A Rational Scheme
for
Conflict Detection and Resolution
in
Distributed Collaborative Environments for Enterprise Integration

Henning Grashoff

A Thesis Submitted in Conformity with
the Requirements of the Degree
of Doctor of Philosophy

Department of Business Computing,
City University,
Northampton Square, London

January 1996

Table of Contents

List of Tables and Illustrations VIII
Acknowledgement X
Declaration XI
Statement of Contribution XI
Abstract XII

1. Introduction

1.1 Preface 1
1.2 Problem Statement 1
1.3 Research Aim 1
1.4 Original Contribution 2
1.5 Research Methodology 3
1.6 Structure 4

2. Distributed Collaborative Environments for Enterprise Integration

2.1 Introduction 6
2.2 Distributed Databases 6
2.3 Distributed Collaborative Environments for Enterprise Integration 8
2.4 Autonomy, Heterogeneity and Integration 13
2.5 Conflict Detection and Resolution in Task Assignment 16
2.6 Rationality in Conflict Detection and Resolution 19
2.7 Information in Enterprise Integration 23
2.8 Evidence, Knowledge and Beliefs of Information Agents
    2.8.1 Knowing and Believing 35
    2.8.2 Belief and Knowledge Sets 37
2.9 Chapter Summary and Conclusion 38

3. Related Research on Conflict Detection and Resolution

3.1 Introduction 40

3.2 Conflicts in Enterprise Integration
    3.2.1 Propositional Conflicts in Enterprise Integration 41
    3.2.2 Other Kinds of Conflicts in Enterprise Integration 45
    3.2.3 Completeness of Conflict Detection in Enterprise Integration Environment 47

3.3 Conflict Detection and Resolution in Distributed Collaborative Environments for Enterprise Integration
3.3.1 Master Model and Unified Approaches ... 48
3.3.1.1 The Master Model in CARNOT ... 48
3.3.1.2 The Unified Approach to Model Integration ... 50
3.3.1.3 Conclusion on Tight Conflict Free Integration ... 53
3.3.1.4 Tight Integration with Inconsistencies Assumption ... 56
3.3.2 Enterprise Integration Based on Federated Architectures ... 58
3.3.3 Integrating Non Persistent Data in Enterprise Integration Environments ... 62
3.3.4 Mediators in Enterprise Integration Environments. ... 64
3.3.5 Conflict Management by Modelling Human Decision-Making ... 65
3.3.6 Conclusion and Summary of Architectural Assumptions ... 66

3.4 Conflict Detection and Resolution in Distributed Artificial Intelligence (DAI)
3.4.1 Introduction to DAI and Enterprise Integration ... 67
3.4.2 Partial Global Planning and Derivatives ... 68
3.4.3 Mainstream Distributed Artificial Intelligence ... 73
3.4.4 Conflict Detection and Resolution in Distributed Planning ... 76
3.4.5 Task Sharing and Result Sharing ... 80
3.4.6 Conclusion Conflict Detection and Resolution in DAI ... 82

3.5 Uncertainty Management in Artificial Intelligence
3.5.1 Introduction ... 83
3.5.2 Quantitative Methods to Uncertainty Management ... 84
3.5.3 Qualitative Methods to Uncertainty Management
3.5.3.1 Non-monotonic Uncertainty in Truth Maintenance Systems ... 89
3.5.3.2 Belief Revision ... 93
3.5.3.3 Uncertainty Management Based on Decision-Making Theories ... 97
3.5.3.4 Uncertainty Management by Argumentation ... 99
3.5.4 Conclusion on Uncertainty Management ... 103

3.6 Conclusion on Related Research ... 104

3.7 Chapter Summary ... 105

4. Theoretical Framework for Conflict Detection and Conflict Resolution

4.1 Introduction ... 107

4.2 A Rational Scheme for Conflict Detection and Resolution ... 108

4.3 Evidence in Law and Information Integration ... 112

4.4 Relevance and Admissibility in Conflict Detection
4.4.1 Relevance and Admissibility ... 114
4.4.2 Overview Conflict Detection Framework ... 118

4.5 Credibility And Weight Of Evidence In Conflict Resolution
4.5.1 Credibility and Weight 119
4.5.2 Domain-Specific Problem-Solving 120
4.5.3 Scientific, Domain-Specific Heuristics 122
4.5.4 Domain-Independent Evaluation 124
4.5.5 Overview Conflict Resolution Framework 128

4.6 Conclusion 129

4.7 Chapter Summary 130

5. A Mechanism for Conflict Detection and Resolution in Enterprise Integration Environments

5.1 Introduction and Design Methodology
5.1.1 Introduction 131
5.1.2 Design Methodology 132

5.2 Conflict Detection - The Gathering Phase 133
5.2.1 Object Identification
5.2.1.1 The Concept of Object Identification 135
5.2.1.2 Limitations Of System-Generated Surrogate Based Identifiers in Enterprise Integration Models 137
5.2.1.3 Deficiencies of Generalised Relations Between Information Sources 139
5.2.1.4 A Novel Object Structure for Enterprise Integration Environments 141
5.2.2 Gathering of Candidates 149
5.2.3 Gathering of Evidence 150
5.2.4 Classification
5.2.4.1 Introduction to Classification 153
5.2.4.2 Classification of Conflict Between Candidates 154
5.2.4.3 Sameness Predicates 158
5.2.4.4 Classification of Sameness of Candidates 166
5.2.4.5 Classification of Evidence 167
5.2.5 Summary Gathering Phase 169

5.3 Conflict Detection - Syntactic Phase 170

5.4 Conflict Detection - Semantic Phase
5.4.1 Object Correspondence and Strengthening Sameness 172
5.4.2 Concept Correspondence 178

5.5 Conflict Detection - Admissibility Phase 183

5.6 Summary Syntactic, Semantic and Admissibility Phase 185

5.7 Conflict Resolution - Credibility
5.7.1 Gathering Credibility Estimates 186
5.7.2 Limitations of Credibility Estimates 190

5.8 Conflict Resolution - Domain-Specific Problem-Solving 192

5.9 Conflict Resolution - Scientific, Domain-Specific Heuristics 197

5.10 Conflict Resolution - Domain-Independent Evaluation - Reliability
5.10.1 Ranking 200
5.10.2 New Alternatives 206
5.10.3 Reliability - Negotiation 208
5.10.4 No Solution 210

5.11 Summary Conflict Resolution 212

5.12 Implementation Concept 213

5.13 Conclusion 214

5.14 Chapter Summary 215

6. Evaluation and Discussion

6.1 Introduction and Evaluation
6.1.1 Introduction 217
6.1.2 Evaluation Methodology 218

6.2 The Enterprise Integration Environment
6.2.1 Overview 220
6.2.2 Cafeteria Integration Environment Scenario 223
6.2.3 Information Agent Model
6.2.3.1 The Agent's View 224
6.2.3.2 The Global Agent View 226
6.2.3.3 Agent Knowledge 228

6.3 Evaluation of the Integration Environment 234

6.4 Demonstrator For Conflict Detection and Resolution
6.4.1 Introduction to the Demonstrator 236
6.4.2 Information Retrieval 238

6.5 Case Study
6.5.1 Gathering of Candidates 240
6.5.2 Gathering of Evidence 243
6.5.3 Classification 244
6.5.4 Syntactic Conflict Detection 252

6.5.5 Semantic Conflict Detection
6.5.5.1 Object Correspondence 256
6.5.5.2 Concept Correspondence 259
6.5.6 Admissibility Phase 263
6.5.7 Summary Conflict Detection 264
6.5.8 Credibility 267
6.5.9 Domain-Specific Problem-Solving 271
6.5.10 Scientific, Domain-Specific Heuristics 274
6.5.11 Reliability - Ranking 276
6.5.12 Reliability - New Alternatives 279
6.5.13 Reliability - Negotiation 281

6.6 Critical Evaluation of Case Study 283

6.7 Conclusion on Evaluation 286
6.8 Chapter Summary 287

7. Concluding Remarks

7.1 Summary 288

7.2 Results and Contributions 290

7.3 Limitations 292

7.4 Future Work 295

Appendix

Appendix A: Distributed Artificial Intelligence

A.1 Example Approaches in Distributed Artificial Intelligence Research 298
A.2 Conflict Detection and Resolution in Distributed Knowledge-Based Systems 304

Appendix B: Object Identity

B.1 Introduction 307
B.2 Identity in Information Systems 307
B.3 External Identification 315
B.4 Integration of Heterogeneous Independent Notions of Identity 317
B.5 Summary and Conclusion 323

Appendix C: The Lews System of Counterparts 324

Appendix D: Information Systems in the Integration Environment 'Cafeteria'

D.1 Object-Oriented Database 'ProductionDB' 326
D.2 Relational Databases 'BookkeepingDB' and 'MaterialDB' 329
D.3 Expert System 'MarketingEXP' 333
D.4 Standard Software System 'ProductionMgmt' 335
D.5 Co-ordination System 'RobotMgmt' 339
D.6 Enterprise Model 342

Appendix E: Overview Implementation Concept 345

References 347

List of Tables and Illustrations

Tables

Table 1: Implicit Conflicts 41
Table 2: Partial Global Planning and Derivatives 72
Table 3: Conflict Detection in DAI 73
Table 4: Distributed Planning 76
Table 5: Conflict Detection Framework 118
Table 6: Conflict Resolution Framework 128
Table 7: Object Correspondence Example 2 270
Table 8: Distributed Knowledge-Based Systems 305
Table 9: Major Notions of Identity in Information Systems 307
Table 10: Overview Integration of Heterogeneous Notions of Identity 317
Table 11: Local Level Object Identification in CARNOT 319

Illustrations:

Figure 1: Architecture Distributed Collaborative Environments for Enterprise Integration 9
Figure 2: Autonomy and Heterogeneity 14
Figure 3: Information in Enterprise Integration Environments 23
Figure 4: Example Conflict Hierarchy 78
Figure 5: Agent Communities within Enterprise Integration 80
Figure 6: Evidence in Integration Environments 113
Figure 7: Classification of Object Sameness 167
Figure 8: Gathering Phase 169
Figure 9: Ranking of Object Sameness 173
Figure 10: Syntactic, Semantic and Admissibility Phases 185
Figure 11: Conflict Resolution 212
Figure 12: Demonstrator Implementation 221
Figure 13: Agent View 224
Figure 14: Global Agent View on the 'Cafeteria' Environment 226
Figure 15: Agent Knowledge Managed by the Demons Program 228
Figure 16: Overview Window in the Agent Knowledge 232
Figure 17: Demonstrator Windows 237
Figure 18: Poet Developer - Class Overview 326
Figure 19: PoetDB Application Interface 327
Figure 20: Overview ProductionDB 328
Figure 21: BookkeepingDB Interface 329
Figure 22: MaterialDB Interface 330
Figure 23: Table Employee and Address Interface 330
Figure 24: Overview BookkeepingDB and MaterialDB 332
Figure 25: MarketingEXP Interface 333
Figure 26: Data Flow Diagram Marketing Expert 334
Figure 27: ProductionMgmt Interface 335
Figure 28: Data Flow Diagram Production Management 337
Figure 29: Interface RobotMgmt 339
Figure 30: Data Flow Diagram Robot Management 340
Figure 31: Enterprise Model Interface 342
Figure 32: Conflict Detection 345

Acknowledgement

This PhD work was supported by an internal grant of the School of Informatics, City University. I am grateful for this financial support which has allowed me to undertake this research.

I would like to thank my supervisor Dr. Allen Long for vigorous assistance in this research. Furthermore, constructive support from my second supervisor Mr. Ian Neale is gratefully acknowledged.

I am deeply indebted to Dr. Gordon Rugg for his assistance in writing up the research. I would further like to thank Paul Collins and Ivor Benjamin for helping with the grammatical format of this thesis.

Many fruitful discussions with many members of the academic staff within the School of Informatics have significantly helped me to conduct this research.

Declaration

The author grants powers of discretion to the University Library to allow this thesis to be copied in whole or in part without further reference to the author.

Statement of Contribution

This disclaimer is to state that the research reported in this thesis is primarily the work of the author and was undertaken as part of his doctoral research.

Abstract

A typical enterprise may have large numbers of information sources such as data stores, expert systems, knowledge-based systems, or standard software systems. These may need to be integrated so that, for example, an application program or a decision maker can access information from all these sources. Such architectures are generally called 'Distributed Collaborative Environments for Enterprise Integration'.

A general problem in these enterprise integration architectures is that information from heterogeneous, pre-existing sources may be obsolete, incomplete, incorrect or, for many other reasons, contradictory. Thus, conflicting results may occur when the same information is requested from semantically related sources. A mechanism is required to detect and resolve these conflicts in a way that is rational to any potential client of the integration environment.

This thesis lays open the design of a general mechanism for conflict detection and resolution that enables intelligent information agents to reason about contradictory information from pre-existing, heterogeneous and autonomous sources. The mechanism's theoretical basis is a **framework** that is drawn from evidence law, which shares some fundamental commonalities with conflict detection and resolution in enterprise integration environments.

Conflict detection opens with **gathering** the results collected by the information retrieval process. These results may have justifications or certainty assessments attached to them. Furthermore, it identifies whether and how these results are conflicting.

Most conflicts in enterprise integration assume object correspondence, for example, that two conflicting candidates are concerned with the same thing. Furthermore, it is examined if concepts used in both results semantically cohere to the same thing. Hence, detection further reviews the information integration process in order to detect that conflicts are a mere **syntactic** (language and translation), or a **semantic mismatch or misinterpretation** of concepts. Finally, the concept of **admissibility** ensures that the results, in principle, are worth considering, for example, according to business rules.

The design of a conflict resolution mechanism is based on a rational scheme for judging the weight of conflicting results. First, the agents assess the reliability or **credibility** of an information source. **Judgement based on the weight** of conflicting results is first applied to any available, domain-specific, resolution strategies. Second, the agent applies any 'general scientific' resolution strategies that are not specific to one domain. When no domain-related expertise can solve the conflict then the agent can only judge on domain-independent evaluation criteria such as the results' reliability. A scheme is sketched out for judgement based on the reliability of conflicting results, involving three steps: Ranking the conflicting results according to their reliability; Ways to redefine conflicting results; and Heuristic decision-making.

The evaluation includes a computational implementation of an enterprise integration environment incorporating a model of an information agent. An example is realised in this environment. The conflict detection and resolution mechanism, and interfaces to each integrated source, are implemented in Visual C++. A case study is conducted on this scenario to evaluate each conflict detection and resolution step. Furthermore, this illustrates both the advantages over existing approaches and the limitations.

1. Introduction

1.1 Preface

This research has been developed from an interest in distributed databases, enterprise integration architectures, and open information systems.

Within this enormously diverse field one small but key problem is addressed: One that is also central to enabling the integration of pre-existing, autonomous information sources.

1.2 Problem Statement

A typical enterprise has large numbers of heterogeneous information systems such as databases, expert systems, or standard software systems. These may need to be integrated in order to provide a large information base for decision makers or application programs. Information agents typically manage the integration of information sources in a collaborative form. Such systems are generally called Distributed Collaborative Environments for Enterprise Integration (DCEEI).

However, an epistemic audit in these environments may produce incomplete, obsolete, incorrect, or otherwise inconsistent results. In other words, information from heterogeneous, autonomous and pre-existing sources may be conflicting. A mechanism for information agents is required to detect and resolve these conflicts in a logical and systematic way that is rational to any potential client of the integration environment. Existing approaches to enterprise integration fall short in providing such a rational mechanism for

(i) detecting conflicts, and

(ii) applying all available resolution strategies to a conflict.

1.3 Research Aim

First, a theoretical basis has to be defined on which a rational framework for conflict detection and resolution can be designed. This framework is then applied to the DCEEI. In other words, all available means for detecting conflicts and strategies to resolve conflicts are embedded in this framework. A general mechanism is laid out that enables intelligent information agents to resolve conflicts about contradicting information from pre-existing, heterogeneous and autonomous sources.

1.4 Original Contribution

Earlier versions of this research have been discussed on workshops and conference, e.g. [GRA92] [GRA93] [GRA94a] [GRA94b] [GRA95a] [GRA95b]. However, this thesis presents the definitive treatment of this research approach with the following contribution:

A **master framework** for a conflict detection and conflict resolution mechanism for enterprise integration is elaborated. Within this structure the following elements were designed:

- The detection framework requires a formal representation that is based on commonly used, mainstream object structures. However, it includes a novel object identifier for heterogeneous integration environments. This novel identifier is the basis for an automatic **mechanism to revise assumptions about the correspondence of objects** across heterogeneous information sources.

- Furthermore, a scheme is designed for identifying whether conflicts are merely **syntactically conflicting** or **semantic mismatch (misinterpretation) conflicts**.

- Within the conflict resolution framework a **specification of rational judgement** of conflicting results is provided that takes multiple alternative resolution strategies into account. It is based on the premise that domain-specific resolution strategies are applied first, followed by more general resolution procedures. This research shows that this principle, which is used in distributed planning, is also appropriate in enterprise integration. A global scheme for **judgement on the reliability of information sources** is sketched. This framework is composed of ranking conflicting results with respect to their reliability, redefining the reliability assessments, and finally proposing a compromise.

The result is a novel design for a **mechanism** to detect and resolve conflicts in DCEEI.

1.5 Research Methodology

The **master framework** for conflict detection and resolution is based on principal concepts in evidence law. It is shown that this field provides a good structure for evaluating evidence, justifications or reasons for potentially conflicting information.

The **mechanism to revise assumptions about the correspondence of objects** across systems is based on a novel object identifier that is an extension of object-oriented structures, for example, as used in most DCEEI. It is more specific about object sameness than existing approaches because it incorporates the identification that objects actually have within their sources of origin.

The scheme for detecting **syntactic** or **semantic mismatch (misinterpretation) conflicts** has been designed by reviewing enterprise modelling and schema integration procedures.

The **specification of rational judgement** with possibly multiple strategies is based on experiences with domain-specific resolution mechanisms in distributed planning systems. **Judgement on the reliability of information** is grounded on existing research in enterprise integration and the wider field of distributed artificial intelligence. This is extended by fundamental observations in social science and value judgement. Composing these approaches produced a scheme including ranking reliability, redefining reliability estimates and their ranking heuristics, and negotiating a compromise.

The design of a conflict detection and resolution **mechanism** is built on the theoretical framework and a synthesis of existing research from related fields.

1.6 Structure

The overall structure of this thesis is divided into five main sections, which incorporate all seven chapters:

(1) An **introductory** part covering the introduction to this thesis (Chapter 1) and an introduction to Distributed Collaborative Environments for Enterprise Integration (Chapter 2).

(2) The **related research** on conflict detection and resolution is provided in Chapter 3).

(3) A **theoretical** Chapter 4 follows the analysis of existing approaches.

(4) The design of the conflict detection and resolution **mechanism** (Chapter 5) is based on the previous chapter's theory.

(5) The **evaluation** of the approach in Chapter 6 leads to the **conclusion** of the thesis in Chapter 7.

Each chapter is opened by a brief introductory overview. A short summary is provided at the end of each chapter.

1. Introduction

Chapter 1 is deliberately very brief to provide an immediate overview of the problem statement, the aim, the contribution, and the methodology of this research.

Chapter 2 covers the basic terminology, introductory architectural issues, and a brief discussion of autonomy versus heterogeneity in integration environments. This is followed by a description of the problems resulting from conflicting information in enterprise integration environments. The concept of rationality is briefly discussed followed by a systematic analysis of the literature on Distributed Collaborative Environments for Enterprise Integration (DCEEI). This leads to a brief clarification of the terms 'agent belief', 'agent knowledge', and 'evidence of information agents'.

2. Related Research

Chapter 3 presents existing research on conflict detection and resolution. First, conflicts are defined. This is followed by an analysis of conflict detection and resolution approaches in existing DCEEI and their predecessors. The wider field of Distributed Artificial Intelligence (DAI) and the particular area of uncertainty management within DAI appear to have the potential to contribute to the detection and resolution of

conflicting information. The section concludes by summarising the contributions and shortcomings of existing approaches.

3. Theory

Chapter 4 covers a theoretical basis for the design of a conflict detection and resolution mechanism. Evidence law is used as a basis for a general framework. This principle structure incorporates existing information, detection mechanism and resolution strategies in enterprise integration. In addition, enterprise modelling theory contributes towards a conflict detection framework. The conflict resolution structure is based on experiences in distributed planning. A general scheme for judging the reliability of conflicting solutions is adopted from existing research in enterprise integration and the wider field of distributed artificial intelligence, social science and value judgement.

4. Mechanism

This framework is used in Chapter 5 to design a mechanism. The basic design principles are outlined followed by a description of the mechanism for conflict detection and conflict resolution. This includes the design of the novel object identifier and sameness predicates. It is subsequently shown how this mechanism is incorporated by a conceptual implementation in the form of a tree schema.

5. Evaluation and Conclusion

Chapter 6 opens the research evaluation by specifying the evaluation method. The evaluation has two phases:

- The implementation of an example DCEEI with multiple heterogeneous information sources, and a computational implementation of an agent model that includes the conflict management mechanism and the Agent Knowledge;
- A case study is undertaken to evaluate each step of the conflict detection and resolution mechanism, the functionality, strength, and weakness of the approach.

This section closes with a critical assessment of the evaluation.

The final chapter 7 briefly summarises this research, its contributions, and limitations in a comprehensive overview. An important result of this research is the identification of potential future work.

2. Distributed Collaborative Environments for Enterprise Integration

2.1 Introduction

This Chapter will provide a basic introduction to Distributed Collaborative Environments for Enterprise Integration (DCEEI) to outline the functionality of these systems, the information agents within these environments, and to clarify some basic terminological issues.

The next section will demonstrate that distributed databases have contributed to DCEEI. The latter are introduced in Section 2.3 It is subsequently shown how enterprise integration depends on the autonomy and heterogeneity of the integrated sources. Section 2.5 describes the problem of conflict detection and resolution in task assignment within enterprise integration. In Section 2.6 rationality in DCEEIs is briefly discussed. This is followed by a systematic summary of the information that is available for such integration environments. Finally, a short introduction into epistemic attributes of information agents is produced.

2.2 Distributed Databases

> "A distributed database (DDB) can be defined as a logically integrated collection of shared data which is physically distributed across the nodes of a computer network. A distributed database management system (DDBMS) is the software to manage a distributed database in such a way that the distribution aspects are transparent to the user" [[BEL92]p.44].

DDBs were initially homogeneous in that they integrated database management systems with the same data model and language (e.g. System R*[WIL82]). Heterogeneous distributed databases integrate data from databases that, for example, use different data models and languages. The SIRIUS-DELTA system [LIT82], for example, provides such an integration that includes a translation mechanism between the different data models and languages.

Decentralised and loosely coupled distributed database systems were introduced as Federated Databases [HEI85] and Multidatabases [LIT90]. These systems are not based on one management system to integrate distributed databases but distributed

management systems that interact directly. Federated Database Management Systems (FDMS) and Multidatabases Systems (MDS) can provide loosely coupled integration when multiple partial global schemata (also called views) exist at each node instead of one uniform global schema. They enable management systems of any source in the distributed system to directly exchange data and conceptual schema information.

Thus, such management systems operate on two levels. Each management system manages its data source, which is called its local level operation. In addition, database management systems operate on a global level in that they:

- Exchange conceptual schema information to describe the information that they are willing to exchange;
- A management system can use this conceptual information from other sources to identify what and where information can be requested;
- Each management system can request data from other management systems, which would then request the data from their local source; and
- Return results in response to that data request.

MDS and FDMS research provide a basic structure for DCEEI, in particular the concept of local and global level management functionality. These management systems are the predecessors of information agents as will be shown in the next section. For a detailed discussion of distributed database management systems see Ceri and Pelagatti [CER84] or Bell and Grimson [BEL92].

2.3 Distributed Collaborative Environments for Enterprise Integration

> "Integrating 'structured text' as found in conventional database systems is only part of the (distributed) information handling picture. Accessing other kinds of data and 'knowledge' is also required, and this adds an additional dimension to the problem" [[BEL92]p. 11].

Open distributed ultra concurrent systems [HEW91] are information systems that span multiple, heterogeneous information sources. These systems are typically integrated by intelligent agents (also called information agents [BAR94a] and Resource Agents [BRO89]). They have been called actor architectures because they are based on multidatabase or federated database principles [CAW92b]. One difference from the latter, however, is that the global operations in these actor architectures are provided by intelligent agents. These agents take the global level activity of traditional database management systems in FDMSs or MDSs. In other words, a local management system is used to manage the local information source. An agent is assigned to one particular information source, e.g. a database or a knowledge-based system, and its management system. This agent provides the following services:

- The agent holds and manages meta information or conceptual descriptions of the information available from the integrated source.
- An agent interacts with its source at least in the form of an information repository. In other words, the agent interacts with the integrated source in that it requests information from the information source, and receives results. In principle, agents may cooperate with an information source much more closely.
- Agents within this architecture may exchange information about the information they are willing to share with other agents. Each agent builds a 'global view' of the distributed system consisting of all the information it can access (also called partial global schema).
- It may be the case that an agent has incomplete information about its own information source and the capabilities of the other sources integrated in the distributed system.
- Agents can cooperate with other agents in solving complex tasks. (communication, exchange of justifications for sub-solutions, the assignment of sub-tasks from one agent to another, etc.).

- One information agent manages a complex information-intensive problem as a 'managing' agent. It is appointed to co-ordinate a complex task and decompose it so that other agents contribute according to their abilities [PAP92a].

A federated or actor architecture is a distributed system that enables flexible integration of highly heterogeneous sources by homogeneous agents. In other words, the information sources may be any variety of systems including data stores, expert systems, or standard software systems. They may have different problem-solving strategies, data and knowledge representations, languages, goals, etc. The only criterion for their integration in the sharing environment is that an information agent must be able to interact with this source. The agents themselves are homogeneous in that they use the same knowledge representation, and communicate with each other by the same protocol and in the same language. Actor architectures are particularly applicable when large numbers of agents cooperate and their expertise is largely distinct [CAW92b]. Hence, the actor architecture is very suitable for enterprise integration environments. An example of an open information system based on an actor architecture with intelligent agents is shown in Figure 1 (such a system is a Distributed Collaborative Environment for Enterprise Integration [PAN91a] [BAR94b] [BAR94a]).

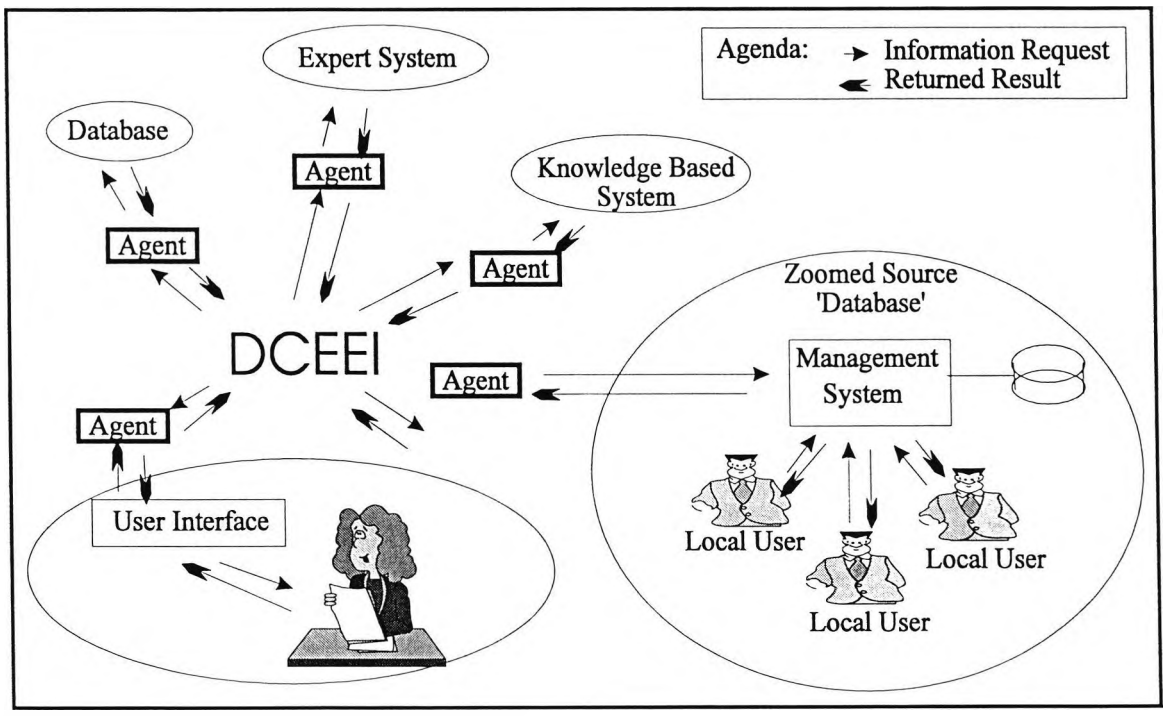

Figure 1: Architecture Distributed Collaborative Environment for Enterprise Integration

First, some differences between distributed databases and open information systems need to be clarified.

> "Today corporate computing environments are heterogeneous, containing many independent information resources of different types, such as a database management with its database, an expert system with its knowledge-base, an information repository, or an application program" [[HUH92]p.38].

If the heterogeneous information sources cannot all be replaced by one homogeneous system, then an environment is required that integrates the heterogeneous sources. Furthermore, open information systems integrate the 'user' as a requester of information as well as an information source [BRO92] [PAN91a] [COE93a] [NOR94]. A decision maker, for example, may be a user who requests information from the integration environment. The information agent integrating the user is specialised in that it knows how to interact with that user [PAN91a] [CAW92c]. Furthermore, it may be tailored to the needs of the 'user' or its expertise. The Aide de Camp project, for example, has shown how the focus of an agent can be tailored to the interests (or expertise) of the human user [COE93a].

A user of the integration environment may also function as an information source. For example, in an engineering environment a controller may provide information on the current stock of a given product A. This user may have an on-line terminal and he may be able to answer specific queries sent to him. If the environment needs to gather information on product A then it could request this information from the 'user'. In simplest terms, the user's expertise may be described by a schema similar to database schemata.

Furthermore, applications may require information from multiple information sources, or multiple applications may need to exchange information via the integration environment [HUH92][BRO89]. Any such applications or users of the integration environment are henceforth called 'clients' of the integration environment.

Two key issues for the practical implementation of the enterprise integration environments are:

1. Communication (between agents, and between agents and their sources);
2. Meta information on where to find information.

The second issue has been briefly addressed by describing 'global views' that information agents build and that allow them to identify what information is available throughout the environment. Section 2.7 will further demonstrate how an information agent can classify this information. Section 3.3 describes enterprise integration

environments including the different ways they manage global views (e.g. in enterprise models).

In enterprise integration environments the inter-agent, and agent-to-source communication are typically based on languages such as the Knowledge Query and Manipulation Language KQLM [CHA92] [FIN94a] [FIN94b] [NEC91] . For example, each KQLM message is composed of a 'content layer', wrapped by a 'message layer', which is embedded in a 'communication layer':

Content Layer: Each agent communicates with the local source it integrates by using the language of this source. These source to agent languages include database languages, e.g. ANSI SQL, or C++, or any other language supported by the local source (e.g. Prolog). To communicate with knowledge sources agents may use languages such as KIF [GIN91], LOOM [MAC91], or CLASSIC [BOR89]. The latter are also typical examples of languages used for inter-agent communication. Moreover, homogeneous information agents, in principle, can apply any language that is rich enough to express the communicated information. Hence, the propositional calculus introduced in Section 3.2.1 and then developed further throughout the design of the conflict management mechanism in Chapter 5, could be used as the inter-agent, and agent-to-source language.

Message Layer: This layer covers the content layer, in that it specifies the language that is used, the kind of speech act, and the ontology on which the content is based. The 'speech act' can be of the kinds 'Content' or 'Declaration'. Content messages contain queries, results in return of a particular query, or error messages in result of a query. Declaration messages are used to register agents, exchange schema objects and their availability, etc. Multiple agents may use different ontologies. These may, for example, be specified in different enterprise models.

Communication Layer: This outer wrapper of a KQLM message identifies the sender, the recipient, and the communication type such as synchronisation or asynchronous. The communication layer is the lowest level of KQLM and is based on a network protocol such as TCP/IP [CHA92].

This three-layered protocol facilitates the communication between agents, and between agents and their sources. The minimum physical link between the communicating partners (their computer notes) is a TCP/IP protocol. In addition, the sources need to have a common application language (in the content layer) such as KIF [GIN91] .

Further detail on inter-agent communication, and agent-to-source communication, is also shown alongside the KQLM description in [CHA92].

Agents in the open systems are intelligent in an Artificial Intelligence sense. Artificial Intelligence (AI) has been proposed, e.g. by Hewitt and Inman [HEW91] or Pan and Tenenbaum [PAN91a], as a support for the agents in open information systems. Agents can use intelligent techniques to integrate an information source. Furthermore, information agents can use intelligence to interact with each other as described in the field of Distributed Artificial Intelligence (DAI).

> "Distributed Artificial Intelligence is the sub-field of AI that has, for over a decade now, been investigating the knowledge and reasoning techniques that computational agents might need in order to participate in societies" [[DUR92]p.858].

One example of how agents may make use of intelligent behaviour is the task of integrating inconsistent information. This task may include decisions on what information is believed to be correct or incorrect.

Open information systems are therefore a subset of 'Intelligent and Cooperative Information Systems' (ICIS).

> "Such systems involve information agents - distributed over nodes with a common communication network- which work together in a synergetic manner by exchanging information and expertise, coordinating their activities and negotiation how to solve parts of a common, information-intensive problem" [[PAP92a]p.169].

'Information-intensive problems' emphasise the integration of information and not the problem-solving structure. The information is integrated from multiple sources that may be semantically related. In other words, there are cases where information can be obtained from more than one source. Agents therefore have to manage potentially conflicting and inconsistent data in an ad hoc manner [HEW91]. A conflict detection and resolution mechanism is required that enables the agent to detect these inconsistencies and possibly resolve them.

In summary the term **'Distributed Collaborative Environment for Enterprise Integration'** (DCEEI) [BAR94a] will be used for open, distributed information systems, managed by collaborating, intelligent agents for information-intensive problem-solving. These systems implement information-sharing and are also called 'information-sharing environments'. Section 3.3 will present a brief survey of existing DCEEI research, and their conflict detection and resolution capabilities.

2.4 Autonomy, Heterogeneity and Integration

The design of a Distributed Collaborative Environment for Enterprise Integration (DCEEI) depends on the autonomy and heterogeneity of the integrated information sources. This will be demonstrated by showing that autonomy and heterogeneity are interrelated which, as a result, has an impact on the close way in which information sources may be integrated.

Autonomy in distributed databases is typically defined on the basis of the following four requirements [SHE90]:

1. 'Association autonomy' is the most basic requirement. It stands for a system's ability to choose if and by how much to associate with other systems or a distributed management system.
2. 'Communication autonomy' measures the degree to which a system has to communicate with other systems or any distributed management system.
3. 'Execution autonomy' enables a system to decide which tasks to execute. This includes the ability of a system to handle any of its tasks independently of any interference from a global or distributed management system. For example, the global system should not influence the command order of the local management system.
4. 'Design autonomy' is concerned with the ability of any system to change its design. The design of a database is, for example, its conceptual description or database schemata. Furthermore, the local design is concerned with any specification of how to manage the information systems including naming, data representation, syntactic and semantic interpretation, etc.

These autonomies are closely interrelated. For example, association and execution autonomy are closely related in that a system that refuses to execute tasks from other systems, executes its association autonomy. It has been shown by Baker *et al.* [BAK92] that the above definition is applicable not only to distributed databases, but also to distributed agent systems in general.

DCEEIs integrate multiple information sources in a sharing environment which will inevitably restrict the autonomy of the integrated sources. For example, if a source agrees to collaborate in any form with its information agent then its association autonomy is restricted.

Local autonomy is much more restricted, for example, by the enforcement of consistency, concurrency control or transaction management in distributed systems:

> "Local autonomy affects the classical requirements for consistency, congruency control, and transaction management" [[LIT90]p.269].

In distributed databases, for example, distributed updates may involve multiple databases that need to commit the update concurrently to ensure consistency.

Heterogeneity in distributed databases is typically defined by variations in data models, data languages, and database systems [HSI92]. In Artificial Intelligence heterogeneity is concerned with "variations in the nature of the data as well as hardware and software platforms" [[JAG92]p.47]. These variations include different knowledge representations, problem-solving capabilities, reasoning mechanism, etc.

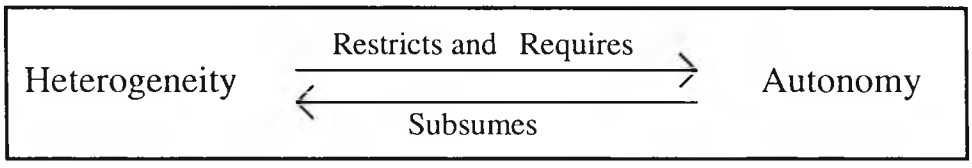

Figure 2: Autonomy and Heterogeneity

However, the concepts of heterogeneity and autonomy as described above are interrelated. Baker *et al.* [BAK92] have outlined that the following relations may occur in distributed databases and distributed intelligent systems in general (Figure 2):

1. Heterogeneity restricts local autonomy;
2. Heterogeneity requires local autonomy; and
3. Strict autonomy regulations subsume heterogeneity.

The previous paragraph has described how any integration of sources will inevitably restrict local autonomy. Furthermore, integrating heterogeneous information sources becomes potentially more restrictive for the local autonomy due to the inherent differences between the sources. For example, integrating a database and a knowledge-based system may put harsh restrictions on the local autonomy of the database if cross system consistency is enforced. Updates to the database may only be possible if they are consistent with the 'facts' in the knowledge-base.

Depending on how heterogeneous information sources are the more complex and difficult it is to integrate these into a tight common concept. For example, two sources with relational data models can be more easily integrated than sources with heterogeneous data models. In the latter case both models use different ways to express

the same concepts. For example, object identity may be implemented differently if based on different notions of identity. A hierarchical database is not able to implement the concept of 'user defined keys' as in relational databases. This heterogeneity of data models requires that their autonomy to implement the identity of the same object must be respected. More heterogeneous systems, such as expert systems and databases, may require even greater respect to their design, execution, and communication autonomy. Hence, system heterogeneity requires a certain degree of autonomy.

In many cases the autonomy requirements of information sources may subsume the problems raised by their heterogeneity. An example might be found in an engineering environment where machines on the shop floor produce information that is also used by other sources. However, such systems will typically not adjust their execution priorities to the other information systems but simply fulfil their role on the shop floor. This is an example of a system that will have very harsh autonomy requirements. However, these requirements may subsume any problems due to the heterogeneity of the sources [BAK92].

In summary, it has been shown that heterogeneity and autonomy are interrelated concepts in information systems integration. Furthermore, heterogeneous systems will always be sensitive to restrictions of their autonomy. The integration of any specific heterogeneous, autonomous sources may therefore be different to any other. In particular the integration may have different degrees of tightness. In principle, integration in a collaborative environment can mean full exploration of the integrated sources with regard to their problem-solving strategy, data and knowledge representation, etc. Furthermore, the integrated sources may be able to commit to the strict control of the global system in respect to execution of tasks at the local sources, changes to the design of the information sources, etc.

However, typical information systems cannot commit to such a restriction or even abolition of their autonomy. Therefore, a heterogeneous, autonomous source may only be integrated as an information repository [HUH93]. In this case, information requests can be sent by the information agent to its source and results may be returned from that system. At the very least, an information agent needs to know how to communicate with the integrated sources, how to request information from that sources, and how to receive results.

2.5 Conflict Detection and Resolution in Task Assignment

Section 2.3 noted that for any given information request issued against a Distributed Collaborative Environment for Enterprise Integration (DCEEI) a manager agent is specified. This agent has to identify which sources can provide the required information. If the request cannot be satisfied by a specific source the multiple sources may be able to provide partial solutions, which together amount to the requested information. In other words, the manager agent may need to decompose and compose complex requests.

In case of either decomposed partial requests or one request, they need to be assigned to the according information agents in the environment. In this way the assigned agents can request the required information from the information sources they integrate. This procedure is generally called task assignment.

Typically, DCEEIs apply a derivative of the contract net approach [SMI80] [SMI81] for task decomposition and assignment. This basic mechanism is enriched with more complex plans such as shown by the Partial Global Planning Algorithm [DUR91a](the algorithm is discussed in Section 3.4.2). An example of how this is applied to enterprise integration is discussed in [PAP92a][PAP91]. One major difference between partial global planning in distributed problem-solving and task assignment in enterprise integration is that DCEEIs are primarily concerned with 'information-intensive problem-solving' [PAP92a](Section 2.3). In other words, the emphasis is on integrating information and not on the problem-solving. The latter is typically a domain-specific activity (see Section 3.4.5 for Task Sharing and Result Sharing in DCEEI).

However, when multiple agents can perform a given task - that is, they can each request the required information from their integrated sources - there, in principle, are two ways to assign this task. Task assignment can be done either by selecting the most adequate agent or by assigning the job to all possible agents. In the first case, the best agent has to be evaluated with the help of heuristics that are typically of the following kind:

- Select the agent that is currently idle [DUR91a];
- Assign the task to the agent with the cheapest bid, e.g. in terms of communication costs [PAN91a].

Such heuristics are typically easy to apply as they provide conflict free task assignment. In addition, the assignment produces very little overhead as there is only one response for every request that is issued.

The problem with these heuristics is that they assume that the results from different agents are consistent. In other words, it has to be assumed that the results are mutually acceptable. For example, agent A and agent B may both be able to provide results on a given query. Agent A is selected because of lower communication costs. This, however, assumes that agent B produces the same solution, or that A is at least equally good. It may, thus, not be the case that one result may be correct and the other incorrect. The question is therefore, 'Is the information in enterprise integration always consistent, and are all results correct?'

> "The information of different parties can be inconsistent, incomplete, and / or incorrect. They are subject to independent outcomes in their operations because of their internal concurrence and the possibility of new information arriving at any time " [[HEW91]p.1409].

Alternatively, any tasks or subtask can be assigned to all possible information agents, that is all agents that can provide the requested information. In this case all possible sources provide results via their information agents. These results may conflict when agents have incorrect or incomplete information.

For example, pre-existing systems can have incorrect or obsolete information if data is inserted incorrectly into a data store. Inconsistencies are also often explained by the concept of different views on the same query. For example, the question, 'How fat is Yogi?' may result in answers that have been correct at different times, or that are correct for different individuals or assumptions.

Furthermore, there are a number of problems that can occur on the integration level. For example, schemata can be integrated incorrectly, or changes may have been made to the information sources that are not known by the integrating agent.

In the last section it was noted that heterogeneous sources, such as integrated in DCEEIs, typically have to be largely autonomous. This means that systems may concurrently and dynamically change with the potential risk of inconstancies between these sources. In other words, a DCEEI that integrates truly heterogeneous sources has to be able to deal with inconsistencies.

It is therefore incorrect to assume consistent information throughout the integration environment:

> **This research is built on the assumption that in an enterprise information system the results of an information request can be inconsistent**

(incomplete, obsolete, or incorrect) so that one agent may have a correct solution and another may have an incorrect solution.

Information agents need to investigate all available alternative results in order to integrate information in a way that is rational to any potential client of the sharing environment (do all that is possible to avoid incorrect results). They need to assign requests to all sources (via their agents) that can possibly produce relevant results. A mechanism is then required that can detect inconsistencies between responses from the sharing environment. Such conflict detection is naturally a prerequisite for conflict resolution. Unless a conflict has been detected there cannot be any resolution.

It follows that the terms 'conflict' and 'rationality' need to be defined for information agents in DCEEI. Agent rationality is, hence, the subject of the following section. Later in this thesis the concept of a conflict will be further defined. These definitions are concerned with conflicts between two propositions, results or partial results.

In other words, more than two results to a given query may exist. For example, the query 'How fat is Yogi' may produce the results A: '18 Stone', B: '20 Stone' and C: 'not 17 Stone'. In principle, these multiple results crystallise down to conflicts between A and B, A and C, and B and C. In other words, an information agent may receive multiple results to a given request. It then gathers these results in some uniform representation (as shown in Section 5.2) and classifies these into pairs of results that may be conflicting. (Section 3.2.1 describes a propositional classification of conflicts and Section 5.2.4 shows how agents can classify multiple results into pairs of conflicting results.). Each pair is then subject to conflict detection and resolution as outlined in this research.

2.6 Rationality in Conflict Detection and Resolution

The opening question to rationality in conflict detection and resolution is: 'Why should information agents be rational at all?'

A possible definition of rationality is:

- "Endowed with reason, reasonable;
- sensible, sane;
- based on, derived from, reason or reasoning;
- not foolish, absurd, or extravagant" [OXF75].

In addition, rationality is concerned with:

- Coherence (e.g. the coherence of goals [JOK95][POL95]);
- Different levels of rationality, e.g. economic rationality [EDM95] [DUR95], or higher levels of goals [EPS95].

Rationality in communities is different to individual rationality [SOL95] in that it concerns itself with the question:

> "Is rational behavior behavior that is rational according to the subject in question or is rational behavior the behavior that is rational according to the observed?" [[SMI95]p.131]

Rationality in enterprise integration is the matter of multiple subjects, of which at least one is an information agent, and at least one is a client. The aspect of rationality in enterprise integration is based on the client. In other words, it is important that the information agent acts rationally from the point of view of any potential client of the integration environment. In the following the concept of Principle and Application Rationality will therefore be introduced [GRA95].

Multiple irrational ways to evaluate conflicting results exist, for example:

- Ignoring the alternatives and judging first come first served, or at random, e.g. [PAP92a].
- Manipulating the conflicting results so that a compromise is reached such as averaging conflicting numerical values, e.g. [SU 91].
- Selecting the solution on another basis such as optimality of the retrieval process, or the response time, e.g. [JAG92] [PAN91a].

Most research in enterprise integration applies similar strategies to avoid the detection of inconsistencies by assuming consistent information throughout the environment and thus:

- Apply heuristics to optimise the retrieval process, e.g., cost evaluation [PAN91a];
- Utilise domain-specific resolution strategies, e.g., credibility of a source of origin and the cost of denying beliefs that were already shared [BAR94a].

However, a rational person (or agent) would **logically and systematically** investigate the results and their evidence. This logical and systematic pursuit produces results that are rational in that all available information is investigated in an appropriate form and complete order. For example, the circumstances of the conflict are analysed, the contributions often need to be concerned with the same thing to be conflicting, results may need to conform with existing business rules and integrity constraints, etc.

Moreover, multiple clients may have different, dynamically changing points of emphasis, priorities, and interests in respect to the integrated information. These are rational in respect to a particular problem or domain. In other words, each client may have a notion of rationality that is not generally shared but is specific to its purpose or application (**Application Rationality**). For example, a sales figure in a business environment is rational for one decision maker if it is manipulated by less than 5 per cent, and rational to another decision maker only if the data has not been manipulated at all.

In contrast, the task of integrating information in enterprise environments has to be rational, in principle, to every client (**Principle Rationality**). Information agents provide a large information base for any potential client. A rational scheme (i.e. a detection and resolution mechanism) and any steps it involves need to be rational to all potential clients that receive the integrated information. In other words, any rational system would come to the same result given the same situation and the same level of knowledge (information) of the conflict and its circumstances.

> "Enterprise integration is concerned with how to improve the performance of distributed organisations and markets. It focuses on the communication of information and the coordination and optimisation of enterprise decisions and processes in order to achieve higher levels of productivity, flexibility and quality. To achieve integration it is necessary that units of the enterprise, be they human or machine base, be able to understand each other" [[FOX92]p.310].

In other words, enterprise integration includes all processes from information retrieval to information use. For conflict detection and resolution, however, information agents need a scheme that is Principle Rational so that it

(i) gathers the results produced by an information retrieval process [PAN91a][JAG92][BAR94a] and language (e.g. KQLM [CHA92]) in the sharing environment,

(ii) to provide rational information integration services to client systems, which may, for example, be concerned with: Human - computer interaction (user interfaces and intelligent retrieval); Human - work flow and process integration (cooperative work); Decision support; They may be application programs or decision makers.

In other words, information retrieval starts with specifying the client's requirements. This is a back-end problem that is concerned with hypermedia interfaces, accessing hypermedia information, providing the back-end with knowledge about the human user, and it is concerned with knowledge about retrieval strategies [CAW92c]. However, retrieval also includes the physical integration of systems, e.g. by applying schema integration, inter language translation, inter source communication. These functions have briefly been mentioned in Section 2.3.

Information retrieval and conflict detection and resolution provide the conceptual integrated by the DCEEI in a Principle Rational way. Following this integration step the information (candidates) may be subject to specific Application Rationality. Application Rationality is implemented not only in software systems or decision makers but is also implemented in systems concerned with presenting information to human users. Furthermore, cooperative work is a subject area that deals with the interaction among humans, their interaction across information systems [COE93a], the interaction between humans and (intelligent) machines or computers [COE92] [FIN93], or with the interaction of (intelligent) sources with each other [CHA90].

An example of such a system is the Aide De Camp approach. This "acts as a buffer between incoming messages and the user"[[FIN93]p.101]. Some routine tasks are provided by the system automatically such as filtering, or storing messages. More complex tasks including replying to standard messages are defined with the help of a language called Task Scripting Language (TSL) [COE92]. It is used to describe tasks that an agent should perform for the human user. This language can also be used for multiple agents to describe cooperative tasks [COE93b]. A task scripted by TSL is compiled into the agent, called an Aide de Camp of a specific user. Hence, the user can

define its notion of rationality by scripting tasks for the agent in the way this specific user prefers it (Application Rational).

Moreover, group decision-making and individual decision-making are areas interdependent of the enterprise integration framework [SHA93] [GOT92] [WON94]. In other words, a decision support system requires information integrated by the sharing environment. It might need meta information about where this information comes from, how it relates to each other (independence assumptions), etc. (Section 3.5.4).

Decision support systems and cooperative work are areas that target the human - computer - process interaction, with emphasis on optimising business goals. Conflict detection and resolution has a different aim in that it is concerned with the integration of information sources to provide a large information base.

2.7 Information in Enterprise Integration

The previous sections have briefly sketched out rational conflict detection and resolution in Distributed Collaborative Environments for Enterprise Integration (DCEEIs). This section will investigate what information is available to agents in enterprise integration environments to undertake this task. Huhns and Singh [HUH92] have outlined a basic categorisation of information in DCEEI, such that:

> "Not just a structural description of the local schemes is used, but all available knowledge, including:
>
> 1. Schema Knowledge, i.e. the structure of the data, integrity constraints, and allowed operations;
> 2. Resource Knowledge, i.e. a description of the supported services such as the data model and languages, lexical definitions of object names, the data itself, comments from resource designers and integrators; and
> 3. Organisational Knowledge, i.e. the corporate rules governing use of the resources" [[HUH92]p.39].

On the basis of this classification an information agent's 'knowledge' (Agent Knowledge) is described in the following overview (Figure 3). The term 'knowledge' may be inappropriate if agents are aware of the possibility of errors in this information. The term 'belief' would be much more appropriate but the term knowledge is used in order to comply with the general literature.

Schema Knowledge (Local and Global)	**Resource Knowledge** (Local and Global)	**Organisational Knowledge**
1. Data Structure and its • *Integrity Constraints* • *Allowed Operations*	1. Environmental Information 2. Services 3. Semantic Matching 4. Extensional Information 5. Comments from Designers, Integrators and Agents 6. Retrieved Information	1. Business Rules *Agent Rules and Procedures e.g. to Integrate The Local Information Source* 2. Decision-Making Knowledge *Agent Rules and Procedures e.g. to Manage Interaction with Other Agents or to Manage the Agent Knowledge*

Figure 3: Information in Enterprise Integration Environments

Schema Knowledge

The ANSI/SPARC three level reference architecture for database systems [ANS75] has the following layers:

- The internal view holds, for example, physical file descriptions of layouts and sizes of fields, and manages the file access such as hashing or indexing.
- The conceptual view holds the logical description of the entire database such as tables and field names in a relational database;
- The external view has a description of the information available to a user of the database. It describes what information is available to that particular user (or group of users) in a possibly abstract way suitable for that particular user.

An information agent is at least a 'user' with a specific external view to the sources it integrates. The external view contains the information that this system is prepared to share with the information agent and, hence, with the enterprise integration environment. This is also called a local participation schema, e.g. by Bell and Grimson [BEL92]. In other words, the agent integrates its source as an 'information repository' (Section 2.3). In principle, however, a system could be integrated more closely if it is prepared to share its entire database and not just a subset (defined in the external view). Furthermore, the source may allow the agent insight into its conceptual view, or even on the internal view. However, the latter internal view on the storage of information is typically of little interest to sharing data.

Furthermore, information-sharing is concerned, not only with integrating databases, but with integrating all kinds of information systems throughout an information-sharing environment. These systems provide their information agents with schemata that are functionally equivalent to conceptual schemata of integrated databases. This principle is implemented in many different approaches. For example, Su and Park [SU 91] have presented information stored in rule-based systems by extending static database schemata with rules. These rules relate multiple objects and show generalisations about the transactions within a source.

> "To extract only global meaningful information, a general rule can be abstracted into an antecedent-consequent relationship among data items and optional and explicit triggering conditions" [[SU 91]p.226].

These generalised rules help to integrate rule-based systems without having to represent every single fact and rule of the system. These meta rules are included between objects in object-oriented schemata [SU 90] or are defined as methods for single objects [PAP92b].

In summary, information sources present a conceptual description of the information they are willing to share throughout the enterprise integration system. This description is here called a conceptual schema, which may include database schemata, schemata with generalisations, other advanced database schemata, entity relationship diagrams, frame systems of knowledge-bases, or process models [HUH94].

The **data structure**, described in the previous paragraphs, may be linked to other, circumstantial information such as integrity constraints. Traditionally, **integrity constraints** are programmed into the software that interacts with data stores or databases. These are then implemented into database management systems, and today, integrity constraints are used in distributed database schemata [THO90]. For example, an attribute Cost in a database schema may have the integrity constraint 'costs must be negative,' (Cost < 0); or an attribute Price of the schema object Food may be constrained by 'Prices must be greater than 0' (Price > 0).

Seligman and Kerschberg "described a new approach to maintaining consistency between objects in dynamic, shared databases and copies of those objects which are cached in an application knowledge-base" [[SEL93]p.187]. The basic idea is to install different levels and kinds of integrity constraints on objects when these are exchanged between databases and applications [SEL93]. For example, an object may become inconsistent if:

> "It is deleted completely; It is nil; A particular condition becomes true; It becomes 'n' minutes older than the original in the database; or When an attribute varies by more than a particular percentage" [[SEL93]p.191].

In conclusion, different kinds of integrity constraints can be attached to schema objects in conceptual schemata.

Typically a schema has some information on the kind of **operation** that a given user is allowed to perform on a piece of data, such as 'read only' or 'may be changed.' However, this information is of little interest for conflict detection and resolution. In other words, whether an agent is, for example, allowed to change data in the local source is irrelevant for detecting conflicts between results gathered by the agents throughout the integration environment.

In Section 2.3 the information-sharing environment was described as a global system that integrates local information sources. Data structures from the local sources are represented in conceptual schemata. These have to be integrated in the global system to provide a global view of the integrated environment. However, this may be achieved in

many different ways but most research on DCEEI is based on object-oriented schema such as described in [OXB90] or [PAN91a]. It is beyond the scope of this research to describe the different ways of schema integration. Furthermore, this section only needs to outline the available information, here the data structure, and not the way it is efficiently managed. Further information is included in the implementation of an integrated schema in the computational model of a DCEEIs in Section 6.2.

Resource Knowledge

Schema Knowledge is only one kind of information that is modelled in integration environments. A 'meta model' is a model about one or multiple information sources. It should "not only contain a common catalogue encompassing local database [or other] schemata, but more significantly, it should also specify the intended contexts ... within which individual systems operate and interact with each other" [[HSU91]p.605]. "Information agents cannot operate autonomously unless they have an understanding of the environment they are in" [[BAR94a]p.274]. In other words, Schema Knowledge is only one kind of information, which is concerned mainly with the data structure, and, in addition, Resource Knowledge describes context related information.

Intelligent agents may have knowledge about the integrated source as a whole and its connections to the surrounding world, which is a kind of Resource Knowledge called **Environmental Information**. This includes the communication modalities of a source, characteristics of the information sources such as the kind of system involved (database, expert system, neural network, user interface, etc.), its knowledge representation, data modelling or problem-solving strategies, the size of the information source or its components (e.g. number of tables in a relational database), key objects in an information source such as key columns in a database, or the description of the content of a source [ARE93]. The age of a system, its maintenance procedures, data security aspects may all be used to described an information source's environment.

Su and Park [SU 91] have proposed that rule-based systems may allow for requesting data not just once but repeatedly. In other words, "the trend of a series of values derived by a repetitive processing" [[SU 91]p.234] may produce information about the stability of the result over time. Hence, this is circumstantial, or environmental information on the form in which to request information from a specific rule-based system.

Many information systems have an area of expertise such as finance, medicine, production control, etc. This is not only the case for expert systems or knowledge-based systems but also for other software systems. A database in the finance department, for example, may be the central source for payroll information. This can be seen as its expertise or role. The following presents a basic structure for characteristics of information systems:

> "An enterprise consists of people having relationships between each other like 'subordinate of', 'belong to group', 'work in project'. People also have characteristics like 'is manager', 'has experience', 'is out of the office'. Machines

are not much different. Relationships exist like 'is immediately following," belongs to group,' and characteristics like 'is fastest', 'is reliable', 'is standalone', 'is cheap', 'is shut down'. Processes (or better servers) also follow structuring rules like 'belongs to group', 'uses', and 'is reliable', 'during night only' " [[BUß92]p.389].

In other words, heuristics may exist that group objects or sources according to their characteristics, for example: All systems on the shop floor; or All systems in the marketing department. In addition, to these groups, individual systems, or schema objects may have characteristics such as roles they play within an enterprise, or their expertise.

Another such characteristic is 'authority'. Barbuceanu and Fox define authority as "a kind of priority agents may have concerning the truth of the information they deliver" [[BAR94c]p.4]. In this case the agents honestly assess their authority for providing specific information. Another model of authority has been based on the role concept. "Agents have different authorities depending on the roles they are in" [[BAR94a]p.279]. For example, in a resource planning scenario an agent A may have a higher authority to specify how a given resource R is spent than another agent B, based on agent A's role to plan resource R.

In summary, any one or multiple information systems may have roles, expertise, or authority that is shared by:

1. Abstract objects (e.g. Peter in source D1)
2. Schema Objects (e.g. Employee's Name,)
3. Sources (e.g. the database D1)
4. Groups (e.g. all sources concerned with production, all schema objects concerned with Employee data, all individual objects that are called Peter,...)

In addition, Environmental Information not only describes the environment of the whole information source, but it may also include the circumstances of a particular piece of information. For example, it may be possible to inquire the last update of a particular piece of data retrieved from a database. "This kind of information includes information about constraints, object dependencies, dependencies of object property values (derived attribute values), statistics about object usage patterns, or object access overheads" [[PAP91]p.409].

Domain-specific information may be included that focuses on particular information from an information source. For example, in SHARE design, information is attached to integrated objects including formal and informal design information [TOY94].

In DCEEI, domain-specific information, as described in the previous paragraphs, is often defined by an enterprise model or a common knowledge-base. The enterprise model provides the agent with Resource Knowledge like an external reference model without being part of the Agent Knowledge. These models are discussed in Section 3.3.1 but the basic concept is briefly introduced below:

> An example of a common knowledge-base is CYC [LEN90] which stores large amounts of 'common knowledge' in the form of rules and facts [COL91] [HUH92]. Further, this common knowledge can be extended with expert knowledge where needed. CYC is presenting a model of the world. An enterprise model typically represents only a subset of this world view that describes the enterprise, e.g. MIND [JAG94], MKS [PAN91a] or TOVE [FOX93]. In addition, multiple, possibly overlapping models within an organisation may exist that have different levels of abstraction and different purposes [PET92].
>
> Every object, including data processes, or physical objects, may have a counterpart in the enterprise model. In order to express an agent's Environmental Information strong links are necessary between the agent's schema knowledge and the enterprise model.

Expert domain knowledge may be seen as a Service to information agents. Hence, enterprise models can function as 'expert knowledge' sources to provide concept explanation and matching **Services**. This includes descriptions of results or concepts, and conflict resolution services. For example, the result 'The Beetle is an Automobile' may be described by definitions of the concept automobile in the enterprise model. Furthermore, Collet *et al.* [COL91] have described ways to check concepts for synonyms, or sub type and super type relations (these are described in Section 5.4.2). The CYCCESS project [GUH94] has taken this idea further and provides a Service for checking if concepts, e.g. results from information sources such as databases or spreadsheets, are consistent with the common knowledge stored in the CYC knowledge-base [LEN90]. In Section 5.8 this process is described further.

Another way to apply expert knowledge is the integration of a user interface as described in [PAN91a] (Section 2.7). The integration environment may request information from a

human expert via its interface. Hence, a request is sent to the user interface requiring, for example, the working hours of employee 'Peter'. The human expert, e.g. a production manager, returns a result that is unrealistic, e.g. the result '200 hours a week' that violates the integrity constraint 'Employees can at maximum work 50 hours a week'. In such a case the information agent could simply return the result to the production manager for verification. This is the simplest from in which an expert can verify its own results and thus provides this service to the integrating agent.

Not only an enterprise model, or a human user, but also information sources themselves, may be able to provide a service to information agents. Many sources can describe the reliability of results received from them. A neural network may be able to produce information on the statistical probability (confidence level) of results it produces. Furthermore, knowledge-based systems may give explanations of results they propose. An inference engine may give a justification based on an internal belief network when this is requested [DOY79]. Other systems may include information on the degree of confidence for a belief including probability [PEA93] or the strength of belief for individual grounds [FOX91].

The problem of **semantic matching** can be put as:

> "Data obtained from remote and autonomous data sources often will not match in terms of name, scope, granularity of abstractions, temporal units, and domain definitions" [[WIE92]p.39].

Matching these differences can be done by independent software components [WIE92] or lists that relate objects in one system to objects in another system, e.g. implemented in auxiliary schemata in distributed database systems [BEL92] [KIM91a]. Data matching information is essentially resource bound as it is knowledge about

- which data objects from different sources are equivalent and
- how a piece of information from one source needs to be transformed to gain its semantic meaning in another specific environment.

These matching rules are also called generalisations [HUH92]. They are further described in Section 3.3.1 including their use in connection with enterprise models. Problems with generalisations are discussed in the following of Section 3.3 and in Section 5.2.1.

Information-sharing environments may not have complete enterprise models and thus gain **Extensional Information** from analysing schema knowledge. Information about concepts may be derived from schema information (intentional information) [PAP91].

For example, one may know the concept of a 'Good Employee' by analysing a relational table Employee: A good employee may have to be older than thirty years, have ten years with the company and earn more than fifty thousand dollar. Given an employee X that is not in this table, the following extensional information can be derived: Any person X is a good employee if they fulfil the investigated criteria of the concept (intentional information), e.g. age, years with the company, and earnings.

Comments from administrators, designers, or integrators may relate to the reliability, condition and set-up of information systems. Examples of such comments are 'the database is new,' 'the expert system has been tested,' the 'Knowledge-Based System is checked and adjusted by an expert daily', or the 'database is very reliable'. Comments can also include messages of caution such as troubles recently experienced with a given source. In principle, comments could be any kind of information, advice, or regulation that an administrator (designer or integrator) may want to make.

In addition, to system managers, information agents themselves could also make comments that may be based on their past experience. For example, an agent may make a comment of caution if information from a specific source, e.g. a database, is frequently incomplete. This requires, however, that the agent has some accepted way to determine a reason for a specific comment. For example, it may have a statistical confidence level of ninety per cent that a source produces incomplete results and this may be a good basis to make the comment s 'frequently incomplete' on this source.

Furthermore, information agents may be able to learn as, for example, implemented by Vital *et al.* [VIT91]. These agents give a critique (a vote) on conflicting, alternative results. The result with the most votes is accepted and then 'learned' by that agent. For example, agents may learn that information on product design from the marketing department is often not consistent with other information sources. This learned information may be stored as comments by information agents. Another form of agent comment is described in Kim *et al.* [KIR91b] where agents can give ratings for the quality of the output expected from them. These ratings are a kind of comment on the quality of the information shared via an information agent.

All the information that can be retrieved from information sources, can be regarded as Resource Knowledge (**Retrieved Information**). A result statement that an information agent receives in return for its request is essentially Resource Knowledge. However, a result statement is external information in that it is never stored as Agent Knowledge.

In principle, global Resource Knowledge includes local Resource Knowledge about an agent's own, local source. In addition, the combination of local Resource Knowledge from multiple sources provides global Resource Knowledge. In other words, information agents can share and combine their Resource Knowledge and build global Resource Knowledge bases just as global views for database schemata [BEL92].

New information may, however, be gained when Resource Knowledge from multiple information sources is integrated. For example, incomplete concepts may be synthesised to produce more complex concepts. Thus the concept 'automobile' may be known to include cars and vans according to one source. In addition, another source may know that automobiles include cars, buses and trucks. Synthesising both concepts produces a new, more complex concept of automobiles including cars, vans, trucks and buses.

Organisational Knowledge

Two different types of Organisational Knowledge exist within an information agent: Procedural Knowledge and Problem-Solving Knowledge. "Procedural Knowledge contains the well-thought-out, well-organised, well-tested and well-adapted Organisational Knowledge" [[WOO92]p.211]. An example of a protocol is the contract net "in which agents use specific message types for communication, along with expectations about the impact of a message (that a task announcement will elicit a bid, for example)" [[DUR92]p.861]. Problem-Solving Knowledge "provides assistance .... when the procedural knowledge is insufficient to perform an organisational activity" [[WOO92]p.212]. No generally accepted terminology of Organisational Knowledge exists. Thus, it is defined, henceforth, that all Procedural and Problem-Solving Knowledge (handbook information) is called Organisational Knowledge, which is subdivided into:

- Business Rules;
- Decision-Making Knowledge.

Information agents need some procedural information about how to interact with their information sources or other agents in the system. They have to have procedures for managing their meta information of their integrated sources or about building global views. Procedures are required to enable an information agent to fulfil its roles, which has been briefly outlined in the introductory Section 2.3.

Hsu defines business rules along the following lines [HSU91]:

> Business rules may concern a particular information system (or a subsystem) such that these rules embody triggers, processes, and integrity constraints that an information agent knows about. All conceptual information that is necessary to integrate the local sources is classified as Business Rules or Procedures.

Conceptual information about a source has already been defined as Resource Knowledge. Therefore it is pragmatic to define Business Rules as rules or heuristics that an agent has to integrate information from heterogeneous sources in enterprise integration environments.

The principle difference between Decision-Making Knowledge and Business Rules is that the latter are agent specific, and may vary from one agent to another, and the former are the same for every agent. However, Decision-Making Knowledge also includes procedures for interaction with other agents and the way an agent manages itself (e.g. its

Agent Knowledge). Among other things this covers the conflict detection and resolution mechanism.

Within the integration environment, authority is an important concept for the management of multiple agents' knowledge. Agents need to know if they can accept (believe) information, e.g. Business Rules or integrity constraints from other agents. Regulations on what information it can believe and from which sources are embodied by an agent's Business Rules.

For example, it has been mentioned that sources may have roles and that hierarchies may exist between sources. This information is Environmental Information (Resource Knowledge) and has, for example, been implemented into enterprise integration environments by Pan *et al.* [PAN89] (Section 3.3.1). In principle, comments from other agents with higher authority may be accepted as binding. However the decision of the system administrators or integrators on these matters, an agent's Business Rules will tell it whether or not to accept other agents' knowledge and use it as if it were its own..

2.8 Evidence, Knowledge and Beliefs of Information Agents

2.8.1 Knowing and Believing

The outline of Distributed Collaborative Environments for Enterprise Integration (DCEEI) in Section 2.3 defined the collaborating character of information agents in these systems [BAR94a]. Collaboration in a distributed artificial intelligence sense is concerned with "knowledge and reasoning techniques that computational agents might need in order to participate in societies" [[DUR92]p.858]. The intelligent, collaborative agents are therefore based on a concept for representing their 'knowledge', together with techniques to reason about or with this knowledge.

In principle, agents can represent the information they have about themselves and the world in any format, ranging from simple propositional calculus to complex epistemic models. For example, the symbolic notions of Hintikka [HIN62] include an agent (a) that may know (K) a proposition ($O.R_k$), which composes to '$K_a(O.R_k)$'. Furthermore, an agent (a) that believes (B) the proposition ($O.R_k$) is expressed as '$B_a(O.R_k)$'. In enterprise integration environments the information that agents exchange is typically propositional [BAR94a].

> "Knowledge representation is immediately concerned with reasoning, because an Artificial Intelligence system will almost always need to generate explicitly some of what has been implicitly represented" [[LEV89]p.35].

It follows that a criterion for a suitable knowledge representation for intelligent agents is that it is rich enough to support the necessary reasoning of these agents. This includes, for example, that:

- They need to provide the agent with the flexibility, generality and modularity that is needed to integrate complex information systems and interact within complex enterprise integration environments.
- Epistemic attitudes need to allow the information agents to express the degree of 'confidence' they have in a proposition.

In a general epistemic model the 'degree of confidence' can be expressed by an agent. It may, for example, know a proposition is false, disbelieve a proposition, believe a proposition, or know a proposition. Furthermore, reasoning may include justifications for and possibly against a proposition. An agent may have reasons to believe a proposition and others to disbelieve it.

A sophisticated model for epistemic attitudes of agents revolves around finding a good semantic model of knowledge and belief [HAL85]. Most research applies the definitions that:

- Knowledge is of true things (facts);
- Believing is based on having evidence pro and contra an entertained proposition and that such evidence makes one to overcome the doubts about the proposition [PRI67].

However, many further definitions exist that are nonconvergent, for example:

- Knowledge has to be closed under logical consequence such that "there must not be anything inconsistent about a state of affairs in which [a proposition] q is true and in which I know what I say I know" [[HIN62]p.17] (I know that I know, that I know...);
- Knowledge is learned and not forgotten information[LEM67];
- Knowledge is a one hundred per cent certain belief [KEY21];
- The concept of knowing is distinct from believing where knowledge implies that something is true and believing always implies the possibility of the belief not being true [KNE49];
- Belief could also be based on preferences such as 'feeling sure' or 'feeling certain' about a proposition [PRI67] [PRI69].

Furthermore, for the multiple agent case, common and implicit knowledge need to be defined. For example, it may be said that "a group has common knowledge of a fact p exactly when everyone knows that everyone knows that everyone knows ... that p is true ..... A group has implicit knowledge of p if, roughly speaking, the agents pool their knowledge together [so that] they can deduce p"[[HAL85]p.481].

Very briefly knowledge representation of information agents has been revealed. What is the impact of any knowledge representation on a conflict detection and resolution mechanism for information agents? Conflict management is part of an agent's reasoning and, hence, needs to be expressed in its knowledge representation (model). Whatever the model of the information agent, the reasoning scheme provided by this research should be the same. In other words, the rational scheme for detecting and resolving conflicts in enterprise integration, in principle, is independent of the model an agent uses for its reasoning. Typically, information agents have epistemic models but they could also have any other, non epistemic model, if this is rich enough. This research, for example, will use a basic propositional calculus which is introduced in Section 3.2.1.

2.8.2 Belief and Knowledge Sets

An important question for conflict detection and resolution is whether agents have closed belief, or knowledge sets, and if agents are omniscient.

An information agent has Schema Knowledge, Resource Knowledge and Organisational Knowledge (Agent Knowledge Section 2.7). Resource Knowledge also includes retrieved information from the integrated sources. These sources are typically autonomous; They can change their information without having to declare that to the integrating agent (Section 2.4 Autonomy). Thus, the information agent has an open set of beliefs, which are retrieved from the integrated source and may change dynamically, and concurrently.

The Agent Knowledge that is defined for the agent, in principle, may be closed. However, as Section 2.3 outlined, this is typically not the case in dynamically changing environments with heterogeneous, autonomous sources. Inconsistencies and incompleteness of Agent Knowledge may exist, e.g. data structure information (Schema Knowledge) may be incomplete or out of date [HEW91].

> "Logical omniscience means that agents are assumed to be so intelligent that they must know all valid formulas, and that their knowledge is closed under implication, so that if an agent knows p, and knows that p implies q, then the agent must also know q" [[FAG87]p.491].

According to this definition, information agents are not logically omniscient. They lack a closed belief set (they do not know all valid formulas).

2.9 Chapter Summary and Conclusion

Distributed Collaborative Environments for Enterprise Integration (DCEEI) (Section 2.3) are based on information agents that integrate heterogeneous information sources such as databases, expert systems, standard software systems, or knowledge-bases. Each information agent integrates one information system into the sharing environment. This integration may be tight or loose. In the case of loose integration, an agent may treat a source as an information repository from which it can request information and receive results. DCEEIs perform 'information-intensive problem-solving'. That means agents are mainly concerned with the integration of information from diverse sources and not specifically with problem-solving structures. The latter is addressed by agents in mainstream Distributed Artificial Intelligence.

A systematic summary of all the information available to agents in enterprise integration environments has been provided. It is termed Agent Knowledge and is classified into Schema Knowledge, Resource Knowledge and Organisational Knowledge (Section 2.7).

A given request for information can be issued to any information agent in the sharing environment, which makes the agent the 'managing agent' for this request. This agent may be able to request the necessary information from its own local source. However, it might be necessary to decompose a complex task and assign subtasks to other agents. In other words, the managing agent can request information from its local sources and also from other sources via other agents.

Pre-existing, heterogeneous, autonomous information systems potentially hold conflicting information. In other words, information may be inconsistent where any result can be incomplete, obsolete or incorrect. Multiple pre-existing sources that have operated independently are potentially inconsistent. Furthermore, they may contain information that became obsolete over time, or they may simply contain incorrect information in the absence of an efficient control mechanism. It has been shown that heterogeneity is an inherent character of large information-sharing environments (Section 2.4). Further, the integration of heterogeneous sources requires that these stay largely autonomous. Hence, information in these sources is independent and may be inconsistent.

It follows that a given task, that is an information request, is assigned not only to any single source but rather to all sources (via their agents) that can presumably provide the

requested information (Section 2.5). In this way the agent systematically gathers the information that is available in response to a given request.

Thus, integrating potentially inconsistent information from heterogeneous, autonomous sources requires a mechanism to allow information agents to detect and resolve conflicts. This mechanism receives the results from the information retrieval process and provides information to any potential user of the integration environment. These users are called clients and may, for example, be decision makers or application programs. The resolution mechanism needs to be Principle Rational (Section 2.6) in that any potential client can accept the way this result was produced,

(i) given the same situation, and

(ii) the same level of knowledge of the conflict and its circumstances.

In a case where the resolution is not rational, the result generally would not be acceptable. In principle, it could not be used by any application and decision maker independent of their specific application and point of view (Principle Rationality versus Application Rationality).

Agents in DCEEI typically have epistemic models to represent information they reason with or about. However, the kind of epistemic or other model is independent of the design of a rational scheme for conflict detection and resolution. Moreover, it is important to note that information agents have potentially incomplete Agent Knowledge and may believe information received from their integrated sources. Hence, they typically lack a closed belief (knowledge) set. One consequence is that information agents typically are not omniscient.

It is concluded that conflict detection and resolution is a key issue in integrating heterogeneous, autonomous sources in Distributed Collaborative Environments for Enterprise Integration. The following section will investigate the existing approaches.

3. Related Research on Conflict Detection and Resolution

3.1 Introduction

In this chapter, existing approaches to detecting and resolving conflicts are analysed. A precondition for this analysis is a structure of the kinds of conflicts that may occur in information-sharing environments. Hence, in Section 3.2 different kinds of conflicts and their detection in information-sharing are briefly surveyed.

Section 3.3 will show existing approaches to conflict detection and resolution in Distributed Collaborative Environments for Enterprise Integration (DCEEI). These systems integrate persistent and transient data from any source throughout the enterprise. They are based on intelligent information agents and are inherently distributed. This discussion and survey will include distributed information systems that are closely related to DCEEIs.

An area that potentially may provide a conflict detection and resolution mechanism is Distributed Artificial Intelligence (DAI) (Section 3.4). Within the field of DAI the subject of uncertainty management is of particular interest and, hence, observed in Section 3.5. Conflict detection and resolution can be seen as a special case of uncertainty management. It will be shown that research in this field provides a framework for conflict detection that is applicable to enterprise integration (Section 3.5).

Conclusions are drawn in Section 3.6 followed by a chapter summary.

3.2 Conflicts in Enterprise Integration

3.2.1 Propositional Conflicts in Enterprise Integration

In Section 2.5 it has been outlined that a retrieval process in enterprise integration environments may produce multiple results (also called candidates). These need to be assessed in a uniform representation. Furthermore, a requirement for conflict resolution is that the results are compared so that it can be determined if, and how the candidates conflict. Furthermore, in Section 2.5 it has been briefly outlined that multiple results to a given query essentially produce multiple pairs of possibly conflict results. For example, three results A, B and C produce the pairs 'A and B', 'A and C', and 'C and B'. The following of this Section will present a basic propositional structure to classify the possible ways in which pairs of candidates, in principle, can conflict. Please see Section 5.2.4.2 for a demonstration how multiple results on a query form pairs of possibly conflict candidates.

A propositional calculus has been adopted to formally represent conflicts in enterprise integration scenarios. It is composed of an object O_i (out of a range of Object O_1, O_2, ... O_i) with the attributes R_1, R_2, ... R_k. Furthermore, it may be specified if the attributes are members of classes of attributes, which range from v_1, v_2, ... v_m.

Two kinds of conflicts, explicit and implicit, in principle, can occur in information integration. Two results, or candidates such as '$O.v_m:R_k$' and 'not $O.v_m:R_k$' conflict **explicitly** when it is claimed that both candidates are true at the same time (where the external negation 'not $O.v_m:R_k$' includes the internal negation 'O does not have R_k in the attribute class v_m'). For example, one proposition 'Yogi (O_i) has the weight(v_m) 18 stone (R_k)' ($O.v_m:R_k$), and the proposition 'It is not the case that Yogi has the weight 18 stone' (not $O.v_m:R_k$), are explicitly conflicting.

Two results conflict **implicitly** when they are logically inconsistent so that they have one of the following six forms:

1.	$O_i.v_m:R_1$ and $O_i.v_m:R_2$	where $O_i.v_m$: R_1 logically excludes $O_i.v_m:R_2$
2.	$O_i.v_1:R_1$ and $O_i.v_2:R_2$	where $O_i.v_1$: R_2 logically excludes $O_i.v_2:R_2$
3.	$O_i.v_1:R_k$ and $O_i.v_2:R_k$	where $O_i.v_1$: R_k logically excludes $O_i.v_2:R_k$
4.	$O_1.v_m:R_1$ and $O_2.v_m:R_2$	where $O_1.v_m$: R_1 logically excludes $O_2.v_m:R_2$
5.	$O_1.v_1:R_1$ and $O_2.v_2:R_2$	where $O_1.v_1$: R_1 logically excludes $O_2.v_2:R_2$
6.	$O_1.v_1:R_k$ and $O_2.v_2:R_k$	where $O_1.v_1$: R_k logically excludes $O_2.v_2:R_k$

Table 1: Implicit Conflicts

This means in the first case, for example, that an implicit inconsistency over the attribute class 'weight of object' (v_m), exists when it is known that the proposition 'Yogi weighs 18 stone' ($O_1.v_m:R_1$) and the proposition 'Yogi weighs 20 stone' ($O_1.v_m:R_k$) cannot be true at the same time. This inconsistency is based on the assumption that O_i is concerned with the same individual (O_i) in both cases and that the individual can only have one value (R_k) for the attribute class v_m. The second case varies in that the same object (O_i) has different attributes for different classes of attributes. For example, 'Yogi weights 18 stone' and 'Yogi eats very little' may be two propositions that are contradicting. The third case only varies from this in that different attribute classes (v_1 and v_2) of the object (O_i) have the same attribute (R_k). For example, the propositions 'Yogi has red hair' may be concerned with the attribute class 'colour of hair of the head' and 'colour of hair of beard'.

The last three cases are concerned with two different objects O_1 and O_2 that have conflicting attributes. For example, the attribute 'fattest bear' (R_k) may only be true for either Yogi (O_1) or another bear Peter(O_2). In this example the objects are concerned with the same attribute class, e.g. 'rank in corpulence'. However, in the fifth case an implicit conflict exists between two different objects over different attribute classes. For example, it may be conflicting, that Yogi (O_1) is 'the fattest bear' (R_1) and his mother (O_2) has only given birth to slim, little bears. This may be logically conflicting based on a number of assumptions such as: It is true that Yogi is the son of the bear O_2; and She has always had slim, little bears; and Yogi is not slim but fat. The final case describes a situation where the two different objects have the same attribute for different attribute classes. This may be conflicting such as 'Yogi has red beard hair' and 'Yogi's father has red head hair' if it is true that son bears always have different coloured beard hair than the colour of their father's head hair.

From the examples in the previous paragraph it becomes obvious that a key question in enterprise integration is 'What kinds of conflicts can **practically** be detected by information agents?' Detecting explicit conflicts, once the arguments have been described in this propositional calculus, is a simple matching operation. In other words, the proposition and the negation of an identical proposition constitutes an explicit conflict. However, implicit conflicts are only detected if they are 'known', that is if the information agent knows when two propositions are 'logically-exclusive'. In other words, it needs to be specified what is 'known' by an information agent to identify implicit conflicts.

Typically, enough information is available to identify the implicit conflicts of cases 1 ($O_i.v_m:R_1$ and $O_i.v_m:R_2$) and 4 ($O_1.v_m:R_1$ and $O_2.v_m:R_2$). For example, the information agent may know that the object (O_i) 'Peter' has only got one property for the

attribute 'address' (v_m) which may be either 'Pentonville Rd.' (R_1) or 'Roseberry Gardens' (R_2). Other implicit conflicts require information that is typically not available in enterprise integration environments. In information-sharing there are two reasons for this:

1. The information agents lack the domain-specific knowledge to identify conflicts between different attributes of objects. For example, complex circumstantial information is required to know that the result from one database 'Peter has blue shoes' is conflicting with the proposition 'Peter has a green shirt' from another database. Typically, these conflicts are identified within a problem-solving scenario or by expert reasoning. For example, a scheduling agent as described by Klein and Lu [KLE89] (Section 3.4.4) may identify implicit conflicts within a planning scenario based on its domain-specific knowledge. Section 5.2.4.2 will specifically describe what data is available to information agents to detect implicit conflicts.
2. A typical query in enterprise integration will request a specific attribute of an object, for example: 'What is the name of the employee with the national insurance number 123?' or 'Is it true that the employee Peter has the national insurance number 123?' This kind of query typically investigates one specific attribute class (v_m). It will not detect inconsistencies between different attributes of the object employee. Furthermore, these queries will typically be concerned with the same object, e.g. the employee 'Peter'.

The term 'same objects', however, requires further specification. The last three kinds of implicit conflicts are concerned with different objects, which may mean that:

- Two objects O_1 and O_2 are known to be different individuals, e.g. called 'Peter' and 'Mark;
- It is believed that two objects are concerned with the same individual but no proof of a common identity is available.

An example of the latter case is an object 'Peter' from one database and an object called 'Peter' from another database may be the same individual but unless this is verified as a fact, it has to be specified that there are two objects. It might be possible to prove that the objects are the same individual but until this is done, the conflict is one of 'different' objects.

In contrast to these conflicts concerned with 'the same object', it may also occur that attributes of different individuals are conflicting, for example, 'Peter's colour of shoe' and 'Mark's length of hair'. In this case objects are known to be different individuals and the

emphasis lies on the semantic problem evaluation and enterprise modelling. For example, a photographer may identify the conflict based on his opinion that these colours do not match when Peter and Mark are together in a picture.

In summary, a conflict detection and resolution mechanism for enterprise integration environments will have to deal with any kind of explicit and implicit conflict. Information agents will have to identify all explicit conflicts and all known implicit conflicts. These typically include conflicts between results that are concerned with the same object, and often the same attribute class.

3.2.2 Other Kinds of Conflicts in Enterprise Integration

In information-sharing many other ways exist to structure the possible kinds of conflicts, other then explicit and implicit conflicts. Conflicts can occur on different levels of the integration process, which include the physical integration and conceptual integration levels. For example, on the physical level sources may use the same communication concept in different, conflicting, ways, or network services may conflict when they transfer data between notes. This research is primarily concerned with the conceptual integration of information throughout the environment. Within this field of conceptual information integration there are at least two classes of conflicts [KIM91b]:

- Schema conflicts; and
- Data conflicts.

Schema conflicts, in principle, are due to inconsistencies in the conceptual descriptions of the integrated sources. Schema conflicts are therefore also called type-level conflicts [HAM93]. Such schema conflicts can occur between multiple conceptual (auxiliary or export) schemata of multiple systems. Alternatively they may occur between a conceptual schema and its internal, or external schema or the data it represents.

By analysing the cause of schema conflicts these could be classified into semantic, descriptive, heterogeneity, or structural schema conflicts [SPA92]. In the case of semantic schema conflicts different schemata use different concepts to represent the same objects. For example, the concepts 'car' and 'vehicle' may describe the same real world object. Descriptive schema conflicts are based on different naming, or different property attribution to objects. For example, the concept car can be described by the properties 'model', 'engine', 'manufacturer', or by other properties such as 'price'. Heterogeneity conflicts are due to different data models used in different schemata such as an object-oriented and a relational data model. In structural schema conflicts the same real world object can be described by multiple constructs in the schema. In other words, within one data model such as the relational one, there are multiple ways to model a real world concept. Schema conflicts are part of schema integration [QUT92] which typically is performed when distributed systems are installed.

> "**Data conflicts** are due to inconsistent data in the absence of schema conflicts" [[KIM91b]p.14].

In other words, data conflicts exist between data items that cannot possibly both be true at the same time because they are explicitly or implicitly contradicting. In distributed information systems these conflicts can only be detected when there is no schema

conflict. Data has to be 'translated' or 'mapped' correctly between multiple sources before its inconsistency or consistency can be evaluated. Schema conflicts therefore need to be resolved first before data conflicts can be detected. Thus, conflict detection in this research is ultimately directed at evaluating whether a data conflict has occurred among responses from multiple systems.

In principle, data conflicts can be based on either inconsistent data or different representations of the same data [KIM91b] The latter may, for example, be due to incorrect syntactic translation between different representations, semantic mis-interpretations, or mis-matching. Furthermore, data conflicts may be due to incorrect, incomplete, obsolete, or, for other reasons, inconsistent data. For example, data may be obsolete in any source or it may be entered in an incorrect form. Data conflicts can also be "factual conflicts [which] relate to the difference in viewpoints about the same entity" [[DUO93a]p.282]. Intelligent and other software systems may have these kinds of conflicts if they produce incorrect or at least inconsistent results. For example, an expert system may use rules that are inconsistent with other systems [PRE92]. Schema and data conflicts can occur in large, distributed, heterogeneous enterprise integration systems as will be shown in the following Sections 3.3.

Data conflicts can occur when more than one result to a given problem exists. For example, for the question 'How fat is Yogi?', the two answers '18 Stone' and '15 Stone' may exist. This conflict is **non-essential** if both results are correct and it is a question of selecting priorities [RES75]. If however, one result (or both) is incorrect then, "a true conflict ... needs to be resolved, and no mere choosing among mutually acceptable outcomes" [[ZLO91]p.1321]. For example, it may be known that Yogi only has one weight so that either '18 Stone' or '15 Stone', or neither result is correct. This conflict is called an essential, or true data conflict.

In summary, a conflict detection mechanism in DCEEI should identify different kinds of conflict:

1. Mere syntactic physical integration or schema conflicts;
2. Data conflicts based on different representations or inconsistent data (syntactic or semantic integration conflict). Data conflicts can be classified as explicit, or implicit propositional conflicts.
3. Essential conflicts or non-essential conflicts where the latter is based on priorities, goals, notions of optimality, etc.

3.2.3 Completeness of Conflict Detection in Enterprise Integration Environments

Ideally a complete conflict detection mechanism would detect all explicit and implicit inconsistencies throughout the distributed system. However, conflict detection depends on the completeness of a retrieval algorithm in enterprise integration. In other words, the detection mechanism relies on the selection of all relevant information to the case in issue. Information retrieval may be incomplete in integration environments (Section 2.3) that integrate heterogeneous, autonomous sources (Section 2.4). Please note that 'information retrieval' as described in Section 2.3 and 2.6 includes not only the investigation of what information is needed but also the physical retrieval of the results, e.g. distributed query processing. Thus, a measure for the completeness of conflict detection is necessarily limited to those results that are provided by the retrieval process.

Among these results, the mechanism should be complete in respect to explicit conflicts. A sentence (result) and its negation (in a second result) are easily identified as conflicting. However, implicit conflicts can only be detected if they are known to be conflicting [LEV84]. For example, truth maintenance systems detect inconsistencies according to a principle of coherence such as 'valid justification' [HUH90] (Section 3.5.3.2). In general, domain or context information can be used as 'rules' of coherence in information systems [SU 91] [DUO93a] [HUH92]. Implicit conflicts (Section 3.2.1) should be detected in a complete fashion in respect to all notions of coherence known to the information agent (called 'known implicit conflicts').

Conflict resolution would ideally investigate the truth of information representing real world notions that lie outside the computational system. In other words, the agent would investigate the real world and determine which result is incorrect. However, this is an impossible task for an information system [KEN91]. Agents cannot investigate the truth of information; rather, they apply some reasoning about what rationally appears to be most likely from the point of view of the information system. Thus, conflict resolution is limited to the resolution strategies know to the information agent, which is its only way to eliminate incorrect results. A rational scheme is designed for solving conflicts with the available strategies.

In summary, information agents should detect all explicit and all known implicit conflicts between results provided by the information retrieval process. It should try to resolve these conflicts with all rational strategies known to it. In the following sections existing approaches to conflict detection and resolution are investigated against this background.

3.3 Conflict Detection and Resolution in Distributed Collaborative Environments for Enterprise Integration

3.3.1 Master Model and Unified Approaches

Semantic unification is a central problem of enterprise integration and can be approached in three different ways [PET92]. The master model approach requires that all models of individual sources be instances of a pre-defined universal model. The unified approach relaxes this requirement considerably in that the global model is a unification of the local models. The federated approach seeks to completely relax model conformance by means of active mechanisms that dynamically relate models as needed.

3.3.1.1 The Master Model in CARNOT

The CARNOT project [HUH93][COL91] integrates all varieties of information systems via its global schema or master model. This schema is based on a common knowledge-base CYC [LEN90] which, in principle, represents every possible concept. Information sources are mapped into CYC. In other words, their schemata are linked to CYC, the master model, and not directly merged with each other.

Equivalence relations between objects in the local sources and CYC are called 'articulation axioms'. These are similar to accessibility relations in the possible worlds semantics [KRI63][DUO93a]. They are implemented by relations based on the 'ist' operator and the equivalence sign $\Leftrightarrow$. It means that an expression ψ that is true in the local source C_i is equivalent to an expression ϕ that is true in the global context G (ist(G ϕ) $\Leftrightarrow$ ist($C_i\psi$)). Such mappings are called generalisations. Concepts are further divided into subgroups in a tree-like schema that leads to a generalisation hierarchy. In other words, every concept is generalised into the tree schema on the most general, possible point or the highest subdomain in the hierarchy [[HUH93]p.39]. Thirdly, the information is not just integrated with its schema description but also with its Resource and Organisational Knowledge [HUH92]. Resource Knowledge includes all environmental information about an information source such as its language or its local data model. Organisational Knowledge embodies corporate rules that govern the use of information from integrated system. Section 2.7 has outlined the different sources of schema, organisational and Resource Knowledge.

The master model approach enables the integration of heterogeneous concepts from data stores, derivation systems, other software systems, etc., into one central model. Schema conflicts cannot occur as CARNOT presents a "coherent integration of models [based

on] its use of the CYC common-sense knowledge-base as a global context and federating mechanism" [[HUH93]p.34]. Furthermore, the builders of the CYC knowledge-base have assumed that it is extremely rare that information systems collide and "a statement and its direct negation both get asserted, let alone nearly simultaneously" [[LEN90]p.34]. Whether this assumption is realistic will be discussed in the following sections but it is clear from this statement and Section 3.2.2, that data conflict may occur in enterprise integration. Moreover, conflict free model integration should be very difficult to enforce, for example:

- Integrators may make mistakes when they integrate an information model;
- Information models may be inconsistent internally (the way they describe the integrated information system), and hence introduce inconsistencies into the integration model.

It follows that the CARNOT system, as all master model approaches, has to be restricted to integrate only conflict free, consistent information systems. A general integration system, without this restricting consistency assumption, requires a mechanism to detect and resolve these possible conflicts.

3.3.1.2 The Unified Approach to Model Integration

The unified approach is based on the merging of existing schemata into one 'all system', coherent schema that may be called a Universal Conceptual Schema [PAP90] [PAP92a], a Common or Canonical Schema [SHE93]. Examples include distributed databases [KIM91a] [HSU91], and enterprise integration systems [PAN91a] [PAP92a].

A unified schema involves in the simplest case of homogeneous distributed databases, a merger of parts or entire local schemata. Local schemata represent the information in a local source. These are (possibly partially) exported into an external schema that may also be called a local participation schema [BEL92]. This schema can be used by the global database management system. The global system has an import schema for each export schema presented from each integrated source. The conceptual schema is the sum of all import schemata. Global external schemata present, possibly partial, views of the distributed system to external clients of the distributed database. An early example of such a heterogeneous distributed database system is Sirius-Delta [LIT82].

However, the global schema of a distributed database typically provides name and data value binding, the definition of some basic consistency rules (integrity constraints), and confidentiality restrictions. Enterprise integration environments based on a global unified schema also integrate circumstantial information as described in Section 2.7. For example, the unified approach by Hsu *et al.* [HSU91] not only integrates existing schemata but also the environments and conditions in which a system operates. Hsu's system is based on a unified metadata representation and management system.

> "[The unified schema] should not only contain a common catalogue encompassing local database schemata, but more significantly, it should specify the intended contexts (i.e. the operating knowledge, control knowledge and decision knowledge.....) within which individual systems operate and interact with each other" [[HSU91]p.605]. Operating knowledge includes business rules, triggers, integrity constraints and process descriptions. "Control knowledge for sequential interaction includes data transfer rules and global equivalence knowledge for all data items pertaining to the same logical object but [are] implemented differently" [[HSU91]p.604]. Decision knowledge includes "global decision processes and their implementation into information flows, localised decision rules and control procedures" [[HSU91]p.604].

In other words, this system provides a unification of all the information, including schemata, that is available from the integrated sources. It is, hence, called a unified model and not just a unified schema.

Enterprise integration systems are typically managed by Intelligent Cooperative Information System architectures (Section 2.3), which use a canonical schema [PAP90]. An example of such an environment is the integration system by Pan and Tenenbaum [PAN91a]. It is based on the enterprise-wide model called MKS.

> "MKS thus serves agents as a repository for shared knowledge, and a centre for information exchange" [[PAN91a]p.206]. "The core of the MKS is a comprehensive object-oriented model of the enterprise and how it functions. The MKS model includes descriptions of personnel, facilities, equipment, inventory, manufacturing processes, and other corporate assets. It also captures the flow of information, decisions, and materials through the enterprise ... The model is wired into the enterprise's information infrastructure ... so that it continuously reflects the actual state of the enterprise" [[PAN91b]p.224].

This approach shares much with the master model approach discussed above. However, the processes, schemata, descriptions of facilities, etc. are not pre-existing such as the CYC common knowledge-base [LEN90] [COL91], but are composed of descriptions of enterprise entities and their interaction. The approach is therefore a unification of not only schemata, but all available information. This dynamic unification is facilitated by special tools.

> "MKS consists of a set of tools for modelling a manufacturing environment (including processes, equipment, facilities, and operational procedures) and a complementary set of application-specific shells that use the models to perform common manufacturing tasks such as monitoring, diagnosis, control, simulation, and scheduling" [[PAN89]p.35].

Conflict management facilities are implemented based on the MKS model and the inter agent communication. In other words, the MKS model is used by information agents, which integrate information sources, and users that request information from the distributed system. The agents have different types of messages and activities that they perform. One is called 'Inform Message' and is designed to request textual information from all other agents. If multiple agents respond, then a request is sent to the agent that is 'the most appropriate' for this task [PAN91a]. The appropriateness is decided based on the context. For example, "Mary, the equipment operator on duty" [[PAN91a]p.209] may justify appropriateness.

An 'Enquiry Message' requests information from a specific agent and the existence of other solutions is not examined. Thirdly, a 'Bid Message' may be sent to all agents that

can possibly provide a response. Which agents can possibly contribute to the tasks in question, is identified through their descriptions in the MKS model. Their responses are evaluated if they arrive within the time limit for submitting tasks. Each result is a 'bid to produce the requested information'. The bid with the lowest cost for carrying out the task is selected. For example, communication costs, or utilisation aspects are used to determine the cost of carrying out the task.

Other research that uses the unified approach for enterprise or schema integration in a similar fashion has been mentioned above such as [JAG92], [PAP92a], [DUO93b], [ARE93]. All these systems are:

> "Based on the assumption that all data used by a number of related, though separately developed, information sources should be first uniformed and then unified into a single conceptual (virtual) accumulation of data" [[MAR91]p.11].

The unified, global model provides an integration that is free of schema conflicts. Furthermore, these distributed information systems are assumed to contain completely consistent information within each individual source, and among the distributed systems. No essential conflicts between information sources are assumed possible. In other words, these systems assume that semantically related information, which can be retrieved from multiple sources, will always bring identical results. Only non-essential conflicts between mutually acceptable solutions are assumed possible. These systems present heuristics such as 'cost evaluation' [PAN91a] which guide the agents in deciding which sources to select in order to optimise the retrieval process.

> "Each component information system (IS) has its own set of desires and goals and an area of expertise; while conflict, which is not total [not a true conflict], may exist among the different component ISs, once these are transformed into information agents compromise is achieved and mutual beneficial activities are performed.....This strategy results in overall solutions which are incrementally constructed to converge on a set of complete local sub problem solutions that guarantee global consistency" [[PAP92a]p.182-183].

In other words, this "model of cooperation implies not only some degree of mutual predictability but also the lack of information agent conflicts" [[PAP92a]p.185].

3.3.1.3 Conclusion on Tight Conflict Free Integration

In summary, tight integration into a consistent global master or unified model requires at least the following:

1. The integrated sources are fully explored at the integration phase [QUT92].
2. Sources are fully and consistently described to the global system. Every concept of the local sources has to be mapped into the global master model such as the CYC concept [LEN90]. In the unified approach all the globally shared information has to be merged.
3. The concepts need to be consistent internally. For example, the integration of multiple databases requires that "the databases contain consistent information, so the choice of database only affects the efficiency of the query and not the accuracy" [[ARE93]p.140]
4. The systems have to stay consistent to these definitions in the global system and therefore lose local autonomy. This consistency requirement will potentially restrict autonomy of the local source to change its design [BAK92]. Furthermore, the autonomy of the local source to communicate with the global system or to execute a request from the global system is violated [SHE90] (For a definition of autonomy please see Section 2.4).

But are these assumptions and requirements realistic for enterprise integration environments? Full exploitation of an integrated source requires omniscience by the integrator. It is necessary to explore all explicit and implicit conflicts. Omniscience of intelligent information systems has been questioned in much research, e.g. [LEV84] [FAG87], and in Section 2.8.2

Several schema conflicts have been listed above including semantic, descriptive, heterogeneity and structural conflicts. As one specific example, the notion of identity in different models may lead to inconsistency that cannot always be overcome. Identity will be discussed in much more detail below. In the following paragraph it will only briefly be used as an excellent example of a potential source of inconsistency in schema integration.

> For example, 'Yogi the bear' may be identified by its name, its social security number, its function in comic series, or a system-defined number (surrogate). More specifically, a relational database would implement a key property of 'Yogi' such as its name, as an identifier. The latter, value based form of identification potentially includes the danger of being too weak. For example, other 'Yogis' may have the same name but different real world identities [PAT88]. An object-

oriented database would, for example, implement a much stronger notion of identity based on a system-defined identifier such as a unique number. Different systems represent the object 'Yogi' in different forms and with different strengths of identity [KHO90]. This may potentially lead to inconsistencies if the locally inaccurate concept 'Yogi' is mapped into an exactly defined concept in the global system. In other words, the 'Yogi' in a relational database and an individual 'Yogi' in an object-oriented database cannot be mapped on one object - with only one notion of identity - without having to manipulate the notion of identity of at least one of these objects. Typically in enterprise integration, an object 'Yogi' from a relational database will be mapped to an individual that is identified with a strong surrogate based identifier. However, based on the notion of identity of objects in relational databases it is impossible to guarantee that this mapping is correct. Hence, unifying a weaker and a stronger notion of identity in a static, schematic way will potentially result in information loss, and incompleteness.

"One of the characteristics of engineering data is that the schema does not change frequently" [[JAG92]p52]. In such an environment, consistency can be enforced by, for example, restricting updates of any schema to times when the sources are not operating. In principle, this is possible if there are very few updates to the schemata. However, in cases where large numbers of information sources are integrated and need to be changed frequently, or where sources cannot be restricted in changing their design, then this assumption may be difficult to maintain. Enterprise integration environments integrate all kinds of systems, including those that change over time and, hence, may become internally inconsistent or inconsistent with other information systems.

"It is absurd to believe that in real organisations contradictions will never occur" [[BAR94b]p.153]. In other words, data conflicts do occur within one source and so consequently also between sources. Without considering whether or not it is possible to construct consistent schemata for one or multiple sources, it seems unavoidable in practice that data is inconsistent and, therefore, conflicting.

Closely related to the fourth assumption (consistency to definitions in the global model) is the concept of autonomy. It has been shown in Section 2.4 that heterogeneity subsumes autonomy [BAK92]. In other words, the more heterogeneous the systems are, the more important is the autonomy of these systems. Tight integration violates the autonomy of information sources. For example, an integrated information system may need to be coherent with production requirements on the shop floor in an engineering environment. This source will need to be autonomous in order to fulfil its function in the

production flow. It may become inconsistent with information stored elsewhere in the enterprise. However, execution autonomy is crucial for such a source to function correctly in its environment. Even for distributed databases, which store persistent data, harshly restricting local autonomy has proved unrealistic:

> "Building a global conceptual schema and making all the data behave as one classical database ... is unrealistic since the local databases must give up their autonomy" [[LIT90]p.271].

In summary, the assumption of fully locally and globally consistent information systems is ill founded in enterprise integration environments:

- Tight integration of information systems is not able to guarantee consistency of the global model and schema conflicts may still occur;
- Essential data conflicts are possible in realistic enterprise integration environments.

Existing research that uses tight, unified or master model integration without assuming inconsistencies, as the example systems described in the previous sections, fails to address essential conflicts. Conflict resolution is limited to proposing resolution heuristics for non-essential conflicts only.

3.3.1.4 Tight Integration with Inconsistencies Assumption

Fox and Barbuceanu [FOX92][BAR94a] propose an approach to enterprise integration that includes a mechanism for handling inconsistencies that also manages essential conflicts.

The research is based on a master model called TOVE [FOX93]. It is a "computer based data model which provides a shared and well defined terminology of an enterprise, and has the capability to deductively answer common sense questions" [[FOX93]p.425]. It includes a data model with objects, attributes and relations. Furthermore, these constructs are defined by generic concepts of time, causality, activity and constraints.

> "The Information Agent (IA) maintains two kinds of proposition. Premises are propositions sent to the IA by other agents that consider them true. The IA has no access to whatever justification the sending agent may have for the proposition. Derived propositions (or simply propositions) are propositions inferred by the IA based on the available premises and on the IA's knowledge of the domain" [[BAR94b]p.153].

The knowledge of the domain is encoded in the TOVE model. It can be used to determine which propositions are disjoint and constitute conflicts of the kind 'p&q⇒ false' [BAR94c]. This means that there may be two beliefs p and q of information agents in the enterprise integration system that cannot both be true at the same time. These conflicts are possible because information persistently stored or derived from multiple source is not necessarily consistent.

The detection in this system includes conflicts based on:

- Terminological inconsistency, which is the conceptual vocabulary employed by an information agent;
- Assertional inconsistencies of beliefs held by information agents while sharing information;
- Temporal inconsistency that occurs when information becomes incoherent over time.

In other words, inconsistencies are assumed possible and may lead to essential or non-essential conflicts.

Conflict resolution for the first kind of conflict is provided by a description logic (T-Box service). The other two conflicts are solved with the help of a 'credibility / undeniability

model' [BAR94a] [BAR94c]. One way to determine credibility is the agent's self assessment of their authority. Furthermore, all agents are ranked in their credibility or competence. The credibility of an agent concerning a specific 'belief', results from a possibly incomplete (partial) order of the agents' roles to accomplish particular goals. Undeniability is the cost or effort to retract information that has already been distributed and used by agents. The undeniability is very high if the information has been much distributed and the effort to change it at a later point is very high. Agents have to self-asses the deniability costs for their beliefs and their authority.

For example, it is possible that one belief has a high credibility and the other a low deniability cost so that the second belief is retracted. Furthermore, it may be the case that one belief has a low authority and the other a high deniability cost. In this case the first belief is denied. However, there are cases where both beliefs have high authority and / or high deniability values. In these conflict cases negotiation strategies are necessary. This negotiation is grounded on two concepts:

- Mediation that tries to find a compromise by redefining conflicting propositions, similar to Sycara's approach [SYC89](described in Section 3.4.3); and
- Cooperation that evaluates the overall cheapest solution (cost), such as Zlotkin and Rosenschein's [ZLO91] game theoretic approach (Section 3.4.3).

In summary, the mechanism by Fox and Barbuceanu [BAR94a] [BAR94c] [FOX92] proposes a credibility / deniability model to conflict management. This research accepts the existence of essential conflicts. However, no general framework for detecting or resolving conflicts is provided by Fox and Barbuceanu. One consequence from this is that the approach lacks generality. Furthermore, it falls short in providing comprehensive conflict detection and resolution for enterprise integration environments because:

- It is limited to three kinds of conflicts including terminological, assertional, temporal. Other conflicts, e.g. other schema conflicts, are not investigated.
- Conflicts are not formally detected and classified but resolution heuristics are applied regardless of their suitability for the conflict case. For example, in case a schema conflict occurs it would falsely be addressed by investigating the authority of the agent that has shared this belief.
- The conflict resolution mechanism only includes the concept of credibility and the cost of retracting already shared beliefs. It opens with accepting that essential conflicts can exist but it does not propose resolution strategies, other than credibility, that can handle essential conflicts in which one result may be correct and another may be incorrect.

3.3.2 Enterprise Integration based on Federated Architectures

The environments described so far were based on a tight integration. More loosely coupled enterprise integration environments are Multidatabases.

> "[They] aim to permit any kind of pre-existing, heterogeneous, independent databases, whether on the same computing facility or not, to be interrogated via a uniform, integrated logical interface" [[BEL92]p. 72].

In other words, the integrated sources can be very heterogeneous and are integrated in such a way that they stay largely autonomous. For example, an integrated database may have local users that use only this database without noticing that it also participates in a distributed database.

> "The need for the existence of a global conceptual schema in a multidatabase system is a controversial issue. There are researchers who define a multidatabase management system as one which manages several databases without a global schema" [[ÖZU90]p. 290].

An example of the former is the Experimental Distributed Database System (EDDS) by Bell *et al.* [BEL87] [BEL89]. The EDDS provides each global user with a global view via its global relational schema. This schema is constructed by the global management system. It composes local participation schemata of each integrated source, with the schematic information on the data that this source is willing to share.

> "One of the objectives of EDDS is to leave the pre-existing local database management systems (LDBMS) unaltered, so they can retain their autonomy and their local database users can ignore the fact that an EDDS distributed database exists" [[BEL87]p.363].

In order to implement design autonomy (the autonomy of the local source to change its local schema) a schema manager is installed.

> "The schema manager (SM) is intended to automate maintenance of the schema in the global data dictionary (GDD)" [[BEL87]p.365].

A copy of the GDD is located at each site. It contains a copy of the global relational schema and all necessary mapping rules, integrity constraints, access rights, etc. that are necessary to integrate the local source. In other words, schema conflicts can be avoided by the SM for the shared data, which is the data defined in the participation schema. Data conflicts, based on inconsistent data stored in multiple databases, may occur. The GDD

shares much with the federated dictionary of federated databases. Before these are discussed other multidatabases will be briefly revised.

Multidatabase without a global schema include Litwin's MRDSM [LIT90] or the PEGASUS system [AHM91]. These systems emphasis the integration of heterogeneous information systems while their local autonomy is strictly retained. Instead of a global schema each integrated source has a dependency schema that contains the mapping and integrity regulations necessary to integrate information from the local source and remote systems. In the absence of a global schema the local management system has to fulfil all the functionalities of the global management system described in the previous sections. This includes the allocation of relevant information, requesting this information and integrating it. These systems have, as a result, provided the most loosely coupled distributed databases. A disadvantage of these distributed databases is the lack of global management facilities. For example, distributed transaction management for updates has not been established in the PEGASUS system [AHM91].

Federated architectures originate from fundamental work on federated distributed databases such as Heimbinger and McLeod [HEI85], or systems such as the ORION-2 [KIM91a]. The federated database architecture has replaced the global schema as used in master model or unified approaches by three component schemata:

- A private schema;
- An export schema; and
- An import schema.

The private schema describes the information in the private or local source. The export schema describes all the information that is made available for other systems. In other words, elements that are put forward by a local system in the export schema can be requested from it through the distributed system. The import schema describes the access and the information that the local source would like to import or share with other sources. In order to import information the other systems' export schemata are investigated and the schematic information is integrated with the existing private schema to form a global view. In [HEI85] sources "guarantee that they will not modify the definitions (structure or semantics) of the exported element unless it notifies the importing systems" [[HEI85]p.262].

A federated system builds a unified schema composed of imported schema information it has received and its own local schema.

"In either approach [federated, master model or unified] one is faced with schema integration - the process of developing a conceptual schema that encompasses a collection of local schemata" [[THO90]p.139].

In other words, federated systems integrate schemata in a similar fashion to tightly coupled systems. The difference lies in the decentralised and ad hoc form. Every source develops its own integrated schema and there is no one central or global integrated schema (model).

In order to provide consistent schema information over time, an integrated source may have to notify all relevant systems of changes to its export schema. This is called a notification of change [HEI85] as described in the last paragraph. Another solution would involve importers repeating the import of schema information in order to incorporate any changes that may have affected the imported schemata since the last import.

The federated system described by Heimbinger avoids any schema or data conflicts based on the following two assumptions.

1. Schema changes are communicated to any importers of that schema using the above mentioned 'notification of change'.
2. "Objects be given unique names that are unique with respect to an entire federation. Such unique names must themselves contain some tag indicating the component that contains the specified object. These unique names are generated by concatenating the component name with a local object name" [[HEI85]p.259].

In other words, all objects in the entire federation are consistent in that each real world object is represented only once and has a unique identifier, which is its name.

The federated architecture allows for a more loose integration than a global schema architecture in two respects:

- A local system directly manages its shared information, including the information it is willing to share with others and the information it can import from other systems;
- The federated schema specifically allows for shared, exported and private, not shared, data. The global view includes all the information, private and imported, that is available at a given site. Hence, it enables the integration of sources that may have inconsistent information privately and only share consistent data.

The modularity together with object-oriented data modelling capabilities has provided the flexibility needed to develop heterogeneous, federated database systems. The ORION-2 system implements the federated architecture in an object-oriented distributed database. In other words, this distributed database has an all system spanning naming convention and assumes that shared data is kept consistent throughout its sharing. Conflicts can occur when data is transferred from the private to the shared data portion of the database. A conflict is that already shared data is not coherent with the new data. However, no conflict resolution but rather a conflict avoidance mechanism is implemented such that the new data cannot be shared.

Huhns and Bridgeland's [HUH90] [HUH91] have implemented a federated system based on information agents. This allows agents to exchange partial schemata of the information they are willing to share. In the same way as the federated system ORION-2, data can only be shared if it is consistent with the already shared data. In other words, local sources can have inconsistent data in their local source and schema. If an agent wants to share a specific piece of data then it has to declare this data from its 'private' into the 'shared' status. This is done by checking if the information is consistent with all the shared data with the help of a distributed truth maintenance (DTM) system. This justification based DTM guarantees that under the notion of coherence all shared data is consistent. The DTM is described in more detail in Section 3.5.1.2 as a means of non-monotonic uncertainty management.

In conclusion, conflict management in federated systems can have two forms:

- Traditionally in distributed databases, conflicts are avoided by mechanisms such as naming conventions and strict consistency rules for shared data. These ensure that only data is shared that is consistent with the other shared data. Inconsistent data is kept in the private portions of the local sources and can, hence, not conflict with data across systems.
- Conflicts may occur and the management system or agent needs a mechanism to detect and resolve conflicts.

This research will propose such a conflict management mechanism. If a rational scheme for managing inconsistent data exists then a federated system can be used to implemented open, ultra concurrent systems as proposed, e.g., by Hewitt and Inman [HEW91] (Section 2.3).

3.3.3 Integrating Non Persistent Data in Enterprise Integration Environments

> "Logical programming ... and relational database systems have the same underlying mathematical background, namely first order logic" [[BEL90]p.41].

Hence, deductive databases typically combine relational database management and querying techniques with logic programming, as used in expert systems. This combination is very desirable because "deductive databases generally have a very large number of facts but only a few rules, whereas logic programs generally have more rules than facts" [[BEL90]p.41]. In principle, the relational database is a provider of data, in the form of facts and rules, to be used by an 'intelligent', inference based, software component, e.g. an expert system. This inference based system may either form part of a deductive database, e.g. described by Bell [BEL90], or it could be a stand-alone component that receives facts from a database.

Enterprise integration environments provide this integration of persistent data stores, including deductive databases, that may provide information (including especially 'facts') to application programs, including logic programs. It is the core functionality of enterprise integration environments (Section 2.3) to provide integrated information for any 'decision makers' and 'application programs'.

In addition, an application program may itself produce information that it may share across the integration environment. In this way, an expert system may infer 'knowledge' via its rule-based inference. This information may, for example, be requested by a decision maker. Such an integration is, however, different from integrating persistent data from databases. Two approaches are commonly used, the integration as:

- Information Repositories; and
- With the help of functions.

The most pragmatic solution is the integration as 'information repositories' [HUH92] (Section 2.3 and 2.7 Schema Knowledge). Just as is done with databases, the agent simply sends a request for information to the integrated system and receives results in return. This approach ignores the specific dynamics of problem-solving software systems but it also provides a pragmatic solution to integrating heterogeneous, autonomous systems [HUH92].

Conners and Lyngbaek [CON88], for example, use functions to integrate transient data from software systems with persistent data in a database system. The approach is based on the object-oriented data model IRIS. It is composed of the constructs object, type and

function. Objects are the entities that are modelled in the database, they have identifiers and can be related hierarchically to other objects. A subtype, supertype structure is therefore implemented. Functions can be implemnted as attributes of objects to enforce relations to other objects or to initiate operations on objects. However, functions can also be used to implement links between the central database and the integrated sources. Such a 'foreign function' is an independent programme that is implemented in an object in the central databases and can be evoked when information is requested from this object. The foreign function can then obtain data from an integrated source as specified in the program of the function, and present it to the requester.

The approach provides transaction and distribution transparency. In principle, full autonomy of heterogeneous information sources is implemented and not even a global schema is necessarily needed. However, the consistency of the 'foreign function' can only be guaranteed by restricting the autonomy of the local source. The integrated source could not be changed unless permission is given by the global database or the administrator.

These approaches are typically used to integrate software system in a uniform way into the sharing environment [PAP92a] [HUH92]. However, much more emphasis can be put on a closer integration and coordination (cooperation) of 'problem-solving' software systems. Enterprise integration environments typically allow a direct interaction of processing systems that can participate in a common problem-solving scenario based on Partial Global Planning [DUR91a]. This cooperation does not integrate other systems which are not part of the problem-solving community, e.g. a database. However, this is a separate area that is discussed in some depth in Section 3.4 on distributed artificial intelligence, and specifically the PGP (Section 3.4.2).

In conclusion, transient data from software systems can be integrated by the information agents as information repositories, or by specialised functions (programs). Either local autonomy needs to be restricted, or schema conflicts may become obsolete over time. Data conflicts, due to inconsistent data may always occur when software systems not only receive information from the sharing environment, but also produce new information that is shared via the environment (i.e. this information may or may not be inconsistent with the persistently stored data in the sharing environment).

3.3.4 Mediators in Enterprise Integration Environments

The most loose integration of information from heterogeneous sources is the ad hoc search by independent information retrieval software. These 'agents' operate in open architectures and are, for example, implemented in the envoy system [PAL92] or mediators [WIE90] [WIE92].

> "Envoys carry out missions for users by invoking envoy-aware applications called operatives and inform users of missions' results via envoy-aware applications called informers" [[PAL92]p.233].

Envoy aware applications are monitoring a source and/or executing a task at a scheduled time. An application is integrated when it is enabled to interact with an envoy system. This 'integration' makes them envoy-aware. The approach is based on an analysis and combination of standard envoy-applications that perform mining, browsing, and other information search operations.

The envoy system also implements conflict detection that is not automated but based on decisions by the user. The system has a front-end that gathers information and presents it to the user in a 'mission summary' where all results are presented. Completeness of these results depends upon the user who has initiated the envoy to search all possible sources. The results are expected to be free of syntactic mismatches. The detection is therefore incomplete in detecting syntactic conflicts. A semantic conflict within the results is a problem for detection by the user of the information. The user in an envoy system is generally a 'human user'. In contrast, enterprise integration is concerned with clients that may be human users, or applications programs and that need automated conflict detection. Envoys fail to provide automated conflict management independent of the conflict detection or resolution capabilities of a 'human user'.

Mediators [WIE92] are independent software modules conceptually similar to envoys. They support an end-user or decision maker with information retrieved throughout an information-sharing environment.

In conclusion, the envoy system, or Wiederhold's mediators, are standalone software utilities that integrate information in an ad hoc manner. They do not try to establish integration environments. However, these utilities present a rational solution for detecting and resolving data conflicts. The task is simply shifted to the user that initiates the information retrieval. The research therefore proposes a rational solution in that it lets the user, which has to be a human expert, decide. However, no rational scheme is proposed to examine how a conflict is detected or resolved.

3.3.5 Conflict Management by Modelling Human Decision-Making

Human decision makers in traditional distributed information systems have to integrate data more or less manually. For example, in an office environment there may be a database, an expert system, a design system and some data from the shop floor. The integration of, possibly conflicting, information is therefore based on human decision-making. This process could theoretically be used to model how information should be integrated correctly (including conflict management). However, this approach has some inherent problems in it, including:

1. It is difficult to prove that a human integrator is purely rational and not biased in supporting his expectations when selecting alternative information. In Wong [WON94] the short term and long term memory is investigated in respect to its effects on decision-making by preferences. It was found that humans can only remember short term preferences, and very long term preferences. This, however, is not objective or rational but highly subjective.

2. No human expert exists in large integration environments that *de facto* integrates information. The task is typically too big to be done completely manually. No 'human information agent' exists that is as representative as, e.g., a medical expert for diagnosis expert systems, or a financial expert for a financial advisory system.

In conclusion, human decision-making may potentially provide very interesting and ground breaking solutions to conflict detection and resolution. However, no model of human decision-making in conflict detection and resolution for enterprise integration is currently available.

3.3.6 Conclusion and Summary of Architectural Assumptions

No conflict detection or resolution mechanism exists in Distributed Collaborative Environments for Enterprise Integration (DCEEI) that is complete and rational. Hence, the following section will investigate conflict detection and resolution in the wider field of distributed artificial intelligence

Furthermore, chapter 2 outlined the fundamental architectural issues of DCEEI. Further analyses of existing distributed databases and enterprise integration environments have produced the following characteristics:

- Enterprise integration environments typically have some form of **global schema or model**. This may be a master model, a unified global schema, or a federated schema. (In the case of multidatabases without global schemata (e.g. Litwin's MRDSM [LIT90]) the information typically provided by the global schema has to be gathered by the local management system.)
- Operational services, such as the construction of global schemata or models, query language translation, inter agent communication, etc., are implemented in various ways. All these implementations either have to restrict **local autonomy** or allow for schema **inconsistencies** that may lead to conflicts.
- Non-persistent data (from software systems such as an expert system) is typically integrated in the form of **information repositories** or by **functions**, which are software utilities that retrieve information from software systems. Specific dynamics of data retrieved from these sources are typically ignored in order to provide an integration with persistent data.

3.4 Conflict Detection and Resolution in Distributed Artificial Intelligence (DAI)

3.4.1 Introduction to DAI and Enterprise Integration

In Distributed Artificial Intelligence (DAI) agents are intelligent software components, e.g. a robot that communicates with other robots over achieving a joint goal. The general field of enterprise integration can be described as:

> "Research that have been investigating the use of artificial intelligence and DAI to support human organisations" [[DUR91b]p. 1303].

In other words, information agents in enterprise integration environments are a special kind of agent that integrates multiple software systems and / or processing systems [JEN92]. In this sense an information agent is a cooperation and coordination component, which integrates information from domain level data stores and processing systems. The emphasis of conflict management in DAI is that:

> "Agents can deal with ... conflicts through negotiation, the process by which the agents act to resolve inconsistent views and to reach agreement on how they should work together in order to cooperate effectively" [[LÄA92]p.292].

In DAI problem-solving, conflicts play a central role where the detection and exploitation of conflicts is an essential part of effective agent coordination [McL92] [GAL90a]. However, conflicts in distributed problem-solving are typically non-essential conflicts, between mutually acceptable solutions. Information agents in enterprise integration environments typically encounter essential and non-essential conflicts.

The following sections will investigate conflict detection and resolution mechanisms in DAI. This analysis is directed at identifying a general framework for conflict detection and resolution. The next section briefly analyses the partial global planning algorithm and its derivatives. Then a small representative selection of mainstream DAI research is discussed in an overview (Section 3.4.3). Section 3.4.4 will describe a selection of distributed planning systems. A brief discussion of task sharing versus result sharing is included in Section 3.4.5. A conclusion based on all DAI research is presented in Section 3.4.6.

3.4.2 Partial Global Planning and Derivatives

This section will begin by briefly introducing the the contract net [SMI80] and the partial global planning algorithms [LES91] [DUR87]. This research provides the basis for task decomposition, task assignment and result composition as typically used in many enterprise integration environments (Section 2.5). Hence, it is a key issue in integration environments. However, the discussion of other mainstream DAI will be left until the next section to allow the reader to explore this mainly fundamental section separately.

In a traditional contract net approach [SMI80] for a given task one agent is appointed as manager. This manager agent may be faced with a task that it may either be able to carry out itself or it has to employ other agents to help. Furthermore, dividing the task may be more efficient such that it may make better use of the distributed resources, or it could improve performance (speed).

First, the manager agent has to decompose and assign subtasks to other agents. However, this requires that a task can be subdivided into (sub)tasks that can be solved autonomously and completely by at least one agent. In other words, the manager agent needs to find subtasks that fit the abilities of each agent. Furthermore, much research assumes that the subtasks may not require any interaction with other subtasks, but that there is a perfect fit between each subtask and the resource (agent) undertaking this subtask.

> "However, there are applications that do not fit cleanly into this model of cooperation: Tasks may be inherently distributed among nodes [agents], and coordination is not a matter of decomposing and assigning tasks but instead is a matter of recognising when distributed tasks (or partial results) are part of some larger overall task (or result) and, when this is the case, how to interact to achieve the larger task (or result)" [[DEC89]p.506].

An extension to the original contract net approach is the factual accurate / cooperative paradigm (FA/C). It provides a model in which "agents need not have all the necessary information locally to solve their sub-problems, and agents interact through the asynchronous co-routine exchange of partial results" [[LES91]p.1347]. In other words the FA/C provides asynchronous, concurrent interaction between agents within the co-ordinated problem-solving. Agents can both have and exchange inconsistent and incorrect information. By exchanging partial results these inconsistencies will eventually converge on a correct solution.

The partial global planning (PGP) algorithm [DUR91a] is based on agents that build partial global plans and goal structures. In principle, these plans contain descriptions of the local action, local goals, and how they are affected by the other agents action and their goals. Each agent can theoretically have its own set of partial global plans. In practise, however, exchange of partial global plans between large numbers of agents proved not feasible [DUR91a]. Hence, some basic hierarchical structures had to be introduced to guide the problem-solving.

When an agent is activated to solve a given complex task, it first analyses its local plans and the model of the global interaction. It can then identify which agents cooperatively work on a common goal. Secondly, the agent makes a partial global plan to achieve its partial global goal (this agent's part of the overall goal). This plan includes its own plans and plans received from other agents:

> "A PGPlanner forms a plan-activity map from the separate plans by interleaving the plans' major steps using the predictions about when those steps will take place. Thus, the plan-activity map represents concurrent node activity. To improve coordination, a PGPlanner reorders the activities in the plan-activity-map using expectations or predictions about their cost, results and utilities" [[DEC89]p.507].

The optimisation strategy includes finding a better solution cheaply. In addition, the PGPlanner builds a solution-construction-graph which specifies the interactions with other agents, e.g., when to receive partial results from them.

> "To control how they [(the agents)] exchange and reason about their possibly different partial global plans, nodes [(agents)] rely on a metalevel organisation that specifies the coordination roles of each node" [[DEC89]p.508].

This metalevel organisation defines domain level cooperative problem-solving. For example, it specifies what roles agents play such that they are hierarchically organised. "For limited times, agents can work on partial global plans that are inconsistent since the same agents views of other agents is out of date" [[LES91]p.1358]. These inconsistencies are solved when agents exchange partial results.

Potential problems with the partial global planning algorithm are about "how to tolerate and resolve inconsistent and incorrect information, and about how to control the formation and propagation of partial solutions to bound the combinatorial complexity of distributed problem-solving" [[DUR91b]p.1302]. The original paradigm was developed

for, and successfully operated in, the Distributed Vehicle Monitoring Testbed [DUR91a]. One improvement of the original mechanism is to make it domain-independent:

> "Generalised Partial Global Planning (GPGP) tries to extend the PGP approach by communicating more abstract and hierarchically organised information, detecting in a general way the coordination relationships that are needed by the partial global planning mechanism, and separating the process of coordination from local scheduling" [[DEC92]p.320]. In principle, "the emphasis in this work is to define the existing goal relationships generically and to add new relationships so that more complex interactions among the local search spaces can be captured" [[LES91]p.1360].

Another problem with the original PGP is that it "cannot dynamically reason about the most important goals for generating a global solution and the best information to satisfy these goals" [[CAR91]p.192]. Simple heuristics are used such as 'avoid redundant work', 'provide predictive results', or 'shift tasks to idle notes' [DUR91a].

> "However, the appropriateness of such heuristics depends on the situation. If there is a great deal of uncertainty in overlapping solution areas then "redundant" work could be very useful" [[CAR91]p.192].

Section 2.5 has outlined that much research in enterprise integration has adopted the PGP algorithm as a means of task decomposition and assignment. Nevertheless, this mechanism fails to provide adequate conflict detection. Conflict resolution is based on heuristics to optimise the problem-solving process ('Shift tasks to idle nodes'), and retrieval optimisation in information-sharing ('Request the result that is cheapest to be produced and communicated [PAN91a]). However, these heuristics may not be appropriate for any specific conflict in DAI or in enterprise integration. The following two approaches address this problem by enriching the inter-agent collaboration (more information is exchanged and multi-step cooperation mechanism for the agents are introduced).

Carver *et al.* [CAR91] propose an explicit representation of the 'solution convergence process'. In other words, each agent has a representation of what is necessary to solve the global goal and problem. The sub-goals and sub-plans that need to be terminated to achieve the global goal and that are not secure are called uncertain.

> "Termination in interpreting problems requires that the systems not only consider whether existing hypotheses are sufficiently proved or discounted, but must also consider whether enough of the data has been examined to be sufficiently sure

that no additional answers may be found - without having to examine all of the data" [[CAR91]p.194].

In principle, this requires very well structured problems, and / or time dependent results. Therefore, satisfying these uncertainties dictates the problem-solving process. A local and an extended global model (Model PS) contain the information on the status of the sub-goals and global goals respectively.

In addition, to holding explicit goals, and status information to reduce inconsistencies, agents also exchange explicit evidence for a hypothesis. For example, evidence includes "partial evidence, possible alternative explanations, possible alternative support, alternative extensions (hypothesis versions), negative evidence, and uncertain constraints" [[CAR91]p.194]. If this evidence is provided by other agents then it is called 'external evidence'. Please note the difference to the original PGP which was solely relying on exchanging partial results.

Krin *et al.* [KIR91a][KIR91b] describe a federated agent system that is based on a derivative from the contract net approach with similarities to the partial global planning algorithm. It adopts the coordination mechanism of the PGP and adds negotiation / self assessment heuristics. Each agent describes its own capabilities and distributes these. For a given task one agent is appointed 'manager'. It first analyses if it can fulfil the task itself or if it needs to find other agents to cooperate with it in solving the task. Task analysis leads to identifying all agents that can possibly do the given sub-task.

> "[It then] communicates the subtasks to all agents which then behave as bidders. They asses their own competence for each of the subtasks and return the results to the manager ... Finally the manager optimises the allocation table (which has been defined as a relation: [task_id, agent_id, competence_rating, constraints]) to find an optimal solution ... Constraints may define time-out points, restricted capacities of the agents, etc." [[KIR91b]p.195].

No central control is provided, so the agents need to bid for solving sub-tasks based on a self assessment. This is specified in the 'competence' variable. The following are examples for competence assessment:

- Agents could give a rating as a "measure to rate the quality of output expected from the knowledge evaluation" [[KIR91b]p.198];
- Given a number of input data that is required for a given task, an agent's competence may depend on how many of these input data it can evaluate;
- Particular problems (e.g. construction, simulation, diagnosis) may fit particular problem-solving strategies (e.g. skeleton planning, different search strategies) better than others;

- An agent's physical resource capability to carry out a given task such as memory size, or processor capabilities.

An overview of the contract net based / PGP related mechanisms is listed in Table 2. For each mechanism the kind of conflict is specified such as task assignment (task allocation), or goal coordination. It is then defined if the mechanism is based on a derivative of the contract net (e.g. the partial global planning algorithm). The type of conflict that occurs between agents can be described as:

- Agents co-ordinate (coor.) their action, typically in a benevolent fashion in order to achieve a common overall goal;
- Cooperating (coop.) agents agree to negotiate with each other but they may have not only common goals but also individual goals so that they may compete for having their individual goals fulfilled .

The third column in Table 2 indicates the resolution strategy for every paper as described here. The last column lists the paper references. The structure of the table is uniform for Tables 2,3,4 and 8.

Kind of Conflict	Contract Net	Conflict Type	Resolution Strategy	Paper Reference
Task Allocation	Yes	coor.	Partial global planning;	[DUR91a] [DUR87]
Task Allocation	Yes	coor.	General partial global planning, domain-independent with more complex relations between sources;	[DEC92]
Task Allocation & Negotiation	Yes	coor.	Partial global planning with explicit solution convergence and explicit evidence, negotiation to reduce inconsistency;	[CAR91]
Task Allocation & Self-Assessment	Yes	coop.	Competence rating based on multiple criteria including self-rating by the agent, best fit of problem, and applicability of the local resolution strategy;	[KIR91a] [KIR91b]

Table 2: Partial Global Planning and Derivatives

In conclusion, most of the PGP research is concerned with solving the decomposition problem, the task assignment problem, and the composition problem [DUR89]. The research listed in Table 2 demonstrates how this mechanism provides a method for information-intensive problem-solving. In other words, it is specifically useful for information agents that have only limited knowledge of the problem structure. For this reason enterprise integration environments described in Section 3.3 have been using the PGP or its derivatives [PAN91a] [PAP92a] [JAG92] [BAR94a]. Extensions to the original PGP mechanism provide the agent with more complex problem-solving capabilities (negotiation) and enable it to also investigate other evidence than partial planes. However, the mechanism does not account for essential and data conflicts (incomplete, incorrect or obsolete information) that require conflict detection and further resolution.

3.4.3 Mainstream Distributed Artificial Intelligence

Mainstream distributed artificial intelligence (DAI) is concerned specifically with complex problem-solving structures and negotiation. Negotiation is, in general, "the process of improving agreement (reducing inconsistency and uncertainty) on common viewpoints or plans through the structured exchange of relevant information" [[DUR89]p.230]. Table 3 briefly lists a few general negotiation mechanisms for conflict resolution. Each approach is shortly described in Appendix A. Only some core ideas from these approaches will be covered in this section.

Kind of Conflict	Contract Net	Conflict Type	Resolution Strategy	Paper Reference
Task Assignment	Yes	coop.	Negotiation consists of proposal, critique, explanation and resolution heuristics;	[LÄA92]
Goals in DPS	No	coop.	Compromise based on case-based reasoning and preference analysis, persuasion to change agents goals;	[SYC89]
Goals in Design Systems	No	coop.	Compromise by alternatives and domain-dependent heuristic decision-making if compromise fails;	[WER91]
Goals in DPS	No	coop.	Compromise by redefining goals and integrative negotiation (most important goals of all parties are integrated);	[LAN89]
Goals in DPS	No	coop.	Fit to the problem and time/cost optimisation of resolution mechanism, human decision for essential conflicts;	[STE90]
Goals in DPS	No	coop.	Game theory in adverbial situations, including a rating of 'worth';	[EPH91] [ZLO91]
Human / Computer	No	coop.	Human user makes a decision;	[STO91]
Learning in Co-ordination of Agents	No	coop.	Manager agent decides with the help of a learning system which is building in form of a case-base for use in coordination (similar to SYC89);	[VIT91]

Table 3: Conflict Detection in DAI

The emphasis in DAI lies on the resolution of very specific, well defined problems and / or conflicts. Very well defined conflicts can be easily detected by domain (problem specific) heuristics. Resolution strategies are based on some form of optimality, priorities, strength of goals, urgency of resource requirements, expected utilities of the alternative solutions, etc. The following are some examples of resolution strategies with these characteristics:

- Agents in the approach by **Läarsri** *et al.* [LÄA92] detect conflicts in the concurrence of action they have already taken. The negation is blackboard based including the phases proposal, and critique which leads to a new proposal until the conflict is resolved.

- Compromise is proposed by Sycara's managing (persuading) agent that applies case based reasoning mechanism and preference analysis (Persuador) [SYC89].
- An extension to proposing compromises is domain-dependent heuristic decision-making as, e.g., described by Werkman [WER91].
- Furthermore, compromise may be achieved by redefining goals. If this fails then integrative negotiation may be able to integrate the most important goals of all parties in Lander and Lesser's approach [LAN89].
- The optimisation strategy 'fit to the problem' evaluates which agent can perform a given task most completely. Steiner *et al.* [STE90] also propose time and / or cost optimisation, and for essential conflicts human decision-making.
- Game theory is a very efficient way to resolve non-essential conflicts in adverbial situations. It requires that all solutions and their pay off are known to the agent [EPH91] [ZLO91]. Agents in enterprise integration, for example, typically lack this omniscience (Section 2.8.1).

These mechanisms have in common that they can only address conflicts where the alternatives are mutually acceptable and not conflicts in which one alternative may be correct and the other incorrect. Thus, these resolution strategies can only be applied by a conflict detection and resolution mechanism as a means to resolve non-essential conflicts.

A proposal to resolve essential conflicts is, for example, presented by Stolze and Gutknecht [STO91] or Steiner *et al.* [STE90], who propose conflict resolution by a human user. However, this is a rational approach to conflict resolution but not one that is feasible for complex conflicts or non-expert users. It is, hence, unrealistic for large enterprise integration environments.

Conflict managing agents, e.g. the Persuador by Sycara [SYC89] or Arbitrator by Werkman [WER91], propose solutions to the other agents involved in a conflict until a compromise or solution is found. Hence, such mediators architecturally suit the concept of an information agent that manages a conflict in an enterprise integration environment (Section 2.5).

Finally, agent learning mechanisms are proposed, for example, by Vittal *et al.* [VIT91] or Gasser and Ishida [GAS91]. The latter approach also emphases the learning of meta information, which has been termed Agent Knowledge in Section 2.7. Learning and adoption are key issues in dynamically changing environments. They are, therefore, particularly important to information agents in enterprise integration environments.

The coupling of traditional rule-based systems has been the focus of much research, for example, by Carlson and Ram [CARL91] or Su and Park [SU 91]. These approaches are described in Appendix A. The problem addressed in these approaches is one of merging multiple rule and facts bases rather than establishing cooperation or co-ordination mechanism between these systems. For this reason, conflict management is only required when the global system is installed. Detection is limited to detecting that multiple results are not identical. Resolution is provided by predefined heuristics or left to the integrator.

Distributed knowledge-bases are concerned with traditional, standalone systems that are tightly integrated into a global derivation system. Thus they are not 'real-time' in respect to their conflict resolution such as rule-based agents as the briefly mentioned approaches by Steiner *et al.* [STE90] and Kirn and Schlageter [KIR91a]. Both worlds are merged by research such as the HOPES approach by Dai *et al.* [DAI93]. It is concerned with real-time distributed knowledge-bases that apply DAI techniques. In other words, the knowledge-bases cooperate via a hierarchical implementation of a blackboard architecture [NII86]. Thus, this conflict management shares much with mainstream DAI approaches described in the previous Section.

The research by Su and Park [SU 91] elaborated the dynamic behaviour of knowledge in rule-based systems. In other words, it is shown that single, and specifically multiple rule-based systems may produce information that needs to be investigated in respect to its behaviour. For example, results may vary in respect to the time and the frequency that they are requested. Information agents potentially have to integrate not only persistent but also dynamically changing, transient data. Integrating the latter may require special processing, for example, with respect to the time and frequency this information is requested.

In summary, multiple negotiation strategies exist for resolving non-essential conflicts. These are typically very domain and conflict dependent. Learning and adoption are crucial requirements for information agents in dynamically changing integration environments. Finally, 'knowledge' or information from derivation systems has a dynamic behaviour that needs to be investigated in order to ensure semantically correct integration.

3.4.4 Conflict Detection and Resolution in Distributed Planning

Distributed planning systems are a sub-field of distributed artificial intelligence that is particularly concerned with conflict detection and resolution in the area of planning and designing. Three examples are briefly investigated here: Adler *et al.* [ADL89], Polat and Güvenir [POLA92], and Klein and Lu [KLE89] (Table 4).

Kind of Conflict	Contract Net	Conflict Type	Resolution Strategy	Paper Reference
Goals in Planning & Control	No	coop.	Conflicts are avoided by goal coordination, or conflicts are resolved at run-time by goal relaxation or removal;	[ADL89]
Design	No	coop.	Conflict detection is domain-dependent, conflict resolution has the four stages: Conflict statement; Resolution proposal; Evaluation; Critique;	[POLA92]
Goals in Design	No	coop.	Conflict classes and according resolution strategies are matched by a resolution expert;	[KLE89] [KLE91]

Table 4: Distributed Planning

Adler *et al.* [ADL89] deal with homogeneous agents in telephone network management. The agents have responsibilities, goals and plans within their own region, which have no overlaps with other agent's regions.

> "Conflicts occur when two or more agents want to use the same resource ... that cannot handle both their demands simultaneously (a resource level conflict), or when one agent has goals that cannot be achieved if another agent's goals are to be realised (a goal level conflict)" [[ADL89]p.146].

Conflicts can be detected while agents execute their plans, which leads to 'conflict driven plan merging'. This conflict resolution, therefore, is real-time rather than advanced planning to avoid conflicts. Resolution involves negotiation by agents that are involved in a conflict. They successively revise their goals starting with goals that are at the lowest (least important) level of a goal tree until the conflict is removed.

Polat and Güvenir [POLA92] have developed a conflict resolution mechanism that is suitable for implementation in systems with varying numbers of agents. It achieves a degree of openness (as defined in [HEW91] Section 2.3) by giving each agent its own "conflict resolution expertise separate from its domain-level design expertise, and that this expertise can be instantiated in the context of particular conflicts into specific advice for resolving these conflicts" [[POLA92]p.107].

A conflict can be detected by a design agent that will then become the 'originator' for this conflict. The originator has to post the conflict centrally on a blackboard. The conflict is

described by its 'originator', the goals it needs to accomplish, the constraints it needs to satisfy, and an initial problem description. All agents are informed of the conflict and can then post proposals on the blackboard. A proposal includes the originator, the proposed action, if possible any reasons the agent has for supporting its proposal and a degree of confidence it has in this proposal. Agents evaluate proposals and critique on them. A result of an evaluation is either conflicting or non-conflicting. If agents have evaluated the proposal as conflicting then they continue negotiating for further compromises. However, it may be that some problems cannot be solved at all.

Agents can 'learn' in two respects: An agent can maintain information on solution generation and, secondly, it can memorise information on the conflict resolution process. In other words, the agent can build its own case-base for later problem-solving.

In addition, agents may detect conflicts in planing their future action, which is called 'shared plan development'. This advanced planning results in a network that is optimised by running conflict free and / or more cost effectively. The resolution is based on a joint planning activity of the agents. They start with planning their most important, top level goals. These are then co-ordinated with the other agents. The next, lower, level is planned and agreed, until the agents have developed conflict free plans.

The previous approach searches for any resolution strategy, which is proposed on the blackboard, and that is an acceptable compromise for all agents. However, this is only any solution, and may not be the best resolution for this specific conflict. The conflict detection and resolution approach of **Klein and Lu** [KLE89] overcomes this problem in that it specifies how agents can apply multiple resolution strategies to a given conflict. It is implemented for cooperative design agents that focus on benevolent cooperation and not competition such as in [EPH91].

Conflicts are detected by human experts, for example, "by noting two incompatible design commitments, or when a design agent has a negative critique of another agent's actions [such that] a description of this conflict is given to a conflict resolution expert" [[KLE89]p.172].

The resolution strategy is based on the resolution expert's conflict resolution knowledge. The latter is composed of explicit resolution strategies that are assigned to conflict classes.

> "When we choose a particular strategy for a conflict, then we are in effect making the hypothesis that the conflict is one that can be addressed by the given piece of

advice, and must be able to respond appropriately if the advice fails" [[KLE91]p.1382].

Hence, for every resolution strategy there are specific conflicts that they can solve and there is a degree of generality attached to them. Consequently, a conflict hierarchy is developed in which conflicts are organised hierarchically from the least specific, domain-independent conflicts to increasingly specific domain-dependent conflicts. These conflict classes are assigned conflict resolution strategies. A less specific conflict resolution strategy is less efficient. However, a domain-independent, very specific resolution is only applicable to a very specific conflict or range of conflicts.

> "When a conflict occurs, we can find the most specific conflict class that subsumes that conflict, and try the conflict resolution strategies associated with that class" [[KLE89]p.172].

The domain-independent portion of the conflict hierarchy can be designed such that it can be used for a very wide range of domains. The domain-dependent part has to be developed for every specific domain. Thus this part of the approach is particularly domain-dependent and will prove difficult for adoption by other areas. .

An example of how general conflict resolution expertise can be classified is demonstrated in [KLE91]. In this example, cooperative problem-solving is divided into multiple subsections including cooperative diagnosis or meta planning formalisms. The latter is further divided into conflict resolution, cooperative schedule maintenance and routine design. Figure 4 outlines a possible hierarchy for conflict resolution based on the resolution strategies described in this Section.

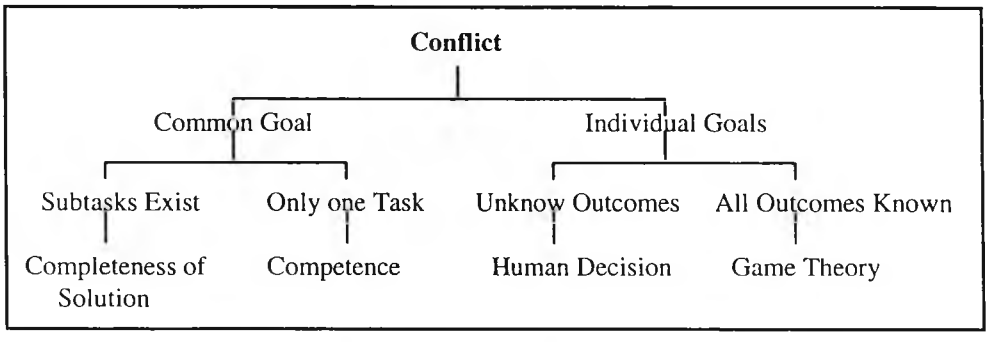

Figure 4: Example Conflict Hierarchy

Some resolution strategies are applicable when all agents have a common goal, and other strategies are applicable if agents pursue individual goals. In the first case, the common task may be divided into subtasks, in which case strategies such as 'best fit to the problem' [KIR91a] are applicable. If there are no subtasks then the agent that is most competent for the whole job has to be identified. Game theoretic rating of worth

[ZLO91] is only applicable if the agents do not share a common goal and if all outcomes are known. Otherwise decision by a human user [STO91] may be applicable.

One problem with the approach is that resolution of a conflict is a one shot approach [LÄA92]. Complex problems may require the consideration of multiple, interrelated resolution steps

In conclusion, traditional planning systems propose conflict resolution by goal relaxation [ADL89]. However, the previous two approaches identified conflict classes, either as goal directed [POLA92] or conflict resolution strategy directed [KLE89]. In the latter case the appropriate resolution strategies are assigned to hierarchically organised conflict classes. The principle for systematising these strategies is to:

> 'Apply domain level strategies first, before more general resolution strategies are applied.'

Information agents are no domain experts. In other words, they have no knowledge of local problem-solving goals, such as planning agents described by Polat and Güvenir [POLA92]. However, they have different resolution strategies which they need to apply to a given conflict (outlined in Section 2.7). Hence, the strategy 'domain level resolution first', in principle, should be applicable to conflict resolution by information agents. Other aspects of conflict resolution need to be added to this principle in order to prevent making this a 'one shot approach' [LÄA92], and to include learning [POLA92].

3.4.5 Task Sharing and Result Sharing

> "Result sharing concentrates on problem domains where tasks are inherently and possibly unpredictable distributed" [[DUR91a]p.1170].

If the systems have related tasks to other systems but not enough information about the meta level goals and coordination with the other sources, these can only exchange possibly partial results. In other words, these systems cannot jointly carry out a task (share a task), but they can exchange results from their isolated problem-solving activity.

> "Result-sharing through iterative exchange of tentative, uncertain information has been termed functionally accurate, cooperative [(FA/C)]" [[DUR91a]p.1170].

The FA/C mechanism has been described in Section 3.4.2 as it is typically used in enterprise integration environments.

Distributed Collaborative Environments for Enterprise Integration are result sharing systems. In other words, information agents share information. This exchange is benevolent, the agents freely exchange information that is requested from them. However, the integrated source itself may be inherently and unpredictably related to other sources.

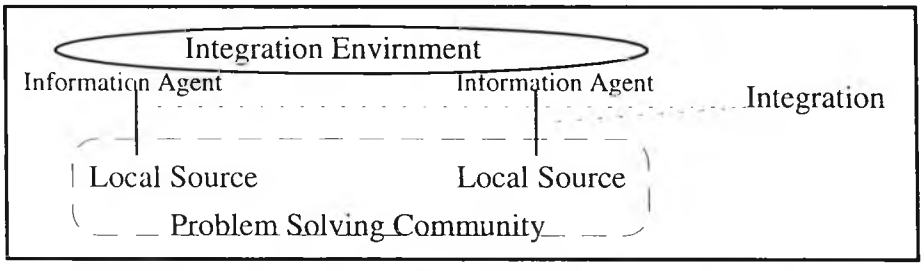

Figure 5: Agent Communities within Enterprise Integration

In other words, cooperative problem-solving communities may exist within the enterprise integration framework (Figure 5). These communities share not only results but typically share a task they jointly resolve. An example may be a community of agents (systems) that cooperate based on game theoretic negotiation protocols [EPH91]. These communities are closed in respect to their problem-solving activity.

The previous sections have described some example problem-solving systems and mechanism. A system or agent that participates in a problem-solving community needs to be aware of the negotiation modalities, goals, it may need to be registered with the other agents, etc. An information agent is typically not part of this domain level

problem-solving (Section 2.3). However, it can integrate this community into the open enterprise framework. This integration, in principle, can include the following:

1. An information agent may enable sharing results from this community and / or provide it with information from throughout the enterprise.
2. The agent may integrate a particular system which is also integrated in a problem-solving community. An information agent typically knows if the source it integrates is involved in a task sharing commitment with other sources (Environmental Information - Resource Knowledge in Section 2.7). The agent can then make assumptions about the stability of results, the interaction with other sources and take the dynamics of the integrated system into account.
3. The agent may use a problem-solving community as a source for expert conflict resolution (Services - Resource Knowledge in Section 2.7). In other words, the community may be able to resolve certain, typically non-essential, conflicts. An information agent may be able to present a specific kind of a conflict to this community which resolves the conflict and returns the result to the agent.

In conclusion, multiple levels of integration exist in the enterprise environment. The information agent based sharing environment is the highest integration level. Information systems may form integrated communities within the sharing environment. They may have multiple links to information agents that communicate the results from a local, possibly distributed problem-solving process on the higher enterprise wide level.

3.4.6 Conclusion Conflict Detection and Resolution in DAI

Conflict detection in DAI is typically based on either detecting that '*multiple results are not identical*' or the detection is highly domain-specific. An example of the latter case is distributed planning where conflicts require a domain expert or the user.

The resolution of non-essential conflicts is the main focus of DAI research. Multiple strategies are presented that suit specific conflicts. For example, compromise is specifically suitable for agents that negotiate deals, or otherwise optimise goals. The research by Klein [KLE91] suggests a way, which is suitable for information agents, to apply these strategies such that the most domain level strategies are applied first.

However, no general framework for conflict detection and resolution is proposed. Furthermore, conflict detection is not specifically in the focus of mainstream DAI. It is clear that uncertainty management is a field that emphasises, typically formal, detection of conflicts.

3.5 Uncertainty Management in Artificial Intelligence

3.5.1 Introduction

The previous section has looked at distributed problem-solving with particular focus on conflict detection and resolution. In principle, all previously described systems deal with uncertainty as do most tasks within the field of Artificial Intelligence [CLA90b]. Most research in enterprise integration (Section 3.3) draws heavily from the area of uncertainty management. For example, Huhn's [HUH90] approach to integrating information sources is based on non-monotonic truth maintenance systems, and decision theory is used by Papazoglou *et al.* [PAP92a] or Pan and Tenenbaum [PAN91a].

Uncertainty is, furthermore, central to this research as resolving conflicts is concerned with resolving uncertainty about conflicting propositions or results. The field of uncertainty management may, hence, provide a scheme to assess uncertain propositions based on evidence and information available to information agents in enterprise integration environments (Section 2.7). However, such a scheme needs to suit the circumstances of conflict management in enterprise integration as outlined in Chapter 2. It includes, for example, the following cornerstones:

1. Results and Descriptive Information (e.g. schemata or environmental information) may be incomplete, incorrect or obsolete (Section 2.5).
2. Any result typically is not from a single closed body of evidence (Section 2.8.2 and 3.4.5) so that the evidence for and against a proposition may be incomplete.
3. Evidence that warrants or refutes any hypothesis (candidates) may not be ordered or ranked in a pre-defined way, and typically is not numerical, such as probability factors to measure trust in a solution. (Section 2.7).
4. A Principle Rational scheme for uncertainty management is required and not one, e.g., based on subjective preferences (Section 2.6).

Uncertainty management, in principle, can be done by quantitative methods and by qualitative methods. Quantitative approaches include Bayesian networks, certainty factors attached to heuristics, belief functions, and possibility theory (Section 3.5.2).How qualitative justifications and assumptions can be assigned to propositions in information systems will be demonstrated by non-monotonic uncertainty management in Truth Maintenance Systems (Section 3.5.3.1). Section 3.5.3.2 will look at the closely related field of belief revision. Decision-making theories are briefly discussed as a source for modelling conflict detection and resolution (Section 3.5.3.3). Argumentation is a way to deal with uncertainty by modelling human reasoning in a generic manner (Section 3.5.3.4). This is followed by conclusions drawn from uncertainty management.

3.5.2 Quantitative Methods to Uncertainty Management

Quantitative methods can be defined as "traditional (i.e. probabilistic) approaches to uncertainty management [which] have tended to focus upon viewing uncertainty either as a frequentistic measure of randomness or in terms of a subjective measure of confidence satisfying well-circumscribed propositions" [[KRA93]p.3].

This section will look at Bayesian probability, Dempster-Shafer theory, and possibilistic methods. These are well-known quantitative methods of uncertainty management that make it possible to demonstrate the basic notions of the quantitative approach.

Bayesian probability is based on an ideal person's degree of belief in a hypothesis. This degree is specified as a number between 0 and 1. It is based on a possibly non-monotonic function of belief defined for an axiomatic system, e.g., as described by [COX46]. The representation and manipulation of probabilistic knowledge in graphical form is called 'belief networks', 'Bayesian networks' or 'causality networks' [PEA93]. In other words, given a piece of evidence E1 then this is the cause for someone to believe a proposition P1. The probability of the proposition P1 in the light of the evidence is evaluated by multiplying the probability of the evidence with the probability of the proposition P1. This result is then normalised by dividing it by the probability of the evidence.

The graphical representation of probabilistic knowledge makes explicit cases in which multiple evidence supports one hypothesis. However, for these cases strong independence assumptions have to be made. Otherwise the dependencies of multiple pieces of evidence can be intrinsically incorporated by a single rule [PEA88].

Typically Bayesian probability is numeric. However, there are approaches based on non-numerical coefficients. For example, Halpern and Fagin [HAL92a] have described a way to use not only absolute probabilities but to put one proposition in relation to another.

Dempster-Shafer theory [SHA76] [DEM67] extends the Bayesian model in that:

1. Belief functions are set functions and not point values. Set functions consign a range of values, a set or a group of things (e.g. a football team) without reference to a particular instance (e.g. one player).

2. Beliefs may have complete non-commitment (neither believed nor disbelieved).

3. Dempster's rule for combining probabilities is similar to weights on a hypothesis. The rule is based on multiplying the probabilities of two alternative rules (A and B) and to divide that by one minus that probability ((A∗B)/1-(A∗B)). The probabilities lie between one (total belief) and minus one total disbelief. In the case of total conflict (A=1 and B=-1) then the probability is zero (0 = (1∗-1)/(1-1)). If there is partially conflicting evidence such as 'A = 0.9' and 'B = 0' then the combined probability is also zero. If there are multiple pieces of evidence by the parties A and B, for example, the pairs 'A_1&B_1, A_2&B_2, A_3&B_3' then those pairs that partially conflict sum up to zero. Furthermore, all conflicting pairs are normalised and the remaining evidence is taken to determine the case. This means that strong, partially conflicting evidence may not be considered and very weak evidence for or against a case may be decisive. For example, very weak but joint support for one solution in the form of 'A = 0,1' and 'B = 0,1' turns the scale.

The Dempster-Shafer theory may be applicable in cases where evidence from experts is integrated (such as medical diagnosis) or where sensory data is evaluated that has hierarchical belief structures. In other words, the rule of combination typically follows hierarchical structures of hypotheses such as a group of hypotheses sums up into one combined hypotheses (the group). The system is therefore very applicable for hierarchical problems, e.g. interpretation or diagnosis systems, where the system can follow down well described trees of probability. Kruse and Schwecke [KRU91] present an approach to combine knowledge-bases built on the Dempster-Shafer theory. Uncertainty is assigned to the information according to the faith the expert puts into a piece of information. However, in a knowledge-based system information is hierarchically and rule-based. Enterprise integration systems not only combine information from expert systems or knowledge-bases but from the whole range of information systems (Section 2.3).

However, the theory allows for incomplete knowledge of the members of a set. In the Bayesian model the evidence for and against the hypothesis always sums up to one. The Dempster-Shafer rule also allows for combining the existing evidence without having to judge on the completeness of evidence for and against the hypothesis.

In the Dempster-Shafer model meta-level reasoning may be required to decide which value out of a range (set) of values to take. Either the lowest, the highest or the average could be selected. Furthermore, it requires numerical certainty values and can, for

example, not incorporate fuzzy sets. It can deal with probability values for a range such as '45%' but not with 'very high', 'high' or 'good'.

Probability theory is not suitable for most scenarios in enterprise integration because of its basic emphasis:

> "Probability theory is centred on the notion of uncertainty as it applies to atomic or conditional propositions. Therefore, concepts such as ambiguity, inconsistency, incompleteness and irrelevance are not permitted, and are not formally part of the Bayesian model [or the Dempster-Shafer theory]" [[KRA93]p.46].

The previous paragraphs have shown that classical Bayesian theory often is inadequate for uncertainty management processes because of the problem involved with identifying the numerical estimates (e.g. of human decision makers). This principle finding has been specifically demonstrated for evidential reasoning by An, Bell and Hughes [AN 93] who demonstrated that:

> "It is difficult to justify decisions based on numerical degrees of belief and to answer questions such as 'How do changes in the numbers affect the answers'"[[AN 93b] p.231]

In this paper it is then shown how relation-based reasoning can overcome the problem of defining numerical degrees. Alternative judgements, that support or refute a hypothesis, can be related by comparing their strength. The approach is based on an 'evidential mapping' that uses mass functions to express uncertainty relationships between evidence and hypothesis groups that are based on this evidence [GUA92].

In contrast to traditional probability theory, **possibilistic logic** is used to deal with uncertainty and / or vagueness.

> "Possibilistic logic involves certainty and possibility degrees which are not compositional for all connectives and which are attached to classical formulae ..." [[DUB91]p.53].

In other words, propositions are combined by first order logic. Each clause can have a possibility label, giving the maximal probability of the clause being true. In addition, each clause can have a necessity label attached describing its minimum probability. When the clauses are combined to prove a consequence then this is

(a) no less certain than the least of its parent clauses, and

(b) at most as possible as the greatest of its parents.

The formal definition and the application for combining information sources can, for example, be found in [DUB92]. However, this approach does not allow for reinforcement such as the Dempster-Shafer rule of combination. Thus no matter how many clauses exist to prove a hypothesis, its certainty cannot increase over the certainty of the strongest single clause.

In principle, uncertainties can be described as probabilities or less precise values such as 'very high', 'low', which are called fuzzy sets. Fuzzy sets consist of a set of values that are assigned to a particular concept by a membership function. For example, the concept 'poor student' can be defined by the membership function 'have less than 400 pounds per month'. If, however, the concept is not so clearly defined, then it may be graded where the degree of poverty increases as the monthly income gets closer to '400 pounds per month'. If, however, there are also other concepts such as 'normal student' and 'rich student' then there may be an intersection between these concepts. The membership functions can be used to define if a given entry (student) is a member of a particular set or multiple sets. In addition, it may be determined how strong the membership is such as '400 pounds per month' is a strong member of the set 'poor student', but '600 pounds per month' is a very weak member of the function 'poor student'. Cut off points can be defined to limit the membership (called 'α cuts').

The derivation of what is possible, necessary or what are the correct membership functions is clearly more difficult than the election of single probabilities.

> "The bottom line with respect to elicitation, therefore, is that it is either highly laborious, as with the direct technique requiring many trials, or it is assumption ridden" [[KRA93]p.139].

In the opening section information-sharing environments have been described as exchanging very poorly structured information that is typically not even ranked or ordered. In enterprise integration fuzzy approaches to reason about uncertain information often fail because of a lack of information on membership functions.

In summary, quantitative methods are based on assigning (un)certainty values to rules or propositions. Syntactic measures are used to combine these uncertainties, which makes them computationally very efficient. Typically, however, very strict requirements exist for qualitative approaches concerning:

- The qualitative nature of the evidence (uncertainty values), for example, "the attachment of numbers to pieces of knowledge according to some experts valuation of the respective sources" [[KRU91]p.24];

- The structure of the problem, for example, in Bayesian networks.

Most quantitative approaches are designed for expert systems or knowledge-based systems which can easily meet these requirements. However, in enterprise integration conflicts often do not fit these requirements. As indicated in the opening Section 3.5.1 evidence may, for example,

(i) not have certainty values specified for it,

(ii) different systems may use certainty estimates that have different basis and are, hence, incommensurate, or

(iii) different problem structures exist which may have to be changed generically.

The attachment of numbers to evidence or to mutually exclusive hypotheses which result from the evidence, is difficult to realise for the information agents themselves. First, they generally lack the domain expertise. Second, this approach is difficult, for example, for the integration of knowledge-bases:

> "Numeric uncertainty factors ... are downright dangerous in huge knowledge-bases, as any two numbers can be compared with each other, and the three digit numbers were often little more than fabricated ways of expressing the fact that one assertion was a bit 'more likely than' another assertion" [[GUH94]p.128].

Non-numerical relation-base approaches overcome the problem of numerical certainty values but they require domain experts. This makes the approach very suitable for modelling expert knowledge. However, as mentioned in the introduction (Section 3.5.1) information agents typically lack such domain expertise. It may integrate domain experts (e.g. in the form of an expert system) but it is no domain expert itself. In other words, an information agent would never have the domain expertise in order to identify that one result A, is 'more reliable' than another result B, base on problem (domain) specific facts.

Because a strictly mathematical framework is not very suitable for mainly non-numerical, qualitative evidence, a semantic (or qualitative) framework is required. A conflict detection and resolution scheme will need to be based on a semantic framework. However, such a scheme would take both the qualitative and quantitative, mainly numerical information, into account (Section 2.7 lists the available information in the sharing environment .).

3.5.3 Qualitative Methods to Uncertainty Management

3.5.3.1 Non-monotonic Uncertainty in Truth Maintenance Systems

Qualitative methods are based on the assumption that "epistemic probability are derived from an appraisal of arguments and need not necessarily be numerical" [[KRA93]p.241]. Rational schemes for resolving conflicts by means of compromise and deals, or assessing arguments in enterprise integration, e.g. [BAR94a], are examples of domain or problem specific qualitative approaches to uncertainty management. Truth maintenance systems are a very good example of qualitative uncertainty management.

Possibilistic logic can be monotonic and non-monotonic. In the first case, the logic is contingent in that it represents an ultimate state. New information will not change the conclusions. Non-monotonic proofs are contingent proofs, which may change. For uncertainty management the latter case is of particular interest in that it reflects the uncertainty about new, yet unknown information. The most typical example of non-monotonic reasoning in information systems are truth maintenance systems.

> "The truth maintenance system (TMS) is a problem solver subsystem for reasoning programs ... to make assumptions and subsequently revise their beliefs when discoveries contradict these assumptions" [[DOY79]p.231].

In other words, all propositions are formulated as epistemic attitudes of the form 'believed' or 'not believed'. These beliefs are based on reasons in the form of justifications or assumptions. In the latter case, for example, "for each statement or proposition P just one of two states obtains: Either

(a) P has at least one currently acceptable (valid) reason, and is thus a member of the current set of beliefs, or

(b) P has no currently acceptable reason (either no reason at all, or only unacceptable ones), and is thus not a member of the current set of beliefs" [[DOY79]p.234].

A conflict has occurred if a proposition P is believed and disbelieved at the same time. In other words, a conflict consists of a reason for believing and another reason for disbelieving the proposition. The detection of conflicts in truth maintenance systems is therefore based on observing true, logical conflicts. These conflicts are called 'implicit conflicts' in Section 3.2.1.

Reasons for believing a proposition P and those for not believing a proposition P can be attached to that proposition. In a system with multiple propositions or beliefs, and

reasons for these beliefs, networks are constructed [PEA93]. It is then possible to retrace the effect of adopting a new belief on the existing network (non-monotonic). For example, the reason 'If a bear is a comic actor then it is Yogi', may exist and it may currently be believed that 'The bear is called Yogi'. However, if the bear is believed 'not to be a comic actor' then this new belief would make the belief 'The bear is called Yogi' invalid. In other words, the belief 'The bear is called Yogi' only remains believed if it is also believed that the bear is a comic actor. Otherwise a conflict exists where either the new belief ('The bear is not a comic actor') is not accepted, or the belief 'The bear is called Yogi' is dropped.

> A TMS is designed to detect conflicts in that it is a "module of a problem-solving system responsible for maintaining the system's beliefs according to certain principles of coherence and rationality" [[ZLA92]p.67].

Principles of coherence are typically assumptions or justifications by which a proposition is believed. In the same way, rules can be specified for enforcing consistency on a system such as, for example, a database. These rules could form a notion of coherence to maintain, e.g., the database consistent [ZLA92].

In summary, there are different approaches to enforce coherence and consistency in information systems. All approaches detect explicit conflicts. Detection of implicit conflicts is done by analysing the logical consequence of adding new beliefs or changing existing beliefs within the current set of beliefs. How this is done depends on the ways coherence is implemented. It has been briefly shown how this can be done by backtracking reasons for existing beliefs.

Any derivation system is potentially at risk of being inconsistent in two ways:

1. Internal consistency may be in question when, for example, updates or change of the internal rules and facts are performed outside the control of the standard TMS.
2. Inconsistency may occur across derivation systems. For example, results from a knowledge-based system may be inconsistent with other KBSs, other information systems, or the real world.

Distributed truth maintenance systems were therefore introduced that not only address internal anomalies within systems (such as standard TMSs) but, in addition, anomalies among multiple systems.

Distributed systems, just like other truth maintenance systems can be assumption based [MAS89], or justification based [HUH91]. An example of the former is Huhns and Bridgeland's [HUH90] [HUH91] Distributed Truth-Maintenance System (DTMS). The distributed system works with the operators 'in' - for believed, and 'out' - for dis-believed. Beliefs are based on valid justifications. A system that is 'well founded' is one that has no believed data that lacks a valid justification, and no dis-believed data with a valid justification. Furthermore, the TMS cooperate in that they can provide justifications for beliefs held by other systems or agents. In other words, a system can have an external justification for its beliefs, which it can import from other sources.

If a knowledge-base is consistent, then there is not an element in that base that is believed and dis-believed at the same time. Any one system can share beliefs with others at any time and classify these beliefs as 'shared beliefs'. Systems can therefore have private beliefs and shared beliefs, which implement local and shared consistency. This means that all shared beliefs are consistent, and all local beliefs are consistent only within their base. A conflict about a shared belief in [HUH91] is detected if:

- One system A has a reason to believe the proposition, or it has an explicit statement that the proposition is believed (in); and
- Another system B has either no reason to believe the proposition, or an explicit statement that the proposition is not believed (out).

If all private beliefs were also shared beliefs then all information would be consistent and all conflicts due to new beliefs, or changes to existing beliefs could be detected. However, this would mean a very close, complete integration of the systems. In effect the integrated systems would operate logically as one system. If only some beliefs are shared then the integration is less tight. Conflicts are only detected when new beliefs are shared or already shared beliefs are changed. Inconsistencies between beliefs, that are not shared are not detected.

Distributed truth maintenance systems (DTMS) implement a particular notion of coherence such as standalone TMSs. These, in principle, can detect conflicts based on explicit and some implicit contradictions. Explicit conflicts are detected by observing that, for example, a proposition is believed and disbelieved at the same time. Implicit conflicts are detected if a notion of coherence is implemented in such a way that it can be investigated. Implicit conflicts can be analysed with respect to the notion of coherence that is maintained. For example, agents may have a set of integrity or

dependency rules that relate objects in the distributed system and form a notion of coherence.

In conclusion, Conflict detection in distributed maintenance systems falls short in detecting :

- Explicit or implicit conflicts that are not shared propositions / beliefs;
- All implicit conflicts of all notions of coherence in a complete fashion;
- Schema conflicts, or non-essential conflicts. All conflicts are assumed to be true data conflicts. In addition, maintenance systems just as standard possibilistic logic both have no grading (only the states 'believed' and 'dis-believed') and, therefore, lack any classification of different kinds of conflict (Section 3.2.1).

Conflict detection in DTMS is based on close integration of the information systems. As a concept for enterprise integration environments the DTMS would have to keep all integrated information permanently consistent within one source and across sources. In practice the detection is transformed into the initialisation phase of the distributed system. Information-sharing environments are, however, open and therefore inherently inconsistent (Section 2.3, 2.4 and 2.5). In other words, a distributed truth maintenance system can establish conflict free environments provided they are able to initially reach a consistent state. This, however, seems to be impossible in enterprise integration because the ultimate notion of coherence for all domains in all aspects is not known, so that sources in enterprise integration will always be inconsistent to some degree.

3.5.3.2 Belief Revision

> "Reason maintenance systems are Artificial Intelligence mechanisms for belief revision. They maintain consistent sets of beliefs in the light of new evidence" [[GAL92]p.224].

Any non-monotonic reason or truth maintenance system needs a mechanism to revise its current state in the light of new evidence. Belief revision, therefore, aims "to perform conflict resolution in case the set of current beliefs is contradictory" [[WIT93]p.2].

It could be concluded from this definition that belief revision has less to contribute to conflict detection than to conflict resolution. On the other hand, within the area of belief revision studies have elaborated, e.g., Galliers, Cawsey et al. [GAL90a] [GAL90b] [CAW92c], that analysing the nature of a conflict is important, not only for social conflicts, but also for problem-solving. This means that the detection (analysis) of a conflict already forms part of, or builds the foundation for, any constructive resolution.

Belief revision is potentially concerned with defeasable and not deductive reasoning, such that:

> "The problem of belief revision is that logical considerations alone do not tell you which beliefs to give up" [[GÄR92]p.1].

Deductive reasoning produces a single line of logically deduced, monotonic arguments.

> "Defeasible reasoning deals with incompatible lines of reasoning. More precisely, defeasable reasoning supports alternative and mutually exclusive conclusions drawn from incomplete information" [[BES91]p.38]. "Default reasoning is the drawing of plausible inferences from less-than-conclusive evidence in the absence of information to the contrary [and] .... Default reasoning is non-monotonic because, to use a term from philosophy, it is defeasable: Its conclusions are tentative, so given better information, they may be withdrawn" [[MOO85]p.77].

In principle, belief revision can be conducted by simple heuristics such as strictly rejecting new beliefs if these are inconsistent with the existing belief network. Furthermore, the problem could simply be delegated to a human decision maker, who would be invoked in the case of conflicting beliefs. However, examples of automatic revision procedures include:

- The definition of hierarchically related evidence spaces that form a consistent set of beliefs. These spaces have different grades of particularity and therefore

strength of belief. Stronger, more specific beliefs are preferred over weaker beliefs in the case of conflict (Konoligen [KON88]).

- Decision theory based on taking the utility of a belief and its consequences into account while bearing in mind its probability (Doyle [DOY92]).
- A consistent set of beliefs is updated with a new belief if it is more coherent to accept the belief than to disbelieve it in respect to the derivability of core beliefs (Galliers [GAL92]).
- Epistemic entrenchment describes the explanatory power and value of a belief. This is encoded by a complete pre-ordering of the beliefs. The reconstruction of a consistent set of beliefs is possible by ranking the conflicting beliefs according to their value and then abandoning the least important belief. Update operations based on entrenchment are either (a) removing a belief from the set so that the belief state becomes consistent again (Contraction), or (b) adding a belief to the set in a way - which may include removing older beliefs - that the belief set is consistent (Gärdenfors [GÄR88]).
- A typical minimalistic model allows for believing every proposition that is not explicitly challenged (but may not be justified either) [McC86]. In the case of conflicting beliefs a minimal change to the existing belief set is targeted. In addition, maximal coherence may be a subgoal that is characterised by the ability of a belief to also support existing beliefs [HAR86].

Consistency-based models, in principle, are not applicable to information agents because of a lack of exactly that 'consistent belief base' in the enterprise as has been discussed in the previous section on truth maintenance systems and Section 2.8.2. Here, it was concluded that information agents simply integrate information from their source while they have no control over the propositions (potential beliefs) in these systems.

In addition, every belief in a consistency-based systems is necessarily based on deductively closed lines of evidence that are ultimately based on epistemically basic beliefs.

> "The major criticism of foundation [consistency] based theories concerns the explicit representation of justifications for beliefs, and also then the propagation of disbelief" [[GAL92]p.230].

Apart from the availability of deductive justifications, the approach

> "involves excessive computational expense, [and] it conflicts with observed psychological behaviour [in the reasoning of humans]" [[DOY92]p.29].

Minimalistic models are applicable to enterprise integration in that information agents believe information they retrieve throughout the sharing environment unless conflicts are detected.

Along with the approach of accepting all beliefs unless they are challenged comes the question what to do when equally good reasons for believing and dis-believing a proposition exist? An agent could

(a) be sceptical, and not believe the proposition,

(b) be credulous, and select either one by random choice,

(c) neither believe nor disbelieve the proposition.

It can be argued that the first choice is adequate for reasoning about propositions [POL94]. In principle, belief has to convince the enquirer. A situation with equally strong evidence for and against believing a proposition will hardly make one overcome his doubts about the proposition [PRI67]. On the other hand, the second choice may be adequate for agents that take the epistemic belief as a basis for action. These agents may need to undertake some action rather than to do nothing.

The third case cannot occur in systems that have consistent belief sets. In other words, if the reasoner is omniscient then there is always a right and a wrong, or true and untrue. Other reasoners, including information agents, can be undecided. Representations of propositions that are neither believed nor disbelieved but existing are described, for example, in Levesque's [LEV84] explicit and implicit beliefs in a possible worlds framework; or Fagin and Halpern's model of awareness [FAG87]. Furthermore, Galliers supposes that agents may not only lack omniscience but there may also be conflicts in real world problem-solving that are not or should not be soluble [GAL90b]. As indicated in the opening paragraphs of this section, it is necessary to analyse conflicts rather than to apply heuristics that solve the immediate problem but are not Principle Rational (Section 2.6), or may (in complex problem-solving scenarios) lead to other problems at a later state.

Only a very few belief revision strategies have been mentioned. However, they describe the basic notions of belief revision so that the following **conclusions** can be drawn:

1. A typical belief revision systems is based on assessing more information to solve the problem of conflicting beliefs. For example, decision theory by Doyle [DOY92] takes the measures utility and probability of the beliefs into account; Galliers [GAL92] evaluates the relation of existing beliefs in respect to their

coherence to core beliefs; Epistemic entrenchment is based on further investigating the belief's expressiveness and value [GÄR88]. Conflict detection and resolution shares with belief revision systems that both are designed to resolve a conflict by 'assessing more information to resolve the conflict'.

2. The applicability of traditional belief revision systems (as described in this Section) to enterprise integration depends on the availability of the information required by these revision systems. For example, Konolige's hierarchical belief spaces [KON88] are difficult to obtain for information agents unless they are omniscient (Section 2.8.3). Briefly, belief revision systems are designed to help the encoding and management of an expert's knowledge in a knowledge-base that is related to a particular domain. However, integrating information in an enterprise is a problem that deals with information from various domains in which the information agent is no expert.

3. Also following from this lack of domain knowledge, information agents cannot apply consistency-based mechanism (The agents cannot ensure a consistent state of their integrated sources).

4. Furthermore, the character of non-monotonic logic is to allow for rules to be changed and every state may, therefore, be based on partial knowledge. However, "there is no representation of partial conflict; a default theory either has a consistent extension, or it does not" [[KRA93]p.192]. The previously described non-monotonic logics lack a grading of certainty / uncertainty. Propositions are either believed to be true or false. Extensions have been made to add grading which results in a possibilistic logic (Section 3.5.2) such as in the Graded Default Logic by Froidevaux and Mengin [FRO92].

5. Belief revision systems provide multiple, domain-specific resolution mechanism that may, in principle, participate in a general scheme for resolving conflicts. In other words, they may provide problem solving services to information agents (Section 2.7). For example, a knowledge-base that applies a revision scheme, e.g. decision theory [DOY92], may resolve conflicts that are specific to its domain provided the resolution strategy is rational to any potential client of the integration environment (Principle Rational).

Conceptually conflict resolution is a form of belief revision (i.e. an agent revises its belief on conflicting believes or evidence). However, existing approaches emphasis on domain expertise, and / or closed belief sets, and typically are not Principle but Application Rational (Section 2.6).

3.5.3.3 Uncertainty Management Based on Decision-Making Theories

Some domain-specific decision theories have been used in belief revision, for example, by Doyle [DOY92] in his classic utility and probability based model for uncertainty management. Game theory is a decision-making theory that has been used for agent coordination [EPH91]. Decision theory in a broader meaning includes most reasoning mechanisms (e.g. probabilistic reasoning by [PEA88]) or many agent coordination mechanisms described in Section 3.4. All these decision theoretic solutions to conflict resolution are application specific (Application Rational Section 2.6). In other words, decision theory aims to optimise the decision makers choice in an application, or domain-specific way. It does not aim to provide a general scheme for conflict detection or resolution.

A representative example of a decision-making model is Wong's [WON94] preference based decision-making, which is applied to cooperative knowledge-based systems. Essentially, the system is a cooperative problem-solving system for design similar to the planning systems described in Section 3.4.4. It is, however, based on a decision-making model consisting of the following:

- In an identification step the decision candidates are selected, ordered by their importance, and checked for inconsistencies among them.
- In a second processing step the preferences of the competing alternatives are deduced and combined.
- The negotiation step consists of the exchange of information for the purpose of finding an agreement by bargaining and compromise. The underlying theory for this interaction is social choice.

It was claimed above that most research approaches lack generality. In accordance with this it was found that:

> "For many problems, such as building design, there is no a priori logical or philosophical principle on which to base a resolution strategy - correct resolution depends on specific domain knowledge as well as the cooperative behaviour of the participants. The resolution of conflicts in the current scheme is largely driven by the participants, with preference profiles and collective choice as focal points" [[WON94]p.433].

Wong proposes a further investigation into human reasoning steps in order to find a general scheme (called Principle Rationality in Section 2.6).

The cognitive concepts of human decision-making provide a potentially useful but unexplored area. There may be solutions to a general framework that is applicable to conflict management but none has been brought to the attention of the field of Artificial Intelligence. Hence, **it may be concluded** that no general scheme for conflict detection and resolution exists in uncertainty management based on decision theory.

Another approach that takes human decision-making models into account is Argumentation. It is based on adopting human decision-making capabilities to some degree, which has lead to a very basic framework that will be described in the following section.

3.5.3.4 Uncertainty Management by Argumentation

Argumentation is a form of reasoning and decision-making in areas where evidence is not necessarily in form of quantitative parameters and the reasoning process needs to be flexible to adjust to changing decision procedures. In other words, it can be characterised as being "(a) able to reason about the structure of the decision itself, and (b) is not based solely on the evaluation of quantitative parameters" [[KRA93]p.208]. Argumentation is, therefore, based on a model of human reasoning implemented in a generic uncertain management mechanism.

A semi-quantitative approach has been presented by An, Bell and Hughes [AN 93] [AN 93b] (Section 3.5.2). An Argument in this research is

> "the relationships between two judgements expressing that one supports or refutes the other, ie, evidential supports" [[AN 93a]p.205].

Multiple ways to reason non-monotonically based on argumentation exist, e.g., by Pollock [POL92], Cohen *et al.* [COH85][COH87], or the approach by Fox, Krause, Clark *et al.* described below. The conceptual design of an argument is further specified along the following lines [FOX92b]:

> A piece of **data** (also called a candidate or a judgement in the previous paragraph) may exist, a **warrant** that leads from the data to a **claim**, which may be evaluated by a **qualifier**.

For example, the data 'Peter's address in the student registrar database is 'Pentonville Rd" and the warrant 'students live at the address they are registered with' may lead to the claim 'Peter lives in Pentonville Rd.'. This argument may be questioned in respect to the validity of the warrant or the claim. In the first case the warrant may, for example, be challenged by the 'community charge investigation' questioning the correctness of the student's ('Peter's') address. Therefore, a backing of the warrant may exist, e.g. the student address is used for communication with the student and therefore is proven to be correct. More often, however, a specific claim is challenged by a rebut leading to the claim being evaluated by a qualifier. For example, the qualifier 'possibly' may be attached to 'Peter lives at Pentonville Rd.'. This qualifier could be based on the rebut 'Peter would not have the cash to rent a place in Pentonville Rd.'.

The model developed by Fox, Krause, Clark *et al.* [CLA90a] [FOX92b] is a good example of argumentation and is based on the following framework:

1. "Dynamically propose decision candidates.

2. Dynamically generate evidence relevant to the decision candidates.
3. Dynamically identify relationships among decision candidates relating to conjunctivity and exclusivity.
4. Operate in the absence of statistical parameters, but
5. Incorporate these when available.
6. Be generic and not limited to any particular decision task" [[CLA90a]p.59].

This framework applies 'domain facts', everything an agent knows about the domain, and Task Specific Knowledge, which includes any problem-solving procedures and strategies. In addition, the decision-making knowledge generically applies the procedures and strategies to the argument in the decision stages:

1. **Proposal**: Decision candidates (or options) are proposed;
2. **Argumentation:** Generation of argumentation for and against the proposals;
3. **Annotation:** Classic logical considerations are employed to classify the evidence by annotating one out of nine states, e.g. possible, eliminated, supported, or definite.
4. **Relation:** Analysing the relations between the arguments (evidence) to identify which need to be evaluated together, which should be analysed separately, etc. This is based on matching the arguments according to assumptions and compatibility.
5. **Evaluation:** The evaluation stage is trying to produce (partial) orders, selections and other assessments.

In the Proposal step, the mechanism opens with collecting 'decision candidates'. These candidates are conceptually similar to propositions retrieved by an enterprise integration environment. For example, the candidates 'Yogi is a bear', 'Yogi is a comic actor', 'Yogi is a student at City University' may be candidates that have been gathered by the information agents from multiple information sources. These present decision candidates when they are alternative, conflicting results to a given query.

The Argumentation step gathers arguments for and against these candidates. These may include, 'Yogi is known to have played in a comic on Television (TV)' and 'An actor in a TV comic is a comic actor', 'Peter is a student at City University', 'Peter is fat and therefore called Yogi', etc.

In the Annotation step these arguments are associated with the candidates. Furthermore, they are evaluated logically. For example, 'Either Yogi is a bear or a student at City University because there are no bears at City University'. The arguments are classified according to nine certainty measures including possible, plausible, probable, certain, believed, likely, suspected, doubted, assumed (for full definitions [[CLA90b]p.135). This way the uncertainty is not quantified but described qualitatively. For example, lets assume that the predicate 'possible' is defined as: "no conditions that are necessary are violated" [[CLA90b]p.135] then the four candidates gathered in the second step are all possible. If the uncertainty factor 'probable' is defined as "P is possible and there is at least one item of evidence in favour of P" [[CLA90b]p.135] then 'Yogi is a comic actor' is probable because there is only one argument that supports the claim:

'Yogi is known to have played in cartoons on TV'

The Relation step then classifies the arguments, for example, in that 'Yogi is a bear' and 'Yogi is a comic actor' can both be possible at the same time. An alternative candidate to 'Yogi is a bear' is 'Yogi is a student at City University'. Furthermore, the argument 'Yogi is a comic actor' is based on the warrants

'Yogi is known to have played in comics on TV', which subsumes that

'An actor in a TV comic is a comic actor'.

Finally, the evaluation step applies general heuristics called decision knowledge. These may allow for ranking the alternatives into possibly partial orders, to combine evidence, or the heuristics may lead to a form of assessment of the conflict situation.

It is reported from applying this approach to real world problems [FOX92b][KRA93] that the aspects of uncertainty (e.g. possible, probable, assumed, known, etc.) have not been accepted by the applicants. Further experimental research is suggested.

Argumentation allows for a sensitivity analysis. This includes, for example, "determining to what extent the proposed decision would change as a result of changes to the particular context" [[KRA93]p.248]. Furthermore, it acknowledges uncertainties not only in numerical but also in qualitative form. Finally, the solution of the reasoning process results in an assessment and not in a strict 'believed', 'not believed' and possibly 'undecided' classification.

In conclusion, argumentation is based on a common notion of uncertainty measures (strength in believing conflicting arguments). In other words, the notion of uncertainty

needs to be commensurate if an information agent would apply this scheme. In enterprise integration, however, sources typically lack common notions of certainty. Furthermore, information agents typically are no domain experts that can evaluate the certainty estimates, e.g. in the form of strength in believing an argument. This makes existing structures of argumentation not as such applicable to conflict management in enterprise integration environments.

The described research, however, shares much with conflict management in integration environments:

- The focus of argumentation and conflict management is that evidence is compared or evaluated in order to make a decision (if rationally possible), and not necessarily that evidence is scaled within a given standard (e.g. a probability) [AN 93a].
- The framework for uncertainty reasoning may provide a basic structure for conflict detection, for example the first four steps of argumentation (please also see the following Section 3.5.4).

3.5.4 Conclusion on Uncertainty Management

Existing uncertainty management research provides multiple, mainly domain-specific conflict resolution strategies. However, no strategy exists that would be general enough to suit all domains. Information agents need such a general strategy, or scheme, to integrate information from all systems and all domains. Domain-specific resolution strategies should be included into a conflict resolution scheme.

Argumentation is found to be useful to explore conflicts in order to accommodate users' preferences. Augmenting conflicts is important for enterprise integration scenarios (Section 2.5), and for belief revision systems (Section 3.5.3.2). Combing the contributions from existing uncertainty management systems (e.g. [AN 93a]) and argumentation produces [CLA90a] [FOX92b] roughly the following conflict detection steps:

1. **Gathering of candidates** (or hypotheses, or objects of discourse);
2. **Gathering of evidence** concerning these candidate, which typically are uncertainty measures of the candidates but may also be other circumstantial information;
3. **Classification** of candidates to identify if and what kind of conflict exists.

This scheme draws mainly from Argumentation (Section 3.5.3.4). The steps (2) Argumentation, and (3) Annotation are joint into a 'gathering of the evidence' step to make the framework more general. In other words, the Argumentation step may not exist in environments that lack a generally accepted uncertainty measure. Furthermore, the Argumentation step (4) Relation is termed Classification because it will result in identifying which candidates are conflicting in what way. Further steps are needed to 'evaluated' the candidates applying some form of decision-making [AN 93a].

3.6 Conclusion on Related Research

Many forms of rational conflict resolution in enterprise integration (Section 3.3) and DAI (Section 3.4) have been identified. Furthermore, rational uncertainty management strategies have been mentioned that are applicable to conflict resolution. All are potentially useful to information agents provided:

- The agent can acquire the necessary circumstantial information about the conflict and the conflicting results;
- The strategies are rational to any user of the information (Principle Rational Section 2.6).

Information agents lack, however, a Principle Rational scheme to systematically co-ordinate the available, rational methods for detecting and resolving conflicts. No such scheme exists. Two important contributions from existing research may be applicable to such a scheme:

- A basic framework for conflict detection, which is basically drawn from Argumentation;
- A conflict resolution scheme should apply all available strategies. Distributed planning research suggests that domain-specific strategies are applied first, before less domain-dependent ones.

Later in this research it will have to be shown that these contributions are applicable to conflict management in enterprise integration.

The following chapter will be dedicated to identifying a general, rational scheme for conflict detection.

3.7 Chapter Summary

In the opening section conflicts were classified (e.g. into explicit and implicit conflicts) and a propositional calculus was introduced. A survey of existing research on conflict detection and resolution in Distributed Collaborative Environments for Enterprise Integration has been undertaken. It analyses the field, ranging from their predecessors distributed databases to enterprise integration environments. However, no rational, complete and automatic framework exists for detecting and resolving the different kinds of conflicts in information-sharing. No conflict detection or conflict resolution mechanism exists that is applicable to Distributed Collaborative Environments for Enterprise Integration.

Most systems in enterprise integration avoid conflicts by making assumptions about internal and inter-system consistency at the data level as well as at the schema level. It has been shown that these assumptions are ill founded. Based on this assumption previous research in enterprise integration avoids the detection of inconsistencies and thus:

- Applies heuristics to optimise the retrieval process, e.g., cost evaluation [PAN91a];
- Utilises domain-specific resolution strategies, e.g., credibility of a source of origin and the cost of denying beliefs which were already shared [BAR94a].

The fields of uncertainty management and distributed artificial intelligence (DAI) have been analysed for possible contributions to conflict detection and resolution, with the following results.

Integrating conflicting information in enterprise environments is a form of uncertainty management. The information agent has to assess the certainty of the conflicting results in order to assess which results to reject and which to believe. Uncertainty management, in principle, can be done by both quantitative and qualitative methods. These have been briefly analysed. They provide solutions for various, specific levels of domain-specific conflicts (problem structures) and levels of domain expertise (e.g. to relate certainty estimates). However, a partial framework for conflict detection has been found in the field of argumentation [FOX92b] [CLA90a] that consists of three stages :

1. Gathering of candidates;
2. Gathering of evidence;

3. Classification of the evidence as pro- and contra resulting in some description or even classification of the uncertainty of the candidates. In a conflict case the conflict may be classified.

This basic scheme needs to be tailored to enterprise integration environments and the information available to information agents (Section 2.7).

DAI has produced the partial global planning algorithm [DUR91a] [DUR87] as a basis for the exchange of information between information agents. However, the partial global planning algorithm is a task decomposition and assignment algorithm and therefore the evaluation of conflicts is avoided (heuristics for optimising the retrieval process are applied to alternative solutions). Mainstream conflict detection and resolution provides some conflict resolution strategies. However, these are only directed at solving non-essential conflicts, e.g. goal optimisation. These strategies could potentially be included by information agents as a means of resolving non-essential conflicts.

Distributed planning systems use multiple resolution strategies. Klein and Lu's [KLE89] research proposes a way to co-ordinate different resolution strategies with various conflicts:

> A given conflict is best solved by the most domain-specific resolution strategy available for this conflict.

This principle may be applicable to a conflict resolution scheme of information agents. For example, it needs to fit with the theory in Section 4.4.

Many approaches organise their problem-solving capabilities similar to that of Fox, Krause, Clark, *et al.* [FOX91][KRA93]:

- 'Generic decision knowledge' is used to co-ordinate the resolution procedure;
- Domain level procedures (strategies) and knowledge are applied to the conflict.

Such a framework, in principle, can be flexible enough to incorporate the different kinds of conflicts. In the mechanism designed and described in the next two chapters generic decision knowledge is implemented in the form of agent heuristics (Organisational Knowledge - Section 2.7).

4. Theoretical Framework for Conflict Detection and Conflict Resolution

4.1 Introduction

It is difficult to conceive of a non-trivial Distributed Collaborative Environment for Enterprise Integration (DCEEI) that does not adequately address the problem of conflicts. However, the previous sections have shown that existing work in this area has not addressed this problem adequately. A mechanism will be presented that is rational and complete in detecting and resolving conflicts in DCEEI.

An essential requirement for the design of such a mechanism is an underlying theoretical framework for conflict detection and resolution. Such a scheme is developed in Chapter 4 by first presenting the central, theoretical concept for both detection and resolution (Section 4.2). The basic concept is adopted from evidence law and forms a new approach to conflict detection and resolution in enterprise integration. A brief discussion of the concept of evidence follows in Section 4.3. On the basis of this theory, conflict detection is outlined in Section 4.4, and a resolution structure is developed in Section 4.5. Sections 4.6 and 4.7 draw conclusions and summarise the chapter. This theoretical framework will facilitate the design of a rational mechanism for conflict detection and resolution in enterprise integration environments (Chapter 5).

4.2 A Rational Scheme for Conflict Detection and Resolution

Section 2.6 demonstrated that conflict detection and resolution has to be logical and systematic but also rational to any potential client of the sharing environment (Principle Rational). Rational schemes to evaluate evidence can be found in such diverse areas as social science [FAC91], third party intervention, e.g., used in international relations [BER84], value judgement [TRU87], epistemology [PRI67] [POL74], operations research and management science [GOT92], and jurisprudence including evidence law [MUR86] [ELL87].

Evidence law has a long professional and academic history in categorising and stratifying evidence. It provides clearly defined terms that are widely accepted and present a rich domain for classifying evidence in enterprise integration environments. All the areas mentioned above with the exception of evidence law, emphasise conflict resolution rather than detection. Evidence law is particularly concerned with providing a pragmatic scheme for assessing conflicts. Other areas may be similarly well suited, but evidence law shares some key characteristics with detecting and resolving conflicts in enterprise integration, for example:

- Both are human built schemes to compensate for not having an omniscient, superficial decision maker that would always come to the right conclusion (e.g. it would identify the truth in a law case, or the true and the untrue results in enterprise integration);
- They deal with uncertain information, that typically is not quantified, e.g., by numeric probabilities;
- The approaches need to be applicable to all situations, problems and sources of evidence;
- The schemes have to be rational in order to produce generally accepted results.

First, a brief excursion into evidence law will provide a basic introduction into the area and some key definitions. Evidence in legal cases is qualified by satisfying the following four criteria:

1. Relevance to the fact in issue;
2. Admissibility;
3. Credibility of the source of the evidence;
4. Weight of the evidence.

Before examining these criteria the concept of evidence requires a brief explanation:

> "Evidence in legal cases may take the form of witness, oral or written, but it may also take the form of the production of things, including documents (real evidence)" [[ELL87]p.9]. Further, "evidence is everything that may persuade an inquirer of the existence or non-existence of some fact or situation which he is inquiring about" [[ELL87]p.3].

There are multiple ways to structure evidence, such as direct and indirect, primary and secondary, original and hearsay, etc.

> "Direct evidence consists of the witness who perceived the fact to be proved or the production of the document or thing which constitutes the fact to be proved. Circumstantial evidence of a fact is the testimony of a witness who perceived, not the fact to be proved, but another fact from which the existence or non-existence of the fact can be deducted, or the production of a document or thing from which the fact to be proved can be deducted" [[ELL87]p.11].

For example, a witness who has seen person A kill person B can give direct evidence to prove that 'A has killed B'. If the witness has only seen person A with a knife in his hand, he can only give indirect evidence because it is for the jury to decide whether it can be inferred that 'A has killed B'.

In legal cases, evidence has to undergo different stages of evaluation during the course of a case. **'Relevance'** and 'admissibility' are qualities that evidence must have in order to be considered by the jury [MUR86].

> "Evidence is relevant if it is logically probative or disprobative of some matter which requires proof" [[MUR86]p.7].

In other words, the evidence must make any fact more likely or less likely to be true. The rules regulating relevance are therefore rules of logic and common sense [ELL87]. For example, the evidence in the previous example, where a possible murderer has been seen with a knife in his hand, may be clearly relevant to the fact at issue. The fact that he has also got a bread cutting machine at home appears to be irrelevant. A clever lawyer may, however, take a fact that appears to be irrelevant and show with a chain of reasoning that it is indeed relevant.

Admissible evidence must be relevant but not all relevant evidence will necessarily be admissible. Thus, the admissibility of evidence is regulated for the British legal system by a large number of rules which have been developed over the past two hundred years. These may, however, be illogical at times [MUR86]. In order for a fact to be admissible

it needs to be relevant according to the rules of logic and common sense. But a relevant fact may not necessarily be admissible according to the legal rules. The best known example of admissibility is a rule that makes hearsay inadmissible for criminal cases. 'D has heard that A has killed B' is hearsay in a case where A is accused of killing B and would not be admissible in court.

When evidence is relevant and admissible then it is permitted to be considered by the jury. This is says nothing, however, about the **credibility** or weight of that evidence:

> "The weight of evidence is determined by the jury by considering its cogency or persuasiveness ..... Evidence is credible if it is worthy of belief, e.g., if it comes from a trustworthy source. It does not follow that it is entitled to much weight." [[ELL87]p.13].

For example, if a witness is a convict who is known to have perjured himself in the past, his testimony may not be very credible. A lack of credibility may make evidence less persuasive, and may make it less convincing then other, possibly contradicting, evidence from a more credible source.

> "The **weight** of evidence depends on the view taken by the jury of its truthfulness, reliability and cogency. Depending upon such a view, evidence may be of virtually no weight at all, or may rest on one of an infinite number of points on the upwards sliding scale, ending with evidence which is so weighty as almost to conclude the case in itself" [[MUR90]p.14].

After the relevance, admissibility and credibility of evidence has been established its weight will need to be assessed. In general this assessment goes along lines which "any reasonable tribunal would in any event take into account" [[MUR90]p.15]. In summary:

- The weight of legal evidence is typically not numerically rated, e.g. in a zero to one scale.
- The judgement, however, should be rational in that any other tribunal in the situation would come to the same solution. Furthermore, the verdict that presents a rational judgement can be accepted by any rational person.

The weight of evidence, in principle, can be used to determine a case. The standard of proof is the "**degree of persuasiveness** which a case must attain before a court may convict a defendant or grant relief" [[MUR86]p.6]. This standard of proof sets the most basic lines upon which the weight of evidence can be assessed. In civil cases the standard is based on a preponderance of probabilities. Here probabilities do not mean a statistical or mathematical assessment; in order to prove a fact it is necessary to show

that the fact is more likely then not to be true. In criminal cases the prosecution must prove the defendants guilt beyond reasonable doubt. The court is beyond reasonable doubt when it is sure of the guilt of the defendant. This means that it need not reach certainty but it must carry a high degree of probability [MUR86].

4.3 Evidence in Law and Information Integration

The question 'How does evidence law relate to conflict detection and resolution?' will be answered in two steps. First, it needs to be established what 'evidence' represents in enterprise integration (this section). It will then be demonstrated how the scheme for conflict management from evidence law applies to conflict detection (Section 4.4) and conflict resolution (Section 4.5).

Evidence or reasons to believe can be defined as:

- "Data used to support or refute choices from alternative conclusions" [[BEL93]p.967].
- "When one belief justifies another, then the former is said to be a reason for the latter, one belief that justifies another" [[POL70]p.63].
- Evidence is everything that may persuade an inquirer of the existence or non-existence of some fact or situation which he is inquiring about [[ELL87]p.3].

In any general reasoning situation, one has, ideally, evidence that clearly indicates if a proposition is true or not true. For example, the evidence 'I saw A kill B' may lead me to know that the proposition 'A has killed B' is true. In enterprise integration environments, information is not 'directly witnessed by the information agent' but retrieved from information sources. Typically, the retrieved information is not epistemically evaluated in any way by the providing sources (e.g. a database). Hence, the information agent has to:

- Gather the evidence, however little, that may be provided by the source of origin (e.g. statistical proof of the information systems reliability in past retrievals);
- Find ways to investigate the case and provide evidence for the case itself.

Ideally, evidence in enterprise integration would be numerical such as certainty estimates. However, for a given candidate (result) only augmentative evidence to prove its correctness may be available. Section 2.7 listed the information that is available to information agents. For example, it includes comments about the reliability of information systems. This evidence is similar to indirect evidence in legal cases in that it is for the information agent to decide:

(a) How this information may refute or warrant a candidate; and

(b) If this evidence is convincing in this particular conflict case.

Later sections will describe a representation for defining evidence (Section 5.2.3 and 5.7) and how agents can apply domain-specific strategies or services for evaluating this

evidence (e.g. Section 5.8). An example scenario and decision procedure for believing propositions in enterprise integration could be:

> An agent can send a query to its database interface, which will respond to this query. The interface selects information from the database. This result (called a candidate) may be supported, e.g., by a time stamp that indicates that the information is not old. However, a typical database does not give any statement on the information's quality or any justification for it (probative evidence for the result from the database). In any case, the agent could conclude that the retrieved result is true simply based on the result from the database interface. This result may also be called a hypothesis, or a propositional candidate $O_i.R_k$ (object O_i has the attributes R_1 to R_k).

In other words, two kinds of evidence exist: Candidates; and The evidence of the candidates. These correspond to chains of evidence in evidence law where one hypothesis is justified by another; or by epistemic justification, e.g. in Pollock [POL70].

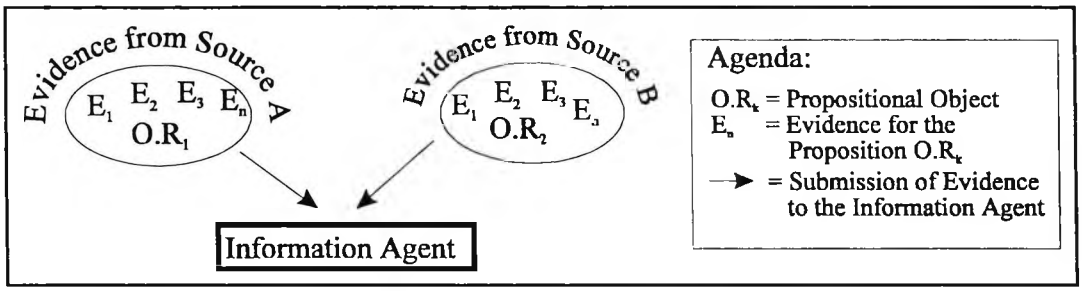

Figure 6: Evidence in Integration Environments

In other words, one hypothesis (or candidate $O.R_k$) has multiple pieces of evidence (E_1, E_2, to E_m) supporting or refuting it. The evidence forms the basis for estimating the uncertainty of the candidate. However, in enterprise integration the conflict case occurs when multiple candidates ($O.R_1$, $O.R_2$, ... $O.R_k$) are explicitly or implicitly conflicting. These hypotheses and their evidence together represent evidence for the information agent when it integrates results from multiple sources to produce one joint result.

In summary, evidence law is concerned with hypotheses and evidence for these hypotheses that may themselves incorporate further justifications. Evidence in enterprise integration consists of candidates and evidence attached to these candidates.

4.4 Relevance and Admissibility in Conflict Detection

4.4.1 Relevance and Admissibility

Information agents in enterprise integration make a typically epistemic audit (Section 2.8) that is very similar to the taking of evidence in legal cases. Agents request information from one or all relevant sources concerning the fact at issue. For example, the address of an employee, e.g. 'Yogi the bear', may be required. Information retrieval identifies which systems can provide this information, and retrieves information from them. The request may be sent to multiple sources, if these can all provide the information but the responses may not all be identical, e.g. any result may be incorrect. The detection starts with receiving the results, also called candidates, from the sources (Gathering of Candidates). These candidates may also include some form of evidence for believing or disbelieving the result (Gathering of Evidence).

In Section 3.5.4 the following concept has been identified for this initial information gathering phase in conflict detection :

1. Gathering of candidates (or hypothesis, or objects of discourse);
2. Gathering of evidence concerning these candidates;
3. Classification of candidates to identify if and what kind of conflict exists .

Steps one and two directly follow the retrieval process in the sharing environment. They initially stratify the retrieved information. The final step results in specifying that (1) no conflict exists or (2) which of the explicit and implicit conflicts defined in Section 3.2.1 exists.

Ideally, the candidates are always syntactically and semantically represented correctly and relevant to the fact at issue. However, in enterprise integration, inconsistencies and incompleteness exists when autonomous, heterogeneous sources are integrated (Sections 2.3, 2.4 for autonomy and heterogeneity, Section 2.5 on task assignment in enterprise integration, and Section 3.3 for enterprise integration environments). For example, the candidates may be misinterpreted, or candidates are proposed even if they are not relevant to the fact at issue. The inconsistencies in enterprise integration systems could have theoretically been overcome by constructing tight, consistent distributed systems. Examples include some homogeneous distributed databases (Section 2.2), integration environments with consistent, master models (Section 3.3.1.1) or unified schemata (Section 3.3.1.2), and distributed truth maintenance systems (3.5.3.1). In other words, these consistent distributed information systems shift the conflict management into the

modelling phase. A closer investigation of enterprise modelling in the following paragraphs will show the principle steps of this management in enterprise modelling.

The process of initially integrating information sources is called enterprise modelling.

> "Enterprise modelling is the cooperate activity that produces models of the information resource, information flows and business operations that occur in an enterprise" [[HUH92]p.38].

An information system in enterprise integration is typically modelled in terms of objects, e.g. in the MKS object-oriented methodology [PAN91a]. The way an 'object' is used, however, originates from distributed databases:

> "The basic modelling element is an object, which corresponds to some (real world) entity or concept (e.g. the person Jane Smith or the number 5). Objects are divided into two categories: Descriptor Objects and Abstract Objects...." [[HEI85]p.258].

Schemata are composed of Descriptor Objects, which describe the Abstract objects stored in, or available from, the information source. Abstract Objects are the data itself that is available from a source. The descriptors are an utility or 'meta information' that shows in a condensed form what information is available from the source. The term schema in this research can mean any number of possibly conceptually related Descriptor Objects.

Schema integration is concerned with integrating different schemata and overcoming potential inconstancies between them. This is a **syntactic** task of matching and translating abstract objects. However, it requires that no inconsistencies exist on the data-schema or the data level. Therefore:

> "The overall task of schema integration can be divided into two main parts, schema analysis followed by schema synthesis. This begins by the process of establishing correspondences and detecting possible conflicts between structural components or semantic contents of local schemes in the so-called pre-integration phase, where these conflicts are resolved by conforming the different schemes so that they can be compared, merged and restructured, before being integrated to form a global schema" [[QUT92]p.3].

Correspondences across information systems are identified on the level of structural components, such as schema items (concept correspondence), and on the data level dealing with data objects (object correspondence). For example, "consolidation begins

by identifying the identical constructs (i.e. those having the same primary key) within and across the [uniformly presented] submodels [of each system]. Each such set of identical constructs is merged into a single construct. The second step in consolidating is representing the notion of foreign keys (referential integrity constraints) arising between constructs in different submodels [or sources]" [[HSU91]p.609].

Schema synthesis is achieved by either merging multiple schemata into one Unified Model (Section 3.3.1.2), merging it with one central Master Model (Section 3.3.1.1), or to merge, possibly partial, schemata directly (e.g. Federated Models 3.3.2).

Enterprise integration environments typically lack a pre-integration or consolidation phase that provides complete internal, and inter source consistency (opening Sections 2.3, 2.4 and 2.5). Hence, the task of detecting these inconsistencies is shifted into the conflict detection phase. In other words, the detection phase needs to revise the above schema integration phase to detect if:

The conflict is only a schema integration problem: or

A data conflict is at hand (incorrect, obsolete or incomplete data).

Conflict Detection should therefore include :

1. Detection of syntactic integration conflicts.
2. Detection of semantic mismatch problems between the candidates:
   - 2.1 Lack of object correspondence;
   - 2.2 No concept correspondence (semantic misinterpretation).

The following will now demonstrate all the proposed steps for conflict detection. These are embedded into a conflict detection scheme structured by the phases Relevance and Admissibility (evidence law). The detection phase opens with gathering the candidates (results) and any evidence that may exists to justify or support them. In the gathering phase a request against the integration environment results in:

1. No candidates;
2. One candidate or identical candidates from multiple sources;
3. Multiple, different candidates.

Information agents apply a minimalistic reasoning (Section 3.5.3.2) in that they accept the first case as 'no result', and the second as a conclusive valid result. In other words, the result is not challenged, the change is minimised, and therefore accepted in the face

of no contradictory evidence. In principle, such minimalistic reasoning is adequate and typical of enterprise integration environments that do not have a consistent, e.g. deductive, belief network but rather an open belief base (Section 2.8.2).

However, in the case of multiple, non-identical candidates, further evaluation is required to identify if a true conflict exists. Determining the **relevance** of multiple results starts with determining if the candidates are syntactically correct (Syntactic Conflict Detection). Syntactic correctness includes, for example, the translation and communication of results (Syntactic correctness is a presupposition for a result to become relevant to the request).

Semantic Conflict Detection is concerned with detecting the 'correspondences' of the possibly conflicting items. In enterprise integration environments this procedure includes evaluating the object correspondence of multiple results or candidates. For example, results from multiple sources can only conflict explicitly or implicitly on the same attribute class if they are concerned with the same thing (Conflict Classification Section 3.2.1). In other words, two candidates cannot conflict on a candidate's property, e.g. the colour of the concept 'car', unless they both

(a) refer to the same car (Object Correspondence) and

(b) understand the same thing under the concepts they uses, e.g. 'car', 'colour', 'red', 'green' (Concept Correspondence).

Another aspect of concept correspondence is that the results need to be presented in the form that is semantically intended by the source of origin. For example, expert knowledge may need to be semantically explained so that the information agent can ensure that a candidate is inconsistent with other results. Furthermore, concepts need to be translated or mapped between sources. For example, the concept automobile in one source may be equivalent to the concept car in another.

The final step for detecting a conflict is to check the **admissibility** of either evidence in legal cases, or results in enterprise integration. In enterprise integration the equivalent of legal 'rules of admissibility' are any guidelines about when to consider a candidate. Business rules (Section 2.7 Organisational Knowledge), for example, may entail that information from a particular source is rejected. Such a rejection on principle may, for example, be justified if a source has had a disk crash.

4.4.2 Overview Conflict Detection

In summary, the following steps build a framework for conflict detection :

1.	Gathering of Candidates	
2.	Gathering of Evidence	Gathering Phase
3.	Classification	
4.	Syntactic Conflict Detection	Relevance or
5.	Semantic Conflict Detection	Syntactic and Semantic
	5.1 Object Correspondence	Phase
	5.2 Concept Correspondence	
6.	Admissibility	Admissibility Phase

Table 5: Conflict Detection Framework

This framework enables the agent to detect a data conflict which requires conflict resolution (Section 3.2.1). This scheme will now be used for the design of a mechanism for information agents in enterprise integration. This mechanism for conflict detection will be described in Sections 5.2, 5.3, 5.4 and 5.5.

4.5 Credibility And Weight of Evidence In Conflict Resolution

4.5.1 Credibility and Weight

In legal cases, relevant and admissible evidence is subject to investigating the **credibility** of the source of origin. The credibility of an information source is described by its reliability. For example, a database may be very reliable, which means that the source is very credible. However, reliability may be related to:

- A whole source, e.g. a database is very reliable [BAR94a];
- A specific result from a source, e.g. a statistical package may have a level of confidence on its results.
- A specific context, where reliability expresses the adequacy of the source to provide particular information [PAN91a].
- An information agent itself may be able to judge on the reliability of a source, or results from that source based on its past experience with information from this system.

This initial resolution will, if possible, value the candidates' reliability or credibility. This is followed by assessing the **weight** of the evidence for the conflicting candidates. In other words, there are multiple candidates with their evidence, including any reliability estimations. In order for the information agents to judge the conflicting candidates, the weight of evidence supporting and refuting the candidates is analysed. The weight of the evidence, including the reliability of the evidence, may provide a basis for the information agent to make a judgement. Reliability may play an important role in this judgement but, in evidence law or enterprise integration, a rational judgement would be based on all the available evidence. In other words, the reliability of evidence is only one aspect used to determine the weight of evidence.

The next step is, however, to assess the weight of evidence and then to evaluate or judge on it. Some research in conflict resolution, such as Klein's [KLE91] concept of general and specific resolution knowledge (Section 3.4.4), provides a partial answer. Multiple strategies to resolve a conflict are applied based on the following premise:

> For a given conflict the most domain-specific resolution strategy is the most accurate one.

This concept will be applied to conflict resolution in enterprise integration. In terms of conflict management this means a conflict is best evaluated on the lowest level of the integration. Each level of integration has its own variety of conflict evaluation or

resolution strategies. Hence, three levels of evaluation and resolution exist in enterprise integration:

- **Local** evaluation is implemented within an information source or among a group (community) of sources and is called **'Domain-Specific Problem-Solving'**;

- **Intermediate** evaluation is undertaken by the information agent but it applies scientific resolution heuristics to domain and problem specific information (**'Scientific, Domain-Specific Resolution Heuristics'**);

- **Global** evaluation is **'Domain-Independent Evaluation'** such as domain-independent evaluation of the reliability of results or their sources of origin.

The minimum requirement for all these resolution strategies is that they all must be **rational**. In other words, all strategies have to be Principle Rational to all potential clients of the integration environment. This is defined in the opening Section 2.6 to be the mastering goal for the design of a conflict detection and resolution mechanism. Hence, the strategies that are embodied in this mechanism also have to be rational in this sense.

4.5.2 Domain-Specific Problem-Solving

The local level of enterprise integration environments is the level of independent information sources such as a database, an expert system, or a knowledge-based system (Section 2.3). Furthermore, individual systems may be clustered to Groups that cooperate to jointly solve conflicts (Section 2.7 Resource Knowledge). For example, a Group of problem solvers may cooperate in solving complex planning and scheduling problems (Section 3.4.4). Any resolution that can be provided by one source or among a group of cooperating sources is domain-dependent. The resolution by domain experts (local problem solvers) is typically, and has to be explicitly identified as Principle Rational. In other words, where a domain expert, such as an expert system or a human expert, can resolve the conflict then this resolution is typically rational to any client of the integration environment.

Groups of cooperating sources form a distributed problem-solving community. The strategies they use are tailored to the use of one specific system or a specific group of systems. Even if these resolution strategies are applicable to other sources, their

applicability needs to be defined. For example, a group of agents that cooperate in a game theoretic cooperation mechanism [EPH91] are a closed group. It is not possible for any system in the enterprise to join in and participate in the problem-solving without being integrated into that community. Integration of a new source may include knowledge of the cooperation protocol, the languages, the rules and goals of problem-solving, etc.(Section 3.4.5.).

It follows, that while information agents can employ these local problem-solving communities they cannot participate directly in the domain-specific problem-solving. For example, Environmental Information (Section 2.7) about the problem-solving strategies of sources enables the information agent to determine that conflicting candidates are from sources that cooperate in the same problem-solving community. It can, in such a case return the results to that community in order to have local problem-solving procedures resolve the conflict adequately.

In Section 3.2.1 conflicts were classified as essential conflicts or non-essential conflicts. In the case of an essential conflict, a deal between mutually acceptable alternatives is not possible, but one result is correct and the other is not correct. It has been outlined in the introduction (Section 3.2.1) that genuine conflicts between candidates may exist in which all candidates are correct and the conflict is one of priorities. Such non-essential conflicts may be based on problems of, for example, optimality or goals. Domain-specific resolution strategies typically resolve such non essential, e.g. goal and optimality, conflicts. Distributed problem-solving, for example, focuses on non-essential conflicts. However, there may be strategies, e.g. outlined by Galliers [GAL90a], that also deal with essential conflicts. Further examples of domain-specific resolution strategies are described in Section 3.4.

In many conflict cases no local problem-solving is possible. No domain-specific resolution strategies are available that can resolve the conflict, that is typically between mutually acceptable results, e.g. non-essential goal or optimality conflicts. More general resolution strategies are described in the next section.

4.5.3 Scientific, Domain-Specific Resolution Heuristics

An intermediate level for evaluating evidence is called Scientific, Domain-Specific Resolution Heuristics. In other words, the conflict resolution is concerned with establishing the weight of evidence and their candidates (Section 5.7) to make a judgement intended to resolve the conflict.

> "Judgements are verbally embodied in statements, sentences that assert that something is or is not the case and can therefore be true or false." [[TRU87]p.7].

The information agent applies any 'sentences that assert something'. This includes any 'scientific' sentences (rules and heuristics), which can be mathematical, logical, empirical or metaphysical judgement [TRU87]. The word 'scientific' indicates the agent's aim to find 'scientific proof' for any conflicting candidates. Further, the resolution strategies are domain-specific because they apply these scientific heuristics to the candidates, their evidence and environmental information relevant to the candidates. The following of this section will briefly describe scientific heuristics that can be applied to domain-specific information (also called 'facts' in many uncertainty reasoning approaches).

An example of a mathematical judgement is the equation '2+3=5'. A logical judgement is the inference 'If A is bigger than B and B is bigger than C then it follows that A is also bigger than C'. These equations are true based on reason. It is known that mathematical and logical reasoning produces these results. In Section 3.5.2 some quantitative methods to uncertainty management have been described, e.g. Bayesian probability, or possibilistic logic and it has been concluded that:

- This reasoning appears rational to any information source.
- However, the methods are limited in the kinds of conflicts they can solve because they are bound to the quantitative nature of the evidence and the structure of the problem.

Qualitative methods, e.g. non-monotonic reasoning and belief revision, or decision-making theories, provide methods for logical judgement (Section 3.5.3). However, much research on qualitative methods is either not Principle Rational (e.g. decision theories based on different notions of optimality), or based on very detailed domain knowledge (e.g. non-monotonic reasoning in truth maintenance systems). Qualitative or quantitative methods are applicable as scientific resolution strategies provided they are:

- Rational to any information source:
- Can be supplied with the necessary quantitative and qualitative information;

- The structure of the conflict suits the methods.

The system administrator defines what strategies are Principle Rational by specifying the scientific resolution strategies in the information agent's knowledge.

In contrast, empirical judgement is based on observation. For example, 'Yogi is fat' may be based on the empirical observation by a person X. However, this observation may be biased because the person X may be short-sighted or weighing only 10 stone he may define anybody over that weight as 'fat'. An empirical judgement that is rational needs to be based on universal assent to produce a generally accepted result [TRU87]. For example, the judgement 'Yogi weights 20 Stone' is based on weighing Yogi on a scale, and is, therefore, empirical and universally accepted.

The previous two kinds of judgement are commonly considered as objective [TRU87]. Metaphysical and empirical judgements are similar in that it must be based on universal assent in order to be objective and rational. However, metaphysical beliefs can never be true or false. They are assertions about the world that cannot be proven. An example of such a metaphysical judgement is: "Space is indefinitely extended" [[TRU87]p.13].

Ideally, an information agent would know all scientific knowledge to reach all scientific judgements. Such an agent would be an 'all domain expert' which has all domain-independent and domain-dependent scientific knowledge. For example, this expert knowledge could be incorporated by an expert system similar to a hypothetical omniscient human. An example of this approach is CYCESS [GUH94] based on the common knowledge-base CYC [LEN90], which is further described in Section 5.9. Typically, however, information agents may be able to provide some scientific judgement and are not omniscient. Hence, they may solve some conflicts with scientific, domain-specific heuristics.

4.5.4 Domain-Independent Evaluation

The third way to evaluate evidence is on a global level, and domain-independent. For this evaluation the following information is available:

- Candidates;
- Any evidence of these candidates and possibly any certainty assessment (reliability) of the candidates or their sources of origin.

However, this evidence could not be evaluated by domain-specific solutions as described previously (local and intermediate levels of resolution).

Judgement can be scientific such as mathematical-logical, empirical and metaphysical, or otherwise it is value judgement [TRU87]. The evidential, scientific judgement of the previous sections has not led to conflict resolution. The information agent is left with the last possible approach to resolve the conflict by a form of value judgement [TRU87]. Reliability is typical of this kind of judgement [BAR94a], which also has general assent and is ,hence, rational.

Reliability, therefore, is a question of standard. Thus, the reliability of a source may be evaluated in respect of certain aspects such as consistency over time, past performance, expected future reliability, etc. Judgement of conflicting candidates based on such a standard of reliability is generally called value judgement:

> "'Value' had always suggested some comparative process of assessing or measuring" [[ENC92]p.1269]. Examples of values include aesthetic values (harmonic), values of usefulness (useful, advantageous), religious (holy, pious), values of justice (just, usurious), values of personality (reliable, lazy), etc. [ENC92]. "The distinctive feature of these judgements is that they are evaluations against a standard that is acknowledged by the person [agent] making the judgement; if the judgement is to be taken as an objective judgement, the standard must command universal assent (as do objective logical and empirical judgements) or it must command substantial agreement (as do some objective metaphysical judgements)" [[TRU87]p.16]. In summary, judgement has to be based on a normative concept (thus also called normative judgement). An example of such a concept can be reliability.

The problem of judging reliability is the lack of a common standard. Multiple diverse standards of reliability exist that may be incommensurate or even conflicting. For

example, the estimate 'certain' from source A may not be comparable with the estimate '70 per cent confidence' from source B.

Criteria for comparing some certainty estimates may be available in enterprise integration environments. These are defined as part of an agent's Decision-Making Knowledge or Business Rules (Section 2.7 Organisational Knowledge). An example of a possible criterion may be: 'A result that is very unreliable can be neglected'. This heuristic may single out one candidate that can be believed (and the conflict is resolved). However, such heuristics often partially rank the results: Very unreliable results are eliminated while others, which may not be rankable, undergo further investigation.

As stated in Section 4.2 a standard of proof exists in legal cases to determine when a case is sufficiently persuasive so that a judgement can be made. The concepts of 'preponderance' and 'proof beyond reasonable doubt' have been mentioned. In enterprise integration conflicting candidates must persuade a rational information agent sufficiently. Organisational Knowledge incorporates the standard of proof. For example, it includes decision criteria on when candidates can be ranked and judgement is possible, or when a case is undetermined and no (or only partial) ranking is possible.

Conflict resolution based on reliability is grounded in mainstream research for solving value conflicts as, e.g., described in [FAC91]. It is a general scheme and based mainly on common sense. Similar approaches have been applied to qualitative uncertainty management systems such as Argumentation [FOX91] [FOX92b]. The emphasis of this, and many approaches in social science, lies on the analysis of a conflict. This emphasis also makes the following synthesis of existing research particularly suitable for conflict resolution in enterprise integration. In principle, the approach is based on the three steps of:

1. Ordering the candidates in respect to reliability (Ranking);
2. Finding New Alternatives (New Alternatives);
3. Negotiation of Compromise (Negotiation).

Ranking is the simplest way to the question 'Which candidate should the agent believe?' The pragmatic answer is: 'The most reliable one'. The easiest way to determine the 'most reliable' candidate is to rank the alternatives according to the reliability attached to them.

Ranking has been used in may conflict resolution mechanisms in all areas from enterprise integration to uncertainty management (as has been described in the related research

section). Typically, this includes ranking preferences, goals and priorities, e.g. Werkmann [WER91] or Adler *et al.* [ADL89]. Uncertainty management systems focus on ranking the expected reliability, e.g. Argumentation [FOX92b] (Section 3.5.3.4). In the case of integrating information, the candidates are typically ranked according to their reliability, e.g. [BAR94a]. The outcome of such a ranking scheme may be that one result may be more reliable than another and, therefore, a judgement is possible. Please note the fact that candidates have rankable reliabilities is not sufficient to make a judgement, the agent also requires heuristics to judge based on this ranking (Decision-Making Knowledge or Business Rules).

However, the certainty estimates may be incommensurate, or no criteria may be found that enables a judgement to be made based on ranking. In these cases, the agent would ideally develop new alternatives [SYC89]. This step enforces flexibility and creativity. In other words, new alternatives can be found by further investigation of the reliability conflict. The agent may find that different certainties are directed at different aspects, circumstances or assumptions. This includes **Alternative Ranking Heuristics** and **alternative certainty estimations**.

In the previous step Ranking Heuristics are used to define the order of certainty estimates from diverse sources. These heuristics are known to be applicable to the conflicting candidates and their certainty estimates. For example,

> a certainty estimate 'possible' in source A is known to be identical to the estimate 'possible' in another source B.

In situations when such heuristics are not available, the information agent would try to propose a ranking scheme or heuristics. In other words, the agent may have Alternative Ranking Heuristics but it may not know if these are applicable to the conflicting candidates. Thus, these heuristics can only suggest a proposed an alternative ranking.

The agent may be able to investigate the reliability estimations further, and come to a more diverse description of the certainty estimates (alternative certainty estimates). This may include descriptions of the circumstances or assumptions under which the certainty estimates are made. New Alternatives are developed based on new, or extended certainty estimates. For example, two conflicting candidate may both be 'reliable'. However, the estimates may be extended when it is derived from the agent's Resource Knowledge that one source is an expert in one field and the other is an expert in another. In other words, each result is more reliable in respect to its expertise. However, this new situation proposes a compromise (i.e. a compromise is the partial satisfaction of the alternatives). Nevertheless, a decision has to be made if this compromise is to be acceptable.

Hence, the Negotiation stage presents to the client of the integration environment the new alternatives which are based on new heuristics or new estimates. The information agent presents a compromise to the decision maker, or possibly to the application, that has requested the information. No actual negotiation between the conflicting parties is possible but the information agent mediates the resolution result to the client of the integration environment. The information agent is a manager agent (Section 2.5) architecturally similar to Persuadors [SYC89] (Section 3.4.3), that seeks a rational result.

However, compromise is limited to cases where new alternatives have been found. Otherwise the conflict case cannot be solved rationally in enterprise integration by information agents. This may be due to:

- The lack of domain level knowledge which would have enabled the agent to resolve the conflict.
- Different business strategies and goals produce alternatives with different notions of optimality (e.g. business strategies such as 'save money' and 'improve working conditions' may conflict).

In the first case there is no Principle Rational solution to resolve the conflict based on the given level of Agent Knowledge. In the second, however, a solution for the conflict may be possible in the scope of the client, e.g. a decision maker or an application. Section 2.6 explained that conflict detection and resolution in enterprise integration lies between information retrieval and information use in applications or by decision makers. Problem specific views may allow for the resolution of such conflicts only for the purpose, and under the rationality assumptions, of this specific application.

4.5.5 Overview

In summary a framework for conflict resolution has been outlined. It includes the following steps:

1.	Evaluating the Credibility of the Results and Their Sources of Origin;
2.	Weighting the Evidence to Make a Judgement:
	A. Domain-Specific Problem-Solving (Local Resolution)
	B. Scientific, Domain-Specific Resolution Heuristics (Intermediate Resolution)
	C. Domain-Independent Evaluation - Reliability (Global Resolution)
	- Ranking
	- New Alternatives
	- Compromise

Table 6: Conflict Resolution Framework

This rational scheme for conflict resolution will be used in Sections 5.7, 5.8, 5.9 and 5.10 to design a mechanism for information agents. in enterprise integration environments.

4.6 Conclusion

The previous sections have outlined a theoretical framework for conflict detection and resolution in enterprise integration environments. An additional construct is, however, needed in order to design a conflict detection and resolution mechanism based on this framework:

- The Gathering phase requires a formal specification for object candidates and their evidence, which is sufficiently expressive that the kind of conflict may be identified and the object correspondence can be investigated (Semantic Conflict Detection).

The evaluation of the theoretical underpinning outlined in this chapter will have to prove that:

- Conflict detection is complete in respect to any known conflicts among multiple results.
- The resolution which is most domain-specific has the greatest accuracy for conflicts in enterprise integration.
- All resolution procedures available to information agents in enterprise integration can potentially be incorporated by the resolution scheme.
- The Principle Rational resolution mechanism is functional in an enterprise integration environment.

4.7 Chapter Summary

The theoretical basis for conflict detection and resolution in enterprise integration has been outlined by adopting a framework from evidence law. It has been shown how evidence law can be applied to conflict detection and resolution so that the design of a detailed multi-step framework is possible. The framework includes the following phases:

Conflict Detection:

1. Gathering of candidates and their evidence (including a possible classification of the conflict);
2. Syntactic and semantic relevance of the candidates;
3. Admissibility of candidates;

Conflict Resolution:

4. Evaluating the credibility of the results and their source of origin;
5. Weighting the evidence to make a judgement.

The Gathering phase includes the uniform assessment of the conflicting results and their evidence and the classification in case of a potential conflict. The semantic and syntactic conflict detection phase draws on a concept to revise the enterprise integration modelling. This ensures that a possible conflict is not merely a syntactic or semantic mismatch. The final step in detecting a conflict is to ensure that the results are admissible, in that no rules set out by the system administrators or integrators generally refute them.

Conflict resolution opens with an investigation of the credibility of the sources of origin and the results derived from these sources. The resolution of the conflict proceeds by systematically applying existing resolution mechanisms and the agent's own heuristics to the conflict. If no domain-specific resolution is possible (Domain-Specific Problem-Solving) then general strategies must be applied. In other words, the information agent may apply general resolution heuristics that are rational to any potential client of the integration environment (Scientific, Domain-Specific Resolution). Finally, a judgement may be based on domain-independent resolution strategies such as the reliability (credibility) of the conflicting results (Domain-Independent Evaluation). A mechanism to judge the reliability of conflicting results and their sources is proposed.

The next chapter will apply this theoretical framework to design a mechanism for conflict detection and resolution in enterprise integration environments.

5. A Mechanism for Detection and Resolution In Enterprise Integration Environments

5.1 Introduction and Design Methodology

5.1.1 Introduction

The previous section introduced a framework for conflict detection and resolution. This framework will now be systematically applied to design a conflict detection and resolution mechanism for information agents in enterprise integration. The methodology for this design approach is introduced in the following section.

Conflict detection requires an adequate formal representation, and this, together with an outline of the Gathering phase will be covered in Section 5.2. Sections 5.3, 5.4 and 5.5 will then describe the Syntactic, Semantic and Admissibility phases of conflict detection. Conflict detection is summarised in Section 5.6.

Conflict Resolution begins with evaluating the credibility of the conflicting candidates (Section 5.7). The resolution steps Domain-Specific Problem-Solving and Scientific, Domain-Specific Heuristics are outlined in Sections 5.8 and 5.9. Domain-Independent Evaluation is the final phase in conflict resolution: Its description is based on the evaluation of the candidates' reliability. A conflict resolution summary is provided in Section 5.10.

Chapter 5. closes with an overview in the form of a decision tree (Section 5.11). This graph provides an Implementation Concept for the following Evaluation (Chapter 6.). Concluding remarks and a chapter summary are covered in Sections 5.13 and 5.14.

5.1.2 Design Methodology

The basic methodology is one of synthesising existing research in the wider field of enterprise integration, uncertainty management and distributed artificial intelligence to function within the detection and resolution framework outlined in Chapter 4.

A major drawback with existing integration environments is the weak form in which object identity and sameness is implemented. In other words, the implementation of the in-depth detection of conflicts, as outlined in Chapter 4, requires a precise formal representation of the identity and sameness of the potentially conflicting objects (candidates).

Hence, the propositional calculus introduced in Section 3.2.1, and existing object structures as typically used in enterprise integration, are extended with a novel object identifier. On this basis a novel notion of object sameness is introduced. This notion is a natural extension of existing research in the field. The underlying ontology is based on counterpart theory in the possible worlds semantics.

5.2 Conflict Detection - The Gathering Phase

Information retrieval precedes conflict detection (Section 2.6). In simplest terms this may involve reading a global schema for the nodes that can provide results and sending requests to these nodes. Requests may, however, need to be translated or matched. Combining the results necessitates that the retrieval procedure will allow the agents to translate the responses into a common language.

Conflict detection and resolution is a rational process for handling inconsistencies between the communicated information in response to an information request. As part of the process the information agent is able to stratify and evaluate the information that has been exchanged through the communication and retrieval processes. It is the follow-up of extensive information exchange between information agents. In the implementation described in Chapter 6., the information communication is based on the Knowledge Query and Manipulation Language (KQML) [CHA92]. This protocol is typically used for inter agent, and agent-to-source, communication in enterprise integration environments.

Section 3.2.1 introduced a basic propositional calculus which described explicit and implicit conflicts. The attributes R_1, R_2, to R_k are a set of attributes of an object O_i that is closed within at least one information source. For example, an object 'Peter' may have a closed set of all its attributes within an information source D1. In addition, to the basic propositional calculus, there is a need to specify attribute classes so that the attributes of different objects can be compared. Attribute classes range from v_1, v_2, ... v_m. These are implemented in relational tables in the form of fields, or in object-oriented databases in the form of tuple objects. For example, an information source may have a set of tuple objects with the attribute classes Name, Address and Phone Number (All names of attribute classes, names of tables or fields in relational databases start with a capital first letter). A particular object within this set may have the Name 'Peter', the Address 'Fridge Rd.' and the Phone Number '609 6486'. In order to increase the readability of the calculus, the attribute names are attached to the attribute classes in brackets (e.g. v_m(Name)).

Conflict detection starts with the information from various information sources that has been communicated through multiple information agents. In propositional calculus a request in the from $O_i.v_m$:? is sent out. For example:

Query: $O_i.(v_1$(E_Number): 123, v_2(Name):?)

'Concerning an object with the employee number (E_Number) 123 what is its name attribute?

Results: $O_1.v_2$(Name): Peter (from BookkeepingDB) or $O_i.v_2$(Name): Mark (from ProductionDB).

Multiple information agents may receive results from their sources and communicate them to the requesting agent. The framework designed in Chapter 4. provides the basic structure for starting the information integration with the Gathering phases (Section 4.4):

1. Gathering of Candidates
2. Gathering of Evidence
3. Classification

The information agent needs a formal representation to express the 'zero to many', results (candidates) to a query. These candidates may or may not be concerned with the same thing, and / or come to the same conclusion. For example, one agent A may have a result that the employee with the number '123' has the name 'Peter', and another agent may claim that the employee with the number '123' is called 'Mark'. Another example of two conflicting candidates would be an agent A has a result 'Peter is 18 Stone' and agent B has the result 'Peter is 20 stone'. In both examples, an investigation of whether the same 'Peter' is intended is required in order to determine if there is a conflict.

In summary, results consist of objects which are claimed to have certain properties (Section 3.2). A notion of identity is required to identify the objects in enterprise integration environments. It will now be investigated how existing research identifies objects and it is outlined why these approaches are inappropriate for enterprise integration. In the following a novel object structure is introduced.

5.2.1 Object Identification

5.2.1.1 The Concept of Object Identification

Here are presented some key issues and shortcomings of existing approaches, to integrating heterogeneous notions of object identity. These form the basis for the object identifier designed in this Section 5.2.1. Heterogeneous notions of identity in standalone information systems have been exhaustively described in the literature, for example in Shave [SHA75], Shave and Bhaskar [SHA82], or Loomis [LOO89]. Appendix B includes a brief overview of existing notions of identity in information systems.

> "Identity is that property of an object that distinguishes it from all other objects" [[KHO90]p.37].

Object identity is implemented in various forms in different information systems. For example, object-oriented databases implement surrogate based identifiers such as system-generated, unique numbers. Relational databases implement user defined keys such as every object in the table Person has a key field Name.

When multiple information sources are pooled in one information-sharing environment, the requirement for implementing a data model arises. Furthermore, within this model a notion of identity, which covers all objects in all systems, has to be implemented. A basic requirement for such a global, or all system spanning model, is uniform access to heterogeneous information bases. This can be provided by a transparent data model, e.g. described by Eliassen and Randi [ELI91]. It hides differences, for example, in query languages, data formats and also notions of identity. Khoshafian and Copeland [KHO90] have outlined that information systems ideally are implemented with:

- A strong notion of identity (system generated surrogate identifiers) because that allows for location (and origin), value and structure independence;
- A notion that incorporates a mechanism to integrate not only persistent object identifiers but also transient identifiers. These will be called session independent identifiers.

Location independent identifiers enable objects to be stored in different places or moved through systems. For example, an identifier which is not based on the physical address of an object can be manipulated by a program and then stored persistently with the same identifier in a different place. In addition, location independence makes controlled replication possible such that multiple copies of the same object can be stored in multiple places. Finally, one object may have multiple versions or temporal variants,

which are all stored under the same identifier, independent of the location of the object or its versions.

Value independence means that the identifier is not based on the value or contents of an object. For example, a relational table may hold the identifier key Second Name. A person 'Miss. F' may change her name into 'Mrs G' so that the object's identifier is changed but, in fact, the object is still the same individual. It is, therefore, important that identifier keys are not allowed to be changed during the lifetime of an object. User defined keys in relational databases, however, are typically changeable by a user and, hence, value dependent.

Identifier keys cannot guarantee one identity for the same object. For example, a table with the fields Employee and Children may have an employee 'Peter' in the Employee field with a child 'Mark'. However, the child 'Mark' may become an employee for the same company, in which case the Employee column would also hold the same object 'Mark'. In this case the object 'Mark' would be stored incorrectly in the table as two different individuals.

Value based identifiers make it very difficult for the data model designer to define classes of objects which always have the same, never changing attributes as identifier keys. In a relational table, for example, all the identifier keys must be valid identifiers for all tuples within that relation. It is not possible for one tuple to have different or changing key attributes.

A further problem with the value based identifier is that the object's attributes themselves are not identifiable. For example, the employee 'Peter' may have the attributes Profession ('Research Assistant') and Name ('Peter'). However, the attributes 'Peter' and 'Research Assistant' are not identifiable themselves in systems with value based identification.

Structure independence means that the identity of an object is independent of the structure and any changes to the structure of an information source. For example, in the relational model, identity can only be implemented, and is only valid, within that one table. Furthermore, changes to a table's identifier fields change the identifiers of all tuples within that relation.

> On a temporal dimension objects have a strong notion of identity "if they preserve their representation of identity within a single program or transaction, between transactions, or between structural reorganisations" [[KHO90]p.38].

A global data model that provides such independence of temporal changes to identifiers provides **session independence**.

In summary, a strong notion of identity in enterprise integration environments is one which provides location, value, structure and session independence. Ideally, an information system, and a sharing environment in particular, should implement a strong, transparent and uniform notion of identity.

5.2.1.2 Limitations Of System-Generated Surrogate Based Identifiers in Enterprise Integration

> "Surrogates are system-generated, globally unique identifiers, completely independent of any physical location. They are associated with each object of any type at the instant the object is created. They cannot be changed; i.e. they represent the identity of the object throughout the lifetime of the object" [[UNL90]p.167].

Most enterprise integration environments use system-generated, surrogate identifiers. In the approach by Pan and Tenenbaum [PAN91a], for example, every object throughout the whole enterprise is given a unique number. Every object is identifiable by this system-generated identifier. The advantages of using such identifiers, rather than other forms of identification, are that they can provide location, value, structure and session independence [KHO90]. A system generated number is not dependent on the location of an object, its value, or the structure of the information system. Nor does it change through processing by any transaction session.

However, there are a number of drawbacks when using surrogate identifiers. First, they require a central management system or enough commitment between agents to ensure that object identifiers are unique for all systems and all times. Open enterprise integration environments, therefore, cannot have system generated surrogate identifiers without loosing much of their openness (Section 2.3). Tight integration, however, fails to provide a platform for inconsistent, dynamically changing information-sharing (Section 3.3.1.3).

Second, these identifiers are not very expressive. For example,

> "value based matching is a straightforward and transparent technique for expressing relationships" [[PAT88]p.280]. "A system which supports objects with unique keys is therefore less expressive than one which supports object

identification as defined above [by value based matching or value based keys]" [[PAT88]p.285].

In most applications these drawbacks are more than outweighed by the benefits of the surrogate based identifiers. However, in enterprise integration there are pre-exiting, heterogeneous notions of identity that coexist with any artificial notion that may be used on the global, all system level. Hiding these differences behind one notion of identity without expressing the different ways in which objects are identified in their source of origin, may incorporate the following problems :

- Objects with value based identifiers react differently from other identifiers when changes occur to their properties [ELI91]. For example, value based identifiers in relational data models may be changed by altering the identifier key attribute. Persistent notions of identity, e.g., based on surrogates, would not change if any attribute of the object is changed. The effect of changing the attributes of an object are hidden by transparent surrogate identifiers. Hence, the effects of changes to the key attribute are hidden by a stronger notion of identity, such as surrogates.

- Persistent identifiers, and session based identifiers, react differently over time. For example session based identifiers, e.g. used for variables in programming languages, only survive a particular session or transaction. Other session based identifiers may only survive until another transaction changes the identifier again. Persistent identifiers, e.g. a surrogate or a tuple identifier as typically used in databases, hide this characteristic of transient objects. In other words, the object may be expected to exist permanently because the transient character of the object is not known.

- The different notions of identity carry different intentions of sameness and relationships to other objects in the same source. Object sameness is implemented on the basis of object identity. Different strengths of identity allow different forms of sameness. In a relational database, an object is defined by its user defined key. If two objects have the same key then common identity is assumed. However, if two objects with the same user defined key from different tables are mapped into a stronger notion of identity, then either they become different individuals with different identifiers, or they are given the same identifier and become one individual. This hides the fact that the lack of implicit support for referential integrity cannot be translated correctly into a stronger notion of identity.

The previously outlined limitations of surrogate-based identifiers lead to the following **conclusion:**

> "A conflicting requirement, however, is that the canonical data model [or any other integrated model] must provide this [strong] notion of identity based on component information bases supporting identity varying from the strongest forms of identity to the weaker forms only... [Further it is argued] that a strong notion of identity at the federated [or in general at the integrating] level can only be achieved by weakening strict autonomy requirements of the component information bases" [[ELI91]p.25].

In other words, the weaker notions of identity are strengthened to support the representation in a strong, surrogate based notion of identity. This research will show a different approach that does not require changing the identities of the integrated objects. It is based on increasing the expressiveness of the global identifiers in respect to the actual notions of identity of the integrated objects.

5.2.1.3 Deficiencies of Generalised Relations Between Information Sources

In most enterprise integration environments rules exist that link schema objects and thereby make generalisations about the correspondence of the objects they describe. String matching with generalised relations (schema objects) has been described in Section 3.3. A typical example are the equivalence relations based on the 'ist' operator and equivalence sign $\Leftrightarrow$ introduced by [COL91]. 'It means that an expression ψ that is true in the local source W_X is equivalent to an expression ϕ that is true in the global context G ($ist(G\phi) \Leftrightarrow ist(W_X\psi)$). Such mappings are called generalisations' (Section 3.3.1.1).

However, a problem is that generalisations are not sufficient for expressing disagreement over the existence of an individual:

1. Disagreement over the existence of an object (existential misconception) can have the form of **compression**, where an agent believes two objects have the same identity, these objects are compressed into one mental symbol in the representation of this agent [MAI91]. In the inter-agent relation one agent A can believe that two objects that are held by another agent B, have the same identity and are, hence, represented as one by agent A.
2. The alternative problem, is called **dispersion** [MAI91]. This is where one agent believes that it has multiple individuals (objects), which have multiple

representations, but these actually are one object in the real-world. Such an object has been 'dispersed' into multiple parts. The problem occurs between agents when an agent A has two objects which are represented by only one object in the model of another agent B.

Generalisations may not be applicable when concepts do not match exactly. For example, there may be cases where the generalised relations hold true except for one, or a few, objects. Concepts may be incomplete, incorrect, or inconsistent. An extension is required that allows for defining relations between individual objects in addition to generalisations. An example of such a generalisation is:

> 'An object in table Person in source D1 is identified by its Name, and is equivalent to an object in class People, in source D2 where objects are matched by their attribute Name'.

It may then be the case that the object called 'Yogi' in D1 is equivalent to 'Yogi' in D2. However, despite the generalised relation there may be cases such that 'Peter' in D1 is not equivalent to every 'Peter' in D2. It should be possible to express these exemptions.

Furthermore, the same real world object can be represented in different information systems where it may be implemented and / or identified differently. Objects can be related to a model of the real world as in CARNOT's CYC knowledge-base [COL91], MKS [PAN91a] or MIND [JAG94]. In loosely coupled information-sharing environments some objects may be defined in such enterprise models and others may not be defined. The latter group of objects start to exist in information systems without known resemblance to a real world object. Existing research fails to provide mechanisms to define relations between objects and their real world counterparts and, in addition, allows for defining relations between objects that exists in information systems without known real world counterparts.

This research proposes an extension to the generalisations that overcomes these shortcomings in the previously described object structure.

5.2.1.4 A Novel Object Structure for Enterprise Integration Environments

The object structure proposed by Khoshafian and Copeland [KHO90] is grounded on the concept of an object O_i which has the structure 'Object.(identifier, type, value)' in the object model.

a. The 'identifier' is one of a set of system-generated identifiers I. The identifier of an object O_i is denoted O_i.identifier.

b. The 'type' is in {atom, set, tuple}. It provides only single-level typing for simplicity. This typing system could be more complete. In distributed information systems, however, the dominate type is that of a tuple and it will also be used in this research. Every object has, therefore, at least one attribute class and a proposition for this class. The type variable, however, provides the conceptual basis for representing objects of any kind including set and atom.

c. The 'value' is one of the following :

   1. If the object is of type atom, then the value is a data element which has no sub-parts. For example, the name 'Peter', the number '1234' or a picture of 'Yogi bear'.

   2. If the object is a type set, then the value is a set of distinct identifiers from I. A set is an object that relates a number of objects. This group is, for example, a class of objects which all have properties for the same attribute classes [MAS90].

   3. If the object is of type tuple, then the value is of the form: $[v_1:R_1, v_2:R_2, \ldots v_m:R_k]$, where the v_m's are attribute classes for which the object O_i has taken the value R_k.

This object structure by Khoshafian and Copeland [KHO90] requires a management system, or a tight integration of systems, that provides unique, surrogate identifiers for every object in the canonical global object model. However, for the research presented here a formalism is required that can express objects that have:

- Object identifiers that are not from a central global model but that are managed within the integrated sources; and

- These identifiers are of different notions and therefore may not be surrogate identifiers by nature.

A novel identifier will be described for the above object structure that incorporates these heterogeneous, independent notions of identity. This novel identifier

[Identifier_Object, Identifier_Class, W_X,]

consists of the following elements :

- The **Identifier_Object** specifies the externally assigned Identifier such as: 'Employee.Name = Peter' for an object that has a value based identifier in the table Employee on the attribute Name; or the Identifier_Object 'Surrogate Identifier = 12345' is a surrogate identifier for an object-oriented surrogate based system.

- The **Identifier_Class** contains the information about how objects are identified in a source. An example Identifier_Class may be 'user defined key for a relational table, the key field is unique'. This variable can be used to interpret the Identifier_Object 'Employee.Name = Peter' as: 'Employee.Name' is the unique, user defined key of a relation data model which has the value 'Peter'.

- The name space or **Naming World** of an object is specified by the variables W_1, W_2 to W_X which indicate where the object can be identified with the Identifier_Object and Identifier_Class specifications. In principle, a world W_X can be one source such as a database, or multiple sources such as a distributed databases that has got a united identification mechanism, as in [KIM91a].

The concepts of Naming World and Object_Class will be considered in further detail in the following paragraphs including a brief discussion of the completeness of the object system.

Naming World

A Naming World is also called a scope and can be defined for a surrogate based systems as:

> "A scope is an arbitrary object such that each token (surrogate) belongs to exactly one scope. An operation in the computational system is executed within a scope" [[KEN91]p.36].

A scope is, therefore, part of one information system, a whole information system, or a Group of information systems (Section 2.7. Resource Knowledge), that have a common notion of identity. However, one kind of identity may include subgroups. For example, a relational database may have user defined keys that are differently implemented in that some keys are unique and others are not. Furthermore, standard software systems may use address based or process based identifiers and, in addition, have a file system that has no implemented notion of identity. A Naming World therefore has one kind of identity including a possible subgroup of this notion of identity.

One potential problem with these identifiers is that they contain information about the Naming World W_X. This makes the identifier location dependent. In other words, if an object moves from one world into another its identity would change. A very pragmatic solution, as used in distributed databases [KIM91a], would be the introduction of a Versioning mechanism that would change the identifier into:

[Identifier_Object, Identifier_Class, W_X,] ***VER***

Thus, if an object moves from one world into another, its identifier would change as the W_X specification is altered. The old version of the identifier, however, could remain and the variable VER would indicate the new successive identifier. This makes the identifier location independent since uniform access to the object via its initial identifier is provided despite the new location of the object.

This Versioning mechanism will, however, not be used in the reminder of this research for two reasons:

1. This Versioning mechanism does not exist in enterprise integration environments. It is novel and proposed here for improving the modelling of object identity in sharing environments. The conflict detection and resolution mechanism should, however, be based on mainstream modelling approaches and not make assumptions about new modelling concepts. In other words, it cannot be assumed that the VER variable is implemented in existing environments when the detection mechanism has to be general in that it is generally applicable to existing environments of all kinds as described in Section 3.3.
2. For the scope of conflict detection the location dependence of the notions of identity given to an object for the duration of the detection and resolution phase is practically of no importance. The conflict management requires a more expressive notion of identity and it, hence, implements for its own purpose this extended notion of identity. While the location dependence of this identifier may be important to enterprise modelling, it is not important to the mechanism itself.

Another extension to existing object structures is necessary to allow for relations between objects from different Naming Worlds, including dispersion, compression, one-to-one relations. Counterpart theory is adopted to overcome the shortcomings of traditional possible worlds frameworks for integrating environments (Section 5.2.1.3).

Lewis' [LEW68] counterpart theory is an extension of the possible worlds semantics. It provides an ontological structure for the novel identifier. Counterpart theory is based on the principle that every object in a world is different from any object in any other world. An object in one world W_x can have a counterpart in a different world W_y, such that:

> "The counterpart relation is our substitute for identity between things in different worlds....Within any one world, things of every category are individuated just as they are in the actual world; things in different worlds are never identical" [[LEW68]p.114].

Research based on accessibility relations between worlds such as Hinttika [HIN62], Halpern and Moses [HAL92b] [HAL91], and most enterprise integration environments (Section 3.3) "have proposed interpretations of quantified modal logic on which one thing is allowed to be in several worlds" [[LEW68]p.115].

This concept corresponds to objects in different Naming Worlds (e.g. information sources). In one Naming World W_x, which has a common notion of identity and sameness, objects are either identical or different. Across worlds there is no common notion of identity so there can be no common identity. Objects in different worlds may have common real world counterparts, they may resemble each other closely or more closely than any other objects, but they are artificial objects that are implemented in information systems with independent notions of identity. The next section, therefore, defines notions of sameness, not identity, between objects from different worlds, and identity only for objects from the same world.

> "Your counterparts resemble you closely in content and context in important respects. They resemble you more closely than do the other things in their worlds. But they are not really you" [[LEW68]p.114].

That relation is described by the object identifiers x and y as

> Cxy where y resembles x more closely than any other object in y's world.

For the complete definition of counterpart relations please refer to [LEW68] and the Appendix C.

There are at least three advantages to conflict detection and resolution of using counterpart theory as an extension to existing mappings between information sources:

1. Generalised relations between information sources have been shown to map every object from one world with exactly one object in another world (Section 3.3.1.1). For example, G (ist(Gϕ) $\Leftrightarrow$ ist($W_X\psi$)) means that the expression ϕ that is true in the world G is equivalent to an expression ψ that is true in the local source W_X. This means that ϕ is a counterpart of ψ and vice versa. As an extension to the original theory such a reflexive counterpart relation will be described as:

   $RCxy =_{df.} (Cxy) \& (Cyx)$ where $RC\,xy = RCyx$

   This, however, is only one, special relationship in counterpart theory. It is also possible that :

   1. Objects can have more than one counterpart in another world.
   2. Multiple objects in one world can have a common counterpart in some other world.
   3. Objects may not have any counterpart in a particular world, or in any world.

   Maida [MAI91] has therefore proposed counterpart theory to overcome the problem of existential misconception where, e.g., two objects are identical and both resemble the same object in another source.

2. Exemptions to generalised relations could be defined with the definition: '~ Cxy', which means that it is not true that x is a counterpart of y. For example, the generalisation may exist that objects identified by their Person_Name attribute from source W_X, are equivalent to objects identified by Student_Name from source W_Y where both identifier fields have the same value. Within this generalisation the definition '~C('Person_Name = 'Yogi", Identifier_Class, W_X)('Student_Name = 'Yogi", Identifier_Class, W_Y)' would define that the object 'Yogi' (Person_Name in W_X) is not a counterpart of the object 'Yogi' (Student_Name in W_Y). This feature is, however, not typically used in enterprise integration and, thus, a suggestion to improve enterprise modelling. It will not be demonstrated in the following of this research in order to comply with existing integration environments.

3. Counterparts in the real world are not defined differently from counterparts in the other worlds. This allows for defining some objects that may have known

counterparts in the real world, while others may not. The real or actual world is defined in [LEW68] as:

$\exists x\ (W_x\ \&\ \forall y\ (Ixy \equiv Ay))$

Some world W_x contains (Ixy) all and only actual things (A_y)

Accordingly Lewis specifies the 'actual world' as "some world contains all and only actual things (P7), ... [and] Something is actual (P8) "[[LEW68]p.114]. In enterprise integration environments, models of the enterprise [PAN91a] [JAG94], or the common knowledge-base CYC [COL91] are used to demonstrate that objects in information systems have counterparts in the actual world. Furthermore, hand-crafted lists are used to establish that conceptual objects, for example in multidatabases [LIT90], resemble the same real world object and are, hence, counterparts.

Identifier_Classes

The basic idea of Identifier_Classes is that different notions of identity are implemented in diverse information sources throughout the environment and each notion has its own particular characteristics defined as Environmental Information (Resource Knowledge in Section 2.7). These characteristics are part of the identifier. The introduction of the Identifier_Classes includes an example for a relational database table in which objects are identified by a unique identifier field. However, there are a number of different ways in which objects can be identified in information sources. A few examples will be given in the following based on the summary of existing notions of identity in Appendix B.

An object-oriented data model typically implements system generated, surrogate identifiers. A management system provides unique identifiers (Identifier_Object) for every individual. The sum of all surrogate identifiers is a closed set within an information source, e.g. ranging from 1 to m, because "the important things about symbols [called surrogates in this paper] is that they constitute a fixed, though infinite set" [[KEN91]p.29].

Surrogate based identifiers can also cover multiple information sources that must be closely integrated into one Naming World. The identifier is then composed of a local surrogate identifier and the specification of the surrogate management system. The specification of the surrogate management system has been implemented for object-oriented distributed databases, e.g. in the ORION-2 system [KIM91a], or for integrating information systems by, e.g., Czejdo and Taylor [CZE92].

The ORION-2 system is also a good example of an object-oriented system that implements Versioning. Objects may have different versions so that they can be opted out of persistent data stores for long periods of time. The same object may therefore have different versions at the same time. It follows that an Identifier_Class specification for such a system would have to make the Versioning mechanism explicit, e.g. 'Surrogate Based Identifier with Versioning'. This way conflicts between different versions of the same object can be identified by the information agent.

Notions of identity other than surrogate identifiers or value based identifiers can be found in mainstream information systems. These include tuple identifiers (INGRES [STO76] or RM/T [COD79], the 'indirection' of objects via an object table (Smalltalk-80 [KAE83] or KBZ [OXB88]), or addresses of objects (e.g. hierarchical and network databases [OLL78]).

In enterprise integration environments many other notions of identity can be found that are less formally defined as in databases or programming languages. For example, objects in standard software systems may be identified by their creation time, or processes and the attributes describing this process may be identified by a process identifier [OHO90]. It is easy to imagine that in a production environment every process is given a number (process identifier) at initialisation.

Objects may also be identified within definitions in an existential framework [RES75] similar to identifying objects in a master model of the enterprise [PAN91a]. For example, for a given software system Student objects are defined to have the essential property Name. Hence, without this notion being defined as a key, e.g. in a relational database, it functions as an identifier that is based on the designers decision to make this attribute an essential attribute of all students.

Different notions of identity, which lead to different Identifier_Classes, may also be defined for the same form of identification. For example, a user defined key in one system may be less strictly implemented than in another. In one system user defined keys may always be unique identifier fields and in another this may not be guaranteed.

In conclusion, the object identifier is complete in that it integrates objects of all notions of identity. In comparison to existing structures it is more expressive and therefore tailor made for the requirements of enterprise integration frameworks.

Consistency of the Object System

"An object system is consistent if

(a) No two distinct objects have the same identifiers (unique identifiers assumption). In other words, the identifier functionally determines the type and the value of the object.

(b) For each identifier present in the system there is an object with this identifier (no dangling identifier assumption)" [[KHO90]p.41].

The system described here may, therefore, be classified as not consistent because it cannot assure that no identifier may be dangling in the global system. Transient data, for example, only has a limited lifetime and the identifier of an object that is still existing globally has actually been changed in the local source. Modelling of the enterprise integration environment would need to be able to manage object identifiers over time. However, for the purpose of conflict detection this inadequacy is insignificant. In other words, the dangling identifier assumption is fulfilled within the results that an information agent has at any time.

The novel identifier is generic in that it is developed according to the existence and characteristics of a particular object at the time of investigation. In order to extend the novel identifier to become a conceptual part, not only of conflict management, but of the enterprise modelling, a mechanism is needed for dangling identifiers. Hence, the unique identifier assumption is fulfilled when the identifier is used within the detection mechanism.

In summary, a sufficiently rich structure has been described that enables information agents to identify candidates and individuate them. The structure has the typical advantages of object-oriented models including a strong notion of identity (e.g. similarities with programming languages, inheritance and encapsulation provide the necessary flexibility to deal with complex, heterogeneous objects). It is a natural extension of existing object-oriented structures in existing enterprise integration environments. In addition, the novel structure is much more expressive, and it is 'generic' in that it incorporates the actual notions of identity that objects in diverse information sources actually posses.

5.2.2 Gathering of Candidates

In Section 5.1 the example query 'O_i.(v_1(E_Number): 123, v_2(Name): ?)' (Concerning an object with the employee number (E_Number) '123' what is its Name attribute?) produced the results:

O_1.v_2(Name): Peter (from BookkeepingDB) and
O_i.v_2(Name): Mark (from ProductionDB)

First, the agent has to identify the sources of origin or Naming Worlds. These are here called the databases BookkeepingDB and ProductionDB. With the information on the Naming Worlds the agent can investigate the object type that is communicated. These are typically of type 'tuple' but they could also be of other types (atom or set) as described by Khoshafian and Copeland [KHO90].

Based on the Naming World the agent can identify the Identifier_Class which in turn specifies the Identifier_Object. Unless the latter is already provided by the results this Identifier_Object is then requested by the information agent from the source of origin of this result.

For example, the result 'O_1.v_2(Name): Peter', with the Naming World BookkeepingDB, has the Identifier_Class: '*User Defined Key* from a relational database'. It follows that the Identifier_Object is called 'User Defined Key'. The agent requests this Identifier_Object for the object 'O_1.v_2(Name): Peter' and may receive the result: 'O_i.v_1(Employee.E_Number):123' from the BookkeepingDB.

The next step is to gather the evidence for the candidates which will be described in the next Section.

5.2.3 Gathering of Evidence

In Section 4.3 a definition of evidence in information systems was outlined such that evidence for believing a proposition ($O_i.R_k$) may be based on:

- A result (also called candidate) from any information source:
- Data that supports (warrants) or refutes the proposition or candidate (E_m).

Formally, any candidate of an object ($O_i.R_k$) may have zero to multiple pieces of evidence (E_0, E_1, E_2, ... E_m) which may support or refute the correctness of this candidate resulting in sets of '($O_i.R_k$)(E_0, E_1, E_2, ... E_m)'.

Uncertainty management systems based on argumentation typically uses similar ways to describe an argument. Section 3.5.3.4 defined that an argument consists of a piece of data, a warrant that leads from the data to the claim, and a qualifier that may further evaluate the claim [FOX92b]. The logic LA [FOX92b] has extended this basic concept by introducing the signs '++' and '+' to describe how strongly the evidence is believed ('--' and '-' can be used alternatively to '¬+' and '¬++'). The evidence itself, however, may consist not only of a single proposition but of multiple propositions, which are connected by the connectives '¬, ∧, ∨, →, ⊥'. For example, the proposition

'Yogi weighs 18 stone'

may be supported by the evidence:

'The doctor said that Yogi weighs 18 Stone' → 'Yogi weighs 18 Stone', ++.

A major advantage of the logic LA is that "one can claim to be able to argue for some proposition P while also being able to argue against P from some different "point of view"" [[FOX92b]p.625]. Fox, Krause and Ambler [FOX92b] have outlined that practical reasoning cannot avoid inconsistencies and that it is necessary to outline these inconsistencies. Information agents need to be able to gather and embrace the conflicting evidence that may typically exist in enterprise integration environments (Sections 2.4 and 2.5 have outlined reasons for inconsistencies in open integration environments).

This basic concept has been applied to the structure of evidence in enterprise integration. Evidence E_m may include any number of propositions, which are connected by connectives, e.g. '¬, ∧, ∨, →, ⊥', to build Formula. These may have certainty estimates assigned to them in the Certainty Estimation variable:

E_m = {(Formula) (Certainty Estimation)}

Any evidence E_m may have only one or multiple formulae, or only consist of a certainty estimate. In the latter case the evidence is a reliability statement, e.g. based on possibilistic or numeric certainty estimates (see Section 3.5.2).

For example, the candidate 'Yogi weighs 18 Stone' ($O_i.R_k$) may have the evidence

E_1 = {('The doctor said that Yogi weighs 18 Stone')(very likely)}

In other words, the proposition 'The doctor has said that Yogi weighs 18 stone' makes the proposition 'Yogi weighs 18 Stone' very likely. In contrast to the formal structure used by Fox *et al.* [FOX92b][FOX91] the candidate is separated from the evidence that supports this candidate. This is necessary because the information agent differentiates between results (candidates) and evidence that supports or rejects this candidate.

Another difference to the original approach is to allow any certainty estimates and not just '++', '+', '--' and '-'. It has been concluded in Section 3.5 that different certainty estimates may be used throughout the integration environment and that the information agent typically is not be able to transform:

> Any estimate, based on any notion of certainty, from any information system throughout the environment,

into one coherent certainty estimate. Hence, any certainty estimates that are used by different information sources may be used in the evidence's Certainty Estimate variable.

Sections 2.3, 2.7 and 2.8 outlined which sources of information an information agent has available. These may be divided into information that:

- An agent has itself (Agent Knowledge);
- Information it can retrieve from its local source;
- Information it can get from other agents.

The Agent Knowledge has been described in Section 2.7. Information retrieved from the local source are the agents' own results (candidates) and any evidence for these candidates. The third source of evidence is the other agent's knowledge and information they have received from their integrated sources. Because the agents are benevolent it is unnecessary to differentiate between information they retrieved themselves, and information other agents retrieved and forwarded to them (Section 2.5). It is a modelling issue how information agents exchange and share their Agent Knowledge. In this research it has been assumed that agents exchange all the Agent Knowledge they want to share instantly in a federated mechanism (Section 2.3). In other words, agents exchange

federated schemata, and any Schema or Resource Knowledge when they join the sharing environment, e.g. described by Huhns and Singh [HUH92].

In a retrieval process some of this information would typically be retrieved automatically by the local sources. For example, a statistical package or justification based expert systems, may always provide evidence for any results from their sources. However, most evidence has to be explicitly requested by the agents from the local sources (e.g. Services - Resource Knowledge Section 2.7). Furthermore, the agent has to explicitly investigate its Agent Knowledge in order to find relevant information that may be used as evidence (e.g. Comments, or Integrity Constraints Section 2.7). Out of all the potential sources of evidence described in the previous paragraph an information agent could:

- Gather all available evidence for every candidate at this stage of the conflict management;
- It could only gather the evidence that is automatically provided by information sources and request further evidence when it is needed in later detection and resolution phases;
- The agent could ignore any evidence.

The latter choice is not rational in that it is not logical and systematic to ignore evidence (Section 2.6). The first two options are both logical and systematic. However, it is much more efficient to leave any further requests for evidence until this is needed. Please note, that the detection phase so far has not even investigated whether or not a conflict is at hand. This will be done in the following Classification step.

In conclusion, an information agent can use the formalism described above to gather evidence. This evidence is assigned to particular candidates. The information agent is most pragmatic if it gathers only the evidence that is automatically provided by the information sources or their integrating agents. The managing agent would typically request and gather, in the same way, any further evidence as it is needed in the following detection and resolution steps.

5.2.4 Classification

5.2.4.1 Introduction to Classification

The previous two steps have provided a structure to represent a candidate ($O_i.R_k$) and its evidence (E_m) in the following form:

> **O_i.([Identifier_Object, Identifier_Class, W_x] Type, $Value_k$)**
> **(E_m={(Formula) (Certainty Estimate)})**
>
> For example, the candidate object O_1 may be identified by the Identifier_Class '*User Defined Key* in a relational database' and the Identifier_Object 'Employee.Name = 'Yogi". It is from the Naming World (W_1) 'BookkeepingDB' and of type tuple. The Value variable has the attribute classes v_1 and v_2 with their properties R_1 and R_2 (v_1:R_1, v_2:R_2) such that: 'v_1(Name): Yogi, v_2(Weight of object): 18 Stone'. The agent received evidence (E_1) from a statistical tool attached to the database BookkeepingDB that specifies the statistical reliability of data from that source as E_1 ={()(99 % confidence)}.

The following step is to classify the retrieval result(s) (candidates) in respect to whether or not they are explicitly and / or implicitly conflicting. The next section will briefly apply the prepositional classification introduced in Section 3.2.1. Section 5.2.4.3 will then outline a novel notion object sameness based on the novel object identifier introduced in Section 5.2.1. These sameness predicates are then used to classify the candidate's sameness (Section 5.2.4.4). Section 5.2.4.5 will then present the classification of evidence step.

5.2.4.2 Classification of Conflict Between Candidates

The information agent receives null to multiple results in return for its request. The candidates (results) include the properties of the objects O_i in respect to the attribute class v_m. In principle, the agent may encounter a situation where :

1. **No candidate** is found throughout the sharing environment;
2. Exactly **one candidate** has been produced by the sharing environment;
3. **Multiple candidates** are produced.

The first two cases do not require any further conflict detection because there cannot be a conflict in case of less than 2 candidates. In the same way, multiple candidates that are all identical in that they are all claiming the same thing are not conflicting. For example, multiple results all claim that 'Yogi is 19 Stone'. In other words, conflict detection is concerned with identifying different results, or candidates, to a given query.

Multiple results form pairs of possibly conflicting results (Section 2.5 and 3.2.1). In other words, in case of 'n' multiple results a given query produces '(n^2-n) /2)' pairs of results that may potentially conflict. For example, the three results (n = 3) 'Yogi is 17 stone', 'Yogi is 18 stone', 'Yogi is 20 stone' form the following pairs ($((3^2-3)/2) = 3$) of possibly conflicting candidates:

1. 'Yogi is 17 stone' and 'Yogi is 18 stone';
2. 'Yogi is 17 stone' and 'Yogi is 20 stone';
3. 'Yogi is 18 stone' and 'Yogi is 20 stone'.

In principle, one result can be internally conflicting. For example, the result 'Yogi weighs 18 stone and is a slim athlete' may be implicitly conflicting. However, typically this result would produce two candidates:

'Yogi weighs 18 stone', and 'Yogi is a slim athlete';

This classification is provided by the propositional representation presented in the Gathering of Candidates phase described in the previous Section 5.2.2.

Section 3.2.1 outlined the different kinds of propositional conflicts and elaborated that in Enterprise Integration:

> 'Information agents will have to identify all explicit conflicts and those implicit conflicts between results that are concerned with the same object. Typically, the conflict is about the same attribute class of the investigated object'.

In the following some described kinds of conflicts will be briefly outlined again. For example, information agents have to identify explicit conflicts, which have the structure:

'$O_i.v_m$:R_k' and 'not $O_i.v_m$:R_k'

Thus, two candidates are explicitly conflicting if one claims that the object O_i has the property R_k for the attribute class v_m, and the other candidates claim that this is not the case.

Conflicts in enterprise integration environments typically are, however, implicit conflicts where a query such as 'What is Yogi's (O_i) weight(v_m)' ($O_i.v_m$(weight): ?) results in different properties for the same attribute class v_m:

$O_i.v_m$:R_1 and $O_i.v_m$:R_2

In order to identify this kind of conflict, the information agent needs to know that the attribute v_m can only be either R_1 or R_2 but not both. For example, 'the weight of Yogi' can only be '18 stone' (R_1) or '20 stone' (R_2). Similarly the conflict may be about different attributes of the object O_i:

$O_i.v_2$:R_2 and $O_i.v_3$:R_3

For example, the object 'Yogi' may have a 'weight' (v_2) of '18 stone' (R_2) and it may be claimed that this object also has the property 'slim' (R_3) for the attribute class 'v_3(body type)'. However, identifying this kind of conflict requires that the information agent has very detailed knowledge about relations between an object's 'weight' and the 'body type' (slim, fat,...).

Other kinds of implicit conflicts have been described in Section 3.2.1. However, in order to identify implicit conflicts the information agent may have the following sources (described in Section 2.7):

- Integrity Constraints (Resource Knowledge); For example, the integrity constraint 'Bears are only of the body type slim if they weigh less than 10 stone' enables an agent to identify an implicit conflict between the candidates: '$O_1.v_1$(Name): Yogi, v_2(weight): 18 stone' and '$O_2.v_1$(Name): Yogi, v_2(Body Type): slim'.
- Environmental Information may be defined as an agent's Resource Knowledge, in enterprise integration models [PAN91a], or common knowledge-bases as CYC [LEN90]. For example, an enterprise model may define essential properties of concepts, such that the concept employee may have the essential property Name as an identifier (which means that every employee should have a Name). Two results that claim that the same employee has two different names are, hence, implicitly conflicting. But also more complex implicit conflicts can be defined with

the help of rules. For example the previously defined integrity constraint may be expressed by a rule in an enterprise model such that: 'The concept Slim is a body type and defined by a weight of less than 10 stone'. Based on this rule it can be identified that the results 'Yogi weighs 18 stone' and 'Yogi is of body type slim' are implicitly conflicting.

- Extensional Information is derived from knowing a concept and applying it to a result. For example, if an agent knows that the body type 'slim' is defined by a weight of less than '10 stone' (Environmental Information - Resource Knowledge), then it can derive the Extensional Information: 'Yogi is not of body type slim', from the result 'Yogi is 15 stone'. This Extensional Information may then explicitly conflict with a result 'Yogi is of body type slim'. Existential Information is different to Environmental Information because the inconsistency is detected not by an explicit rule, but by inference from environmental facts ('Slim means less than 10 stone'; and 'Yogi is 15 stone').
- Comments (Resource Knowledge) include definitions by the designers, integrators or system administrators on consistency regulations. For example, a designer my define that every employee object may only have one E_Number and one Name.
- The agent may assume that all results have to be exclusive unless otherwise defined. This is typically the way distributed databases, and integration environments function (Sections 3.3 and 3.5.3.2). Such assumptions are typically defined implicitly as Organisational Knowledge [HSU91] such that the detection mechanism identifies such conflicts. For example, unless otherwise defined the results 'Yogi weighs 18 stone', and 'Yogi weighs 20 stone' are implicitly conflicting. Thus, unless the agent explicitly knows that someone can have both weights at the same time a conflict is assumed.

Section 3.2.1 demonstrated that it is very difficult for information agents to identify implicit conflicts between candidates that are not concerned with the same object. However, the following two examples based on an enterprise model and external services demonstrate how this, in principle, may be possible:

- Enterprise integration models (e.g. [PAN91a]) may contain domain-specific information defining complex relations between objects. For example, a production rule may define that two products cannot have a daily production of more than one hundred units per day. Hence, the results 'Product P1 has a daily production of 60 units', and 'Product P2 has a daily production of 70 units' are implicitly conflicting.
- A domain expert, e.g. a planning system as described by Klein and Lu [KLE89] (Section 3.4.4), may identify complex implicit conflicts. For example, the

information 'Product BigMac has the daily production 50 units', and 'Product Hamburger has the daily production 40 units' may be implicitly conflicting because the planning expert knows that:

-'Only 120 Meat Balls are available per day;
-BigMacs require 100 Meat balls;
-Hamburgers require 40 Meat Balls;

-Thus 50 BigMacs and 40 Hamburger can not have been produced in one
entailing that the results can not both be true at the same time'.

Hence, the domain expert (i.e. the planning system) can derive that the results are implicitly conflicting. This kind of expert knowledge is a service to the information agent (Resource Knowledge).

One **potential limitation** of attempting to identify candidates and classify possible conflicts among them in the field of enterprise integration, is that the only results considered are those presented to the agent by the retrieval process. Furthermore, the identification of implicit conflicts depends on the knowledge of the relation between the candidates (e.g. see the previous paragraph). The previous paragraphs have outlined how the information agent could manage this knowledge. However, acquiring and maintaining this information in existing systems relies on manual, expert definition. Alternatively, conflict identification can be solely undertaken by human users (e.g. [PAL92] Section 3.3.4). Future research might improve the agent's conflict detection capabilities, together with methods of maintaining this knowledge, by modelling the way human experts identify conflicts. However, current environments can detect all explicit and most implicit conflicts. The impact of limited Agent Knowledge on the detection of implicit conflicts may be small depending on the complexity of the problem domain.

The final step in classifying the candidates is to describe their correspondence assumptions. In other words, all explicit conflicts and those implicit conflicts between candidates that are concerned with the same object assume that the objects correspond to the same thing and therefore conflict (Section 3.2.1). For example, the following candidates are implicitly conflicting:

- O_1.(v_m(Name): Yogi, v_m(Weight): 18 stone)
- O_2.(v_m(Name): Yogi, v_m(Weight): 12 stone)

If these candidates are derived from different information systems then it has to be assumed that both candidates are concerned with the same object ('Yogi'). This requires a definition and analyses of object sameness for heterogeneous enterprise integration environments as presented in the next section.

5.2.4.3 Sameness Predicates

Identity and sameness predicates in homogeneous object models based on Khoshafian and Copeland's [KHO90] object structures can be described as:

1. Identical [$id(O_1,O_2) =_{df.} (O_1.identifier = O_2.identifier)$],
2. Shallow Equal [$se(O_1,O_2) =_{df.} (O_1.R_k = O_2.R_k)$],
3. Deep Equal [$de(O_1,O_2) =_{df.}$
   - Atomic objects : $((O_1.R_k = O_2.R_k)$
   - Tuple objects : $(O_1.v_m{:}R_k = O_2.v_m{:}R_k)$
   - Set objects :$(O_1.cardinality \ \& \ O_2.cardinality) \ \& \ (O_1.v_m{:}R_k \ O_2.v_m{:}R_k$ where R_k entails identifiers of referenced objects)].

In homogeneous distributed systems object identity typically subsumes the question of object sameness. In other words, in a homogeneous notion of identity every object is, for example, identified by a surrogate based identifier. Objects are Identical ($id(O_1,O_2)$) if they have the same identifier ($O_1.identifier = O_2.identifier$). For example, Peter from source D1 with the identifier '12345' and Peter from source D2 with the identifier '12345' are identical. This is called an Identical predicate in contrast to Shallow Equal objects ($se(O_1,O_2)$) which only have the same value 'Peter' but are different objects ($O_1.R_k$, $O_2.R_k$). For example, 'Peter' from source D1 may have the identifier '12345' and 'Peter' from source D2 may have the identifier '12346'.

> "The essential point is that the object equality refers to the sameness of value, while the object identity (sameness) refers to the sameness of objects themselves" [[MAS90]p.186].

Objects are Deep Equal ($de(O_1,O_2)$) if their structure and their corresponding components are equal. In the case of atomic objects, deep and shallow equality have the same meaning. Atomic [KHO90] or primitive [MAS90] objects are character and number objects which do not change their state. In the case of tuple objects their corresponding attribute classes need to have identical properties such that $O_1.v_m{:}R_k = O_2.v_m{:}R_k$. Set objects are those that are composed of references, e.g. identifier numbers, to other objects. Such set objects are Deep Equal if they have the same cardinality ($O_1.cardinality = O_2.cardinality$) and the instances they address are pairwise deep equal ($O_1.v_m{:}R_k \ O_2.v_m{:}R_k$ where R_k entails identifiers of referenced objects).

These sameness predicates, however, were defined in relation to object-oriented structures. In other words, they fail to provide sameness predicates for identifiers other

than homogeneous, surrogate based identifiers in object-oriented structures. For example, these predicates do not integrate tuple identifiers, or user defined keys. Hence, it is proposed to combine:

- The notion of sameness based on identical, shallow equal and deep equal, and
- The novel identifiers [Identifier_Object, Identifier_Class, W_X,] from Section 5.2.1.

This will provide the formal concept for extending the object structure to heterogeneous notions of sameness. These include the following three categories of equalities between objects in the conceptual object model:

1. Identical Predicates

A. Purely Identical

$PI(O_1,O_2) =_{df}$
O_1.[Identifier_Object, Identifier_Class,W_X] ≡ O_2.[Identifier_Object, Identifier_Class, W_X]

B. Derived Identical

$DI(O_1,O_2) =_{df}$
O_1.[Identifier_Object, Identifier_Class] ≡O_2.[Identifier_Object, Identifier_Class]

2. Match Equal

$ME(O_1,O_2) =_{df} (O_1.[\text{Identifier\_Object}] = O_1.v_m:R_k)$
$\& (O_1.v_m:R_k = O_2.v_m:R_k)$

3. Aspect Equal

$ASE(O_1,O_2) =_{df} O_1.v_m:R_k \equiv O_2.v_m:R_k$

In this approach, as with Khoshafian and Copeland's notion of sameness [KHO90], identical objects have to be replicated objects from the same Naming World (in [KHO90] a surrogate management system). Their identifiers Identifier_Object, and Identifier_Class, are identical and they are implemented in the same Naming World W_X. Such a Purely Identical predicate is different to traditional models in that it covers any notion of identity that allows for identical objects, and not just surrogate based notions.

Typically, however, information-sharing environments integrate objects from different sources, or Naming Worlds. Counterpart theory maintains that objects from different worlds cannot be identical (Counterpart Theory Section 5.2.1.4). In other words, objects are Derived Identical if their Identifier_Classes and Identifier_Objects are identical. For example, the objects O_1 and O_2 may both have the Identifier_Object 'Name = 'Peter'' with the Identifier_Class 'Relational User Defined, Unique Key'. In such a case, these objects are Derived Identical.

Objects from different sources, using different data modelling concepts, often have different Identifier_Classes. In the case of Match Equality one object's O_1 Identifier_Object for the Identifier_Class in the Naming World W_x, is another object's (O_2 in world W_y) attribute R_k such that the Identifier_Object and the attribute R_k are from the same attribute class v_m. The objects are Match Equal (ME) if the attribute $O_2.v_m:R_k$ and O_1's Identifier_Object of the attribute class v_m are identical. For example, an object O_1 in a relational database W_x may be identified by its Identifier_Object 'Name = 'Peter'', and its 'Identifier_Class' 'Unique User Defined Key in a Relational Database'. The second object O_2 may be from another database (and world), identified by the Identifier_Class 'System Defined Surrogate', and the Identifier_Object 'Surrogate = '12345''. However, if the object O_2 has the attribute (R_k) 'Peter' in the attribute class 'Name' then the objects are Match Equal.

In contrast to Match Equality, objects may not even partially share their identifiers. In such cases the objects may only have a common characteristic (they are equal in respect to one specific aspect). In other words, Aspect Equal objects have distinct identifiers but have common attributes for the same attribute class. For example, an object O_1 from the database D1 (world W_x) is identified by a surrogate based identifier '12345' and has the attributes 'Name = Yogi', 'Weight = 18 stone', 'Profession = Comic Actor'. The second object O_2 may be identified by its creation time and date '1. Sept. 94 at 00:01:24' in the backup system 'S2' (world W_y). This object has the attributes 'Name = Yogi', 'Profession = Comic Actor' and 'size = 6 feet'. These objects are Aspect Equal in that they both share the properties 'Yogi' and 'Comic Actor' for the attribute classes Name and Profession.

Shallow and Deep Equality in traditional object structures [KHO90] are special cases of aspect equality. They describe events where all known attributes of an object match. For example, Shallow Equal objects with only one value (property R_k) share this value such that '$O_1.R_k = O_2.R_k$'.

All the predicates described so far are only concerned with objects in the conceptual level of information-sharing environments. However, they do not allow for expressing sameness predicates of objects based on common real world counterparts. Real world counterparts can be specified in the sharing environment, for example:

- Section 5.2.1.4 has outlined an extension to generalisations between schema objects in different information sources by counterpart relations in the form of 'C_{xy}' [LEW68]. In other words, when x is an object in the world W_x and y an object in another world W_y, then C_{xy} means that x in W_x resembles the object y in world W_y more closely than any other object in W_x. (Semantic Matching - Resource Knowledge).
- Enterprise models [PAN91a] [JAG94] or models of the real world (CARNOT [COL91]) provide reference models of the enterprise (or the real world). Objects that reside in an information source can reference particular object in the reference model (e.g. global knowledge-base CYC [LEN90] (Section 3.3.1)). These generalisations (G) are defined by the equation G (ist(Gϕ) $\Leftrightarrow$ ist($W_x\psi$)) such that object ψ in world W_x is equivalent to the object ϕ in the global context G ([COL91] and Section 3.3.1.1). Objects that are related by these generalisations can be tested for counterpart relations as described in the previous paragraph.

For the scope of this thesis it is defined that real world objects (AO_i) may be equal based on logical equality. In other words, the equations

($AO_1 = AO_3$) and ($AO_2 = AO_3$),

imply that ($AO_1 = AO_2$).

Sameness of conceptual objects, which support a strong, surrogate based notion of identity to real world objects has, for example, been defined by Masunaga [MAS90] as:

1. **Trivial Equal** (te) objects (O_1, O_2) are identical (O_1.identifier = O_2.identifier) and have a common real world counterpart ((C_{O_1} AO_3) & (C_{O_2} AO_3) ,

   [te(O_1,O_2) =$_{df.}$ (O_1.identifier = O_2.identifier) & ((C_{O_1} AO_3) & (C_{O_2} AO_3))].

2. **Referential Equality** (re) means that "there may exist two objects O_1 and O_2 in class C which refer to the same real world object although they are not trivial-equal" [[MAS90]p.189]. Class C is the superclass of the minimal common attributes (R_1, R_2, to R_k) of the referential-equal objects (O_1 and O_2),

$[re(O_1,O_2) =_{df.} (O_1.R_k = O_2.R_k] \& ((C_{O_1}\ AO_3) \& (C_{O_2}\ AO_3))]$.

3. **Arbitrary Equality** (are) covers any form of equality other then trivial or referential equality. It covers all cases where conceptual objects have distinct real world counterparts but these real world counterparts have some common characteristic, predicate, function etc.

   "For example, suppose ... two submarines with the same type. In such a case one might want to regard the two submarines equal because of the same type. Suppose that these two submarines are registered as objects s1 and s2 in class Submarine. Notice that since the two submarines are different objects in the real world, s1 and s2 may be neither trivial-equal nor referential-equal" [[MAS90]p.191].

$$[are(O_1,O_2) =_{df.} (AO_1.R_k = AO_2.R_k) \& ((C_{O_1}\ AO_1) \& (C_{O_2}\ AO_2)) \& \text{not } (TE(O_1, O_2) \text{ or } RE(O_1, O_2))]$$

where R_k is an attribute, any characteristic, type, or function of a real world object AO_i.

These three classes of sameness based on real world counterparts by Masunaga [MAS90] are now extended by the heterogeneous notions of identity of the conceptual objects using the novel identifier [Identifier_Object, Identifier_Class, W_x,].

1. TRIVIAL IDENTICAL PREDICATE:

A. Trivial Purely Identical (Trivial Equal in [MAS90])

$TPI(O_1,O_2) =_{df.}$
$(O_1.[\text{Identifier\_Object, Identifier\_Class, } W_x] = O_2.[\text{Identifier\_Object, Identifier\_Class, } W_x]) \&$
$((C_{O_1}\ AO_3) \& (C_{O_2}\ AO_3))$

B. Trivial Derived Identical

$TDI(O_1,O_2) =_{df.}$
$(O_1.[\text{Identifier\_Object, Identifier\_Class,}] = O_2.[\text{Identifier\_Object, Identifier\_Class,}])$
$\& ((C_{O_1}\ AO_3) \& (C_{O_2}\ AO_3))$

2. REFERENTIAL EQUALITY:

$RE(O_1,O_2) =_{df.} ((C_{O_1}\ AO_3) \& (C_{O_2}\ AO_3)) \&$
$\text{not } (TDI(O_1, O_2) \text{ or } TPI(O_1, O_2))$

Which means that Referential Equal objects can be :

- Match Equal objects with a common real world counterpart;
- Aspect Equal objects with a common real world counterpart;
- Not have any equality between the conceptual objects other than having a common real world counterpart.

3. ARBITRARY EQUALITY:

$[ARE(O_1,O_2) =_{df.} (AO_1.R_k = AO_2.R_k) \& ((C_{O_1}\ AO_2) \& (C_{O_1}\ AO_2)) \&$
$\text{not } (TPI(O_1, O_2), TDI(O_1, O_2) \text{ or } RE(O_1, O_2))]$
where R_k is an attribute, any characteristic, type, or function of a real world object AO_i.

Trivial Equality in [MAS90] is the equality of objects that have identical identifiers in the conceptual level of an information system (Identical Predicate), and additionally have a common counterpart in the real world. In other words, these objects resemble the same real world object and, additionally, these objects are identical in the same Naming World. This case is described as Trivial Purely Identical (TPI), because the objects are Purely Identical (PI) and also Trivial Equal as defined in [MAS90].

In information-sharing environments, Derived Identical objects are from different information sources (Naming Worlds) which have the same notions of identity. Such objects have identical Identifier_Classes and Identifier_Objects. If such objects have a common real world counterpart then they are Trivial Derived Identical (TDI). They are Trivial Equal according to [MAS90] but the conceptual objects sameness is only Derived Identical because they are from different Naming Worlds.

In general, a common real world counterpart is less important for conceptual objects that are Purely or Derived Identical (PI or DI). But in such cases the degree of sameness is already very strong. However, objects that are Referential Equal (RE) have a common real world counterpart but are not purely or derived identical. The conceptual objects

themselves can be Match Equal (ME), Aspect Equal (ASE), or have no conceptual equalities, but they need to refer to the same real world objects.

Objects that are not Trivial Equal or Referential Equal have no common real world counterpart. However, objects can refer to different real world counterparts but these counterparts may have common instances, share any attributes or are equal in any other aspect. This relation is called Arbitrary Equal.

In enterprise integration environments real world relationship of objects may not necessarily resemble the way this relationship is implemented in the various information systems. In other words, it is possible that Arbitrary Equal objects are conceptually:

> Purely or Derived Identical, Match Equal, Aspect Equal or not equal in any aspect.

Typically, however, objects are implemented in the various information sources according to their real world appearance, and, therefore, Arbitrary Equal objects are conceptually implemented as Match, or Aspect Equal.

Three remarks for classifying the sameness between candidates are necessary:

1. Two objects O_1 and O_2 in world W_1 cannot both be Derived Identical or Trivial Derived Identical to another object O_3 in a different world W_2 without that O_1 and O_2 are Purely or Trivial Purely Identical themselves. In other words it has to hold true that:

   If (O_1 [Identifier_Object, Identifier_Class, W_X] = O_3 [Identifier_Object, Identifier_Class, W_Y])

   and (O_2 [Identifier_Object, Identifier_Class, W_X] = O_3 [Identifier_Object, Identifier_Class, W_Y])

   Then (O_1 [Identifier_Object, Identifier_Class, W_X] = O_2 [Identifier_Object, Identifier_Class, W_X])

2. Derived Identical and Trivial Derived Identical objects cannot be identified by, for example, surrogate, or address based identifiers. In other words, derived identical objects O_1 and O_2 are related such that :

   (O_1.[Identifier_Object, Identifier_Class] = O_2.[Identifier_Object, Identifier_Class])

If both Identifier_Classes are surrogate identifiers based on alphanumeric signs and both Identifier_Objects have the value 'Surrogate = 1234' then these could formally be Derived Identical. However, this is not semantically correct when these objects are from different Naming Worlds (different surrogate management systems may, in principle, use the same surrogate for distinct objects). It is, therefore, necessary to specify which Identifier_Classes can be used to determine Derived Identical objects such as value based identifiers.

3. Some information systems may be able to identify candidates from their sources as Purely Identical internally but are unable to provide the information agent with the evidence (e.g. Identifier_Objects) to enable the agent itself to determine the degree of sameness. For example, an object-oriented database system may be unable to provide its surrogate identifiers (e.g. POET database Version 2). Objects from this source have to be identified in the sharing environment based on value matching, e.g., the employee Name attribute, to identify a candidate concerned with employees. However, if two candidates are from this same database then the object-oriented database management system may provide the information agent with the information that the candidates have the same surrogate identifiers. Because they are from the same database and database management system they share the same Identifier_Class, and are from the same world W_X. In these specific cases, it is pragmatic for an information agent to infer that the objects are Purely Identical, based on the statement from the database management system without investigating the actual surrogates (Identifier_Objects).

5.2.4.4 Classification of Sameness of Candidates

In this section the previously designed notion of sameness for enterprise integration environments will be bound into the conflict detection process. In other words, the conflict detection mechanism would first investigate if the potential conflict between the candidates, represented in a propositional calculus, is based on a correspondence assumption. In such a case the candidates are assumed to be concerned with the same object. Hence, it is important to initially investigate whether the object correspondence can be established.

Investigating the Identifier_Classes and Identifier_Objects may reveal that the investigated objects actually have the same Identifier_Classes but distinct Identifier_Objects:

$(O_1.[\text{Identifier\_Object, Identifier\_Class, } W_X], O_2.[\text{Identifier\_Object, Identifier\_Class, } W_X])$

$O_1.\text{ Identifier\_Class} = O_2.\text{Identifier\_Class}$

and $O_1.\text{Identifier\_Object} \neq O_2.\text{Identifier\_Object}.$

or, in case of candidates from different Naming Worlds:

$(O_1.[\text{Identifier\_Object, Identifier\_Class, } W_X], O_2.[\text{Identifier\_Object, Identifier\_Class, } W_Y])$

$O_1.\text{ Identifier\_Class} \equiv O_2.\text{Identifier\_Class}$

and $O_1.\text{Identifier\_Object} \neq O_2.\text{Identifier\_Object}.$

In these cases it can be positively detected that the objects have no correspondence and are not concerned with the 'same object'. For example, the objects 'Peter' from source (and Naming World) D1, and 'Peter' from source (and Naming World) D2 may both have the same Identifier_Class '*User Defined Key* from a Relational Database'. However, their Identifier_Objects for the 'User defined Keys' are 'Employee.Number = 123' and 'Employee.Number = 1245'. Therefore, the sameness assumptions for the candidates O_1 and O_2 can positively not hold.

In any other case it is necessary to make further investigation of the sameness between candidates. In order to classify the candidate's sameness into the different categories of sameness outlined in the previous Section 5.2.4.3, it is necessary to establish if the objects have any known real world counterparts. As mentioned above, the information agent may, for example, have information of counterpart relations in its Agent Knowledge (Semantic Matching - Resource Knowledge). This includes that candidates

may have common real world counterparts in an enterprise model (or a common knowledge-base such as CYC [LEN90]). For example, if the object 'Peter' from source D1 and 'Peter' from source D2 both have a common counterpart called 'Peter F' in an enterprise model then it may be inferred that these have a common real world counterpart.

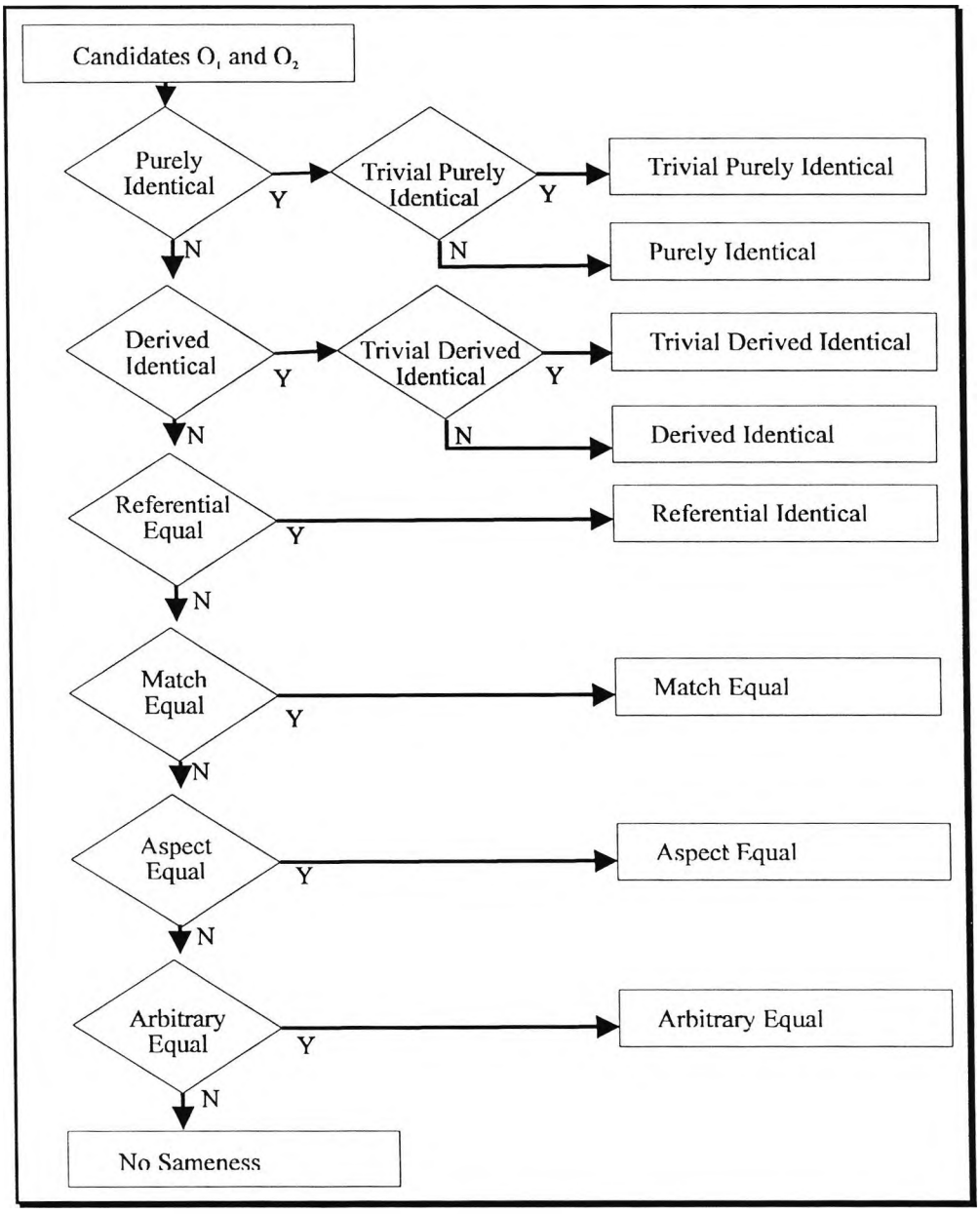

Figure 7: Classification of Object Sameness

Furthermore, investigating the semantic matching information may, in principle, result in finding that two candidates are defined as 'not counterparts'. Most existing research does not account for these exemptions to generalisations. However, in Section 5.2.1 it has been shown that, in principle, it is advisable to implement Comments in the Agent Knowledge (Resource Knowledge) or define exemptions in the form $\sim C_{xy}$, or $\sim C_{xy}$

(Counterpart theory Section 5.2.1.4). Candidates that are defined to be distinct objects cannot conflict if their conflict is based on a counterpart assumption.

The classification includes a systematic comparison of the candidates to the different notions of sameness starting with the 'strongest degree of sameness' (conceptual and then real world to conceptual notions). Candidates may have multiple kinds of sameness. For example, two Purely Identical candidates typically share most or all their attributes for shared attribute classes. These are then not only Purely Identical but also Aspect Equal. However, Aspect Equality is practically of no importance for object sameness in cases where a stronger predicate (Purely Identical) has been identified for these objects.

The 'strongest degree of sameness' between two candidates is, hence, identified by the chart in Figure 7. Thus, if two candidates from the same world share their Identifier_Class and Identifier_Object then they are Purely Identical (PI). If they additionally have a common real world counterpart, then they are Trivial Purely Identical (TPI). However, if the candidates are from different worlds but share the Identifier_Class and Identifier_Object then they are Derived Identical (DI), unless the candidates also have a common real world counterpart and, hence, are Trivial Derived Identical (TDI).

After the mechanism has checked for the identical predicates, a possible common real world counterpart of the candidates indicates that they are Referential Equal (RE). Match Equal (ME) candidates share at least one attribute that is also the Identifier_Object of one of the candidates. Very loosely corresponding objects only share a common attribute and are, hence, Aspect Equal (ASE). If the candidates share no attributes but only their real world counterparts share at least one property then these are Arbitrary Equal (ARE). In any other case there is no sameness between the objects.

5.2.5 Summary Gathering Phase

The Gathering phase collects all the candidates and their evidence from external sources. This information is brought into a uniform, precise novel representation that is an extension of existing object-oriented structures as commonly used in enterprise integration. Further, the mechanism detects if multiple candidates exist that can conflict. These form one or multiple pairs of conflicting candidates. These pairs are the subject of investigation in the succeeding phases. The next step is to investigate if the pairs of candidates fit any kind of a propositional conflict as outlined in Section 3.2.1. The degree of sameness is then elaborated for candidates that are assumed to be concerned with the same object. Figure 8 provides an overview on the gathering phase. The following Section will check if the candidates are syntactically and semantically correct, admissible and, thus, conflicting. This includes a judgement by the agent whether the degrees of sameness between candidates is sufficiently strong to conclude that these are conflicting.

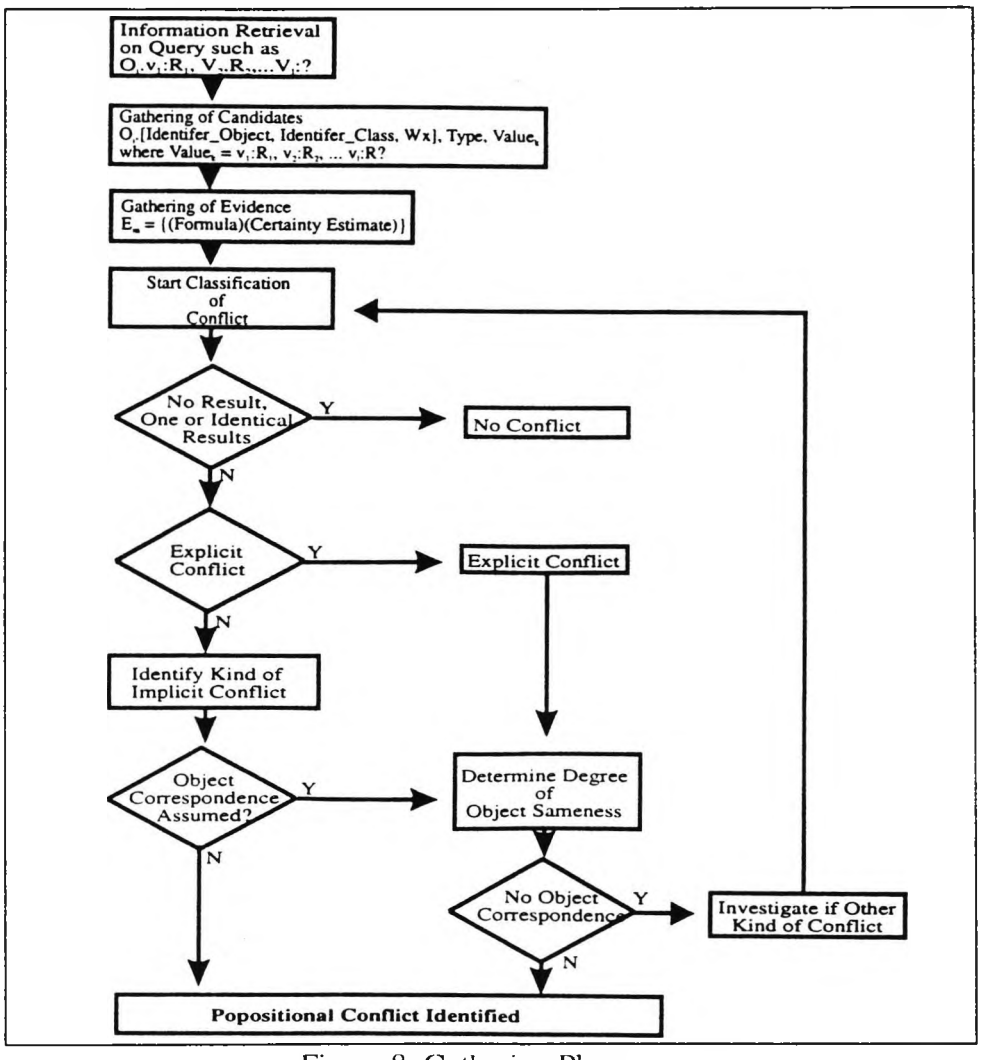

Figure 8: Gathering Phase

5.3 Conflict Detection - Syntactic Phase

Section 5.2 described the Gathering phase which starts with collecting the candidates and their evidence. The Classification stage identifies whether a conflict exists and if so, which kind. However, in order to ensure that a conflict has been detected the information agent needs to ensure that the results are syntactically correct. In other words, in this phase, conflicts are detected based on syntactic errors, e.g. due to inadequate integration of the sources in the enterprise integration environment (Section 4.3). Examples of this syntactic conflict detection include:

1. Explicit error messages may be evoked on the inter-agent, and agent-to-source communication, e.g., shown for KQML in [CHA92]. KQML includes messages about any failures of the communication process, e.g. messages may be incomplete in the face of TCP/IP or network problems.

2. Errors may, in principle, occur within the source of origin, e.g., a relational database management system, which may include error messages in its results. For example, the request from a relational database may be interrupted by a memory or disk problem at the source of origin. It may follow that the result is incomplete. An error message may, however, be attached to the result by the database management system.

3. Inter-agent and agent-to-source communication may fail on the language level. For example, all agents may use KIF [GIN91], LOOM [MAC91], or a basic propositional calculus introduced in Section 3.2.1 and Section 5.2. Within this interaction level agents may encounter errors while analysing the messages from other agents.

The previous paragraphs have briefly outlined the extent to which information agents in existing integration environments can detect syntactic conflicts. Limitations of these mechanisms and ways to improve them include:

- Syntactic conflict detection requires that the agents apply system, and / or 'language specific' information to investigate the physical integration and inter-language translation. This is 'specific' because it requires that the agent understands the error messages, or has knowledge of procedures to assess the correctness of the communication process. For example, most research on enterprise integration environments uses the KQML communication protocol

[CHA92]. In these cases, the agent requires knowledge of the KQML protocol and error messages. Hence, to improve syntactic conflict detection the information agent needs more details on the transmission and translation protocols.

- Existing languages and protocols in enterprise integration such as the KQML protocol include only the modes:
  - 'Result message' for complete results;
  - 'Success-reply messages' or simply no result which may indicate a failure of the result transmission.

  However, further message types which allow a result transmission and, in addition, indications of possible transmission errors could be used by the conflict management scheme. For example, if a result had been transmitted a number of times by the sending agent before it went through to the managing agent then the sending agent could attach a message of caution. In the case of conflict, the managing agent could request the same result again to ensure it has been transmitted correctly. This kind of message is currently not used in existing integration systems but could be very useful to the described information agents in syntactic conflict detection.

- Traditional enterprise integration environments (e.g. described in Section 3.3 [COL91] [PAN91a] [FOX93] [JAG94]) assume that information gathering is never inadequate or incomplete. This assumption is, however, not realistic in large sharing environments with autonomous, heterogeneous information sources that operate concurrently (Section 3.3.6). It follows that existing research has not placed much emphasis on analysing the completeness of information gathering or the correctness of, for example, result translation. Future research could be directed at improving ways in which information agents could evaluate their information gathering processes (on the inter-agent level). Furthermore, information systems might be better equipped with ways to detect if their communicated information has been transmitted correctly (on the agent to source level, and within the information source). Modelling the integrators management of translation errors would be a way to develop such schemes.

5.4 Conflict Detection - Semantic Phase

> "Semantic heterogeneity occurs when there is a disagreement about the meaning, interpretation, or intended use of the same or related data" [[SHE90]p. 186].

Semantic conflict detection is concerned with:

- Object Correspondence and
- Concept Correspondence.

In other words, the candidates are syntactically correct and their correspondence is defined in the Classification phase. The semantic phase, however, investigates the information agents' questions: 'Are we really talking about the same thing?', and 'Do you really mean that?'

5.4.1 Object Correspondence and Strengthening Sameness

In the Gathering phase a novel object identifier for enterprise integration environments has been outlined (Section 5.2.2). It is an extension of object structures typically used in enterprise integration [KHO90]. Furthermore, six categories of object sameness have been defined and used in the Classification phase including, Identical and Trivial Identical Predicates (Purely and Derived), Match and Aspect Equality, Referential Equality and Arbitrary Equality. These notions of sameness are used to describe object correspondence (Section 5.2.4). Semantic conflict detection follows this procedure by examining whether the degree of sameness is strong enough to assume correspondence of the candidates. That is, the propositional conflict identified in the Classification phase may be based on the assumption that the objects are concerned with the same thing, e.g., this is the case in all explicit conflicts. This assumption requires a strong degree of sameness. However, the terminology for 'strong' and 'weak' degrees of sameness requires clarification and this is now described, followed by the steps to strengthen weak notions of object sameness.

Identical predicates can be of the kinds Purely Identical, Derived Identical, Trivial Purely Identical and Trivial Derived Identical. Object sameness in these cases is very strong and the correspondence assumption can easily be accepted. Thus the information agent is justified in concluding that the candidates are concerned with the same thing. This assumption is in convergence with mainstream research on object identity and sameness including Khoshafian and Copeland [KHO90], Masunaga [MAS90], Maida [MAI91], or Kent [KEN91].

Referential Equality requires that the information agent knows that the two objects have a common real world counterpart. So despite their weak sameness based on the conceptual identifiers in their information sources, there is strong evidence that undermines the correspondence of the candidates. Whether referential equality is considered strong enough to assume object correspondence can be disputed. The decision has to be made by the administrator of the integration environment. Mainstream research in enterprise integration, e.g. MKS [PAN91a], Mind [JAG92] or CARNOT [COL91], uses enterprise models as the only means of defining object correspondence and, hence, accept Referential Equality as a sound basis for object correspondence.

Weaker notions of object sameness include Match Equality, Aspect Equality or Arbitrary Equality. Match Equal candidates have no common real world counterpart, as to the knowledge of the information agent, and only the identifier of one object matches attributes of the other object. Whether Match Equality represents a strong enough degree of sameness to assume object correspondence is a question of policy which must be decided by system administrators. However, it is here suggested that Match Equality is too weak to assume object correspondence. This corresponds with much research on object identity such as the work of Khoshafian and Copeland [KHO90].

A similarly weak notion of sameness exists when two objects share any attributes (Aspect Equality). The weakest form of equality is Arbitrary Equality because only the real world counterparts share some attributes. These notions of sameness are generally not assumed to establish object correspondence [MAS90].

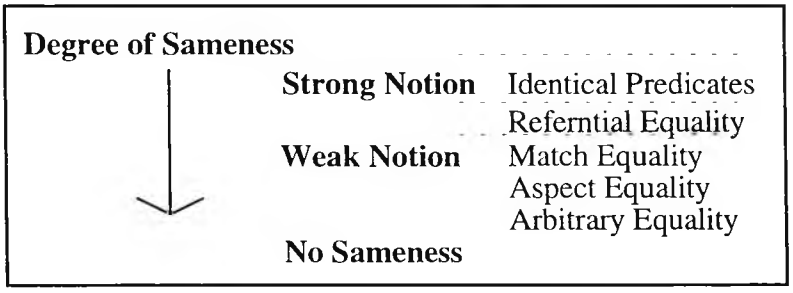

Figure 9: Ranking of Object Sameness

Thus, these weaker notions of sameness (see Figure 9) are not sufficient to support a sameness assumption and require further investigation. Hence, the information agent tries to strengthen these sameness assumptions. In practice, the agent strengthens the sameness of two object O_1 and O_2 by justifying the hypothesis: 'Candidates O_1 and O_2 are concerned with the same object O_i'. This investigation includes the examination of the following three questions:

- 'Which properties are shared by the candidates?'
- 'Are any essential characteristics (Properties) shared?'
- 'Is O_1 in world W_x the closest resemblance of O_2 in its world W_y and vice versa?'

Shared Properties

An information agent initially gathers all properties for any shared attribute classes of the candidates. In other words, two objects O_1 and O_2 may have a number of attribute classes of which they share a subset v_1,v_2, v_m. If the properties (R_k) of these attribute classes (v_1:R_1, v_2:R_2,... v_m:R_k) are identical for both candidates then they share all properties of common attribute classes.

Essential Characteristics

In the next step the agent investigates if the candidates share any essential characteristics. A very brief excursion into essential characteristics will be presented in order to demonstrate the difficulties of applying essentialistic models to information systems:

> In principle, objects can have essential characteristics and accidental characteristic. "Let it be supposed that an individual x answers to any of a number of different 'basic' or 'canonical' identifying descriptions of it" [[RES75]p.23]. These descriptions are essential characteristics. Accordingly, any other properties of an object are accidental. However, the use of essential properties is very complex. For example, no ultimate construction of essential properties is possible for many objects that change in the light of different view points. For example, the Employee Name may be an essential characteristic on the shop floor but only the Employee Number may be essential in the book-keeping department. Furthermore, properties may be essential only in the absence or presence of other characteristics. For example, the Employee Name may be an essential character only in the absence of the characteristic Employee Number.

In enterprise integration pragmatic solutions are typically found to define essential characteristics of concepts. For example, an enterprise model may include definitions of the essential properties of most concepts. In the same way, an information agent may have domain-specific information (Environmental Information - Resource Knowledge) that defines essential characteristics of schema objects. For example, it may define that the schema object 'Employee' has the essential characteristic 'Employee_Number'.

Closest Resemblance

Counterpart relations have been identified between some candidates during the classification phase (Section 5.2.4.3). However, many candidates lack such explicit counterpart relations, for example, in the enterprise model or the Agent Knowledge. At this stage the information agent should investigate the closest resemblance for those candidates that are not already related by a counterpart relation.

Counterpart relations were defined in Section 5.2.4.3 such that a counterpart "resembles you more closely than do the other things in their worlds" [[LEW68]p.114]. Hence, the information agent should investigate if the candidates resemble each other more closely than any other objects in their worlds:

- It identifies for each candidate any attributes that are also part of the candidates Identifier_Object (which means that these identifiers are value based). For example, the attribute class 'Employee_Name' with the property 'Peter' may be part of the object O_1's Identifier_Object for the Identifier_Class 'User Defined Key in a Relational Database'.
- The essential characteristics of the candidates are identified as outlined in the previous subsection. For example, an object O_1 may be of concept employee which has the essential characteristics 'Employee_Number' and 'Employee_Name' in the enterprise model. These characteristics are attributes of the object O_i with the values 'Peter' and '123' for the classes Employee_Name and Employee_Number.

For two objects O_1 from W_x, and O_2 from W_y, the information agent requests from:

- W_y any object that matches O_1's essential attributes, and O_1's attribute from any value based identifier.
- W_x any object that matches O_2's essential attributes, and O_2's attribute from any value based identifier.

For example, the agent requests an object where the attribute class 'Employee_Name' is 'Peter' and the attribute class 'Employee_Number' is '123' from the sources of the potentially conflicting candidates.

In the case of n conflicting results that have no counterpart relations and have a weak notion of sameness, '(n-1)*n' objects are requested from the sources. For example, three conflicting results based on weak notions of identity exist and twelve ((4-1)*4 = 12) requests are necessary to determine if the objects are counterparts of each other.

This investigation will, in general, provide one of the following results for every request:

1. The candidate's closest resemblance in the other object's world is the potentially conflicting candidate (Closest Resemblance);
2. Another object resembles the candidate more closely than the supposedly conflicting candidate;
3. No object matches the request.

At this point the information agent has information on the candidates' shared properties, common essential characteristics, and whether they are each other's closest resemblance. It will now be outlined how this information may be used to enrich the current classification based on counterparts and degrees of sameness, based on the following characterisation of counterparts:

> "Your counterparts resemble you closely in content and context in *important respects*. They resemble you more closely than do the other things in their worlds. But they are not really you" [[LEW68]p.114] (and Section 5.2.1.4)

The question '**Are we really talking about the same thing**' may now also be answered for the candidates with weak notions of sameness (ME, ASE, ARE) based on the analysis undertaken above. In other words, the agent has investigated if the candidates are each others' 'closest resemblance in their sources'. This resemblance is based on *important respects* (see Lewis' definition above) such as the object's identifier, or essential characteristics. The agent may be able to identify one of the following:

1. Object correspondence can be assumed if the candidates resemble each other closest, and (a) they share all common properties, or (b) they have the same property for at least one shared essential characteristic, and no shared essential characteristics are conflicting. This may lead the agent to conclude that the candidates are counterparts and, hence, at least Referential Equal.
2. No object correspondence has to be assumed in cases where candidates are not each others 'closest resemblance' and do not share at least one essential property and / or all properties for all common attribute classes.
3. No object correspondence exists when the candidates differ in at least one essential characteristic and are not each others 'closest resemblance'.

In any other case the weak object correspondence cannot be strengthened

In the first case, further conflict detection and possibly resolution is required. There are no doubts about the object correspondence. In the second and third case, however, the conflicting candidate can be rejected based on a lack of object correspondence. The

retrieval process then has to determine which of the candidates is relevant to the information request.

In any other case, however, object correspondence cannot be assumed nor definitely be rejected. The information agent could, for example, do one of the following:

1. It could take the candidates, with such a weak notion of object correspondence, into account and continue resolving the conflict. Thus, as far as the conflict detection is concerned it has to detect a conflict that is based on a weak notion of object correspondence. This weak correspondence is part of the evaluation of the conflict case. It should, therefore, be enclosed in the solution statement for clients of the sharing environment that can make use of this complex information (e.g. human, domain expert decision makers).
2. Alternatively, the enterprise modelling process could include definitions on the minimum correspondence of candidates from specific sources. For example, a Business Rule (Organisational Knowledge) could define that candidates from relational databases and object-oriented databases have to share at least one essential characteristic to justify at least a weak sameness assumption (i.e. Correspondence).
3. Conflicts based on object correspondence require that the candidates' degree of sameness is at least Referential Equal, or Match Equal and the candidates are each others closest resemblance in their worlds. In all other cases no conflict exists between these objects because of a lack of object correspondence.
4. Conflicts based on object correspondence require that the candidates degree of sameness is at least Referential Equal. In all other cases no conflict exists between these objects because of a lack of object correspondence.

Which strategy is adequate for a given environment must be decided by the system administration. However, the last three approaches would best fit with existing research on object identity and sameness in enterprise integration, e.g. Pan and Tenenbaum [PAN91a] or Khoshafian and Copeland [KHO90]. Option two requires that Business Rules are defined. If these do not exist the third approach is most realistic.

Existing research on enterprise integration environments, e.g. by Fox and Barbuceanu [BAR94a], fails to detect cases where no conflict actually exists because of a lack of correspondence. Furthermore, no existing scheme includes an explicit evaluation of the correspondence assumption so that weak correspondences are made explicit to evaluate the integrated information.

5.4.2 Concept Correspondence

Object correspondence is concerned with the sameness of candidates or data objects. For example, the correspondence of the individual 'Peter' in sources (world) W_1 and the 'Peter' in source (world) W_2. Concept correspondence is concerned with matching the semantic meaning of the concept of discourse. For example, two candidates may be concerned with the same object Peter but one claims that Peter is 'fat' and the other claims that Peter is not 'fat'. Both candidates deal with the concept 'fat'. It is necessary to ensure that both propositions are concerned with the semantically same meaning of this concept, e.g. 'fat', in order to detect that their propositions, including this concept', are conflicting. In addition, the information agent would want to ensure that any concept included in any candidate has been retrieved in a way that its semantic meaning is captured as intended by the source of origin. For example, the concept 'cool' from a music database may be used to describe 'interesting music'. It has, therefore, a semantically different meaning than 'cool' used by an expert chemist to describe the temperature of a product. Integrating both concepts requires that these semantic meanings are interpreted correctly in order to evaluate a possible conflict between candidates using these concepts.

Concept correspondence plays various roles in the different kinds of conflicts that can occur in enterprise integration environments (Section 3.2.1).

1. Concept correspondence of the conflicting attribute (e.g. 'fat')is assumed when candidates conflict explicitly as described in the previous paragraph in the example 'Peter is fat' and 'Peter is not fat'.
2. Implicit conflicts in the same attribute class (v_m) assume that the concepts (e.g. 'fat' and 'slim') of the conflicting attributes are semantically exclusive (e.g. 'Peter is fat' and 'Peter is slim').
3. All implicit conflicts (including those in different attribute classes) require that the semantic meaning of the concepts used in the conflicting attributes (e.g. 'Peter is fat' and 'Peter has blue shoes') correspond to the concepts in the heuristic(s) defining the implicit conflict. For example, the heuristic 'It can only be true that 'Peter has blue shoes' or that 'Peter is fat' applies to candidates that use the semantically same concepts of 'fat' and 'blue shoes'.

Enterprise integration environments typically ensure concept correspondence of the first two kinds. The latter kind of concept correspondence is less typical and is closely related to enterprise modelling (e.g. how to define integrity heuristics, or rules).

The problem of semantic matching has been defined as (Section 2.7):

> "Data obtained from remote and autonomous data sources often will not match in terms of name, scope, granularity of abstractions, temporal units and domain definitions"[[WIE92]p.39].

Concept correspondence is traditionally a problem of mapping the semantic differences between autonomous information systems [PAP92a] [HEI85]. For example, the concept 'car' may be mapped onto the concept 'automobile' in another source. Such mapping is, for example, provided by dependency schemata in multidatabases [AHM91] or 'knowledge transformators' (also called Mediators by Wiedehold [WIE91] Section 3.3.4). The latter are software components that mediate information between sources, or between information sources and human users [WIE91]. However, this hardwired mapping has become a rather syntactic task of translating 'car' into 'automobile' and is falsely called 'semantic matching'. Lenat, for example, argues that it is a "syntax mirrors semantics" [[LEN90]p.26] approach. Mutation and exportation by dependency tables does not investigate the semantic meaning of the candidates but simply 'translates' names of concepts.

Hence, the translation of concepts is part of adequate information retrieval. In the Agent Knowledge the required matching information is defined as Resource Knowledge - Semantic Matching in Section 2.7. Section 5.3 on syntactic conflict detection was concerned with checking the translation including the translation of concepts (also called matching from a syntactic point of view).

In this section the information agent attempts to evaluate the actual, semantic meaning of the candidates and their concepts. For example, an information agent may investigate whether a result from a user interface is 'really' what was meant by simply returning the result for **verification**, e.g. by the source of origin. A typical example is the verification by human users, e.g. implemented in Stolze and Gutknecht [STO91] (Section 3.4.3). In KQML [CHA92] (Section 2.3), for example, such queries are of the type 'Assign-Truth-Value' which means that the recipient can answer this query only with 'True' or 'Not True'.

Furthermore, potential conflicts may arise over concepts that are assumed to be corresponding but actually mean different things. As a form of local problem-solving **expert knowledge** may be used to semantically evaluate concepts. For example, two medical expert systems may produce a result such as 'Peter has fever' and 'Peter has high

temperature' which semantically means the same thing. In other words, the concepts do not conflict but are subtype supertype related in the form:

$$O_i.v_m{:}R_k \subset O_i.v_m{:}R_k.$$

The results 'Peter has fever' and 'Peter has high temperature', from the previous example, may require domain expertise (e.g. by a doctor) to interpret their semantic relation. It may be the case that fever is a form of high temperature so that the concepts are not semantically conflicting but the patient's fever includes high temperature.

Expert Knowledge may potentially be available from a human expert, or a software system. The latter provide the information agent with problem-solving capabilities or Services (Resource Knowledge Section 2.7). Another source of Expert Knowledge is an agent's Environmental Knowledge (Resource Knowledge) or an enterprise model (Section 3.3.1 CARNOT [COL91], MKS [PAN91a], TOVE [FOX93]). Both can potentially be used to interpret concepts so that their correspondence is evaluated. For example, the candidate 'Peter has fever' uses the concept 'fever'. This concept may have other synonyms and may be a supertype or subtype of other expressions as outlined in the previous paragraph. An enterprise model may outline these relations between objects and / or have information on synonyms. CARNOT [COL91], for example, has an automatic procedure for matching concepts. It is based on finding 'the best match' of a concept in the tree-like enterprise model called 'subgraph matching'. "Subgraph matching [is based] on simple string matching between names or synonyms" [[COL91]p.60]. As outlined in Section 3.3.1.1 the highest common subdomain of a concept and its counterpart in the model is its best match. Different concepts from multiple, possibly conflicting candidates can be matched to concepts in the enterprise model. The concepts may be related in the enterprise model as:

1. Synonyms;
2. Subtype or supertype concepts;
3. Different concepts so that the candidates are not conflicting.

However, difficulties with the approach have been reported [COL91] in integrating queries of all types and formats. In addition, no enterprise model exists that incorporates all common and domain knowledge so that there are always synonyms and matches that are not identified. Thus, concept matching based on enterprise models or agents' Environmental Information may be incomplete. Despite this incompleteness, a rational scheme would have to apply all available strategies, including relevant enterprise models and Services.

Finally, the semantic meaning of results may be particularly dependent on the **form and time** it is requested. A good example are different versions of an object, e.g. stored in an object-oriented database. The agent may know about different versions of objects from its Identifier_Class or from its Environmental Information - Resource Knowledge. Different versions of an object may produce different results that are not conflicting but only correspond to different versions of the same object.

Furthermore, information sources may not be able to produce useful snapshot results but need to be monitored over time. In Section 3.4.3 the dynamic aspect of knowledge in rule-based information sources has been mentioned. Results from such dynamic sources may be inconsistent because they are not derived in the semantically correct form.

For example, "special processing is required for parallel paths [parallel results from one or multiple sources], which can derive alternative values for the same data item" [[SU 91]p.236]. Schemata may describe generalised rules or relations between objects as described in Section 2.7 on Schema Knowledge. This is one way in which an information agent may determine whether parallel processes in one or multiple information sources may be based on a constant exchange of data between these processes. In other words, this dynamic interaction may have to be monitored over time in order to obtain a result that is 'relevant'. For example, an agent may request multiple results from such a source over a longer period of time and determine if the results have a trend, or if they converge after multiple iterations [SU 91].

Another source to detect whether 'special processing' is necessary to ensure the retrieval in the semantically correct form and time is the agent's Environmental Information (Resource Knowledge). The agent can ensure that this processing is (or has been) applied to the retrieval of information from the sources with the potentially conflicting results. It may follow that no conflict exists but only a retrieval in a semantically incorrect form or time.

In summary, it has been shown that concept correspondence of potentially conflicting candidates can be evaluated by:

1. Result Verification;
2. Employing Expert Knowledge to semantically related concepts (subtype, supertype relations or synonyms);

3. Investigating if the form and time in which the concept has been requested is semantically correct.

These are representative ways in which concept correspondence can be investigated in current enterprise integration environments. **Limitations** of these approaches and possible future improvements are:

- Result verification is typically based on verification by human experts or a human user that has manually inserted data into a system. Ideally, other information systems (e.g. expert systems) could also 'verify' their results in a similar way to human users. This would be particularly desirable for complex systems that have dynamically changing data such as production management or control systems. A pragmatic implementation of verification could be a repeat of the original query and a comparison of the new and the original results.
- Much research exists on semantic matching from distributed databases (e.g. 'ISA-relations' [HUL87]) and enterprise integration (e.g. CARNOT [COL91]). However, subtype / supertype relations and synonym information is based on extensive domain knowledge. For example, the CARNOT approach described above depends heavily on the manually implemented common knowledge-base CYC [LEN90] (Section 3.3.1.1). Future research should be directed at further automating the definitions and maintenance of these bases (e.g. learning). In addition, common knowledge-bases could be used in more than one system. This would be one approach to reusing the once defined common knowledge in many applications and systems.
- It is easy to perceive an approach that investigates the form and time of the retrieved information with the help of heuristics defined by database administrators or integrators. However, the limitation of this approach is the manual definition of these heuristics. Some research aims to automate this task such as an approach by Su and Park [SU 91]. As described above and in Section 3.4.3, the approach outlines the dynamic exchange of information between sources which leads to a cyclic result exchange between these sources. Because of this interaction, information requests from such sources may have to be of a specific form or be repeated multiple times. Information agents could potentially use this mechanism and automatically analyse schemata for cyclic relations. Where information is retrieved from schema objects that have cyclic relations the agent could repeat the information request to ensure the result is retrieved correctly.

5.5 Conflict Detection - Admissibility Phase

Admissibility of a candidate is based on the rules of the information agent to consider a candidate (Section 4.4). An agent may have heuristics that let it reject a result (candidate) without further investigating its semantic content. These heuristics may be in the form of:

1. Business Rules and Decision-Making Knowledge;
2. Comments, e.g. from System Administrators;
3. Integrity Constraints.

Business Rules and Decision-Making Knowledge are Organisational Knowledge (Section 2.7) that an information agent has as part of its 'handbook' information. Some Organisational Knowledge may induce an information agent to reject any information from a particular source, concerning a particular domain, of a specific format, etc. For example, the Business Rule 'Any Information From the Database D1 is not admissible' will let the agent reject any result from this source. Such a heuristic may, for example, be installed if the source produces unpredictable results after having had a disk crash.

Comments from administrators (Designers or Integrators) are Resource Knowledge (Section 2.7) that describe an information source, critique its reliability, make statements concerning particular results, etc. This may include comments that on principle discharge particular information just like Business Rules.

Integrity constraints are related to schema objects and, thus, are classified as Schema Knowledge in Section 2.7. For example, the constraint 'Prices must be larger than or equal to zero may be attached to the attribute Price of a Product object called 'Beer'. This integrity constraint may direct an information agent to reject a result if it does not fulfil this basic requirement. For example, the result 'Beer has the Price £-5' may be inadmissible because of the constraint 'Prices must be larger than or equal to zero'.

The use of integrity constraints for deciding on admissibility is a question of policy. Comments or Business Rules and Decision-Making Knowledge are explicit rules on the condition of admissibility. Integrity constraints, however, are not specific conditions of admissibility. They are a condition for the integrity of results. Schema information in enterprise integration environments carries some risk of being inconsistent with the data it describes in autonomous information sources (Section 2.4). It may, hence, be assumed that integrity constraints are potentially at risk of not being correct for all data items they

should constrain at all times. In the previous example, the price of beer may be '£ -1' in a particular situation, such that beer is tested and the volunteering people are given a one pound reward per pint. This aspect may not have been considered when the information agents' integrity constraint was defined. Furthermore, if the agents' constraints are derived from local schemata then it is possible that the autonomy of the local sources will permit them to change their local schemata without instantly informing all information agents.

In conclusion, the use of integrity constraints is limited in distributed systems in that extra caution has to be taken when applying locally defined constraints (i.e. within one information system) in a global context (i.e. in the integration environment). However, within these limitations integrity constraints are successfully used in enterprise integration environments (e.g., [PAP92a]) or database schemata [THO90].

In legal systems the concept of admissibility has been based on rules that may not necessarily be logical or generally accepted [MUR86] (Section 4.2). It is also a question of information management policy which rules in enterprise integration environments determine if results are possible or worth considering. Conservative data policies may not permit integrity constraints as a reason to reject a result *per se*, but pragmatic implementations may do.

5.6 Summary Syntactic, Semantic and Admissibility Phases

The following Figure 10 provides a chart of the phases Syntactic Conflict Detection, Semantic Conflict Detection and Admissibility. These follow the Gathering of the candidates, their evidence and the propositional classification of the possible conflict. The admissibility phase is the final step to ensure that a genuine, data conflict (Section 3.2.2) has been detected that requires conflict resolution.

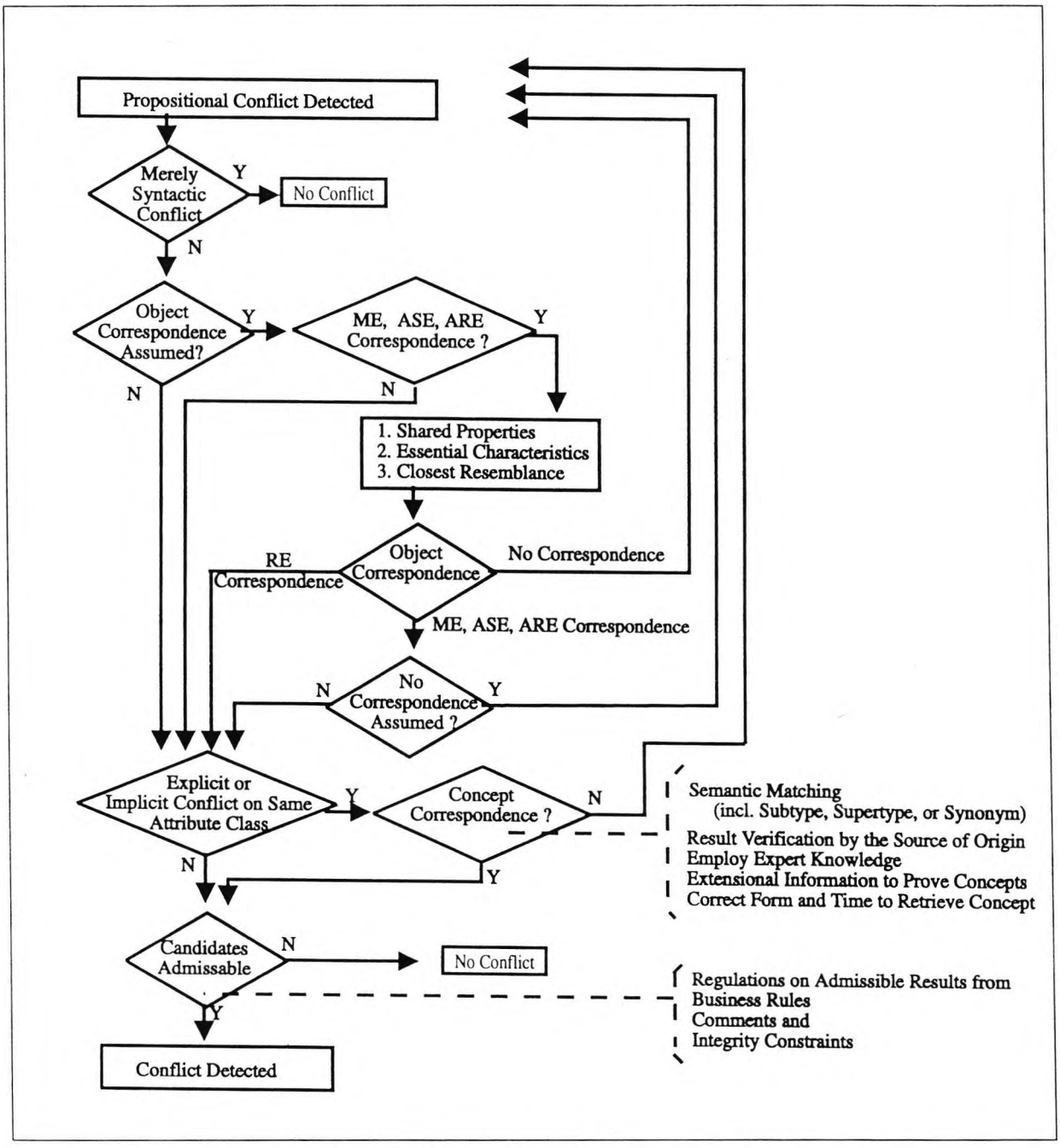

Figure 10: Syntactic, Semantic and Admissibility Phases

5.7 Conflict Resolution - Credibility

5.7.1 Gathering Credibility Estimates

In the conflict detection phases described above either no conflict, or a genuine, data conflict among multiple candidates has been identified. The candidates are gathered in a uniform, propositional structure. In addition, evidence that warrants or refutes the correctness of the candidates may be related to them. Evidence has been formally described in Section 5.2.3 in the form of:

$E_m = \{(\text{Formula})(\text{Certainty Estimation})\}$.

The Formula variable consists of zero-to-multiple interconnected propositions that constitute evidence concerning a candidate. This evidence may include an estimate of the certainty of the candidate (in the Certainty Estimate variable), which is based on any propositional evidence in the Formula variable. Furthermore, the formula variable may be empty and the evidence may become a pure certainty estimate, which is included in the Certainty Estimation variable.

Section 5.2.3 has further outlined that the credibility of a source of origin is based on its reliability. A reliability estimate can be related to the whole source, a specific object, a schema object, Groups of objects or sources (Resource Knowledge in Section 2.7). Finally, it has been shown that it is rational for the information agent to gather only the evidence that is automatically provided by an information source at the point of the initial request. In other words, the agent did not collect all evidence, including all available certainty estimates, in the gathering phase. This complete collection of all information that can be utilised by the information agent to provide certainty estimates on the candidate is undertaken in this Credibility phase.

In principle, the credibility can be estimated on the grounds of:

1. Business Rules and Decision-Making Knowledge (Organisational Knowledge) and Comments (Resource Knowledge) that **directly** determine the credibility of sources, or candidates.
2. Business Rules and Decision-Making Knowledge (Organisational Knowledge) and Comments (Resource Knowledge) that **derive** the credibility of sources, or candidates based on Environmental Information (Resource Knowledge).
3. **Services** that estimate the reliability of a specific candidate (Resource Knowledge).

Direct Credibility Estimation

All evidence that has been gathered in the conflict detection phases that includes a certainty estimate, is of the kind 'direct'. In other words, it directly evaluates the credibility of the candidate to which it has been attached. Examples of direct certainty estimates include, 'Database D1 is very reliable', or 'Data from Source D2 concerning Schema Object Employee Name is, based on past experience, very unreliable'.

Furthermore, an agents Organisational Knowledge and Comments may not only be heuristics on the Admissibility of candidates (previous Section 5.5) but they may also directly specify the credibility of sources, or candidates. Comments may, for example, incorporate an agent's own experience with integrating a source. In this case, the agent may have developed a statistical reliability on results from a specific source.

For example, Sadreddini, Bell and McClean [SAD90] have described ways to implement statistical analysis into distributed systems. Following these architectural considerations, the information agent could have a 'Global Statistical Module' (GSM) with which it undertakes statistical computations and a 'Global Statistical Database' (GSDB) to store precomputed statistics. The information agent could monitor how often 'local results', which it has retrieved from its local source, vary from the 'integrated result'. The 'integrated result' is based on multiple, possibly inconsistent, results being retrieved and integrated by the DCEEI including conflict detection and resolution. The GSM could determine how often the 'local result' has varied from the 'integrated result' and store this reliability estimate in the GSDB. This information in the GSDB would be a 'comment' in the terminology of the Agent Knowledge in Section 2.7 (Resource Knowledge). An example of such a comment could be:

> 'Results on the schema object 'Employee.Name' from the local source BookkeepingDB have been identical to the integrated result in 90 out of 100 cases of a global request in the DCEEI'.

Derived Credibility Estimation

Derived reliability is more complex. In the field of enterprise integration the following examples of how an information agent can derive reliability estimates based on Environmental Information have been introduced in Section 2.7 (Resource Knowledge):

- Fox and Barbuceanu [BAR94a] [BAR94c] (Section 3.3.1.4) establish the authority of an information source by its ability to accomplish a particular goal. In other words, a partial order exists that ranks information sources according to their importance (roles) in accomplishing a specific goal. This authority rating is directly related to a source's reliability. For example, a system that supervises the production may have a higher authority than a system in the sales department in, for example, determining the daily output. This means that the first system has more credibility than the latter in providing the daily output figure.
- Another way to specify the reliability of the source of origin is called 'appropriateness' [PAN91a] (Section 3.3.1.2). In this model context information is used to determine whether the source is qualified to produce a weighty result. The role that a source plays in a company may be used to determine its appropriateness to produce an accurate result. Appropriateness in this context is equivalent to reliability. For example, 'Mary, the equipment operator on duty' may be an assessment that the source 'Mary' is very adequate to provide weighty results concerning equipment operations.

A scheme is required to apply known Environmental Information in the Agent Knowledge (Section 2.7), or in the enterprise model Section 3.3.3.1 CARNOT [HUH93][COL91], MKS [PAN91a], TOVE [FOX93]) to be used by Organisational Knowledge and Comments so that these can derive reliability estimates. A pragmatic approach that implements the existing strategies to evaluate credibility estimates, is the following scheme:

1. The agent analyses its Environmental Information (Resource Knowledge) to identify if the candidates are part of any **Groups**. Example Groups are: Sources are jointly working on the shop floor, or in the finance department; All Employee schema objects are of the Group 'employee' (Please see Section 2.7 - Resource Knowledge on Groups).
2. Environmental Information or Comments may describe the **characteristics** such as expertise, roles and authority of the candidate's and their sources, or Groups, for example: Sources D1 is an expert on finance data; The expert system E2 has a high authority on production management data; or The software system S2 has the role 'Calculating the Demand'. Characteristics can, in principle, be of various kinds but of primary importance in the research described in Sections 2.7 and 3.3, are the roles, expertise and authority of Groups and sources [BAR94a] [PAN91a].

3. The **reliability** of the candidates is derived by Organisational Knowledge and Comments based on the characteristics of the candidates themselves, their sources, or Groups they belong to. For examples, 'all new sources are very reliable' may be a Business Rule that allows the agent to derive that because the database D1 is new it is also 'very reliable'. Another example includes a system E3 that is an expert scheduling staff. The rule 'expertise in scheduling makes a system very credible' lets the agent determine (i.e. derive) that E3 is 'very credible'.

Formally the derived and direct reliability, e.g. 'very reliable', is specified in the variable Certainty Estimate. The basis of this certainty estimate is specified in the variable Formula. The heuristics that lead to this estimate are enclosed in the Formula variable, e.g. 'Expertise of the system E3 is scheduling staff'. Both variables can be attached as evidence of the form 'E_m = {(Formula)(Certainty)}' to the candidates (as described in Section 5.2.3).

Credibility Estimations by Services

A third way to obtain certainty estimates are **Services**. These have been described in Section 2.7 such that they are provided by the information sources to evaluate their results. For example, a statistical package may produce a confidence level for its results. Section 5.2.3 has described how Services can provide evidence automatically, or on request from an information source. Services include any certainty estimation that is provided by the local source. Examples are numerical Bayesian probabilities, possibilistic and fuzzy certainties, or any qualitative certainty estimates. Some methods of describing certainties in information systems are discussed in Section 3.5 and example Services are mentioned, e.g., in Section 2.7 on Resource Knowledge. These are attached as evidence 'E_m ={(Formula)(Certainty)}' to the candidates they describe.

5.7.2 Limitations of Credibility Estimates

A focal problem with certainty estimates in large, heterogeneous enterprise integration environments is the lack of a common standard - a set of uniformly defined estimates - throughout the environment (Section 3.5.2). Each source may have its own kind of certainty estimates (e.g. Bayesian probability estimates, possibilistic estimates, etc.). In addition, multiple sources may interpret the same estimates differently (e.g. 'very credible' may have different semantic meanings in multiple sources). Methods of combining certainty estimations have been described in Section 3.5 (e.g. Dempster-Shafer [DEM67] [SHA76]). However, these methods can only be used by information agents in enterprise integration to combine numerical credibility estimates if:

- The estimates are from the same source;
- Estimates from different sources are jointly defined. For example, two autonomous sources may be known to use certainty estimates (e.g. probability estimates) in semantically the same way (seventy per cent in source A is equal to seventy percent in source B).

Another way to overcome the problem would be domain knowledge. For example, if the information agent would be a domain expert and could relate the reliability estimates from multiple sources. An, Bell and Hughes [AN 93] have described a method for evidential reasoning called RES that uses the relative strength of evidence (e.g. 'evidence A is more believable than evidence B') rather than absolute measures (e.g. 'A is ninety per cent reliable and B is seventy per cent reliable') (Section 3.5.2). However, information agents typically lack the domain expertise necessary for this kind of evidential reasoning (Section 2.3). They only integrate data from information sources including human and machine experts. For example, an information agent can integrate medical advice systems (Section 3.5.3.4) but it is not a medical expert itself. Hence, it could not identify the relative strength of the medical evidence. The next Section 5.8 will investigate ways in which the information agent can employ existing domain experts, which may apply evidential reasoning schemes, to relate the (credibility) estimates of conflicting candidates.

It may be concluded that information agents require specific heuristics to enable them to rank credibility estimates from multiple, heterogeneous sources. These ranking heuristics have been used in existing integration environments and are further described in this research in Section 5.10.

Another question is whether it is desirable that the information agent, which is not a domain expert, makes a judgement at this point of the conflict management process only based on credibility estimates.

Following the framework designed in Chapter 4, the resolution step Credibility only assesses and stratifies the reliability of the conflicting candidates. This is a first step towards conflict resolution because it provides the most fundamental information on the weight of the conflicting candidates. Section 4.2 outlined that in legal systems

> 'A lack of credibility may make evidence less persuasive, and may make it less convincing then other, possibly contradicting, evidence from a more credible source.'

This principle is also incorporated by the rational conflict resolution scheme introduced in Section 4.5. Hence, no decision is made solely on the basis of the credibility of the candidates but a further evaluation of the evidence, provided in the following Sections, is necessary to rationally attempt to resolve the conflict. The following of this section will briefly present the main points of this strategy as outlined in Section 4.5.1.

For example, a database may be estimated 'not very reliable' by an information agent based on its previous experience. However, this result should not be rejected only because of a low certainty estimation. Reasons to adopt this approach have been discussed in Section 4.5. These include that

(a) certainty estimations are not compatible throughout the environment, and

(b) there may be good reasons to look again at the candidates and to make a judgement on the basis of all available evidence.

An example of the latter is that a source may improve its accuracy over time. In other words, the source may have been recently overhauled and the existing certainty estimate (e.g. the source is 'Not very Reliable') based on previous experience with this source, may have become a poor basis for a judgement. Furthermore, good reasons may exist that warrant a candidate which, from another point of view, is 'very unreliable'. For example, an old database may be judged as 'unreliable' based on a derived certainty estimate on 'old databases'. However, this source may be useful on historic (old) data.

In the following steps conflict resolution takes any evidence that warrants, or refutes a result, into account. This includes the credibility of the candidates. It would, however, not be Principle Rational (systematic or logical) to reject a result as 'unreliable', without taking any available, possibly good, evidence for believing the result into account (Sections 4.5.1 and 2.6). The following Sections will describe this systematic resolution.

5.8 Conflict Resolution - Domain-Specific Problem-Solving

In Section 4.5.1 the strategy to solve conflicts on the most domain-specific level, has been outlined. In the following Section 4.5.2 this has been sketched out for the most domain level resolution, called Domain-Specific Problem-Solving. In summary, the following domain-specific problem-solving strategies have been identified:

1. Domain Expert Resolution;
2. Problem-Solving Communities.

Domain Expert Resolution

Existing DCEEI have integrated human experts into the sharing environment (e.g. Pan and Tennenbaum [PAN91a]). Specific conflict management by human experts in problem-solving systems has been described in Section 3.4.3, including Stolze and Gutknecht [STO91] or Steiner *et al.* [STE90]. Single system domain-specific resolution is concerned with **expert resolution** systems. A system may exist in the enterprise that resolves particular conflicts, they are experts on solving special conflicts. For example, expert resolution strategies by a medical advice system are typically domain-specific such as Cohen's *et al.* [COH87] knowledge-based consultant, the application of Argumentation to medical advice systems [CLA90a] (Section 3.5.3.4), or domain-specific planning systems as described in Section 3.4.4 (e.g. [KLE91]). These domain experts may resolve specific conflicts over correct domain level alternatives such as a diagnosis of a patient's illness.

A domain expert system can internally use any form of problem solving strategy or knowledge representation. For example, the problem solving system may use a truth maintenance system and / or belief revision strategies (Section 3.5.3.1. and 3.5.3.2.), it may be argumentation-based (Section 3.5.3.4), use any quantitative mechanism (e.g. Bayesian probability (Section 3.5.2)), or evidential reasoning such as RES by An, Bell and Hughes [AN 93] (Section 5.7.2. and 3.5.2).

Currently only a few examples of domain-specific problem-solving systems exist of which some have been mentioned in the previous paragraphs. In enterprise integration these experts are typically human [PAN91a]. However, future research and the practical implementation of expert advice systems in the organisations, may produce more machine experts that provide problem solving capabilities [BRO89]. Thus, the sharing environment should allow the flexible integration of autonomous, possibly pre-existing

domain expert systems as they become available to information agents [HEW91] [BRO89]. This section will sketch such an implementation.

Problem-Solving Communities.

Section 4.5.2 revealed that problem-solving communities, which build closed groups of cooperating 'agents' (Please note the difference between information agents and these local, domain-specific problem-solving agents Section 3.4.3), can participate in conflict resolution in the following ways:

1. One or multiple local agents may produce results that are shared via an information agent on the enterprise integration level. In such a case, multiple local agents from the same community may produce conflicting results. The information agent should then request a conclusive, non conflicting result from the community as a whole. This would imply that the community has to resolve any inconsistencies it has internally and, hence, resolve the conflict. In some cases the community may resolve inconsistencies over time automatically and the information agent need only repeat its request after that resolution has taken place.
2. Local agents may be able to resolve conflicts assigned to them by an information agent, that may not necessarily be based on results originating from this community. In such a case, agents jointly provide a resolution, e.g. by negotiating the conflict. For the information agent this kind of community problem-solving is identical to domain expert resolution systems (e.g., a medical advice system) because only the problem solving strategy in the community or in a single problem-solving system is different. From the point of view of the information agent both systems provide an equally valuable result to the conflict.

Architecturally the implementation of the described problem-solving capabilities into the information agents might be considered. However, this is not desirable because:

- Information agents typically integrate information sources, but they themselves lack domain expertise which would be necessary for domain level conflict resolution (Sections 2.7.2. and 2.3). An example of this domain expertise is a medical advice system which typically requires extensive knowledge from the specific medical domain in addition to problem solving strategies.
- The domain experts can be very heterogeneous, e.g., they may internally use any from of problem solving strategy or knowledge representation.

- A domain expert, that is a modular system, can provide domain-specific problem-solving capabilities to multiple information agents.
- Domain experts can be integrated into the integration environment as they become available by current research and development. New expert systems or modifications to existing systems can be made without having to architecturally redesign the information agent.

It can be concluded that conflict management is proposed by this research to be part of the functionality of information agents. Existing research in DCEEI only integrates human or machine experts as information sources [PAN91a]. In other words, the agent can request specific information from these sources. However, the previous paragraph list reasons why 'problem solvers' should architecturally be integrated in the same way. How this integration can be implemented, in principle, has been described for problem solving systems [COH87] [CLA90a] [KLE91]. Following these approaches the next paragraphs briefly outline ways to integrate problem-solving experts to support conflict management in enterprise integration.

One key design question is: **How could the information agent identify that domain-specific resolution procedures exist which are relevant to a specific conflict? Two ways are described here:**

1. An agent's Environmental Information (Resource Knowledge) with explicit descriptions of domain-specific problem-solving capabilities;
2. Schema Knowledge on data structures may enable the agent to derive cooperative interaction among local, domain agents in a cooperative problem-solving community.

Expert conflict resolution capabilities that are provided by any source to assist the information agent on domain-specific conflicts may be known to the information agent in its Environmental Information. For example, a distributed planning system as described in Section 3.4.4 may be a community of domain-specific planning experts (agents). The Environmental Information on problem-solving capabilities may have the heuristic 'Conflicts on Production Planning may be resolved by the Planning Agent Community'. The query 'What is the Current Production of the Product BigMac' may result in the conflicting candidates '100 units' and '500 units'. The agent has to know that queries on the 'Current Production' are concerned with the domain 'Production' (Environmental Information - Resource Knowledge in Section 2.7, or Enterprise Models such as MKS by Pan and Tenenbaum (Section 3.3.1.2)). It can then send the original query to the local

planning experts. If the result matches with any of the candidates then this candidate is correct and the case is resolved. If a different result other than the existing candidates is returned then a new conflict detection and resolution process has to be started with all three candidates. In any other case no local resolution is possible.

The previous paragraphs have briefly outlined ways in which information agents can employee domain experts to domain level conflicts resolution. **The limitations that have to be addressed in further improving this integration include:**

- It is much less complex to present all information on the conflict to a human expert than to machine experts. Existing research on DCEEI has, hence, typically integrated human experts, e.g. by presenting information on the expert's terminal as demonstrated in the enterprise integration system by Pan and Tenenbaum [PAN91a]. The previous paragraphs have outlined a way to integrate 'human experts' and 'machine experts' not only as 'information sources' but as 'problem solvers'. Furthermore, this research has sketched out ways to integrate problem-solving systems into the conflict management scheme. Only a few examples of automated problem-solving systems exist (e.g. medical advice in Section 3.5.3.4). Future developments of expert advice systems can be made available to information agents in the architecture and conflict management scheme presented here. In other words, the development of domain expert systems will automate information integration in the presented architecture (e.g., Brodie [BRO89] or Hewitt and Inman [HEW91]). As described in this section this will include the conflict management scheme's domain-specific problem solving.
- Furthermore, the evaluation of this resolution step in the next chapter will show that it is difficult to identify which conflicts a domain expert can resolve without that the information agent has to become an expert of the field itself. For example, the domain 'medicine' is very general but multiple medical expert advisory systems require the agent to be much more specific on the medical area of the conflict. It might be necessary to provide the information agent with more sophisticated methods to identify domain experts than presented in this section. For example, it might be necessary to introduce mediators (e.g. similar to envoys [PAL92] or mediators [WIE92] as described in Section 3.3.4) that are specialists on identifying the appropriate advice system for a give conflict.
- Another difficulty is that the information agent, once it has identified an expert, needs very detailed knowledge of the format in which this expert can handle the conflict. In summary, current enterprise integration and modelling only facilitates an information agent in very specific cases to use domain level expertise for

conflict resolution. Hence, more standardisation of interfaces between system would greatly improve the practicality of this approach. For example, the implementation in Chapter 6 is based on the use of global schema objects and uniform C++ interfaces to local sources. Furthermore, communication protocols such as KQML [CHA92] provide uniform access to heterogeneous systems.

Section 6.5.9 presents an implementation and case study that integrate an expert advice system into a DCEEI such that it can provide domain-specific problem-solving.

In case any relevant domain-specific problem-solving can be identified by the information agent then this conflict resolution can produce :

- Multiple new results that are sent through the detection phase again in order to evaluate if these are still conflicting;
- Only one result is returned by the local problem solver(s) and, hence, the conflict is resolved;
- No solution can be found and the resolution has to be continued into the next phase.

In case not relevant domain-specific heuristics exist, or these can not resolve the conflict the agent continues with the next resolution step: Scientific, Domain-Specific Heuristics.

5.9 Conflict Resolution - Scientific, Domain-Specific Heuristics

Domain-specific resolution strategies, described in the previous section, resolve a conflict by the most problem specific resolution strategy. Often, however, no such tailored resolution strategies exist for a given conflict. For example, candidates from two worlds (sources) may be in a conflict for which no resolution expert on the domain level exists. Moreover, these cases include the typical conflict in enterprise integration environments called 'essential data conflicts' as described in Section 4.5.2. In other words, a data conflict has arisen that is due to incorrect, incomplete or obsolete information, and not based just on a question of choosing between mutually acceptable solutions.

In contrast to Domain-Specific Problem-Solving this next step applies general resolution strategies to the domain-specific candidates and their evidence. Section 4.5.3 described 'scientific' judgements of the form mathematical - logical, empirical or metaphysical [TRU87].

The CYCCESS approach [GUH94] is a good example of how general heuristics (also called basic common sense) can be used to evaluate information, e.g. candidates. It is an application that uses the CYC common knowledge-base [LEN90], that is, facts and scientific rules to check results for their correctness:

> "Data in the Structured Information Source [e.g. a database or a spreadsheet] can be checked for consistency with basic common sense [which is stored in the CYC knowledge-base]. For example, it is not reasonable for a person to be employed before they were born, or for an automobile to be in two countries at the same time" [[GUH94]p.140].

The approach is, however, limited to the completeness of the concepts defined in the global knowledge-base CYC.

An architecturally different way to make resolution heuristics available to information agents is to adopt the CYCCESS approach for the design of information agents. In other words, heuristics can be defined in the Agent Knowledge to evaluate if candidates are consistent with the agent's general heuristics (its common basic knowledge in CYC terminology). This may evaluate that candidates are incorrect. A conflict can be resolved if one out of two conflicting candidates is incorrect and the other is correct.

The principle limitations of the described and similar approaches are:

- Candidates and concepts throughout the enterprise typically lack a common structure (Section 2.3). Hence, the CYCCESS approach is currently only applied to databases or spreadsheets [GUH94].
- The definition of 'general heuristics' for the information agent is a very complex task. The CYC knowledge-base, if it is as complete as described in the literature [GUH94], would provide this 'absolute knowledge'. A pragmatic application of this approach to information agents is the definition of Scientific Heuristics (Organisational Knowledge). These must, however, be very carefully chosen to fit the absolute applicability to all facts that may be derived throughout the environment. In other words, these heuristics would have to be individually accepted as a 'company policy' (Principle Rational Section 2.6). Examples used in Section 4.5.3 include:
    - Arithmetic calculations ('2 + 2 = 4');
    - Empirical facts such as 'Yogi weights 20 stone', or 'The world is round';
    - Metaphysical information such as 'Space is indefinitely extended'.
- In either case of a common knowledge-base such as CYC or scientific heuristics these definitions would have to be implemented and maintained by human experts. Research in enterprise integration is currently limited by heavily relying on human experts or administrators.

In order to improve the way scientific heuristics are specified a systematic definition and maintenance of the scientific heuristics, such as described by Lenat for the CYC knowledge base [LEN90], could provide guidelines for

(a) defining and testing heuristics,

(b) a common set of heuristics that can be used in multiple integration environments.

Future research should be directed at making the definition and maintenance of scientific heuristics less dependent on human experts. Examples could be (semi-) automated agent learning and reuse of scientific heuristics (Section 5.7.2). Learning could include that the agent constantly, systematic and autonomously interrogates human experts to improve its scientific heuristics. Reusing of the CYC knowledge-base is, for example, targeted by the CARNOT approach [COL91].

The burden of distributing all heuristics among all agents could be reduced by defining only a specific, incomplete subset for each agent. A potential way to allow for

incomplete sets of 'scientific heuristics' is to let the agents share their scientific heuristics. In other words, each agent could issue an inquiry for scientific heuristics relevant to the schema object addressed in the query at issue. For example, the query 'What is the Employee 'Peter's' FirstName' might be concerned with the global schema object 'Employee.FirstName'. The other agents could then contribute relevant heuristics which could be used by the initiating agent to resolve its conflict. It is beyond the scope of this research to further outline these concepts. However, they propose ways to improve the current dependence on human experts.

In conclusion, conflict resolution by information agents is typically not complete in respect to resolving all conflicts that are solvable by scientific reasoning. However, the scheme presented here provides a framework within which the agent can apply all the scientific reasoning available to it. This and the last section have demonstrated how an information agent can apply reasoning with domain-dependent and general heuristics in information-sharing environments.

It becomes obvious that, in the light of the weak approaches to apply general scientific heuristics to conflict resolution, the information agent should make its reasoning as explicit as possible to the clients of the integrated information. For example, it should present the rules that have led to a resolution to any client that can make use of such complex results, e.g., a human decision maker. Such an 'assessment' has also been proposed by Clark *et al.* for results from Argumentation [CLA90a] (Section 3.5.3.4).

5.10 Conflict Resolution - Domain-Independent Evaluation - Reliability

The first resolution step employed local experts and communities. The previous step has applied the domain-dependent information, which is candidates, their evidence and relevant environmental information, to any general, scientific heuristics the information agent is justified to apply. Section 4.5.1 determined that any further resolution has to be domain-independent. Information available for this evaluation is typically the credibility or reliability of the information sources and their candidates. A scheme to resolve a conflict based on the candidate's reliability has been outlined in Section 4.5.4 that includes the steps:

1. **Ranking** the Candidates;
2. Finding **New Alternatives**;
3. **Negotiating** a Compromise.

5.10.1 Ranking

In Sections 5.7 and 5.2.3 the reliability, or the certainty of the candidates (and their evidence) has been identified. The certainty measures may have been taken into account in the previous two resolution steps. For example, the phase Scientific, Domain-Specific Heuristics may use certainty estimations attached to candidates to judge conflicting candidates. However, this final phase of the resolution is concerned with resolving conflicts based solely on certainty assessments.

In cases where **no certainty estimates** exist, no ranking is possible. In any other case a ranking, at least in principle, is possible. In practice, however, most heuristics require that all candidates have certainty estimates.

A general problem in enterprise integration is that certainty estimations from autonomous sources are often not comparable (as outlined in Section 5.7). In other words, there is no rational basis to assume *per se* that certainty estimations such as 'possible', 'certain', '75 per cent', 'very reliable', are used in the same way, based on the same concepts in any source throughout the sharing environment. Furthermore, different kinds of certainty estimates are typically incomparable. For example, a certainty based on high authority and one based on certainty factors provided by a statistical package (Services) are typically not comparable as such.

In addition, certainty estimates may be directed at overlapping but different things, such as a specific source, a schema object, Groups of objects or sources (Section 2.7), or only specific candidates. For example, some estimates are evaluating a whole source, e.g. 'The Database is very reliable'. Another estimate may be concerned with a particular result, e.g. an expert system may rate the reliability of a specific result in a particular situation.

Ideally, all certainty estimates from all sources would be from a uniform set of estimates (Section 5.7.2). In such a case ranking and evidential reasoning would be easily possible by methods such as the Dempster-Shafer rule of combing probabilities [DEM67] (Section 3.5.2), semi-quatitative approaches such as presented by [AN 93] (Sections 3.5.2 and 5.7.2), or the Argumentation-based scheme by [FOX92b] (Section 3.5.3.4). The latter, for example, is based on a pre-defined set of ranked certainty estimates. Judgement on conflicting evidence is possible by evaluating the order of the certainty estimates of the conflicting evidence.

Existing research in DCEEI cannot assume such a set of homogeneous, ordered certainty estimates. Enterprise integration environments integrate autonomous, pre-existing information sources with heterogeneous certainty estimates (Section 2.3, 3.3.1 and 3.3.2). It follows that typically in a sharing environment:

- Existing uncertainty management schemes are not assumed possible in all cases where certainty estimates support conflicting candidates from heterogeneous information sources;
- Ranking schemes are introduced, e.g. by [BAR94a] and in this research, that may on principle incorporate the well known uncertainty management methods (Section 5.7.2), but allow for only partial ranking of some certainty estimates.

The remainder of this section describes a typical way to rank conflicting candidates in enterprise integration under the assumption of heterogeneous certainty estimates.

In principle, two steps are required to solve conflicts between candidates based on their certainty estimates:

1. Ranking the certainty estimates; and
2. Judgement based on this ranking.

Hence, heuristics may exist that rank the certainty estimates and also include a judgement based on these certainty estimates (Ranking and Judgement Heuristics - Organisational Knowledge), for example:

'If one candidate is 'very reliable' and another is 'very unreliable',
then judge in favour of the former and reject the latter candidate'.

Alternatively, Ranking Heuristics (Organisational Knowledge) may only rank certainty estimates. Judgement Heuristics (Organisational Knowledge) are then required that use this ranking to derive a judgement of the conflicting candidates. For example, the Ranking Heuristic:

"Certain' is a higher reliability estimate than '20 per cent confidence"

may exist. Based on this ranking a Judgement Heuristic may specify:

'If one certainty estimate 'has a higher reliability' than any other,
then always judge in favour of the former

Typically, however, these heuristics are more specific. They may also identify that the judgement heuristic is only valid for specific sources, or specific domains. Thus, ranking requires that the agents know how to compare which estimates, from which sources. This can be done in three ways in enterprise integration environments:

A. Specific Certainty Estimations

Ideally, the information agent has heuristics that rank specific certainty estimations from particular sources with each other. These heuristics are typically Business Rules or Decision-Making Knowledge (Organisational Knowledge Section 2.7). Such an example heuristic may be:

If the result A from source D1 is 'certain' and result B from source D2 is 'not certain',

then judge in favour of result A.

However, this kind of ranking certainties is very unrealistic in large, open information-sharing environments simply because the number of sources and their number of uncertainty estimations is vast. Furthermore, it is typically impossible for an information agent to assess all these estimates in a complete fashion from autonomous sources (which may change the estimates they use over time).

B. Related Certainty Estimates

A more general way to rank and relate estimates is on the level of 'estimates from particular sources'. Thus, two or more sources may have been developed under the same

modelling premises and have related certainty estimates. For example, two expert systems may have been developed by the same modelling process and may be known to use comparable, reliable certainty estimations. In addition, the agent needs to know the range and order of the estimates. Such information would also be Decision-Making Knowledge or Business Rules that are related to a whole source. For example, the agent may know that two sources both have semantically related estimates, which both rank in the order: 'Certain, Possible, Impossible and Uncertainty'.

Furthermore, the certainty estimates to assess credibility described in Section 5.7 include those provided by local sources, other information agents, or the assessing information agent itself. Certainty estimates that originate from the assessing agent itself are typically based on the same modelling premises and thus of the kind 'related estimates'.

Related certainty estimates can, in principle, be based on uncertainty management methods. For example, propositional estimates from sources with related certainty estimates can be combined by the Dempster-Shafer rule [SHA76] [DEM67] (Section 3.5.2). In other words, this rule can be included in a ranking heuristic for such probabilistic certainty estimates. However, as outlined in Section 5.7.2 more complex reasoning schemes typically require domain-specific information. For example, complex reasoning methods such as RES [AN 93] and other qualitative reasoning schemes described in Section 3.5.3, are applicable to domain specific conflict resolution (Section 5.8). They cannot be applied by information agents, which lack domain-expertise, to rank related certainty estimates.

C. General Acceptance of Certainty Estimates

A pragmatic solution to assess certainty considerations in information-sharing is to value the estimates themselves. In other words, the information agent may have further information on the general acceptance of certainty estimations from particular sources. It may have Organisational Knowledge or Comments that validate certainty estimations from a particular source. For example, an agent may know that certainty estimations from a particular source D1 have 'universal assent'. In other words, the certainty estimates are of a generally acceptable kind ('universal' is here the community of information agents, administrators, or the enterprise). The information agent can then apply heuristics such as

> If one results is 'very unreliable' (less than 30 per cent confidence, uncertain, impossible, unrealistic, etc.),

and another is 'very reliable' (more than 90 per cent confidence, highly possible, very likely, strongly supported, etc.),

then judgement can be made in favour of the second result.

Ranking

The information agent would rationally first try to apply heuristics based on specific certainty estimates, then those for related certainty estimates, and finally those for general certainty estimates. This order allows the agent to first apply the most specific heuristic and then to move to a more general one. In any case three outcomes of ranking exist:

- Determinate;
- Indeterminate;
- Incommensurate.

Determinate ranking is possible in cases where two certainty estimates are rankable based on any of the previously described methods. Two certainty estimations are "incommensurate if it is neither true that one is better [more certain] than the other nor true that they are of equal value" [[RAZ86]p.322]. For example, two certainty estimates may exist that cannot be compared by the information agent because no common base for these estimates exists.

> "The ranking of A and B is indeterminate just in case it is reasonable to conclude that A is better than B, that A is worse than B, and that A and B are of equal value" [[SEU92]p.802].

In practice, candidates may have multiple heuristics and ways to rank their certainty estimates which are not consistent, or indeterminate. In other words, indeterminate ranking describes an unsettled situation in which there are aspects to rank the candidates when considering one aspect, and different when considering another.

In summary, reliability ranking can conclude with one of the following:

1. Determinate ranking of the candidates is possible which resolves the conflict.
2. Ranking Heuristics exist but no Judgement Heuristics exist for the ranked estimates.
3. Incommensurate certainty estimates make ranking impossible,
4. Indeterminate ranking of the certainty estimates makes a judgement impossible.

In the last two cases no Ranking Heuristics are available to solve the conflict. It may, however, be possible that new alternatives or certainty estimates can be identified. This evaluation will be described in the next section.

Ranking conflicting candidates is **limited by** the judgement and ranking heuristics that have been defined for the information agent. In other words, Section 5.7.2 outlined that potentially any kind of certainty estimate can exist in open, heterogeneous environments that integrate autonomous (Section 2.4) sources. On principle, there may always be a case when an information agent cannot relate certainty estimates because at least one of the estimates is unknown to the agent. In addition, from an enterprise modelling point of view the relation between certainty estimates in large, complex environments might not be clear to the administrator or integrator at any one time. In such cases it might not be intended to instruct the agent to rank specific certainty estimates. Hence, the limitation by incomplete ranking and judgement heuristics is a realistic and pragmatic assumption for complex enterprise integration environments.

However, a closely related limitation is that the implementation of very specific and precise heuristics by human experts is very complex and laborious. On the other side, very general heuristics make the implementation of a ranking and judgement scheme easier and less dependent on the human implementation. However, the problem with these heuristics is the assumption that very general heuristics are adequate to rank very heterogeneous estimates from autonomous, heterogeneous information sources. Section 5.7 has described the heterogeneity of direct estimates (e.g., based on calculating the statistical reliability) and derived estimates (based on the Role, Expertise and Authority of information sources). Future work could be directed at ways to specify which heuristics are adequate for a specific environment. Furthermore, the definition and maintenance of heuristics could be automated further by modelling how this task is undertaken by human experts.

5.10.2 New Alternatives

The previous step was aimed at ranking the credibility of candidates in order to resolve the conflict. However, in this section New Alternatives are applied to rank candidates that have incommensurate or indeterminate certainty estimates. Alternatives are developed by:

1. Alternative Ranking and / or Judgement Heuristics;
2. Developing a Compromise.

Three kinds of **'Alternative Heuristics'** (Organisational Knowledge) are described, which enable the information agent to make suggestions. They are not Principle Rational, that is they are not proven to be accepted by all potential clients of the integration environment (Sections 4.5 and 2.6). However, the information agent believes that they are rational in this sense (i.e. belief without proof in the form of Ranking and / or Judgement Heuristics by a system administrator). Thus, the format and functionality of these heuristics is identical to the heuristics described in the previous section, but the new Alternative Heuristics represent resolution suggestions.

An information agent may have its own certainty order. These are called Alternative Ranking Heuristics in Section 4.5.4. This ranking is alternative, or proposed because the information agent believes that the heuristics are appropriate but their applicability is not proven, e.g. in the form of rules from a system administrator. However, the alternative ranking may be applied to Judgement Heuristics described in the previous section so that a new alternative solution is developed. In the case where no Judgement Heuristics fit the proposed ranking the agent may have some Alternative Judgement Heuristics. These may also be applicable to any candidates that have been ranked in the last section but lack suitable Judgement Heuristics.

In addition, an agent may have Alternative Ranking and Judgement Heuristics that not only overcome the problem of ranking but provide a judgement on the conflicting candidates based on their reliability.

A different way to finding alternative heuristics is to further investigate the circumstances of the certainty estimates in order to redefine them. In other words, the agent investigates the characteristics of the candidates, which have been identified in the Credibility Phase (Section 5.7). In particular, the agent investigates those certainty estimates that have been derived from roles, expertise, or authority characteristics.

Certainty estimates may be redefined by including circumstantial information on the origin of estimates. In this way a **compromise** is developed similar to Sycara's approach [SYC89] (Section 3.4.3). For example, the following conflicts may have resulted in an incommensurate ranking:

Candidate 1: Yogi lives in Pentonville Rd.,
Evidence: E_1 = {('Result form from Source A' $\wedge$ 'Source A's expertise is student related data')(very reliable)};

Candidate 2: Yogi lives in Rosemary Gds.,
Evidence: E_2 = {('Result is from Source B' $\wedge$ 'Source B is an expert on teaching staff')(very reliable)}.

A New Alternative may be derived such that the candidate from source A is more reliable in respect to student data and the candidate from source B is more reliable on teaching staff. In that case the conflict changes to:

Candidate 1: 'Yogi lives in Pentonville Rd.',
Evidence: E_1 = {('Result from Source A' $\wedge$ 'Source A's expertise is student related data')(very reliable **on student data**')};

Candidate 2: 'Yogi lives in Rosemary Gds.',
Evidence: E_2 = {('Result is from Source B' $\wedge$ 'Source B is an expert on teaching staff')(very reliable **on teaching staff**)}.

This alternative may, or may not be a solution to the conflict. This is decided in the following Negotiation Phase.

5.10.3 Reliability - Negotiation

Negotiation in distributed artificial intelligence (DAI) is typically concerned with deals and compromises among competing agents, e.g. described in Section 3.4.2, 3.4.3 and 3.4.4 for some distributed problem-solving systems. A fundamental difference between enterprise integration and mainstream DAI is that information agents have to evaluate results in respect to their persuasiveness in a Principle Rational way (Section 2.6). Thus the negotiation stage aims to summarise the results. Further, New Alternative Heuristics may have been suggested, or a compromise that allows judgement on conflicting reliability estimates may have been developed. These are presented to persuade the client, typically a decision maker. This persuasion and the suggestion of compromises shares commonalties with negotiation protocols, e.g. by Sycara [SYC89].

Not only decision makers but also application programs potentially request information from the integration environment (Section 2.3 and 2.6). However, compromises can only be meaningful to applications that can incorporate such complex results. The example in the previous Section resulted in a compromise where candidate O_1 is 'very reliable on student' data and candidate O_2 is 'very reliable on teaching staff' data. This compromise may lead a decision maker to believe candidate O_1 ('Yogi lives in Pentonville Rd') if it knows that 'Yogi is a student'. A human decision maker might be able to have this knowledge and, hence, be able to apply this complex result. Closer integration of information systems in future integration environments [BRO89] may produce more systems that can make use of complex results. For example, an expert system might request information from the sharing environment, e.g. on Yogi's home address, and may be able to store not just one value but the following complex result from the integration environment:

Compromise: 'Yogi lives in Pentonville Rd.' if Yogi is a student;
'Yogi lives in Rosemary Gds.' if Yogi is teaching staff.

A further development of this conflict resolution scheme should include adaptation and learning. For example, New Alternatives may be suggested to a decision maker. The decision maker might be able to make suggestions about the resolution of this case which in turn could improve the agent's Alternative Ranking and / or Judgement Heuristics. It could learn how efficient its suggestions are, or it could grade its alternative heuristics in order to use them in a more adequate way (e.g.[VIT91]).

The limitations and future work of this Negotiation phase can be summarized such that:

- The Negotiation phase could be improved by a more detailed scheme for the interaction between information agents and human or machine 'users' of the integrated information. A possible concept of how this negotiation could be established has been outlined by research on distributed problem solving agents (e.g. by Sycara [SYC89])
- Learning should be implemented, e.g., in that New Alternatives are developed based on the negotiation with expert users of the integrated information. A compromise developed by these users could become a New Alternative Heuristic in the Agent Knowledge. Furthermore, the negotiation result could be used by the agent to make comments on the applicability of specific alternative heuristics and estimates.
- Systems that use the integrated information should be more qualified to present feedback to the agent (e.g., what circumstantial information on the conflict management of a specific result can be used).

The presented integration environment uses homogeneous, benevolent information agents. However, future research could be directed at integrating heterogeneous agents (e.g. from different integration environments) that may be less benevolent. These agents would not all exchange all their meta-knowledge. Thus, a conflict resolution scheme that has been developed by one agent could be presented to other agents for a critique. Section 3.4 and Appendix A present a number of schemes in which agents could critique their results. For example, Laärsi *et al.* [LAÄ92] have developed a blackboard-based negotiation protocol including the stages proposal of a result, critique of this result by the other agents and a number of resolution heuristics to resolve a possible negotiation conflict.

5.10.4 No Solution

The negotiation process may propose a compromise that is acceptable to the client of the integration environment such as a decision maker. However, as outlined in Section 4.5.4, resolutions may lie in the scope of the application program, i.e. the only resolution to some conflicts may be to manipulate the results by application specific, e.g., risk management procedures. Other conflicts simply lack a rational solution, e.g. if they are based on preferences. It was, hence, concluded in Section 4.5.4 that a rational scheme for information agents should not resolve all conflicts (Section 2.6).

However, the case has been evaluated in depth. The agent would try to present the conflict and its resolution, or in case not resolution was reached the accumulated information, to sources that can operate with such complex information (e.g. Argumentation applies a similar Assessment as described in Section 3.5.3.4). This includes the candidates, their evidence, their relations, any evidential resolution steps applied to them in the phases Domain-Specific Problem-Solving, Scientific, Domain-Specific Heuristics, or Domain-Independent Evaluation (reliability estimates). This information is typically not informative to any client, but it may be important to domain experts, e.g. specific decision makers.

Section 4.5.4 revealed that 'No Solution' may result from a lack of conflict resolution information, or it may be due to the nature of a conflict that presents an inherent inconsistency. Furthermore, research on conflict management has outlined that the resolution attempt, without a resolution to the conflict, plays a positive role in problem solving (e.g. Galliers [GAL90b]). For example, some conflicts typically lay outside the scope of information agent conflict management and may be due to inconsistencies in the organisation's policy. Such a conflict of policy may be 'High prices are good (more income)' and 'High prices are bad (for staying in competition)'. In this respect identifying the conflict and coming to the conclusion 'No Solution' is a meaningful outcome of serious conflict resolution by information agents.

Furthermore, in the introductory sections of this research (e.g. Section 2.6) and as a conclusion on existing research in Section 3.3, the fundamental basis to produce

meaningful, rational results from Distributed Collaborative Environments for Enterprise Integration is that:

No solution to conflicting candidates is better

than irrationally manipulated,

but

consistent information.

5.11 Summary Conflict Resolution

The previous Sections have described conflict resolution. A general overview is provided in the following Figure 11 in form of a flowchart. Conflict resolution is based on conflict detection, hence, resolution begins with a genuine, data conflict that has been analysed and formalised as described in Figures 8 and 10.

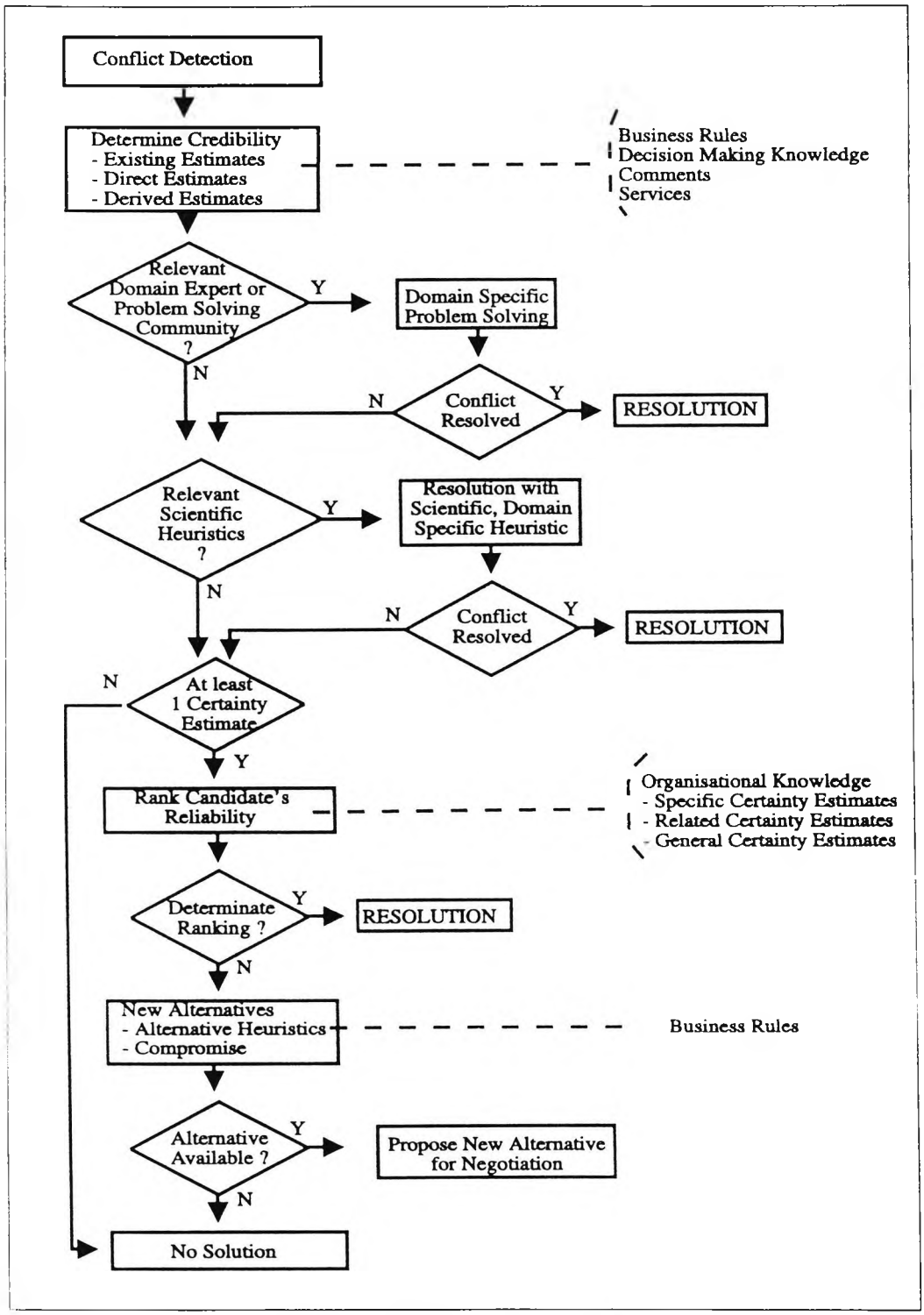

Figure 11: Conflict Resolution

5.12 Implementation Concept

The conflict detection and resolution mechanism described in this Chapter 5 will be implemented in the following chapter to demonstrate the approach in a prototype. The concept of the mechanism has been described in form of flowcharts in the figures:

- Figure 8: Gathering Phase
- Figure 10: Syntactic, Semantic Admissibility Phases
- Figure 11: Conflict Resolution Phases

These have been bound together to provide a single view on the whole implementation concept of the conflict detection and resolution mechanism in Appendix E.

5.13 Conclusion

In Chapter 5 a mechanism for conflict detection and resolution has been outlined based on the framework described in Chapter 4. A formal representation of candidates and their evidence has been designed, which forms the fundamental basis for the mechanism. In particular a detailed conflict detection is possible within this expressive, novel representation.

The conflict detection and resolution mechanism is based on the Agent Knowledge and information available in enterprise integration as outlined in Section 2.7. The following evaluation phase will have to demonstrate how the mechanism operates with the integration environment. Hence, the implementation of a prototype will have to include:

1. The information shared by autonomous information sources within the enterprise integration environment;
2. The Agent Knowledge that incorporates the information agent's information on itself, and the environment;
3. A conflict detection and resolution mechanism based on the implementation concept outlined in Section 5.12.

5.14 Chapter Summary

In Chapter 5 a conflict detection and resolution mechanism for information agents in Distributed Collaborative Environments for Enterprise Integration has been described. The framework from the previous chapter has been applied for this design.

The Gathering of Candidates phase has introduced a novel notion of object identity within the objects structures defined by Khoshafian and Copeland [KHO90]. These are typically used in enterprise integration. The novel identifier is sufficiently expressive to precisely specify sameness of candidates from heterogeneous systems with heterogeneous notions of identity. Each candidate may be supported or refuted by evidence. It is presented in a uniform way in the Gathering of Evidence phase. The Gathering concludes with a Classification Phase, which determines if a conflict is at hand, and classifies it according to the conflict classes outlined in Section 3.2.1. Furthermore, the degree of object sameness is determined in cases when object correspondence is assumed between conflicting candidates.

Syntactic conflict detection identifies if mere syntactic mismatches, e.g. translation and communication problems, have been the cause for a conflict. Semantic conflict detection rehearses the weak objects' correspondence assumption. Conflicts based on the assumption that the candidates are concerned with the same thing (e.g. explicit conflicts of the kind '$O_1.v_m{:}R_k$' and 'not $O_1.v_m{:}R_k$'), cannot be conflicting if the correspondence of the objects cannot be established. A mechanism, called strengthening object sameness, has been designed with which the information agent can analyse the correspondence between the candidates. This analysis may lead it to decide which candidates have a strong enough degree of sameness so that object correspondence can be assumed.

Semantic conflict detection is concerned with ensuring that candidates use concepts with the same semantic meaning (Concept Correspondence). For example, the information agent can ensure that concepts are not simply subtype, supertype relations of the same concept, or Synonyms (Semantic Mismatch - Resource Knowledge Section 2.7).

The Admissibility phase is concerned with investigating whether a candidate on principle can be reliable and rational. In other words, the information agent's Organisational Knowledge, or Comments (Resource Knowledge) e.g. from system administrators, may include rules on the admissibility of specific information sources (or parts of these sources). For example, a database may produce invalid, and, hence, inadmissible results

because it has had a disk crash. Should one out of two conflicting candidates be from an inadmissible source then this result can be neglected, and only one valid result remains (no conflict). After this phase a genuine data conflict has been detected.

Conflict resolution opens with identification of the credibility or reliability of the candidates and their sources of origin (Credibility phase). Domain-Specific Resolution Strategies may be available to resolve domain-specific conflicts for the information agent. For example, a medical expert system may resolve conflicts over some medical issues. Scientific, Domain-Specific Heuristics can be used, as e.g. described in the CYCCESS [GUH94] approach, to apply scientific heuristics to evaluate the candidates (their evidence and any relevant environmental information).

However, often no domain level resolution strategies, or general (scientific) heuristics are available for a given conflict. A conflict resolution scheme solely based on the candidates' reliability estimates is a typical example of a Domain-Independent Evaluation. Such a resolution scheme is outlined in a three step process. First the information agent attempts to rank the candidates' certainty estimates. In case this ranking cannot provide a solution, the agent attempts to develop alternative solutions (New Alternatives). For example, the agent may develop a compromise by diversifying the candidate's certainty estimates (e.g. by specifying the expertise of the source of origin of the estimate). The information agent functions like a 'Persuador' proposing compromises in Sycara's negotiation protocol [SYC89]. However, the information agent proposes the New Alternative as a suggestion to resolve the conflict to the client of the integration environment and not, like Sycara's Persuador, to other agents. This final phase requires that the client is able to operate with such complex results, e.g. a human decision maker.

It may be the case that a conflict is not soluble by Principle Rational (Section 2.6) strategies based on the information agent's knowledge. In other words, the information agent is able to identify that a conflict has no solution under the given circumstances. In this case, a full report on the resolution steps can be presented to clients of the integration environment, if these can evaluate such complex information (e.g. specific decision makers).

6. Evaluation and Discussion

6.1 Introduction and Evaluation Methodology

6.1.1 Introduction

The last chapter introduced a conflict detection and resolution mechanism for Distributed Collaborative Environments for Enterprise Integration (DCEEI). This chapter will now describe the implementation of this mechanism and its evaluation.

The next Section investigates the evaluation methodology (Section 6.1.2). Based on this concept, a Distributed Collaborative Environment for Enterprise Integration has been implemented (Section 6.2). First, a description of the implemented integration environment is provided (Section 6.2.1). This is followed by an example integration environment based on a fictitious university cafeteria (Section 6.2.2).

The integration environment includes multiple autonomous information sources, and a model of an information agent (Section 6.2.3). A demonstrator is built that demonstrates the integration environment from the point of view of an information agent. Section 6.3 evaluates the integration environment.

The conflict detection and resolution mechanism is implemented in a prototype called the Demonstrator introduced in Section 6.4. The case study in the following Section 6.5 evaluates, step-by-step, the detection mechanism. This case study is critically evaluated in Section 6.6, and followed by a conclusion and chapter summary.

6.1.2 Evaluation Methodology

The framework of Section 4 demonstrated the necessity to show that

1. 'Conflict detection is complete in respect to any known conflicts among multiple results.
2. The most domain-specific resolution is the most accurate for conflicts in enterprise integration.
3. All resolution procedures have the potential to be incorporated by the resolution scheme.
4. The Principle Rational resolution mechanism is functional in an enterprise integration environment' (Section 4.6 Conclusion).

Furthermore, a proof of concept requires that the mechanism can be demonstrated to function with the enterprise integration environment. This means that a prototype should be implemented that includes:

5. 'The information shared by autonomous information sources within the enterprise integration environment;
6. The Agent Knowledge that incorporates the information agent's information about itself, and the environment;
7. A conflict detection and resolution mechanism based on the implementation concept outlined in Section 5.12' (Section 5.13 Conclusion).

It follows that the evaluation has to be twofold in that it has to incorporate:

- An implementation of integrated sources and an agent model in an enterprise integration environment;
- A prototype for conflict detection and resolution by an information agent within this environment.

Thus, the following Sections describe an implementation of an enterprise integration environment including a representative number of heterogeneous information sources. A realistic scenario of interrelated information sources is implemented. This is necessary to show that semantically related data can be inconsistent across heterogeneous sources, and may also be inconsistent internally. In other words, this environment can show that data conflicts, and schema conflicts are an inherent property of dynamically changing, complex integration environments (point 5).

A model of an information agent is implemented which shows how the Agent Knowledge is managed within the agent (point 6). Hence, a demonstrator is built that allows the observation of the agent and the environment, from 'inside' the information agent.

Finally, the implementation concept outlined in the previous chapter is realised in a conflict detection and resolution mechanism within this agent model (point 7). This is described in Section 6.4.

A case study is carried out within this integration environment. This is based on information retrieved from the heterogeneous, autonomous sources which allows the inspection of the origin of possibly conflicting data. The case study shows that:

- The Mechanism detects all known conflicts (point 1);
- It manages domain-specific and other strategies in the correct order (point 2);
- Any rational resolution strategy can be incorporated into the resolution mechanism (point 3);
- The rational scheme is functional (point 4).

Furthermore, the case study outlines the contributions of this research to improve existing conflict detection and resolution in enterprise integration environments.

6.2 The Enterprise Integration Environment

6.2.1 Overview

The integration environment and the information agents are implemented in a Microsoft Windows [MSW] environment. This platform was chosen because it facilitates uniform integration that can be easily observed and monitored. For example, it is easy to switch between windows in which different information sources operate. The performance drawbacks of the Microsoft (MS) Windows implementations, for example a lack of parallelism and processing speed, can be neglected for the scope of demonstrating this research approach.

It follows that the integrated sources are preferably Windows applications, which operate smoothly within the Windows environment. Each information source has an interface programmed in MS Visual C++ [MSV]. These are menu driven, uniform windows, that are conceptually like an agents' view on a local source. In other words, two levels exist for every information source:

- The uniform Interface in C++;
- A local system such as the object-oriented database POET [POET], the relational database SQLBASE within the SQLWindows application development tool [GUP] (based on ANSI Structured Query Language SQL [SQL]), or a number of application programs implemented in C++.

The local sources show the interaction of multiple relational databases (called BookDB and MatDB), one object-oriented database (called PoetDB) and some application programs. The latter are implemented such that they simulate:

- An Expert system (called MaktExp);
- A Standard software system with an internal sequential file system (Manage);
- A software system that co-ordinates a small agent community of 'intelligent' robots (RobMgmt);

Furthermore, an enterprise model is used throughout this research as a reference model of the enterprise. This enterprise model is implemented in a C++ program called EntMod which stores data in the object-oriented database POET. The model only includes some example concepts. It is, hence, a reference model as used in federated integration environments (Section 3.3.2) and not a master model (Section 3.3.1.1), or a unified model of all systems (Section 3.3.1.2). This implementation corresponds to the definitions of an open integration environment as outlined in Section 2.3.

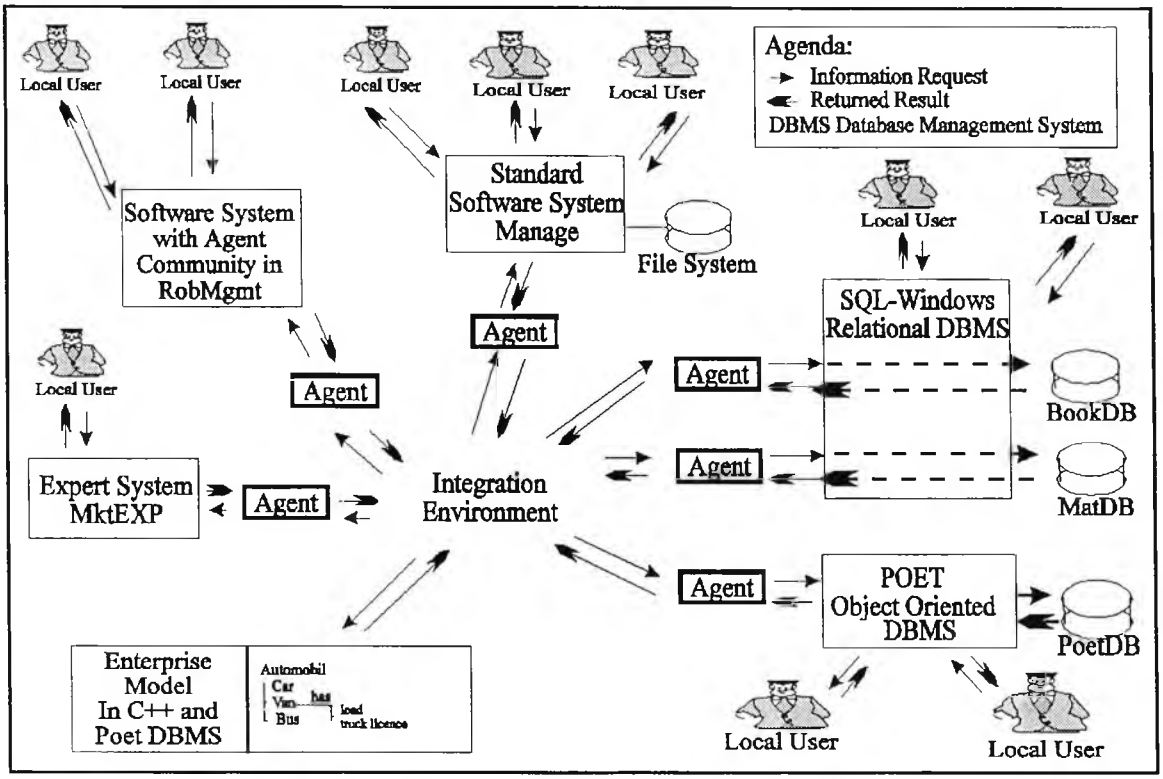

Figure 12: Demonstrator Implementation

The central component of the integration environment is the information agent. It is implemented in a C++ program called Demons. This program implements all the information available to an information agent and its reasoning capabilities. This makes the Demons program suitable to evaluate this research. Each step is explicitly shown to the inspector on the screen, including the reasoning and any data (e.g. the Agent Knowledge).

However, this implementation can only show one information agent. For all other agents only their views on the local sources are implemented. This simplification makes:

- The resulting integration procedure much more apparent;
- Draws the focus of attention to the concurrence of heterogeneous, autonomous information sources that may produce inconsistent results.

The benevolent exchange of results and meta information between homogeneous information agents can be simulated without corrupting the evaluation. Other research has described enterprise modelling and schema integration (e.g. Pan and Tenenbaum [PAN91], Barbuceanu and Fox [BAR94], Papazoglou *et al.* [PAP92a], Jagannathan et al [JAG92], Huhns and Singh [HUH92], Section 2.3 and 3.3), or information agent communication (e.g. KQML [CHA92]) in various ways. This existing research demonstrates how results are retrieved and how meta information (Agent Knowledge) is

exchanged among information agents or to one managing agent (Section 2.5). In other words, it may be assumed that

> the information available through the local views of the information agent in the Demons program,
>
> is identical to the information it would receive from other information agents that could integrate these sources, and forward the results to the managing agent (i.e. in the Demons program).

All integrated sources, their uniform interfaces (Agent Views), and the enterprise model are described in detail in Appendix D. The model of an information agent as implemented in the Demons program is described in Section 6.2.3.

6.2.2 Cafeteria Integration Environment Scenario

The previous Section has described the software and hardware platform of the integration environment. This Section will now introduce the scenario described within this integration environment; a fictitious Cafeteria as might be operating in a University of the 21 century.

A small number of food products are delivered to the Cafeteria. These are then cooked and sold to the customers. This 'cooking' is very basic such as frying chips or assembling hamburgers, and might be described as fast food production.

Two relational databases are available, one contains data on the products (MaterialDB), and the other stores any general book-keeping information (BookkeepingDB). The latter database receives frequent sales data (products and the quantity sold) from the cafeteria staff who work on the cash registers. The products sold in the restaurant are manufactured by automated robots. The program RobotMgmt manages the kitchen including these robots. In addition, an object-oriented database (ProductionDB) has been installed for data concerned with this production related section of the business.

The robots on the shop floor jointly cook all the products according to demand. Hence, they are a small community of cooperating agents. These bid for the assignment of a cooking task (a task is the production of a product). The actual task assignment is decided by the robot management system RobotMgmt.

The automated production allows the company flexibly to produce food according to changing demand. It is therefore necessary to have a production management system (ProductionMgmt) that plans the production in accordance with the current demand for any product. Furthermore, a marketing expert system (MarketingEXP) gives this production management system some basic guidelines on the overall company philosophy on production management.

A partial enterprise model is defined for this company as a service for reference by the information agents. This enterprise model is conceptually similar to the CYC knowledge base [LEN90], though it is a partial model of the enterprise such as Pan's MKS [PAN91a] or TOVE [FOX92].

A detailed description of these sources and the data they exchange is provided in Appendix D. This includes a brief description of the data structure within these sources.

6.2.3 Information Agent Model

6.2.3.1 The Agent's View

In the previous sections the integration environment has been described including the local sources, their local Agent Views on the information available from the source, and the enterprise model. However, in this section the model of an information agent within this environment will be outlined.

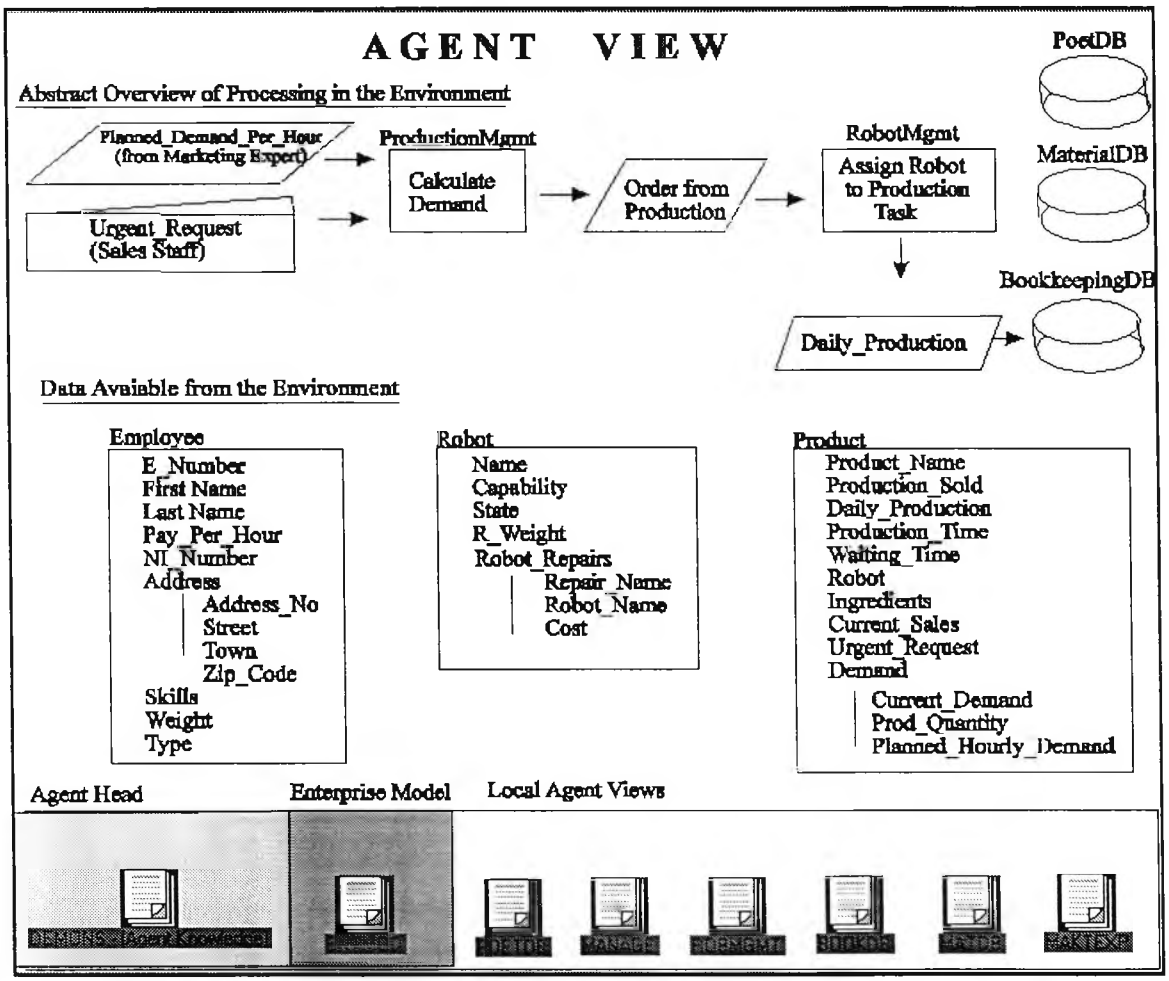

Figure 13: Agent View

Figure 13 shows the environment as it might look 'from the agent's point of view' (called the **Agent View**), including:

- The **Global Agent View** which is a description of the integration environment (systems and their interaction), and the information that is available to the agent from the environment. The Global Agent View is composed of an agent's own local view and schema information from other agents' views.
- An agent has it's own 'window' called **Agent Head**. This includes the 'Demons' demonstrator which manages the Agent Knowledge and the conflict detection and resolution mechanism. The Agent Knowledge includes other Schema

Knowledge, Resource Knowledge and Organisational Knowledge, as described in Section 2.7 (Figure 3: 'Information in Enterprise Integration') and in the description of the conflict detection and resolution mechanism in Chapter 5.

- The **enterprise model** can be accessed as a reference by the agent through the program EntMod.
- Six **Local Agent Views** are installed. These represent the channels with which the agent can interact with its own Local Source, and other agents that manage the other information sources. For example, the information agent's local source could be the system MarketingEXP (MaktExp). The other five views represent the other sources which could be integrated by other agents. The evaluation methodology (Section 6.1.2) is to focus this implementation on a pragmatic demonstration of conflict detection and resolution so that:
    - All local views have uniform interfaces implemented in Visual C++; and
    - All local views directly address the local sources and no other information agents.

6.2.3.2 The Global Agent View

Section 3.3 described different approaches to enterprise integration. These include tight integration based on master models, unified models, federated architectures, or mediators. A federated integration environment facilitates a lose integration of autonomous information sources and, therefore, provides a realistic platform for enterprise integration. References to the functionality of federated information systems, e.g. exchange and integration of schemata in sharing environments, can be found in Section 3.3.2 (e.g. Huhns *et al.* [HUH91] [HUH92]). An example of a loosely integrated, environment could be implemented along the following lines.

Each information agent shares its view on its local source with the other agents. For example, an agent A exports its schema information on its local source, with a reference that this information is available through agent A, to all other agents. In addition, each agent has information about generalisations (Mapping Information - Resource Knowledge) to integrate information from other sources (Agent Views). An example generalisation may be that the schema object Robot_Name in the RobotMgmt system may be related to schema object Robot_Name in the ProductionDB. The generalisations are used to construct a Global Agent View by matching objects in schemata imported from the other Agent's Views, e.g. described by Heimbinger and McLeod [HEI85], or Huhns and Singh [HUH92].

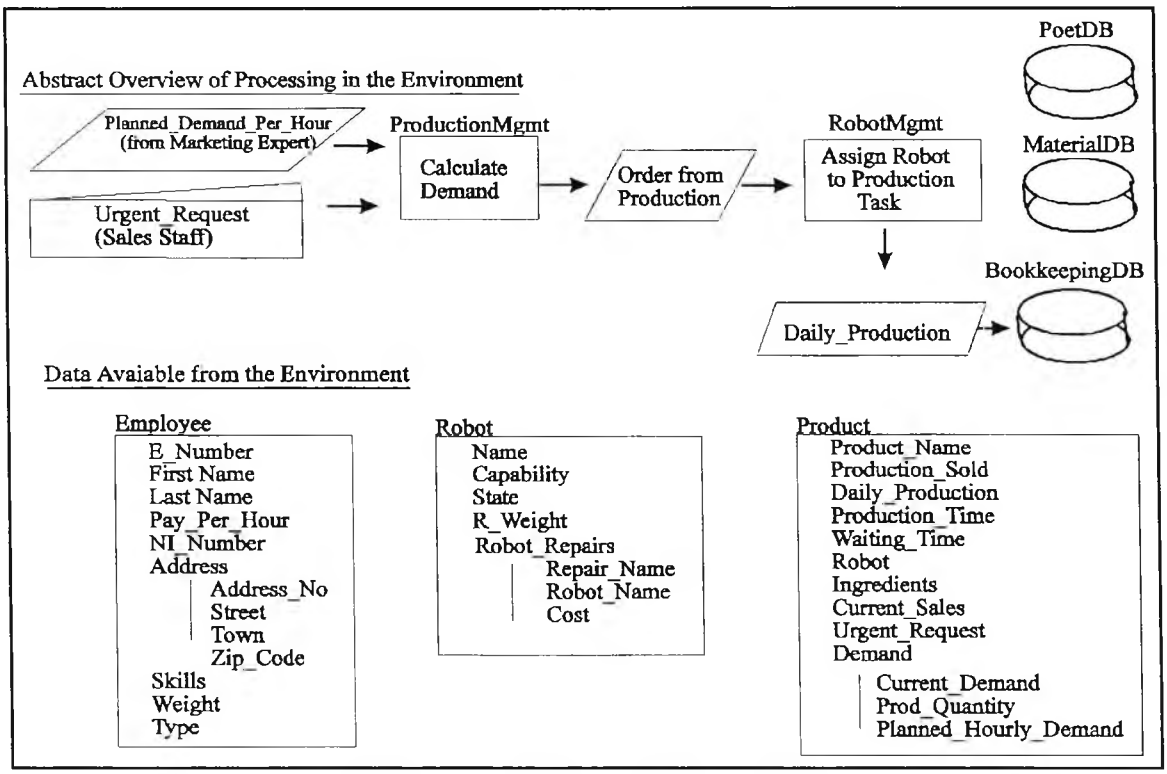

Figure 14: Global Agent View on 'Cafeteria' Environment

A Global Agent View on the 'Cafeteria' environment has been constructed by merging the local schemata in the Agent Views of the sources ProductionDB (program PoetDB), BookkeepingDB (program BookDB), MaterialDB (program MatDB), ProductionMgmt (program Manage), MarketingEXP (program MaktExp) and RobotMgmt (program RobMgmt).

The schematic integration of the information sources in the sharing environment has two levels. The top level is the Global Agent View, which shows an integrated view of the information available throughout the environment. Figure 14 provides a very high level overview of the processing and information sources in the environment. The information available throughout the environment is categorised as Employee, Robot and Product related data. In other words, all information that can be requested by the information agent that has this example Global Agent View is listed in three redundancy free lists. It is composed from the redundant information of all sources. This composition becomes explicit on the second level of the schematic integration. It is implemented in the Agent Knowledge, embodied in the 'Agent Head'. The Agent Knowledge has schematic information that represents the lower level of the Global Agent View. In other words, for each schema item in the Global Agent View all the local schema items that can provide this information are listed (described as generalisations, e.g., Section 3.3.1.1). Each local schema item has a reference to the source of origin that it resides in.

For example, the Global Agent View has the schema item Employee.First_Name. This global schema item First_Name in the category Employee has the local schema items:

Employee.First_Name from Source BookkeepingDB, and

First_Name from ProductionDB.

The latter information is available through the Agent Knowledge that resides in the Agent Head (program Demons).

6.2.3.3 Agent Knowledge

All the information that is available to an information agent has been categorised in Section 2.7 (overview in Figure 3). The information is implemented in the Agent Knowledge and can be investigated within the Demons program. Figure 15 shows the opening Window of the Agent Knowledge. The menu driven program can be initiated to begin the conflict detection and resolution mechanism, or a browser that searches the Agent Knowledge.

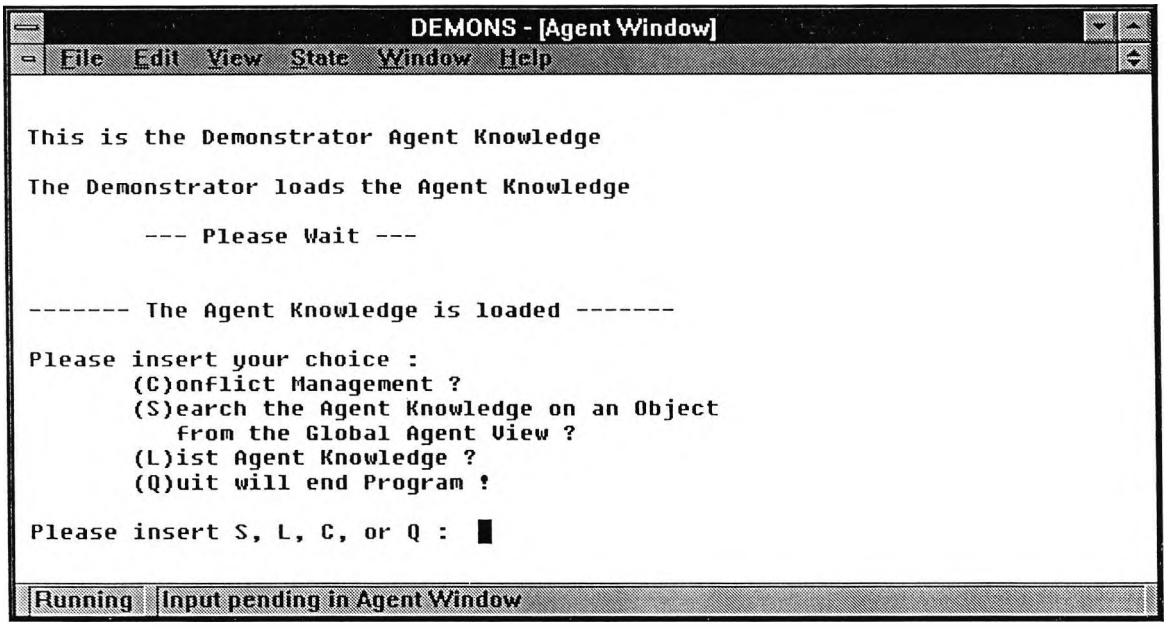

Figure 15: Agent Knowledge Managed by the Demons Program

The conflict detection and resolution mechanism is described and demonstrated in a case study starting with Section 6.4. In this Section the implementation of the Agent Knowledge, which is later used by the conflict management, is described. This description follows the categories outlined in Section 2.7, that is, Schema Knowledge, Resource Knowledge and Organisational Knowledge. An Overview on the information that is stored in the Agent Knowledge is provided in Figure 16 at the end of this Section.

Schema Knowledge

The previous Section has outlined the Global Agent View and its lower level schema objects. Global schema objects, local schema objects, and the references to their source of origin are all Schema Knowledge.

Integrity constraints may reference schema objects (Integrity Constraints - Schema Knowledge). They can define the schema objects, e.g. Production_Time, or individual objects described by that schema object (Product_Name = 'BigMac'). For example, the

schema object Production_Time (defined in the Object variable) may be constrained to only positive time values by a definition in the variable Constraint such as 'Production_Time > 0 Minutes'.

Resource Knowledge

The following information is stored as Resource Knowledge:

- Information on the Identifier_Class is contained in the form of a list with the Naming Worlds W_X and the description of their according Identifier_Classes. The description includes the name of the variable that needs to be requested from world W_X to obtain the Identifier_Object. For example, the description '*User Defined Key* in a relational database' specifies that the variable *User Defined Key* needs to be requested from the agent that manages the relational database (W_X) in order to obtain the Identifier_Object.
- Environmental Information on schema objects includes:
  - A specification when the schema objects are an essential property of any concept. For example, the attribute First_Name of the schema object Employee may be defined as an essential property of the concept Employee.
  - Instructions on how to ensure that the time and form of a certain object (described by a specific schema object) is correct (e.g. objects may have to be monitored over time to ensure that results are relevant);
  - Information on subtype - supertype relations (includes 'is_a' and 'has_a'), and synonyms ('is_equivalent_to') may be attached to schema objects. For example, the synonym definition 'Fat is_equivalent_to Overweight' may define that these concepts in the results 'Yogi is overweight' and 'Yogi is fat' (requested through the global schema object Weight) are synonyms.
- The Expertise variable specifies for any schema object or source one or multiple areas of expertise, e.g. 'Production', 'Marketing', 'Finance', 'Cooking'.
- The Authority identifies an assessment of any schema object or source's authority, such as 'High authority on Production planning'.
- Roles of objects or sources within the environment are specified in the variable Role, e.g. 'Equipment Operator'.
- Information on Problem-Solving Strategies including local problem-solving and cooperative problem-solving strategies. Information on the existence and use (accessibility by the information agent) of strategies is stored. For example, an

information agent may be able to resolve conflicts concerned with specific domains by sending the conflicting results to a specific expert system.

- Resource Knowledge also includes local Service. A service is an object that can be requested from a local source. For example, 'Statistical Reliability of Hourly Demand' may be a service to provide a statistical estimate on the schema object Hourly_Demand that can be requested from the source ProductionMgmt via its information agent.
- Counterpart Relations, Generalisations and Mapping Information may reference schema objects (Semantic Matching). Counterpart relations establish a correspondence either to objects in other information sources or to objects in the enterprise model in the variable Counterparts in the format C_{xy} and RC_{xy} (Section 5.2.1.4). Furthermore, the variable Generalisations stores generalisations between the local source and other information sources, or between the local source and the enterprise model in the format $(G(ist(G\Phi) \Leftrightarrow (ist(C_1;\Psi)))$ (Section 3.3.1.1). Mapping information, for example to map between a difference in granularity, or weight measures between sources, may be attached to these generalisations.
- Comments from designers, system administrators or the agents themselves may be linked to schema objects. Formally, Comments have three components: An Originator, e.g., a system administrator; A comment Description, e.g., Admissibility; and The comment Value, e.g., 'not admissible' or 'temporary out of order'.

Organisational Knowledge

A third source of knowledge for an information agent is its Organisational Knowledge. This includes many procedures and rules that the agent needs as 'handbook information' to function in the integration environment. Examples include the integration of schemata, or the communication with the integrated information source and other agents. It follows that the conflict detection and resolution mechanism also is Organisational Knowledge. However, in this section only that part of the Organisational Knowledge required by the conflict detection and resolution mechanism is described.

Section 2.7 demonstrated that Organisational Knowledge is defined either as Business Rules or Decision-Making Knowledge. The difference between the two is not the type of Organisational Knowledge but the way in which this knowledge is derived. Business Rules are agent specific and may vary from agent to agent. Decision-Making

Knowledge is defined for all agents alike. Hence, Organisational Knowledge can typically be both Business Rules or Decision-Making Knowledge of the following kind:

- Scientific Heuristics are domain-independent, and enable the agent to make a judgement on domain-specific information, e.g., candidates and their evidence.

- Credibility Heuristics:

  I. Heuristics specifying the credibility of schema objects, sources, or Groups directly (Implemented in the variable pairs: Origin and Certainty);

  II. Heuristics relating Roles with credibility (Implemented in the variable pairs: Role and Certainty);

  III. Heuristics relating Authority with credibility (Implemented in the variable pairs: Authority and Certainty);

  IV. Heuristics relating Expertise with credibility (Implemented in the variable pairs: Expertise and Certainty).

- Admissibility Heuristics are referenced to specific objects, sources and Groups. These heuristics are implemented in the variable pairs Origin (i.e. it specifies the source, object or Group) and the variable Admissibility. The latter can have the value 'inadmissible', or it may include a conditional admissibility. For example, 'If today is the 1.1.96 then inadmissible'.

- Ranking and Judgement Heuristics:

  (a) Ranking and Judgement Heuristics rank the credibility of sources, Groups, or specific objects, and include a judgement;

  (b) Ranking Heuristics only rank the credibility of sources, Groups, or specific objects, without including a judgement;

  (c) Judgement Heuristics only judge ranked credibility ratings for conflicting contributions.

  Each category of Ranking and Judgement Heuristic is classified into Specific Certainty Estimates, Related Certainty Estimates and General Certainty Estimates as outlined in Section 5.10.1.

- Alternative Ranking and Judgement Heuristics are believed by the information agent to be suitable for sources, Groups, or specific objects to rank and / or judge

them. However, they are only alternative, or suggested, heuristics because no definite insurance (e.g. a Business Rule defined by a system administrator into the Agent Knowledge) is available for their applicability (Section 5.10.2).

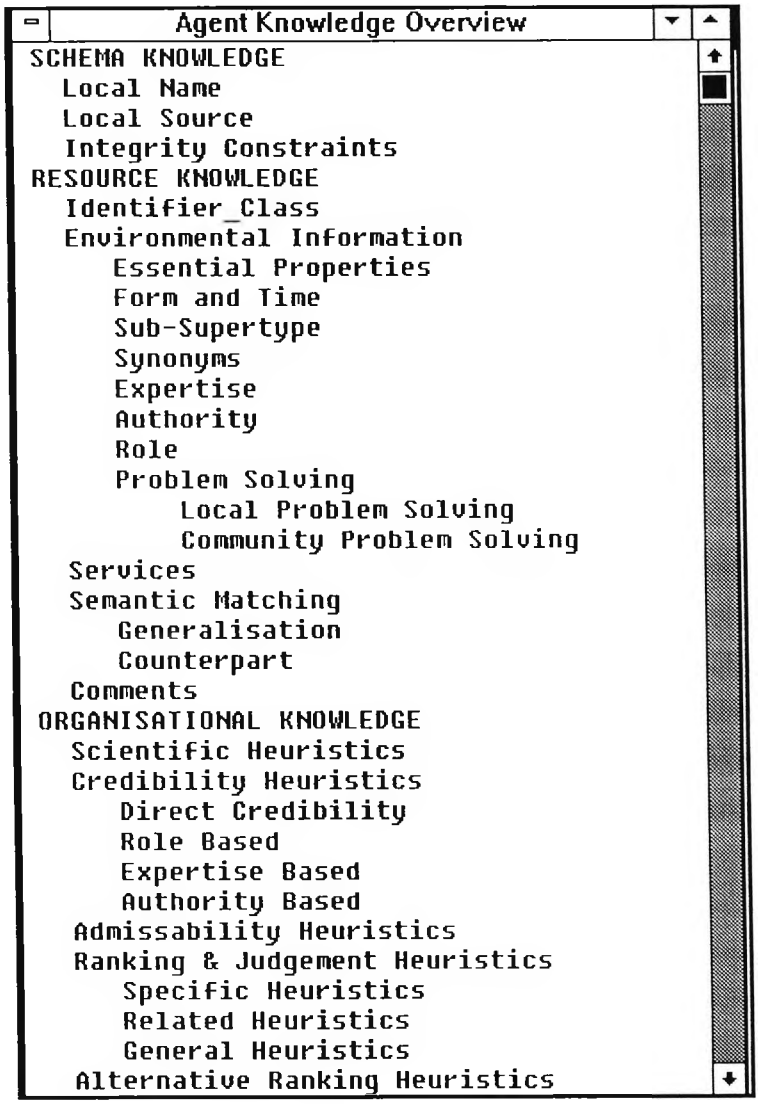

Figure 16: Overview Window in the Agent Knowledge

Figure 16 provides an overview of the information implemented in the Agent Knowledge. This overview differs from the overview in Figure 3 in that:

- Allowed Operations have not been implemented in this model of the agent because it is not concerned with distributed updates but specifically with information relevant to conflict detection and resolution among retrieved results.
- Existential Information (Resource Knowledge) is derived by the information agent based on its intentional information (e.g. Environmental and Schema Information). Hence, it is not explicitly defined in the Agent Knowledge.

- This overview is more specific than that in Figure 3. Hence, it entails a more detailed list of the contents of the subcategories than Figure 3, for example, Essential Properties are listed as a kind of Environmental Information.

6.3 Evaluation of the Integration Environment

The implementation of the integration environment 'Cafeteria' has illustrated how the information described in Section 2.7 can be gathered in Enterprise Integration Environments. Furthermore, it shows how this information can be explored and managed in an agent model. The description of available information in enterprise integration in Section 2.7 is complete. However, some Organisational Knowledge that is the agent's 'handbook information' defined in the Agent Knowledge, may include information specific to the conflict detection and resolution mechanism (e.g. Credibility Heuristics).

The 'Cafeteria' example demonstrates a fictitious enterprise integration environment. However, it makes apparent that the integration of local agent views (schemata) of heterogeneous information sources into a complete global view is, even with this small example, very complex.

Existing research on enterprise integration and the presented implementation rely heavily on human experts. For example, current research in enterprise integration assumes that an 'enterprise model' such as TOVE [FOX93], MKS [PAN91a], or the common knowledge-base CYC [LEN90], is defined and installed by human experts. In other words, such a model is man-made and has to be administrated or updated constantly by human experts. This is a potential limitation of the practicability of current research in the field of DCEEI. However, for some areas of enterprise integration agent learning can reduce the dependence on manual installation and maintenance. Some potential ways to design such learning mechanisms have been briefly mentioned in the previous chapter (Section 5.4.2, 5.9 and 5.10.3). For example, the information agent could automatically, and systematically request new meta-knowledge from the integrated sources to update its Agent Knowledge. Furthermore, agents could improve and update their Agent Knowledge by autonomously interrogating domain experts.

The implementation shows that the Agent View and the Global Agent View are independent of the information that is actually stored in the information sources. The views provide access to the information sources but no control over these autonomous sources. In other words, in dynamically changing, concurrent environments inconsistencies between an information source and the agent view is, at least temporally, unavoidable. Hence, cross system consistency is an unrealistic assumption made by most research in enterprise integration described in Section 3.3.

Matching objects from different information sources and agent views into common concepts in the global view carries the potential risk of semantic misinterpretation. The implementation makes it explicit that it is adequate to assume schematic incompleteness, or incorrectness (The conflict detection and resolution mechanism has therefore been based on this assumption).

In summary, the implemented environment shows that inconsistencies on the global level (between an agent's local view and the local source, or across local views) are inherent in enterprise integration. Semantically related data can be inspected at its origin. No control mechanism of the information agents can control or exclude these inconsistencies within and among autonomous sources. It can potentially become inconsistent across systems.

The implementation of the agent model (Agent Knowledge) shows how the information agent can manage all the information available to it in enterprise integration environments (Section 2.7).

It remains to be demonstrated how the conflict detection and resolution mechanism outlined in the implementation concept can be realised. Furthermore, the functionality of the mechanism has to be shown, and that it fulfils the conditions set out in the conclusion of Chapter 4 (Section 4.6 and Evaluation Methodology Section 6.1.2). The next Section addresses these issues.

6.4 Demonstrator For Conflict Detection and Resolution

6.4.1 Introduction to the Demonstrator

The conflict detection and resolution mechanism is implemented as a demonstrator. It is part of the Agent Knowledge (Organisational Knowledge). In Figure 15 the opening window of the Demons program is shown. In this window a choice can be made to investigate the Agent Knowledge (as described in the previous Section 6.2.3) or to run the conflict management program. The investigation of the Agent Knowledge has been the subject of the previous Section.

In this section the conflict management program is introduced that strictly realises the implementation concept outlined in Section 5.12 (in C++ as is the whole agent model). This is demonstrated in the following case study, which systematically runs through each step of this conflict management program. Hence, each section in the case study corresponds to a section in Chapter 5 describing this step (e.g. Section 5.2.2 Gathering of Candidates provides the design for the Gathering of Candidates in the case study Section 6.5.1).

The conflict management program (called the Demonstrator) has two sources of data input:

- Local Information (results and other evidence originating from local sources)
- the Agent Knowledge.

Local information from the local sources is retrieved through the Local Agent Views (Section 6.2.3.2). The retrieval is a manual, menu driven process that allows the observer to retrieve results and evidence from the local sources. This manual inspection follows the premise that the sole purpose of the retrieval in this evaluation is to prove that (and how) semantically related data and schema information may exist that is inconsistent in concurrent, dynamically changing environments of heterogeneous, autonomous sources (Section 6.1.2 Methodology). An example retrieval process is briefly outlined in Section 6.4.2.

The Demons program automatically searches the Agent Knowledge for the relevant information for each conflict resolution step. However, this process is not a 'black box' that simply produces information from the agent model. Each detection or resolution step is opened by a 'walk through' that automatically selects the Agent Knowledge relevant for this specific step. The selected information is shown on the screen and can be inspected at any time by switching to the **Agent Knowledge Search** window.

The detection and resolution process constantly analyses the conflict (first by assessing if a conflict exists and then to resolve it). Hence, a **Result Window** always provides an overview of the information gathered on this case, and any results that have been produced about the conflict case that is currently managed.

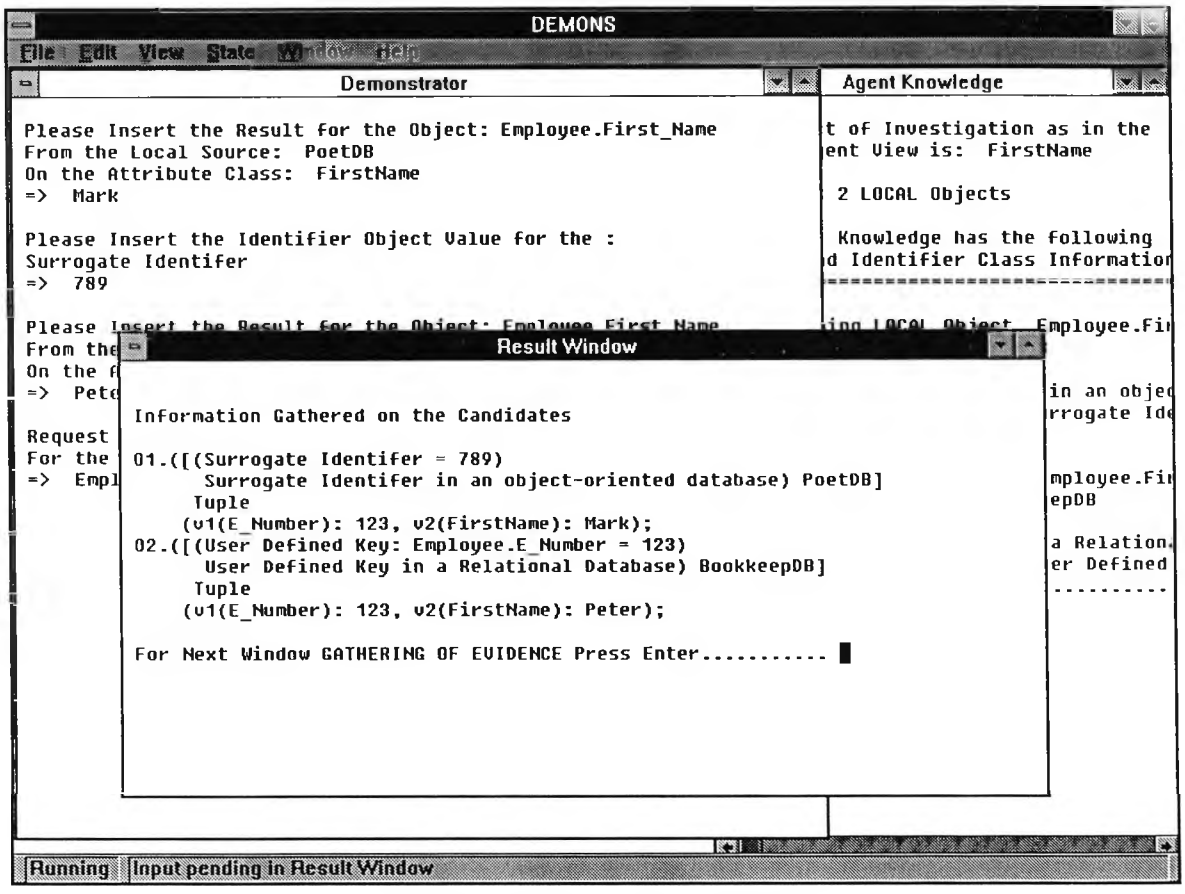

Figure 17: Demonstrator with Result Window

Figure 17 shows the Demonstrator with its three windows:

- The Demonstrator window with the conflict management program;
- The Result Window that shows the current state of affairs ('all that is known about the possibly conflicting candidates'); and
- The Agent Knowledge Search window, which is used to search the Agent Knowledge for relevant information (it presents the latest search results until a new search is initiated by the Demonstrator).

The following case study demonstrates example queries that have been managed by the Demonstrator with the implementation of the 'Cafeteria' environment as described in Section 6.2.2.

6.4.2 Information Retrieval

In this section some example information retrieval processes will be described to demonstrate the functionality of the integration environment. Furthermore, it will show how these processes provide the information from the local sources to the Demonstrator.

Any information system in the enterprise may have a number of 'local users' (Figure 12). These may be humans or applications that request information from this source. For example, a local user of the relational database ProductionDB may request the first name of an employee with the employee number '123'. This is formulated in the local database language SQL:

```
Select First_Name from Employee
where E_Number = 123
```

This query may result in a solution in the form of :

```
First_Name: Peter
```

However, it may also be necessary for the user to ensure that this result is consistent with any other source in the environment. Furthermore, the user may want to request information that is not stored at the local database ProductionDB but is located in another database. In these cases it is necessary that the user requests information from the environment. In principle, two ways exist for a client, which can be a human user or an application systems, to request information from the Collaborative Distributed Environment for Enterprise Integration (DCEEI):

1. A client may request information from its local source, e.g., a database management system. This system may initiate its integrating agent to request information from the distributed system (DCEEI).
2. A user may have a user interface, as described in Sections 2.3 and 2.6. This is an expert that mediates between the information agent and the user. For pragmatic reasons this translation has been reduced by assuming that the user understands the propositional calculus used by the agents. The interface is therefore identical to the Global Agent View (Section 6.2.3.2).

In any case an information agent is initiated and appointed 'managing agent' to receive the information request. The client on the user interface, or the management system that acts as a client to the integrating agent, requests information from the agent. This agent has to then match this request with an object from its global view. This is a value based schema object matching task based on a specific enterprise modelling strategy. For example, many enterprise integration environments provide a common ontology (Section Master Models or Unified Model Section 3.3.1). Otherwise a federated

mechanism would let the information agents directly share schema information. This is conceptually similar to federated databases described by Heimbinger and McLeod [HEI85], or for an object-oriented database with shared and private sources by Kim *et al.* [KIM91a]. Any of these schemata may enable the agent to match the requested object on different, hierarchically related levels of abstraction. The agent would aim to match the object to the most specific schema object possible (lowest possible level of abstraction (e.g. implemented in CARNOT [COL91] Section 3.3.1.1).

An example query (**Example 1**) could be:

> 'What is the First Name (attribute) of an Employee (object) with the employee number 123?

That query translates into the propositional calculus introduced in Section 5.2.2 as:

> $(O_i.v_1(E\_Number)$: 123, $v_2(First\_Name)$: ?)

Provided the agent can match the request to its schema objects (typically in the global view) then it can identify which sources can provide this information. For example, the agent can find the objects Employee and an attribute First_Name in its Global View. The agent would investigate its Schema Knowledge for Local Names corresponding to the global schema object First_Name (please note that the Global Agent View is free of redundancies Section 6.2.3.2). In this example, two entries can be found in the agent's Schema Knowledge (in the variable Local Name):

> Local Name: Table Employee First_Name Source: BookkeepingDB;
> Local Name: Class Employee First_Name ProductionDB.

In other words, generalisations map the global entity Employee to the two Employee entities in the sources BookkeepingDB and ProductionDB. Further investigation of the Agent Knowledge also shows that the entities Employee both have the attributes E_Number and First_Name. In principle, the query would be sent to agents that integrated the two sources BookkeepingDB and ProductionDB. These agents translate and forward the request to the local management systems (e.g. the POET database and the SQLBASE relational database management system). In the Demons implementation this can be done directly via the Agent Views PoetDB and BookkeepingDB. In return the following results can be retrieved:

> First_Name: Peter (from BookkeepingDB)
> First_Name: Mark (from ProductionDB)

The managing agent now has two results provided by the information retrieval process of the DCEEI. The agent now needs a conflict detection mechanism to ensure that the results do not conflict. This mechanism is provided by this research and opens with the Gathering phase in the following Section 6.5.1.

6.5 Case Study Conflict Detection and Resolution

6.5.1 Gathering of Candidates:

The previous Section has described the functionality of a possible information retrieval process in Cooperative Distributed Environments for Enterprise Integration (DCEEI). It provides the conflict detection mechanism with the results to a given query. The example query:

> 'What is the First_Name (attribute) of an employee (object) with the employee number 123?'

has been used to show how the environment produces two results 'Peter' and 'Mark'. These results now need to be uniformly gathered by the detection mechanism. In other words, the results are specified as:

> O_1.(v_1(E_Number): 123, v_2(First_Name): Peter)
>
> O_2.(v_1(E_Number): 123, v_2(First_Name): Mark)

These results need to be uniformly presented in the object-oriented form of:

> [Identifier_Object Identifier_Class W_X] Type Value

Some variables are already known to the agent including the Value, and the Naming World W_X. The Value variable simply contains the object's O_1 and O_2 attribute names v_m (that is, in this example, the attributes E_Number and First_Name) and their data values (e.g. 'Peter' and 'Mark').

The Naming Worlds are known since the agent has requested the results from specific sources (or their agents in other DCEEI). In addition, the results returned from information agents, e.g. of the BookkeepingDB and ProductionDB, would specify their source of origin in the communication protocols. For example, the Communication Layer of the KQML communication package lists the source of origin ([CHA92] [FIN94a] [FIN94b] and Section 2.3). Provided that cases exist where the information source is not automatically the Naming World, a further mapping would be necessary that relates each local schema object to a Naming World (Schema Knowledge).

With the information on the source of origin and the Naming Worlds of the results, the agent can access either its Agent Knowledge, or the agents that integrate these sources to investigate the Type of the retrieved objects. Typically in enterprise integration, however, results of the Type 'tuple' are exchanged. In addition, the object-oriented database POET or the software system ProductionMgmt, which could potentially present other objects of the type set and atom, would specify that type automatically. In

other words, an object-oriented database would typically specify the type along with the result statement. Though not implemented, the Agent Knowledge could hold this information as a kind of Semantic Matching Information - Resource Knowledge.

The Identifier_Class information is an agent's Resource Knowledge. It is implemented as a list of Naming Worlds W_X with a formal specification of the Identifier_Class. The Naming Worlds are BookkeepingDB and ProductionDB. Hence, the list of Identifier_Classes has the following entries that are relevant in this example:

BookkeepingDB (*User Defined Key* in a relational database)

ProductionDB (*Surrogate Based Identifier* from an object-oriented database).

Each Identifier_Class requires a specific Identifier_Object. In this example these are 'User Defined Key' and 'Surrogate Based Identifier'. In other words, the information agent has to identify the key columns of the table Employee in the BookkeepingDB. The definition of key columns in schemata of relational databases can be defined in an Agent's Environmental Information (Resource Knowledge) or it could be attached to schema objects (Schema Knowledge). In this example, it is Schema Knowledge implemented as part of the agent's view on the BookkeepingDB. In the previous example the key E_Number of the object with the First_Name 'Peter' has already been provided by the initial retrieval. Otherwise, the agent could request the User Defined Key E_Number for an object with the First_Name 'Peter' in the Table Employee from the BookkeepingDB, and would receive the following result:

Employee.E_Number = 123.

In the same way, the agent can request the Surrogate Based Identifier for an employee called 'Mark' from the ProductionDB and would receive a result such as

Surrogate Identifier = 0-507#30.

In summary the following candidates are gathered for **Example 1** in the structure ([Identifier_Object Identifier_Class W_X] Type Value):

O_1.[(Employee.E_Number = 123) (*User Defined Key* in a relational database) BookkeepingDB] Tuple (v_1(E_Number): 123, v_2(First_Name): Peter)

O_2.[(Surrogate Identifier = 0-507#30) (*Surrogate Identifier* in an object-oriented database) ProductionDB] Tuple (v_1(E_Number):123, v_2(First_Name): Mark)

Another example query 'Get me the Demand of the Product called 'BigMac' would formally translate into :

O_i.(v_1(Product_Name): BigMac, v_2(Demand):?)

In total seven results can be gathered in this **Example 2** which are described in the structure ([Identifier_Object Identifier_Class] W_X) Type Value as follows:

O_1.[(ProcessID: 229) (Member of a Process with *Process Id's*) ProductionMgmt] Tuple (v_1(Product_Name): BigMac, v_2(New_Demand): 34)

O_2.[(ProcessID: 229) (Member of a Process with *Process Id's*) ProductionMgmt] Tuple (v_1(Product_Name): BigMac, v_2(Hourly_Demand): 44)

O_3.[(Essential Property Product_Name = BigMac) (*Essential Property is Product_Name)* ProductionMgmt] Tuple (v_1(Product_Name): BigMac, v_2(Current_Hourly_Demand): 39)

O_4.[(Essential Characteristic Product_Name: BigMac) (*Essential Characteristics* as in the Enterprise Model) MarketingEXP] Tuple (v_1(Product_Name): BigMac, v_2(Current_Demand): 50)

O_5.[(Surrogate Identifier = 0-508#789) (*Surrogate Identifier* in an object-oriented database) ProductionDB] Tuple (v_1(Product_Name): BigMac, v_2(Expected Demand): 45)

O6.[(Essential Characteristic Product_Name: BigMac) (*Essential Characteristics* as in the Enterprise Model) MarketingEXP] Tuple (v_1(Product_Name): BigMac, v_2(Planned_Hourly_Demand): 55)

O_7.[(Position Number 238481) (*Position Number* address based in Production Table) RobotMgmt] Tuple (v_1(Product_Name): BigMac, v_2(Prod_Quantity): 44)

6.5.2 Gathering of Evidence

In this next step the information agent gathers any evidence that supports or refutes the candidates. That is, any evidence available from the source of origin, such as Services that qualify results (Resource Knowledge), or justifications as they may be provided by expert systems. In any case, the evidence is specified in the form of E_m ={(Formula)(Certainty Estimate)} (Section 5.2.3). Either the Formula, or the Certainty Estimate variable may be empty in case the evidence is only a formula (description or verbal justification), or a only a certainty estimate.

In the previous Example 2 the candidate O_1 has a formula justifying its New_Demand in the form of a reasoning chain (Justification from an expert system):

E_1 = {(Current_Demand is more than 5 units larger than Current_Sales) ()}

This information is provided by the expert system within the ProductionMgmt system. The justification is internally derived from the rules in the expert system (as described in Appendix D Section D.4).

In addition, the ProductionMgmt calculates the statistical reliability of the New_Demand variable. This is done by a small statistical program that calculates how often the New_Demand estimate has varied less than five per cent from the actual sales (Current Sales)(detailed description Appendix D Section D.4).

E_2 = {(New_Demand) (44 % Reliability)}

This form of evidence is either a Service (Resource Knowledge) that can be requested from an information agent or it is, as in this case, automatically produced by the source of origin (ProductionMgmt).

6.5.3 Classification

In the previous two steps the candidates and the evidence of these candidates has been gathered. In case one or no result exists for a given query then a conflict is impossible and further conflict detection or resolution becomes meaningless. For example, the query 'What is the address (attribute) of the employee (Object) with the First_Name (Attribute) George':

O_i.(v_1(First_Name): George, v_2(Address.Street): ?)

The result from the BookkeepingDB would be 'Null' because there is no employee with the First_Name George. If this request would be concerned with, e.g., the employee called 'Peter' then only one result would be returned from the BookkeepingDB:

First_Name = Peter, Address Street: Pentonville Rd., Town: London, ZIP: 123

In order to identify a conflict at least two candidates need to exist and it needs to be determined that multiple results exist that are **not** identical. The rare case where one result is internally inconsistent produces two conflicting candidates as outlined in Section 5.2.4.2. Thus, if there are multiple candidates then these may be identical in respect to the investigated attribute. For example, the query:

O_i.(v1(E_Number): 125, v_2(First_Name): ?)

results in the candidates:

O_1.(v_1(E_Number): 125, v_2(First_Name): Joe) from BookkeepingDB

O_2.(v_1(E_Number) 125, v_2(First_Name): Joe) from ProductionDB

These are multiple candidates but they are not conflicting. These candidates have identical properties for the requested attribute First_Name. In any other case a kind of explicit or implicit conflict may potentially exist that needs to be specified according to the propositional categorisation introduced in Section 3.2.1.

Explicit Conflicts

An explicit conflict could result from the query: 'Is the employee with the employee number 123 called Peter?':

O_i.(v_1(E_Number): 123, v_2(First_Name): Peter).

Using KQML such queries are called 'Assign Truth Value' [CHA92]. In other words, the information agent sends the above sentence to the relevant sources. These sources then identify if the statement is valid in their world. A database typically retrieves this value.

Only in cases where a result (match) is found the sentence is assigned a truth value. The previous query produces the results:

O_1.[(Employee.E_Number = 123) (*User Defined Key* in a relational database)v_1(E_Number) BookkeepingDB] Tuple (v_1(E_Number) : 123, v_2(First_Name) Peter)

O_2.[() (*Surrogate Identifier* in an object-oriented database) ProductionDB] No type (v_1(E_Number): , v_2(First_Name):)

In other words, an explicit conflict exists. The BookkeepingDB results in a claim that an employee called Peter with the E_Number 123 exists. The result from the ProductionDB results in a statement claiming that this is not the case (Null).

Implicit Conflicts

Example 1 is a typical case of implicit conflicts in enterprise integration [BAR94b]. The following two results from Example 1 conflict implicitly provided the information agent assumes that only one First_Name value is correct.

O_1.[(Employee.E_Number = 123) (*User Defined Key* in a relational database) BookkeepingDB] Tuple (v_1(E_Number): 123, v_2(First_Name): Peter)

O_2.[(Surrogate Identifier = 0-507#30) (*Surrogate Identifier* in an object-oriented database) ProductionDB] Tuple (v_1(E_Number):123, v_2(First_Name): Mark)

In other words, all candidates are assumed to be concerned with the same candidate (O_i), and each candidate has a different property (R_k) for the same attribute class v_m(First_Name).

Another kind of implicit conflict is presented by Example 2. It has seven results concerning the demand of the product 'BigMac'. Out of these five results provide a Current_Demand attribute (i.e. the Current_Demand schema object in the Global Agent View described in Section 6.2.3.2 relates to the following attribute objects in local Agent Views (schemata): New_Demand, Hourly_Demand, Current_Hourly_Demand, Current_Demand and Expected_Demand):

O_1.[(ProcessID: 229) (Member of a Process with *Process Id's*) ProductionMgmt] Tuple (v_1(Product_Name): BigMac, v_2(New_Demand): 34)

O_2.[(ProcessID: 229) (Member of a Process with *Process Id's*) ProductionMgmt] Tuple (v_1(Product_Name): BigMac, v_2(Hourly_Demand): 44)

O_3.[(Essential Property Product_Name = BigMac) (*Essential Property is Product_Name)* ProductionMgmt] Tuple (v_1(Product_Name): BigMac, v_2(Current_Hourly_Demand): 39)

O_4.[(Essential Characteristic Product_Name: BigMac) (*Essential Characteristics* as in the Enterprise Model) MarketingEXP] Tuple (v_1(Product_Name): BigMac, v_2(Current_Demand): 50)

O_5.[(Surrogate Identifier = 0-508#789) (*Surrogate Identifier* in an object-oriented database) ProductionDB] Tuple (v_1(Product_Name): BigMac, v_2(Expected_Demand): 45)

All these candidates conflict implicitly. In other words, all candidates are assumed to be concerned with the same product 'BigMac', but for the attribute 'v_2' different propositions (R_k) exist. This implicit conflict is detected by the information agent based on its Schema Knowledge, which indicates that all attributes are concerned with the same thing; the Current_Demand (as in the Global Agent View) despite their different names used in the local Agent Views and local sources (New_Demand, Hourly_Demand, Current_Hourly_Demand and Expected_Demand). Hence, these attributes are conflicting unless they have identical properties (R_k) for the attribute class 'v_2(Current_Demand)'

The detection of implicit conflicts can, however, be much more complex. In Example 2, the Prod_Quantity attribute with the value '44' from the RobotMgmt (candidate O_7) may be conflicting with the candidates O_6 specifying the Planned_Hourly_Demand. This implicit conflict can only be detected if the agent knows that:

- The Planned_Hourly_Demand from the MarketingEXP is supposed to be the actual production at any given time (from the point of view of the Marketing Department).
- Thus, the attributes for the classes Planned_Hourly_Demand (MarketingEXP) and Prod_Quantity (RobotMgmt) concerned with the same Product, conflict if they are not identical.

O6.[(Essential Characteristic Product_Name: BigMac) (*Essential Characteristics* as in the Enterprise Model) MarketingEXP] Tuple (v_1(Product_Name): BigMac, v_2(Planned_Hourly_Demand): 55)

O_7.[(Position Number 238481) (*Position Number* address based in Production Table) RobotMgmt] Tuple (v_1(Product_Name): BigMac, v_2(Prod_Quantity): 44)

More complex implicit conflicts, e.g., based on different individuals (Objects) with conflicting properties, are typically not detected in enterprise integration environment (Section 3.2.1). Hence, these are not of particular interest to this evaluation and not investigated specifically.

Conflict detection is implemented in the mechanism in that it systematically checks all relevant Agent Knowledge. In other words, the Demonstrator searches for the attribute class of investigation (e.g. Prod_Quantity and Hourly_Demand in the last example) in the following Agent Knowledge categories:

- Integrity Constraints (Schema Knowledge) ;
- Essential Properties (Environmental Information - Resource Knowledge);
- Comments (Resource Knowledge).

Furthermore, the enterprise model can be manually investigated on any essential properties and relations of the attribute class of investigation (Called Concepts in the Enterprise Model Appendix D Section D.6)

All this information is gathered from the Agent Knowledge and provided in the Agent Knowledge Search window in the Demonstrator implementation. This, and information from the enterprise model, facilitate the agent to detect any 'known' conflicts described above. 'Known' means based on the available Agent Knowledge (Section 3.2.3). Potentially, conflict detection is **limited** by the completeness of Agent Knowledge (Section 5.2.4.2). In this implementation the detection is limited by the completeness of the Agent Knowledge categories Integrity Constraints, Essential Properties and Comments (Section 2.7).

However, all the described examples are based on a correspondence assumption. In these cases the information agent has to be justified to assume that the candidates are concerned with the same thing, in order to detect a conflict based on this assumption. Section 5.2.4.3 introduced some sameness predicates. These are used by the information agent to classify the sameness between the candidates. The aim of analysis is to identify candidates that are not concerned with the same object. Hence, their correspondence can definitely not be assumed and no conflict based on this assumption will exist.

In order to classify the object sameness the mechanism has to search for generalisations and counterpart relations between the candidates. The agents' Semantic Matching - Resource Knowledge has all the generalisations and counterpart relations for a given object (Global Schema Objects in the Global Agent View and Local Schema Objects in the Local Agent Views) (Section 5.2.1.4). Furthermore, Section 5.2.4.3 outlined the

format for generalisations G (ist(Gϕ) $\Leftrightarrow$ ist($W_X\psi$)) and counterpart relations (C_{XY}). Real world counterparts of two objects are either defined by a counterpart relation (or generalisations) between two objects in information sources, or by a counterpart relation (or generalisations) to the same object in the enterprise model (Section 5.2.4.3).

It follows that the detection mechanism collects all generalisations and counterpart relations concerned with the relevant attributes of the candidates. In case of Example 2 the demand of a given product is investigated. Following the description in Section 3.3.1 the mechanism first tries to match the candidates to any Set Concepts in the enterprise model (Appendix D Section D.6). It then tries to map the concepts (attributes) which is the next lower in the hierarchically organised enterprise model. The following generalisation matches each Product entity in the ProductionMgmt to the Set Concept Product in the enterprise model by its Product_Name:

1. G (ist(Product_Name in ProductionMgmt) $\Leftrightarrow$ (Product Name in Enterprise Model)

Counterpart relations in the MarketingEXP system for the Product_Name entities include

2. C (BigMac in Product_Name ProductionMgmt)(BigMac in Product_Name MarketingEXP).

In other words, the product called 'BigMac' in the ProductionMgmt system has a counterpart in the MarketingEXP with the same name. In addition, the following relation defines that the counterpart relation is also reflexive such that the 'BigMac' in MarketingEXP also has the 'BigMac' in ProductionMgmt as a counterpart:

3. C (BigMac in Product_Name MarketingEXP)(BigMac in Product_Name ProductionMgmt)

The information on generalisations and counterpart relation is required by the information agent in order to determine the sameness predicates. The candidates are checked for their sameness predicates in the order outlined in Figure 7 Section 5.2.4.4. The following list includes an example for each degree of sameness (Format: ([Identifier_Object Identifier_Class] W_X) Type Value):

Purely Identical (PI)

From Example 2 the following candidates are Purely Identical:

O_4.[(Essential Characteristic Product_Name: BigMac) (*Essential Characteristics* as in the Enterprise Model) MarketingEXP] Tuple (v_1(Product_Name): BigMac, v_2(Current_Demand): 50)

O_6.[(Essential Characteristic Product_Name: BigMac) (*Essential Characteristics* as in the Enterprise Model) MarketingEXP] Tuple (v_1(Product_Name): BigMac, v_2(Planned_Hourly_Demand): 55)

Trivial Purely Identical (TPI)

Example 2 has the following candidates that are based on the Generalisation: G (ist(Product_Name ProductionMgmt) ⇔ ist (Product.Name Enterprise Model) which means that all Product_Name attributes in ProductionMgmt can be matched to Product.Name in the Enterprise Model.

O_1.[(ProcessID: 229) (Member of a Process with *Process Id's*) ProductionMgmt] Tuple (v_1(Product_Name): BigMac, v_2(New_Demand): 34)

O_2.[(ProcessID: 229) (Member of a Process with *Process Id's*) ProductionMgmt] Tuple (v_1(Product_Name): BigMac, v_2(Hourly_Demand): 44)

Derived Identical (DI)

A query on the Ingredients of a given Product (Product_Name) called 'BigMac' with a Total_Production of one hundred units may be described as:

O_i.(v_1(Product_Name): BigMac, v_2(Total_Production): 100, v_3(Ingredients): ?)

This produces the following Derived Identical Results:

O_1.[(Product.Product_Name = BigMac) (*User Defined Key* in a relational database) BookkeepingDB] Tuple (v_1(Product_Name): BigMac, v_2(Daily_Production): 100 pcs.)

O_2.[(Product.Product_Name = BigMac) (*User Defined Key* in a relational database) MaterialDB] Tuple (v_1(Product_Name): BigMac, v_2(Material_Name): Bun, MeatBall, BigMacSource)

Trivial Derived Identical (TDI)

The following candidates are defined by the reflexive counterpart relation (Section 5.2.1.4) RC (MacChicken in MaterialDB Product.Product_Name) (MacChicken in BookkeepingDB Product.Product_Name) that makes the following Results Trivial Derived Identical based:

(Query: O_i.(v_1(Product_Name): BigMac, v_2(Total_Production): 100, v_3(Ingredients): ?))

O_1.[(Product.Product_Name = MacChicken) (*User Defined Key* in a relational database) BookkeepingDB] Tuple (v_1(Product_Name): MacChicken, v_2(Total_Production): 100 pcs.)

O_2.[(Product.Product_Name = MacChicken) (*User Defined Key* in a relational database) MaterialDB] Tuple (v_1(Product_Name): MacChicken, v_2(Material_Name): Bun, Chicken, MacChickenSource)

Referential Equality (RE)

The previous counterpart relation between the 'BigMac' objects in the ProductionMgmt and the MarketingEXP makes the candidates O_3 and O_4 from Example 2 Referential Equal:

O_3.[(Essential Property Product_Name = BigMac) (*Essential Property is Product_Name)* ProductionMgmt] Tuple (v_1(Product_Name): BigMac, v_2(Current_Hourly_Demand): 39)

O_4.[(Essential Characteristic Product_Name: BigMac) (*Essential Characteristics* as in the Enterprise Model) MarketingEXP] Tuple (v_1(Product_Name): BigMac, v_2(Current_Demand): 50)

Match Equal (ME)

The following candidates from Example 1 are Match Equal (on the user defined key E_Number).

O_1.[(Employee.E_Number = 123) (*User Defined Key* in a relational database) BookkeepingDB] Tuple (v_1(E_Number): 123, v_2(First_Name): Peter)

O_2.[(Surrogate Identifier = 0-507#30) (*Surrogate Identifier* in an object-oriented database) ProductionDB] Tuple (v_1(E_Number):123, v_2(First_Name): Mark)

Aspect Equal (ASE)

A query on the First_Name of any employees that live in 'James St.' (Adress_No 11) has produced two Aspect Equal candidates ('Frank' with E_Number '126' and 'Gary' with E_Number '136' both live at the same Address 'James St.'):

(Query: O_i.(v_1(Address_No): 11, v_2(E_Number):?))

O_1.[(Employee.E_Number = 126) (*User Defined Key* in a relational database) BookkeepingDB] Tuple (v_1(Address_No): 11, v_2(E_Number): 126)

O_2.[(Employee.E_Number = 136) (*User Defined Key* in a relational database) BookkeepingDB] Tuple (v_1(Address_No): 11, v_2(E_Number): 136)

Arbitrary Equality (ARE)

The following candidates are Arbitrary Equality. Both candidates have no common attributes, and distinct real world counterparts. The generalisation 'G (ist(Product_Name ProductionMgmt) ⇔ (Product Name in Enterprise Model))' and the counterpart relation 'C (MacChicken in MarketingEXP)(MacChicken in Enterprise Model)' exist. Based on these the following candidates have real world counterparts called 'BigMac' and 'MacChicken' but these are distinct. However, in the enterprise model the products MacChicken and BigMac both have the same essential property 'Product_Name'. Hence the following query produces Arbitrary Equal candidates:

(Query: O_i.(v_1(Current_Sales): 22, v_2(Product_Name):?))

O_1.[(ProcessID: 229) (Member of Process with *Process Id's*) ProductionMgmt] Tuple (v_1(Current_Sales): 22, v_2(Product_Name): BigMac)

O_2.[(Essential Characteristic Product_Name: BigMac) (*Essential Characteristics* as in the Enterprise Model) MarketingEXP] Tuple (v_1(Current_Sales): 22, v_2(Product_Name): MacChicken)

No Object Correspondence

Finally, the agent may determine that candidates are definitely not concerned with the same object. The previous query has actually produced three candidates. The first candidate O_1 is from the same source ProductionMgmt, has the same Identifier_Class ('Member of Process with Process Id') but different Identifier_Objects ('Process Ids 229 and 230') as the third candidate O_3. Hence, the following candidates O_1 and O_3 are not concerned with the same object:

O_1.[(ProcessID: 229) (Member of Process with *Process Id's*) ProductionMgmt] Tuple (v_1(Current_Sales): 22, v_2(Product_Name): BigMac)

O_3.[(ProcessID: 230) (Member of Process with *Process Id's*) ProductionMgmt] Tuple (v_1(Current_Sales): 22, v_2(Product_Name): MacChicken)

At the end of the Gathering phase the Demonstrator presents in the Result Window all cases in a uniform representation (candidates and their evidence), the classification of the possible kind of conflict, and in cases where the conflict is based on a correspondence assumption of the candidates, the sameness between the candidates.

6.5.4 Syntactic Conflict Detection

Syntactic conflict detection is concerned with ensuring that the candidates are not merely syntactically conflicting. In particular, the transmission and translation of candidates is analysed. The example communication language for inter-agent, and agent-to-source interaction described in the previous Sections (e.g. Section 2.3) is KQML [CHA92]. Some examples using KQML are described to demonstrate how information agents can detect mere syntactic conflicts. The following displays a typical query that an agent (in this example the agent that integrates the ProductionMgmt system) can issue to request information from another information agent (in this example the agent that integrates the BookkeepingDB). This query is taken from Example 1. Comments are attached to each line of this protocol in italics and brackets:

(PACKAGE
:FROM ProductionMgmt-Agent
:TO BookkeepingDB-Agent
:ID A23 *(Identifier of the message generated e.g. by the information agent)*
:COMM block *(This can be either 'block' or 'nonblock')*
:CONTENT *(Beginning of the Message Layer)*
(MSG
:TYPE **query** *(This can, e.g., be query, assert, retract,...)*
:Qualifiers (:number-answers 1) *(An agent can request multiple results)*
:Content-Language Propositional *('Propositional' is the pseudo language used in this research)*
:Content-Ontology (BookkeepingDB) *(Each agents local view is an ontology as well as the enterprise model)*
:Content-Topic (Employee) *(Terms describing the topic in that ontology)*
:Content *(Beginning of the Content Layer which is the actual query)*
(O_i.(v_1(E_Number): 123,v_2(First_Name): ?))
))

Typically, the reply from the agent of the BookkeepingDB would look like this:

(PACKAGE
:FROM BookkeepingDB-Agent
:TO ProductionMgmt-Agent
:ID B24
:COMM block
:In-Response-To A23
:CONTENT (MSG
:TYPE **content-reply** *(a reply to a previously sent query)*

:REQUEST-ID A23 *(The Id of the query)*

:REPLY-NUMBER 1 *(The number of requested results)*

:CONTENT (O_i.(v_1(E_Number):123,v_2(First_Name): Peter))

))

However, in syntactic conflict detection interest is directed at identifying any possible errors that may occur. For example, if an information agent has encountered any communication errors on the KQML level than it expresses these in the following type of message:

(PACKAGE

:FROM BookkeepingDB-Agent

:TO ProductionMgmt-Agent

:ID B24

:COMM block

:In-Response-To A23

:CONTENT (MSG

:TYPE **success-reply**

:VALUE failure *(This is either success or failure)*

:REQUEST-ID A23

:EXPLANATION *(Typically an explanation in English)*

(The ProductionMgmt is currently not accessible.)

))

A success-reply message is typically used when no content-reply can be sent. For example, an information source is currently not available. This kind of message is important to the retrieval process and indicates if the agent has been successful in collecting all possible results from all sources. For conflict detection, however, the syntactic correctness of the results is of interest.

A **content-reply message** may include warnings or error messages that explain any possible difficulties that have occurred while a result has been issued. This information will allow the information agent to rehearse the result in that the agent may request the information again in order to ensure that a result is syntactically correct.

For example, the agent integrating the MarketingEXP may receive a query from the managing agent A requesting the Product_Name of any products that have a Planned_Demand of '40':

O_i.(v_1(Planned_Demand): 40, v_2(Product_Name): ?),

This query may produce the following results from the MarketingEXP:

O_1.[(Essential Characteristic Product_Name: File-o-Fish) (*Essential Characteristics* as in the Enterprise Model) MarketingEXP] Tuple (v_1(Planned_Demand): 40, v_2(Product_Name): File-o-Fish)

O_2.[(Essential Characteristic Product_Name: FishMac) (*Essential Characteristics* as in the Enterprise Model) MarketingEXP] Tuple (v_1(Planned_Demand): 40, v_2(Product_Name): FishMac)

The two results are returned from the MarketingEXP system to the integrating agent. This agent also knows that the objects in the MarketingEXP system are matched to concepts in the enterprise model. It is this agent's 'job' to translate the results with the help of the enterprise ontology. For example, the name 'FishMac' in result O_2 translates into 'File-o-Fish' with synonym definitions provided by the enterprise model. However, it may be the case that the enterprise model is currently not accessible to the integrating agent. Because the agent has two results it will forward these to the requesting (managing) agent despite the fact that the results may not have been translated correctly.

In other words, the agent would not send a 'success-reply' message of the value 'Failure' because it has received two results from the MarketingEXP. It would, however, attach a 'warning' to the result forwarded. Hence, the agent would forward a content-reply message with the results (O_1 and O_2), and a warning such as 'Not checked against the enterprise ontology'.

(PACKAGE
:FROM MarketingEXP-Agent
:TO ProductionMgmt-Agent
:ID B24
:COMM block
:In-Response-To A23
:CONTENT (MSG
:TYPE **Content-reply**
:REQUEST-ID A23
:REPLY-NUMBER 1
:Content
O_1.[(Essential Characteristic Product_Name: File-o-Fish) (*Essential Characteristics* as in the Enterprise Model) MarketingEXP] Tuple (v_1(Planned_Demand): 40, v_2(Product_Name): File-o-Fish)
O_2.[(Essential Characteristic Product_Name: FishMac) (*Essential Characteristics* as in the Enterprise Model) MarketingEXP] Tuple (v_1(Planned_Demand): 40, v_2(Product_Name): FishMac)
('Not checked against the enterprise ontology))
))

The previous example has illustrated that mere syntactic conflicts within the framework of KQML may occur. However, it also becomes obvious that the detection of syntactic conflicts is extremely domain-driven. It has already been outlined in Section 5.3 that this detection phase is very language, protocol and system specific. In this example, the agent needs to be able to

(i) analyse the KQML protocol,

(ii) find the error messages in the Content section of the KQML message, and

(iii) also interpret this message correctly.

Only then is the agent able to analyse the results and, in this example, determine that 'File-o-Fish' and 'FishMac' actually translate into the same thing with the definitions of the enterprise ontology.

The KQML protocol is a typical example of the limited communication in enterprise integration where messages are only of the kind correct ('Content-reply') or not correct ('success-reply' message with the value 'failure'). Currently, information agents fail to further assess their own syntactic result manipulation (transmission and translation). The introduction of a kind of 'warning' or 'caution' message to the KQLM protocol, for example, would enable the information agent to make a comment on any difficulties encountered while information was translated or transmitted (Section 5.3).

However, the described scheme is complete in that it contains all means to detect mere syntactic conflicts based on current research. Future work may facilitate information agents, and possibly other integrated sources, to assess their own results in respect to their transmission and translation. The presented conflict management mechanism enables the information agent to employ such new forms of assessment (e.g., messages of warning).

6.5.5 Semantic Conflict Detection

6.5.5.1 Object Correspondence

The Gathering phase has outlined different degrees of sameness for those candidates that are assumed to be concerned with the same object. However, the degrees Match Equality, Aspect Equality and Arbitrary Equality are very weak. Hence, this section will further investigate these degrees of sameness in order to assess the object correspondence assumption.

Existing Research fails to investigate assumptions about object correspondence when these are weak. Example 1 includes the two Match Equal candidates:

O_1.[(Employee.E_Number = 123) (*User Defined Key* in a relational database) BookkeepingDB] Tuple (v_1(E_Number): 123, v_2(First_Name): Peter)

O_2.[(Surrogate Identifier = 0-507#30) (*Surrogate Identifier* in an object-oriented database) ProductionDB] Tuple (v_1(E_Number):123, v_2(First_Name): Mark)

If these candidates were integrated tightly into an object oriented model (Section 3.3.1) then it would typically be assumed that these candidates are identical. For example, they would be given the same global identifier such as a system generated surrogate in a system such as MKS by Pan and Tenenbaum [PAN91a].

Without any further investigation of Example 1 there is equally much justification to establish the alternative hypothesis about the sameness of the candidates O_1 and O_2 such that:

> These candidates O_1 and O_2 are distinct objects that only share the attributes employee number (E_Number), which happens to be part of the identifier of one candidate.

Following this approach these objects could be integrated into a tightly coupled distributed environment as different objects with distinct surrogate identifiers (e.g. Ahmed *et al.* [AHM91]). However, the novel representation of object identity and sameness developed in this research makes it obvious how weak the basis for the object correspondence is in cases such as this, and allows for a more distinct (accurate) classification.

In the first step of analysing weak notions of object sameness, such as Match Equality, Aspect Equality and Arbitrary Equality, the Demonstrator selects from the Agent Knowledge any information on essential properties (Resource Knowledge). Furthermore,

where the candidates correspond to any concepts in the enterprise model these concepts may have information about any essential characteristics (properties).

These essential properties (and any other properties of the candidates) are retrieved from the local sources. Furthermore, the agent investigates if the candidates are each others closest resemblance in their sources. Section 5.4.1 described how the agent systematically investigates the other candidate's world for the closest resemblance. In the case of Example 1 the attribute First_Name is an essential characteristic according to the agent's Resource Knowledge. Hence, it requests any employee objects from the ProductionDB that have the First_Name 'Peter', and any employee objects from the BookkeepingDB that have the First_Name 'Mark'.

This investigation produces no result, no object called 'Peter', from the ProductionDB. However, the BookkeepingDB has an employee called Mark:

O_3.[(Employee.E_Number = 123) (*User Defined Key* in a relational database) BookkeepingDB] Tuple (v_1(E_Number): 1234, v_2(First_Name): Mark), v_3(NI_Number):24343, v_4(Pay_Per_Hour):6, v_5(Address_No):12);

The candidates are listed with all their properties:

O_1.[(Employee.E_Number = 123) (*User Defined Key* in a relational database) BookkeepingDB] Tuple (v_1(E_Number): 123, v_2(First_Name): Peter, v_3(NI_Number):4567, v_4(Pay_Per_Hour):6, v_5(Address_No):12)

O_2.[(Surrogate Identifier = 0-507#30) (*Surrogate Identifier* in an object-oriented database) ProductionDB] Tuple (v_1(E_Number):123, v_2(First_Name): Mark, v_3(NI_Number):4567, v_4(Body Type): Fat, v_5(Weight): 12 stone)

The agent retrieves from the Agent Knowledge that the candidates are of concept 'Employee' and have the essential properties E_Number and First_Name. The enterprise model defines the essential properties of the concept Employee with E_Number and NI_Number.

Based on this information the information agent can conclude that:

- The candidates share the two essential properties E_Number and NI_Number, but not the First_Name;
- The candidates share all common properties except the First_Name;
- Candidate O_2 is the counterpart of O_1;

- Candidate O_1 closer resembles O_2 than other objects in the BookkeepingDB (world). This includes object O_3 because the latter only shares 1 (First_Name) essential property with the object O_2, where as O_1 shares the E_Number and the NI_Number with O_2.

Following the categorisation in Section 5.4.1 the agent is justified in assuming object correspondence. However, the degree of sameness is no stronger than Match Equality. In order to become Referential Equal the candidates need to be real world counterparts, which cannot be established because the candidates conflict on an essential property. In any case, the agent's evaluation can ensure that the candidates are not a mere semantic mismatch.

The same procedure has been applied to the Aspect Equal candidates described in the previous Section:

O_1.[(Employee.E_Number = 126) (*User Defined Key* in a relational database) BookkeepingDB] Tuple (v_1(Address_No):11, v_2(E_Number): 126, v_3(First_Name):Frank, v_4(NI_Number):5678, v_5(Pay_Per_Hour):6)

O_2.[(Employee.E_Number = 136) (*User Defined Key* in a relational database) BookkeepingDB] Tuple (v_1(Address_No):11, v_2(E_Number): 136, v_3(First_Name): Gary, v_4(NI_Number):45786, v_5(Pay_Per_Hour):3)

Corresponding to the previous example the essential properties are E_Number, NI_Number and First_Name. Hence, the agent can conclude that:

- The candidates share no essential properties;
- The candidates are not counterparts because they are from the same world and are not identical (Same Identifier_Class different Identifier_Object).
- All common attribute classes, except the address (Address_No), are distinct.

Following the categories outlined in Section 5.4.1, these candidates correspond to different objects. Provided these candidates are not conflicting in a way that is not based on this correspondence assumption, the information agent can positively conclude that these candidates are not conflicting.

Existing research on conflict management in enterprise integration, e.g., by Barbuceanu and Fox [BAR94b], fails to detect that these cases cannot conflict because of a lack of object correspondence. For example Barbuceanu and Fox apply conflict resolution mechanism to any case without detecting a conflict. In this example conflict resolution is inadequate as it carries the risk of providing a resolution to a conflict that does not exist.

6.5.5.2 Concept Correspondence

Semantic conflict detection is also concerned with investigating the correspondence of concepts used by the, potentially conflicting candidates. For this step in conflict detection Section 5.4.2 outlined that the agent requires the following Resource Knowledge (Section 2.7):

- Environmental Information on Subtype - Supertype relations, Synonyms, or the correct Form and Time of requests from specific sources;
- Services that may provide relevant expert knowledge.

All this information is gathered in the Demonstrator from the Agent Knowledge. In addition, the agent may investigate expert knowledge such as a human expert, or the Enterprise Model for the information on subtype - supertype relations or synonyms. For example, provided the candidates' propositions of the conflicting attribute class each have a counterpart in the enterprise model. If these objects in the enterprise model are related as subtype - supertypes, or synonyms then the agent can conclude that the candidates are not conflicting but semantically match into the same concept.

An example of how this expert knowledge can detect a semantic mismatch conflict is provided by the results to the following query on a robot's repairs:

O_i.(v_1(Name):Yogi, v_2(Repair_Name):?)
(What are the Repairs of the robot called Yogi?)

This query may result in the following two candidates:

O_1.[(Surrogate = 0-1108#1777) (*Surrogate Identifier* in an object-oriented database) ProductionDB] Tuple (v_1(Name): Yogi, v_2.(Repair_Name): NewWheels)

O_2.[(Robot.Name = Yogi) (*User Defined Key* in a relational database) BookkeepingDB] Tuple (v_1(Name): Yogi, v_2(Repair_Name): NewChassis)

The candidates are Match Equal and possibly are implicitly conflicting over the attribute Repair_Name. However, investigating the two concepts 'NewWheels' and 'NewChassis' in the enterprise model illustrates that these are actually synonyms. In other words, the agent has gathered the information that a synonym exists:

'NewWheels is_a NewChassis'
which is valid for the attribute class Repair_Name of the entity Robot.

With this information the agent can conclude that no conflict exists but that the candidates' properties 'NewWheels' and 'NewChassis' match into the same thing.

Another example based on the same query but for the Robot 'Tim' produces the results:

O_1.[(Surrogate = 0-1308#2061) (*Surrogate Identifier* in an object-oriented database) ProductionDB] Tuple (v_1(Name):Tim, v_2.(Repair_Name): OilChange, ElectricityCheckup, BatteryCheck, Cleaning)

O_2.[(Robot.Name = Tim) (*User Defined Key* in a relational database) BookkeepingDB] Tuple (v_1(Name):Tim, v_2(Repair_Name): Inspection)

These candidates are also Match Equal. However, they are not conflicting because the repairs 'OilChange', 'ElectricityCheckup', 'BatteryCheck' and 'Cleaning' are all covered by the supertype 'Inspection' (defined in the enterprise model and the Agent Knowledge). In other words, every inspection includes an 'OilChange', 'ElectricityCheckup', a 'BatteryCheck' and 'Cleaning'.

A typical form of expert knowledge is advice by human experts. An engineer on the shop floor exists (Service - Resource Knowledge) who has a terminal to which the conflict could be sent. This expert would be able to clear the conflict and answer the question "Do these concepts correspond to the same thing?" in the affirmative. In other words, the candidates semantically express the same thing (concept) which only have different descriptions (i.e. names).

An example for a validation of the semantically correct interpretation of a result is provided by the following query on the Current_Sales for the product MacChicken (O_i.(v_1(Product_Name): MacChicken, v_2(Current_Sales):?)). This query may produce the following Referential Equal results:

O_1.[(ProcessID: 249) (Member of Process with *Process Id's*) ProductionMgmt] Tuple (v_1(Product_Name): MacChicken, v_1(Current_Sales): 300)

O_2.[(Essential Characteristic Product_Name: MacChicken) (*Essential Characteristics* as in the Enterprise Model) MarketingEXP] Tuple (v_1(Product_Name): MacChicken v_2(Current_Sales): 22)

The Current_Sales in the ProductionMgmt system are constantly updated by the shop floor staff (See Appendix D Section D.4 for a system description). The MarketingEXP has a file system with some estimated Current_Sales figures. In effect the quantity of three hundred MacChickens might easily be a typing error. It would be easy to imagine an employee on the shop floor accidentally adding an extra zero to the thirty MacChicken he wanted to order ('30' and the digit '0' forms '300'). Thus, the information agent could verify the result from the human employee (candidate O_1) by sending it back to the shop floor. The employee that has placed this Current_Sales figure is asked to verify whether this order is what he intended. In KQML such a message is sent as a 'Assign-Truth-Value' query to the user interface. The employee would verify this query

with as 'true' or 'not true' and it would be returned to the sender (information agent). This procedure ensures that the semantically correct, or intended, quantity is ordered.

This is a typical case where authority, reliability or role based approaches fail (e.g. Barbuceanu and Fox [BAR94], or role-based approaches by Pan and Tenenbaum [PAN91a]). In other words, the shop floor staff would be given a higher authority, be judged more reliable, or have a superior role then the MarketingEXP system. This may be based on the assumption that the staff are humans, or because they are actually managing the shop floor sales. Their higher authority would verify the three hundred MacChicken as a correct Current_Sales and not solve this conflict's semantic misinterpretation.

The **limitation of result verification** in present systems is the dependence on human experts. With respect to the previous example involving a human expert, it is easy to imagine how future developments of machine experts could provide 'automated validation'. For example, a machine expert system could be developed that has knowledge of commonly made errors when data is inputted. One rule in such a system could be 'It easily happens that a digit (e.g. the zero in figure thirty) is accidentally inputted twice (e.g. the input is three hundred instead of the intended thirty)'. Such an expert might then tell the information agent to request the same information again from the shop floor. Finally, the expert could compare if the initial and the new results are identical, or if the first result was only inputted incorrectly.

Finally, the form and time of the candidates can be investigated to check if candidates use corresponding concepts. Example 2 includes the following three candidates:

O_1.[(ProcessID: 229) (Member of a Process with *Process Id's*) ProductionMgmt] Tuple (v_1(Product_Name): BigMac, v_2(New_Demand): 34)

O_2.[(ProcessID: 229) (Member of a Process with *Process Id's*) ProductionMgmt] Tuple (v_1(Product_Name): BigMac, v_2(Hourly_Demand): 44)

O_3.[(Essential Property Product_Name = BigMac) (*Essential Property is Product_Name)* ProductionMgmt] Tuple (v_1(Product_Name): BigMac, v_2(Current_Hourly_Demand): 39)

It may also be determined from the Agent Knowledge that these candidates are produced by concurrent processes. In other words, the Agent Knowledge contains Environmental Information (Resource Knowledge) on the Form and Time of how to receive semantically correct attributes Current_Hourly_Demand from the ProductionMgmt system, such that:

'The Current_Hourly_Demand may need to be monitored over a short time'.

This information may be based on the fact that the Current_Hourly_Demand is updated by the Hourly_Demand which is actually recalculated periodically. This calculation includes the old Current_Hourly_Demand and the New_Demand to derive the Hourly_Demand. The Hourly_Demand then updates the Current_Hourly_Demand in the ProductionMgmt's internal file system. It follows, that the information agent requests the Current_Hourly_Demand again to ensure that it is not simply a conflict based on retrieving the candidates at a time when it was just recalculated. In this example, the conflict between the candidates O_2 and O_3 is resolved in that the semantically correct result for the Current_Hourly_Demand is '44'.

6.5.6 Admissibility Phase

The final stage of conflict detection considers the admissibility of the conflicting candidates. In other words, the Agent Knowledge includes Admissibility Heuristics which are implemented as Business Rules and Decision-Making Knowledge. Furthermore, Comments (Resource Knowledge) or Integrity Constraints (Resource Knowledge) which may exist to regulate the admissibility of results. All this information is collected by the Demonstrator from the Agent Knowledge.

For example, a new cook in the kitchen may want to check a list of ingredients of the product FishMac with the query:

(O_i.(v_1(Product_Name): FishMac, v_2(Ingredients):Bun, Fish, FishMacSource))

The following conflicting results are returned:

O_1.[(Surrogate = 0-1318#2073) (*Surrogate Identifier* in an object-oriented database) ProductionDB] Tuple (v_1(Product_Name): FishMac, v_2.(Ingredients): Bun, Fish, FishMacSource)

O_2.[(Product.Product = Null) (*User Defined Key* in a relational database) MaterialDB] Tuple (v_1(Product_Name): Null, v_2(Ingredients): Null)

In other words, the MaterialDB claims that the product FishMac does not exist and / or that the ingredients specified in the query are incorrect. The Demonstrator has collected the following Admissibility Heuristic from the Agent Knowledge:

'Results with null values from the MaterialDB are inadmissible'

This heuristic may be installed because the MaterialDB is new and still being completed. Thus, the candidate O_2 is rejected as inadmissible so that no conflict is actually at hand but only one admissible result from the ProductionDB.

6.5.7 Summary Conflict Detection

In the previous Sections on conflict detection any explicit, and any known implicit conflicts between the candidates are detected. The results are investigated in depth. The Result Window in the Demonstrator implementation provides an overview of all the information that is known on the conflict case. In the following all this information is presented for the cases of Example 1 and Example 2.

Example 1:

O_1.[(Employee.E_Number = 123) (*User Defined Key* in a relational database) BookkeepingDB] Tuple (v_1(E_Number): 123, v_2(First_Name): Peter, v_3(NI_Number): 4567, v_4(Pay_Per_Hour): 6, v_5(Address_No): 12)

O_2.[(Surrogate Identifier = 0-507#30) (*Surrogate Identifier* in an object-oriented database) ProductionDB] Tuple (v_1(E_Number):123, v_2(First_Name): Mark, v_3(NI_Number): 4567, v_4(Body Type): Fat, v_5(Weight): 12 stone)

Both candidates are Match Equal and the agent is justified in assuming object correspondence because each resembles the other closer than any other object in its world. Thus, the candidates are assumed to conflict implicitly on the properties (R_k) for the attribute class v_2(First_Name):

$O_1.v_2$(First_Name): Peter $O_2.v_2$(First_Name): Mark

Example 2:

O_1.[(ProcessID: 229) (Member of a Process with *Process Id's*) ProductionMgmt] Tuple (v_1(Product_Name): BigMac, v_2(New_Demand): 34)

O_2.[(ProcessID: 229) (Member of a Process with *Process Id's*) ProductionMgmt] Tuple (v_1(Product_Name): BigMac, v_2(Hourly_Demand): 44)

O_3.[(Essential Property Product_Name = BigMac) (*Essential Property is Product_Name)* ProductionMgmt] Tuple (v_1(Product_Name): BigMac, v_2(Current_Hourly_Demand): 44)

O_4.[(Essential Characteristic Product_Name: BigMac) (*Essential Characteristics* as in the Enterprise Model) MarketingEXP] Tuple (v_1(Product_Name): BigMac, v_2(Current_Demand): 50)

O_5.[(Surrogate Identifier = 0-508#789) (*Surrogate Identifier* in an object-oriented database) ProductionDB] Tuple (v_1(Product_Name): BigMac, v_2(Expected Demand): 45)

The following evidence has been gathered for the candidate O_1:

E_1 = {(Current_Hourly_Demand is more than 5 units larger than Current_Sales ()}
E_2 = {(New_Demand) (44 % Reliability)}

Semantic Conflict Detection has detected that the candidate O_3 actually has a Current_Hourly_Demand of '44' instead of '39' such that no conflict exists between the candidates O_3 and O_2.

Object correspondence is assumed between all candidates based on the following degrees of sameness:

	O_1	O_2	O_3	O_4	O_5
O_2	TPI	-	-	-	-
O_3	RE	RE	-	-	-
O_4	RE	RE	RE	-	-
O_5	ASE / RE	ASE / RE	ME / RE	ME	-

Table 7: Object Correspondence in Example 2

The sameness between candidate O_5 and the other candidates has been strengthened because these candidates are each other's closest resemblance in their worlds, and share the only known essential property. This essential property is the Attribute Product_Name as defined for the set concept Product in the Enterprise Model (This is also defined in the Environmental Information - Resource Knowledge).

The following Generalisations and Counterpart Relations have been identified:

1. G (ist(Product_Name in ProductionMgmt) ⇔ (Product Name in Enterprise Model)
2. C (BigMac in Product_Name ProductionMgmt)(BigMac in Product_Name MarketingEXP)
3. C (BigMac in Product_Name MarketingEXP)(BigMac in Product_Name ProductionMgmt)

In conclusion, six implicit conflicts over the value of the attribute R_k for the attribute class v_2 Current_Demand (as specified in the Global Agent View) have been **detected**:

1. $O_1.v_2$:34 ($O_2.v_2$:44 & $O_3.v_2$:44)
2. $O_4.v_2$:50 ($O_2.v_2$:44 & $O_3.v_2$:44)
3. $O_5.v_2$:45 ($O_2.v_2$:44 & $O_3.v_2$:44)
4. $O_1.v_2$:34 $O_4.v_2$:50
5. $O_1.v_2$:34 $O_5.v_2$:45
6. $O_5.v_2$:45 $O_4.v_2$:50

The remaining two candidates of Example 2 are Match Equal:

O_6.[(Essential Characteristic Product_Name: BigMac) (*Essential Characteristics* as in the Enterprise Model) MarketingEXP] Tuple (v_1(Product_Name): BigMac, v_2(Planned_Hourly_Demand): 55)

O_7.[(Position Number 238481) (*Position Number* address based in Production Table) RobotMgmt] Tuple (v_1(Product_Name): BigMac, v_2(Prod_Quantity): 44)

Further orders for the product 'BigMac' exist in the production table in the RobotMgmt system. Hence, no 'closest resemblance' between the objects in their sources could be established so that their degree of sameness could not be strengthened. The degree of sameness is very weak. The agent questions the object correspondence assumption of the candidates. For this implementation, the Business Rule has been established that the agent can **provisionally** classify this case as an implicit conflict of the case:

$O_6.v_3$(Planned_Hourly_Demand): 55 $O_7.v_4$(Prod_Quantity): 44.

This is provisional, because such a conflict assumes that the candidates are concerned with the same object, here the Product, 'BigMac'. However, this assumption can be established by the information agent in this case.

6.5.8 Credibility

At this opening stage of conflict resolution the information agent needs to investigate the credibility of the conflicting candidates and their sources. Some evidence that includes reliability estimations, is already known from the Gathering Phase. However, no exhaustive search for reliability estimates was undertaken by the agent. Thus, the Demonstrator searches the Agent Knowledge systematically for any direct reliability estimates. Furthermore, it investigates any Environmental Information on the candidate's expertise, roles, or authority that may enable the agent to derive a candidate's reliability. Finally, the agent investigates any Services (Resource Knowledge) that can potentially provide reliability estimates for the candidates but have not yet been investigated. The Demonstrator derives all relevant certainty estimates from this information and illustrates it, together with the Direct Certainty Estimates, in the Agent Knowledge Search window. This includes the evidence E_1, E_2 which has been specified in the Gathering Phase, and evidence E_9 supporting the Hourly_Demand from the ProductionMgmt:

- E_9 = {(Hourly_Demand) (90 % confidence level)}.

In the case of Example 2 this collecting and derivation of certainty estimates includes the following. Environmental Information (Section 2.7) is available from the Agent Knowledge search:

- Each candidate from the ProductionMgmt has the Role 'Calculating the Demand';
- The schema object Hourly_Demand from the ProductionMgmt system has the Authority specification: 'High Authority' (because it determines production supervised by the RobotMgmt)
- Current_Demand from the MarketingEXP system has the Authority specification: 'No Authority'
- The Expertise of the candidates from the MarketingEXP is: 'Planning'

These characteristics are used by the following heuristic available in the agent's Organisational Knowledge (Credibility Heuristics):

- "Calculating the Demand' is a Role that makes a system 'very credible" (E_4).

This heuristic, for example, makes the candidates O_1, O_2, and O_3 in Example 2 'very credible'. The expertise of the MarketingEXP makes it very credible based on the heuristic:

- 'A system's Expertise (e.g. 'Planning') makes it probable' (E_7).

Furthermore, the following heuristic makes the candidate O_2 very credible:

- 'A system with a 'High Authority' (Authority) is 'very credible" (E_5).

In addition, the succeeding heuristic makes the candidate O_4 'not credible':

- 'A system with 'No Authority' (Authority) is 'not credible" (E_6).

Finally, the following heuristic on the Group production systems supports candidate O_1:

- 'All production systems are, in principle, 'very reliable" (E_3).

These credibility assessments will be related to the candidates as evidence. In other words the existing evidence, e.g., the services defining the certainty of the first two candidates (O_1 and O_2) in Example 1, will be completed by the new evidence on the candidates credibility. The candidates in Example 2 have, therefore, got the following evidence (including E_1 which has no certanty estimate) in the form of 'E_m={(Formula)(Certainty)}':

O_1 has E_1 = {(Current_Hourly_Demand is more than 5 units larger than Current_Sales ()};

O_1 has E_2 = {(New_Demand) (44 % Reliability)};

O_1 has E_4 = {('Calculating the Demand' is a Role that makes a system)('very credible)};

O_1 has E_3 = {(All Production Systems are) (very reliable)};

O_2 has E_9 = {(Hourly_Demand) (90 % confidence level)};

O2 has E_3 = {(All Production Systems are) (very reliable)};

O_2 has E_4 = {('Calculating the Demand' is a Role that makes a system)('very credible)};

O_2 has E_5 = {(A system with a High Authority is)(very credible)};

O_3 has E_4 = {('Calculating the Demand' is a Role that makes a system)('very credible)};

O_3 has E_3 = {(All Production Systems are) (very reliable)};

O_4 has E_6 = {(A system with No Authority is) (not credible)};

O_4 has E_7 = {(A system's Expertise (Planning) makes it) (probable)}.

O_5 has E_8 = {(An object-oriented database system) (is typically very reliable)}.

Furthermore, candidate O6 has a credibility assessment such that:

O_6 has E_7 = {(A system's Expertise (Planning) makes it) (probable)}.

However, candidate O_4 now has evidence that supports (E_8) and another (E_6) that refutes its credibility. Such conflicts of credibility will be subject to the last three steps of the conflict resolution phase (Ranking, New Alternatives and Negotiation).

Heterogeneity of Credibility Estimates and Evidence:

The previous example provides a good overview of the heterogeneity of credibility estimates that exist in typical enterprise integration environments, including:

- Numerical (e.g. E_2 '44% reliability') and non numerical certainty estimates (e.g. E_3 'very reliable');
- Direct estimates (e.g. E_2 '(New_Demand) (44 % reliable)') that are typically provided by the information sources themselves, and estimates that are derived by the information agent (E_4 '(Calculating the demand is a Role that makes a system) (very credible)') ;
- Different kinds of derived estimates exist, e.g., based on the characteristics Role (E_4), Authority (E_5) and Expertise (E_7).

It can, in principle, be envisaged that evidence is probabilistic and that the agent is justified in assuming that these estimates can be combined, e.g., by the Dempster-Shafer rule of combination [SHA76] [DEM67] (Section 3.5.2 and 5.10.1). However, typically the information agent also has evidence containing other than probabilistic certainty estimates, e.g., evidence E_2 ('44% reliability') warrants O_1 and E_3 ('very reliable') warrants candidateO_2.

For the information agent to rank more heterogeneous estimates such as:

E_3 ('very reliable') is more believable than E_2 ('44% reliability')

it needs to know that the derived estimate 'very reliable' is more believable than the estimate '44 % reliable' from source ProductionMgmt. This information is provided by heuristics such as described in Sections 5.10.1 and 6.5.11.

Finally, evidence such as E_1 (('Current_Hourly_Demand is more than 5 units larger than Current_Sales')()) may lack a certainty estimate. A ranking that is also based on this kind of evidence cannot be based on the pure management of uncertainty estimates but needs to be based on domain expertise. The approach by An, Bell and Hughes [AN 93], for example, can be used to combine evidence that is not necessarily numerical (Section 3.5.2). In other words, a domain specialist may identify that evidence E_1 (('Current_Hourly_Demand is more than 5 units larger than Current_Sales')()) is more believable than E_4 (('Calculating the Demand' is a Role that makes a system')('very credible')). However, the agent needs to be a domain expert of the ProductionMgmt system, for example:

To interpret Evidence E_1 such that the Current_Hourly_Demand is calculated based on the Current_Sales which means that candidate O_1 is a more adequate estimate of the Current_Demand than O_2 which also includes Urgent_Requests (Section D4).

Thus, these evidential reasoning mechanisms may be applied by domain experts in the next phase of Domain-Specific Problem-Solving.

For example, a human expert as described in the following section may be able to make a judgement based on any kind of evidence. Furthermore, a domain expert may also make a judgement based on the candidates themselves and not just based on the certainty estimates that support or refute these candidates (Sections 5.8. and 6.5.9).

In conclusion, existing methods of uncertainty management can be incorporated in the design of ranking heuristics as described in Sections 5.10.1 and 6.5.11. However, the use of uncertainty management methods is typically limited because of the heterogeneity of the conflicting certainty estimates.

6.5.9 Domain-Specific Problem-Solving

In the second step of conflict resolution the information agent applies domain-specific strategies, such as provided by domain problem-solving or expert knowledge, to the conflict. The Demonstrator searches the Agent Knowledge for information on strategies for Problem-Solving or Services (Resource Knowledge) that are able to resolve the current conflict of candidates. For example, the query 'What is the waiting time of the product BigMac?' may translate into the query:

O_i.(v_1(Product_Name):BigMac, v_2(Production_Time): ?)

The following two candidates result from this query:

O_1.[(Essential Characteristic Product_Name: BigMac) (*Essential Characteristics* as in the Enterprise Model) MarketingEXP] Tuple (v_1(Product_Name): BigMac, v_2(Waiting_Time): 4),

O_2.[(Surrogate Identifier 0-508#789) (*Surrogate Identifier* in an object-oriented database) ProductionDB] Tuple (v_1(Product_Name): BigMac, v_2(Waiting_Time): 5)

In other words, the MarketingEXP system has a Waiting_Time of five minutes because it expects that customers will wait that long for the product BigMac. The RobotMgmt system has a Waiting_Time of four minutes, which could, for example, be based on an average of the actual production time of the robots that cook the product BigMac. In any case, this is a genuine implicit conflict between two candidates.

How could the information agent identify domain-specific problem-solving expertise to resolve this conflict? Existing research is limited in that it integrates expert advice systems as information sources and not as 'problem solvers', which may be conducted by the information agent to resolve a conflict. It is relatively easy to imagine how a human expert would resolve such a conflict. Implementations of such domain-specific expert resolution by humans is, for example, described by Steiner *et al.* [STE90], or Pan and Tenenbaum [PAN91a]. Section 5.8, however, has introduced a possible way to integrate human and machine advice systems as 'problem solvers'. This concept is implemented in the Demonstrator.

This implementation shows how a software program, the RobotMgmt, can provide expert advice to conflict resolution. Appendix D Section D.5 describes how the RobotMgmt receives orders from the ProductionMgmt system and assigns robots in the kitchen to process these orders. By doing this the RobotMgmt actually provides local problem-solving functionality. Conflicts over the correct Waiting_Time for a product can be answered precisely. In other words, the system's production table lists all the products

currently produced, the robots that carry out these tasks, and the time it takes until the products are produced. This provides the 'actual' Waiting_Time.

The identification of a relevant domain specific-problem solver is implemented in the Demonstrator such that schema objects in the information agent's view on the integrated system include information on relevant problem-solving experts. In other words, an agent A may retrieve information concerning the schema object 'Current_Sales' from its local source ProductionMgmt. Provided a conflict occurs including this result then the following heuristic indicates a relevant problem-solving expert:

'A conflict including the Current_Sales can be determined by the RobotMgmt'.

In the previous example, the following candidate can be received from the RobotMgmt system:

O_1.[(Position Number 238481) ((*Position Number* address based in Production Table) RobotMgmt] Tuple (v_1(Product_Name): BigMac, v_2(Production_Time): 5)

Provided the RobotMgmt system is integrated as a domain expert then it could provide a resolution in a sentence such as:

The actual Waiting_Time for the product BigMac is currently: 5 Minutes

However, this is only a solution to the conflict if the RobotMgmt system's resolution is Principle Rational (Section 2.6). In other words, the agent needs to know that resolving this domain-specific conflict with the decision from the RobotMgmt provides a solution that is rational to any potential client of the integration environment. Precisely that is defined implicitly in the Agent Knowledge which provides the agent with the option of applying this expert knowledge. Hence, the resolution to the conflict over the correct Waiting_Time may be formulated as follows:

O_1.[(Essential Characteristic Product_Name: BigMac) (*Essential Characteristics* as in the Enterprise Model) MarketingEXP] Tuple (v_1(Product_Name): BigMac, v_2(Waiting_Time): 4),

O_2.[(Surrogate Identifier 0-508#789) (*Surrogate Identifier* in an object-oriented database) ProductionDB] Tuple (v_1(Product_Name): BigMac, v_2(Waiting_Time): 5)

$\Rightarrow$ The actual Waiting_Time for the product BigMac is currently: 5 Minutes

Limitations and Future Work on Domain Experts

In this research only one system provides domain expert resolution. In the light of this case study many more experts would be able to resolve more conflicts. For example, ideally an expert advice system on 'Current_Demand' figures would be able to resolve the conflict in Example 2. Further research on machine advice systems that employ reasoning methods such as described in Sections 3.5.2 and 3.5.3, and the further development of advise systems to be integrated into existing sharing environments will provide more domain-specific conflict resolution [HEW91] [BRO89].[1]

This research presents one possible way how agents could identify domain-specific problem-solvers. However, a large number of advice systems might require more sophisticated mechanisms for the information agent to identify the best, available domain expert for the conflict at hand (Section 5.8).

Existing integration environments, including this research, assume that the Agent Knowledge on domain-expertise is defined by a system administrator. This is potentially a complex, laborious task in large integration environments. Furthermore, conflict resolution is potentially incomplete if the information agent's knowledge of all existing domain-specific problem-solving capabilities is incomplete. In other words, a lack of Resource Knowledge (Services and capabilities of Problem-Solving) may prevent the information agent from detecting all existing resolution mechanisms, and / or associating them correctly with their domain-dependent conflicts.

The presented implementation has been built with a windows-based tool to define the Agent Knowledge. This is a first step to making the implementation and maintenance of Agent Knowledge manageable in large environments. However, future work on ways to define the Agent Knowledge or ways in which the agents could identify domain-experts automatically are needed. For example, new advice systems could automatically define themselves in a standardised form to the agent. The agent could then try to automatically match the resolution procedure to any schema information it has and place a reference if appropriate.

The mechanism is, however, still Principle Rational (Section 2.6). It applies all available rational strategies, that is all strategies 'known' to the agent by its Agent Knowledge, to the conflict in a way that is systematic, complete and adequate. In case no resolution is possible with the 'known' domain-specific expertise then the agent has to apply less specific resolution strategies as described in the next Section.

6.5.10 Scientific, Domain-Specific Heuristics

In the previous step of conflict resolution, domain-specific strategies and heuristics were applied to resolve the conflict. Where the conflict could not be solved (e.g. no suitable domain-specific strategies exist) general scientific heuristics are applied. Existing research such as the CYCCESS approach provide tools to check if a candidate is correct under general scientific rules (Section 5.9 and Guha and Lenat [GUH94]). The information agent has a conceptually similar approach based on its enterprise model and, in addition, on Scientific Heuristics collected by the Demonstrator from the Agent Knowledge.

An example of how the information agent can use its enterprise model to resolve conflicts is demonstrated by the following example query on the cost of the robot Yogi's repair called 'NewEnergySystem':

O_i.(v_1(Name):Yogi, v_2(Repair_Name): NewEnergySystem, v_3(Repair_Cost):?)

This query may result in the following two candidates:

O_1.[(Surrogate = 0-1416#2209) (*Surrogate Identifier* in an object-oriented database) ProductionDB] Tuple (v_1(Name): Yogi, v_2(Repair): NewEnergySystem, v_3(Repair_Cost): 0)

O_2.[(Repair.Number = 2) (*User Defined Key* in a relational database) BookkeepingDB] Tuple (v_1(Robot.Name): Yogi, v_2(Repair): NewEnergySystem, v_3(Repair_Cost): 10,000)

In other words, the RobotDB has information about a repair called 'NewEnergySystem' for the robot called 'Yogi'. However, the price for this repair is zero. The BookkeepingDB also has information about this repair but it has a price of ten thousand (pounds). The information agent can find a resolution to this conflict in the enterprise model. It can identify that the concept Repair_Cost has a Relation that defines that Repair_Costs must be higher than zero. In the same way, the following Scientific Heuristic from the Agent Knowledge could let the information agent conclude that a cost of zero is invalid:

'Repair_Costs must be larger than 0'

This example forms a good basis for demonstrating the linitations of scientific or general heuristics. The previous example seems to be right at first sight. However, if it is assumed that some repairs might be undertaken within the warranty of a robot it is possible to imagine the case of a Repair_Cost of '0'. In conclusion, 'scientific' reasoning is

limited to those Scientific Heuristics and definitions in the enterprise model that have assent throughout the enterprise, for all potential clients of the integration environment.

The following will provide a brief discussion of the question: **'Should the information agent apply domain specific resolution strategies before less domain specific ones?'**

The domain-specific conflict resolution example in the previous Section included two candidates that conflict over the correct value of the Waiting_Time of a product BigMac. The following resolution was provided:

'The actual Waiting_Time for the product BigMac is currently: 5 Minutes.'

This solution is provided by an expert that can ensure that its result and resolution is adequate (Principle Rational as in Section 2.6). In this case, the RobotMgmt system actually determines the current Waiting_Time for a given product. In comparison to general scientific heuristics as introduced in this section, more domain-specific resolution is:

- More difficult to find (only few expert advice systems exist, typically only human advice for conflict resolution is realised in existing integration environments); but
- It is less problematic to use than less domain-specific, Scientific Heuristics.

It can be concluded that in case multiple resolution strategies are available for a conflict it is preferable to apply the most domain-specific resolution. The answer to the previous question is, therefore, affirmative. The information agent should apply domain specific resolution strategies first, before applying less domain-specific ones.

Furthermore, this section includes a resolution based on general Scientific Heuristics. It has been elaborated that a **potential limitation** of this resolution strategy is the definition of heuristics that can be applied to any domain specific information (candidates and their evidence). The example presented here is representative for current approaches in enterprise integration. However, it can also be easily imagined that it is very complex and typically dependent on human expert definitions. Section 5.9 has proposed solutions in the form of learning, methods for the definition and maintenance of common knowledge-bases or scientific-heuristics, and current efforts to make common knowledge-bases widely available.

6.5.11 Reliability - Ranking

Judgement on the reliability of conflicting candidates is concerned with Domain-Independent Evaluation. It only investigates the reliability or certainty estimations of the candidates. In other words, at least one candidate needs to have evidence that specifies its certainty or no resolution based on reliability is possible. Most heuristics, however, require that more than one conflicting candidate has certainty estimates. Evidence is formally specified in the Certainty Estimate variable which may be further specified in the Formula variable:

E_m = {(Formula)(Certainty Estimate)}

Section 5.10.1 outlined that Ranking and Judgement Heuristics relate Specific Certainty Estimates, Related Certainty Estimates and General Certainty Estimates. The Demonstrator selects all relevant Judgement and Ranking heuristics from the Agent Knowledge starting with those relating specific estimates, then related estimates, and finally it searches for general estimates. For pragmatic reasons this agent model only has Judgement and Ranking Heuristics (no separate Ranking and Judgement Heuristics).

The certainty estimates for the Example 2 are outlined in Section 6.5.8. For example, the following Ranking and Judgement Heuristic for specific certainty estimates is applicable to the evidence E_9 and E_6 from Example 2:

If Evidence1 (Hourly_Demand of the ProductionMgmt) (Certainty Estimate: 90% confidence level))

And Evidence2 (Current_Demand from MarketingEXP) (Certainty Estimate: Not credible based on the candidate's authority)

Then judge in favour of the candidate with the first evidence.

It follows for the evidence E_2 and E_6 that:

O_2 has E_9 = {(Hourly_Demand) (90 % confidence level)};

O_4 has E_6 = {(A system with No Authority is) (not credible)};
$\Rightarrow$ refute O_4

Based on this heuristic the information agent can evaluate the evidence E_8 and E_6 and rank the candidates such that O_2 is believed and O_4 is refuted. However, in Example 2 further evidence, that is certainty estimates, exist. The following Ranking and Judgement Heuristic is based on the related certainty estimates from the ProductionMgmt and the MarketingEXP:

1. ProductionMgmt 90 % confidence ⇔ MarketingEXP certain;
2. ProductionMgmt 70 % confidence ⇔ MarketingEXP probable;
3. ProductionMgmt 50 % confidence ⇔ MarketingEXP possible;
4. ProductionMgmt > 50 % confidence ⇔ MarketingEXP unreliable.

⇒ Judge in favour of the higher class of certainty (1 to 4). (The certainty estimates are rankable so that the first level is certain and the next lower level less certain, etc. Judgement can be made in favour of the candidate with the higher certainty level.)

It follows for the evidence in Example 2:

O_2 has E_9 = {(Hourly_Demand) (90 % confidence level)}
O_4 has E_7 = {(A system's Expertise (Planning) makes it) (probable)}
⇒ refute O_4

O_1 has E_2 = {(New_Demand) (44 % Reliability)};
O_4 has E_7 = {(A system's Expertise (Planning) makes it) (probable)}
⇒ refute O_1

In other words, evidence E_9 has a higher level of certainty than E_7 according to the Related Certainty Estimate. The certainty estimate of E_9 is from the ProductionMgmt and matches the highest level of certainty. The certainty estimate ('probable') in E_7 is derived by the information agent, based on the systems expertise, and therefore matches the second level of certainty. In addition the evidence E2 and E7 can be related in the same way such that object O_1 is not as reliable as O_4. Based on this evidence, the information agent can reach a judgement in two conflict cases:

- In favour of the candidates O_2 to refute O_4; and
- In favour of candidate O_4 and refute O_1.

In contrast to Related Certainty Estimates, General Certainty Estimates are universally accepted, where 'universal' may mean within the enterprise including all potential clients of the integration environment (Section 5.10.1). A ranking order is defined for Groups of certainty estimates. In other words, certainty estimates, such as 'very reliable' from the ProductionMgmt system, are related to a certainty level in the general certainty table, for example the following may be included:

Certainty Level 1: 'Very credible' (for results from ProductionMgmt)
Certainty Level 2: .....
Certainty Level 3: 'Probable' (for results from MarketingEXP).

This hierarchy is the basis for relating the following evidence:

O_1 has E_4 = {('Calculating the Demand' is a Role that makes a system)(very credible)};
O_4 has E_7 = {(A system's Expertise (Planning) makes it) (probable)}.
$\Rightarrow$ refute O_4

O_2 has E_4 = {('Calculating the Demand' is a Role that makes a system)(very credible)};
O_4 has E_7 = {(A system's Expertise (Planning) makes it) (probable)}.
$\Rightarrow$ refute O_4

O_3 has E_4 = {('Calculating the Demand' is a Role that makes a system)(very credible)};
O_4 has E_7 = {(A system's Expertise (Planning) makes it) (probable)}.
$\Rightarrow$ refute O_4

In other words, E_4 matches the first certainty level in that it has the certainty estimate 'very credible' for a result from the ProductionMgmt. E_7 includes the certainty 'probable' for a result from the MarketingEXP. The hierarchical order in this case of Ranking and Judgement Heuristics enables the agent to conclude that it should make a judgement in favour of the candidates supported by evidence E_4 (O_1, O_2, O_3), because E4 is in a higher certainty level than candidate O_4's evidence E_7.

In conclusion, determinate ranking and judgement has been made on the conflicting candidates of Example 2 such that the candidates O_2 and O_3 are more reliable than O_4 and a judgement is possible to refute O_4 in these cases. Section 5.10.1 outlined that ranking is **limited** by the heterogeneity of the evidence and their credibility estimates (Section 5.7.2, 5.10.1 and for this case study in 6.5.8). In addition, Section 5.10.1 sketched that the Ranking and Judgement Heuristics are typically incomplete. Thus, in this case study a lack of heuristics has produced incommensurate candidates for the following four conflicts:

1. O_1 and (O_2 and O_3);
2. O_1 and O_5;
3. (O_2 and O_3) and O_5;
4. O_4 and O_5.

An indeterminate ranking situation has occurred between the candidates O_1 and O_4. Evidence E_2 (44 % reliable) and E_7 (probable) make the agent refute O_1 where as E_7 (probable) and E_4(very credible) make the agent refute O_4 (O_1 and O_4 are indeterminate). The following Section will describe how these incommensurate and indeterminate cases may be resolved by a New Alternative.

6.5.12 Reliability - New Alternatives

For cases of incommensurate and indeterminate ranking described above, new alternatives can be proposed by the information agent based on its own Alternative Ranking and Judgement Heuristics. The form of these heuristics is identical to the Ranking and Judgement Heuristics for General Certainty Estimates. In other words, the agent has multiple classes of hierarchically ordered levels of certainty estimates. Each certainty estimate for results from a specific sources (or Group) relates to a ranking level in the Alternative Ranking Scheme. The Demonstrator retrieves all relevant Alternative Rankings based on these new alternatives for all estimates of the conflicting candidates. This produces a New Alternative solution to the conflict provided the Alternative Heuristics can meet the estimates of the conflicting candidates.

In Example 2 the New Alternative ranking would propose to rank the candidate O_5 and its evidence E_8, over the candidate O_1 and its evidence E_2:

O_1 has E_2 = {(New_Demand) (44 % Reliability)};
O_5 has E_8 = {(An object-oriented database system) (is typically very reliable)}.
$\Rightarrow$ refute O_1

In contrast to proposing a ranking based on the agent's Alternative Ranking Heuristics the agent may be able to define a New Alternative (i.e. a compromise) to incommensurate ranking cases based on the circumstantial information of the candidates. Circumstantial Information is defined as Environmental Information (Resource Knowledge) in the Agent Knowledge. The Demonstrator specifically searches the Agent Knowledge for the relevant information on the candidates roles, expertise and authority. A compromise can be developed in Example 2 such that incommensurate ranking of the candidates O_1 and O_2 can be redefined in the following way:

O_1 has E_1 = {(Current_Hourly_Demand is more than 5 units larger than Current_Sales ()};
O_1 has E_2 = {(New_Demand) (44 % Reliability)};
O_1 has E_4 = {('Calculating the Demand' is a Role that makes a system)(very credible)};

O_2 has E_9 = {(Hourly_Demand) (90 % confidence level)};
O2 has E_3 = {(All Production Systems are) (very reliable)};
O_2 has E_4 = {('Calculating the Demand' is a Role that makes a system)(very credible)};
O_2 has E_5 = {(A system with a High Authority is)(very credible)};

The investigation of the candidates' (O_1 and O_2) Environmental Information produces that both candidates are from the ProductionMgmt system and have the same Role ('Calculating the Demand'). However, O_2 has a high authority 'because it determines the production supervised by the RobotMgmt' (Section 6.5.8 introducing Authority to derive certainty estimates). Hence, evidence E_5 can be made more specific such as:

O_2 has E_5 = {(A system with a High Authority is)(very credible **on determining the Production in the RobotMgmt**)};

It is then possible to propose this compromise such that O_1 is relevant for Current_Demand in the ProductionMgmt, and O_2 determines the result from the ProductionMgmt, which is relevant for the Production (in the RobotMgmt system,).

Further compromises can be developed in the same way for the other candidates of Example 2 and as a result the following compromise would resolve the conflicts between the candidates:

Candidate O_1 New_Demand is relevant to calculate the Current_Demand from the point of view of managing the production (ProductionMgmt).

Candidates O_2 Hourly_Demand, and O_3 Current_Hourly_Demand determine the production in the RobotMgmt system.

Candidate O_4 Current_Demand is relevant for the Current_Demand in the Marketing Department.

Candidate O_5 Expected Demand is a statistical value for the long term prediction by the production database ProductionDB.

However, up to this point in the conflict management, the agent has only received a compromise that is a suggestion, which may lead to a conflict resolution. In other words, this compromise needs to be negotiated as described in the next section.

6.5.13 Reliability - Negotiation

The Negotiation phase presents the alternative solution to the client, typically a decision maker. This presentation includes all candidates, their evidence, any information gathered for these candidates, their degrees of sameness, and any resolutions to conflicts among the candidates. In case no compromise has been developed (e.g., the candidates have no certainty estimates) the No Solution stage presents all the information gathered on the conflict case. However, in the previous example a compromise was found and the Negotiation Phase presents not only all information gathered on the conflict but also the compromise.

In the case of Example 2, the following information can be listed as an assessment and a compromise presented to a client of the integration environment:

Conflict Detection: All the information from the conflict detection as described in Section 6.5.7 (pp. 254-255) and presented in the Demonstrator's Result Window (the candidates, their evidence, the kind of conflict, possibly the candidates' sameness, etc.)

Conflict Resolved:

Refuted : $O_4.v_2$ (Current_Demand):50

Supported: $O_2.v_2$(Hourly_Demand):44 and $O_3.v_2$(Current_Hourly_Demand):44)

based on:

O_2 has E_4 = {('Calculating the Demand' is a Role that makes a system)(very credible)};

O_4 has E_7 = {(A system's Expertise (Planning) makes it) (probable)}.

$\Rightarrow$ refute O_4

O_3 has E_4 = {('Calculating the Demand' is a Role that makes a system)(very credible)};

O_4 has E_7 = {(A system's Expertise (Planning) makes it) (probable)}.

$\Rightarrow$ refute O_4

O_2 has E_9 = {(Hourly_Demand) (90 % confidence level)};

O_4 has E_6 = {(A system with No Authority is) (not credible)};

$\Rightarrow$ refute O_4

The New Alternative

1. New Alternative Ranking and Judgement:

O_1 has E_2 = {(New_Demand) (44 % Reliability)};

O_5 has E_8 = {(An object-oriented database system) (is typically very reliable)}.

$\Rightarrow$ refute O_1

2. Compromise:

Candidate O_1 New_Demand is relevant to calculate the Current_Demand from the point of view of the production management ProductionMgmt.

Candidate O_2 Hourly_Demand and

Candidate O_3 Current_Hourly_Demand determine the Production in the RobotMgmt system.

Candidate O_4 Current_Demand is relevant for the Current_Demand in the Marketing Department.

Candidate O_5 Expected Demand is a statistical value for the long term prediction by the production database ProductionDB.

In Example 2 six implicit conflicts over the value of the attribute R_k for the attribute class v_2 Current_Demand (as specified in the Global Agent View) have been detected in the Conflict Detection Phase (Section 6.5.7 p. 255). The previous paragraphs of this section have described the resolution of these conflicts. The following will provide an overview of how these conflicts have been resolved:

1.	$O_1.v_2$:34	($O_2.v_2$:44 & $O_3.v_2$:44)	Compromise
2.	$O_4.v_2$:50	($O_2.v_2$:44 & $O_3.v_2$:44)	Resolved by refuting O_4
3.	$O_5.v_2$:45	($O_2.v_2$:44 & $O_3.v_2$:44)	Compromise
4.	$O_1.v_2$:34	$O_4.v_2$:50	Compromise
5.	$O_1.v_2$:34	$O_5.v_2$:45	New Alternative: Refute O_1
6.	$O_5.v_2$:45	$O_4.v_2$:50	Compromise

Section 5.10.3 states that a **potential limitation** of this resolution phase is that the developed solution is not embedded by a further scheme for learning. In other words, the alternative solution developed above for Example 2 could be presented to a domain expert, e.g. a decision maker. This expert could then comment on the applicability of this solution. For example, the usefulness of the compromises developed or the applicability of the New Alternative Ranking. These comments could be used by the information agent to rank its Alternative Heuristics. Furthermore, such a domain expert could also propose new Alternative Ranking heuristics which might then be 'learned' by the information agent.

6.6 Critical Evaluation of Case Study

The case study has demonstrated the functionality of the conflict detection and resolution mechanism. It has proved that the mechanism detects any 'known' conflict and demonstrated which implicit conflicts are 'known' by an information agent. Furthermore, it is clearly shown how difficult and complex it is to detect implicit conflicts.

In particular it has been revealed by example that domain level resolution procedures provide expert resolution that is

(i) very well suited as a source of Principle Rational (Section 2.6) conflict detection, but

(ii) it is difficult to identify and to apply to conflict resolution by information agents.

Current integration environments are limited in the way they apply domain expertise. Furthermore, future enterprise integration environments (e.g. also called 'Future Intelligent Information Systems' [BRO89]) may improve the integration of many more human and specifically machine domain experts [HEW91]. This may ultimately provide conflict resolution that will be based mainly on domain expert resolution. Less domain level resolution procedures have been shown to be able to resolve conflicts adequately. However, domain level resolution, once it is identified and applied to the conflict, is much less problematic to use as a form of conflict resolution and it has much less chance of being inadequate. For example, it has been shown that general, scientific heuristics that can be applied to any domain information appropriately, are extremely difficult to define. Furthermore, Domain-Independent Evaluation, e.g., based on reliability or certainty estimates, carries much more risk in providing semantically inappropriate solutions.

The shortcomings of judgement based on credibility assessments of the candidates have been made explicit. Gathering reliable certainty estimates is a very difficult task for information agents and deriving them from circumstantial information such as the expertise, role or authority has been demonstrated. This is a well established way to determine reliability and to solve conflicts [BAR94]. However, heterogeneous certainty estimates are only comparable if the agent has extensive Organisational Knowledge. The concepts for reliability measures are very heterogeneous, e.g., derived from the role of an information system or its essential characteristics. This research has demonstrated how difficult it is to define heuristics that relate these certainty estimates. However, previous research described in Section 3.3 relies solely on credibility as a measure for resolving

conflicts. It fails to even investigate domain-specific resolution strategies that are, as shown in this research, appropriate to resolve these conflicts.

Example 2 of this case study has investigated all steps of the conflict detection and resolution mechanism. The final resolution is a New Alternative that is, the conflicting candidates are redefined in order to find a compromise. This redefinition is based on circumstantial information, such as the roles, or expertise of the sources of origin. For example, the first two candidates of this example are :

O_1.[(ProcessID: 229) (Member of a Process with *Process Id's*) ProductionMgmt] Tuple (v_1(Product_Name): BigMac, v_2(New_Demand): 34)

O_2.[(ProcessID: 229) (Member of a Process with *Process Id's*) ProductionMgmt] Tuple (v_1(Product_Name): BigMac, v_2(Hourly_Demand): 44)

They conflict because the two New_Demand and the Hourly_Demand are both generalised into the concept Current_Demand in the Global Agent View. Common sense would intuitively suggest to investigate the 'meaning' of the Hourly_Demand and the New_Demand. This process is included in the Semantic Conflict detection phase as an investigation into concept correspondence (Section 6.5.5.2). Hence, thorough conflict detection, as proposed by this research, will make obsolete most conflict resolution proposed by existing research in enterprise integration, e.g. Barbuceanu and Fox [BAR94]. However, further effort to improve the information agent's detection capabilities promises to improve its conflict management capabilities significantly (see also Future Work in Section 7.4).

The approach shows detailed conflict detection and rational conflict resolution. The results of conflict detection are explicitly available to a decision maker. It includes the candidates, their evidence, possibly an evaluation of the candidate's object correspondence, and the kind of conflict. In addition, conflict resolution provides an explicit explanation of any judgement based on credibility.

Some pragmatic assumptions in the implementation and demonstration of the environment have been made in order to focus on conflict detection and resolution. In particular the Demonstrator assumes consistent Agent Knowledge. Thus, these assumptions are representative of the potential **limitations** of current implementations of information agents. In other words, conflicts could potentially occur, for example, between Admissibility Heuristics or Reliability Heuristics, and Comments in the Global Agent View.

A potential solution to inconsistent Agent Knowledge could be agent learning and adaptation. In addition, learning, adaptation and reuse of once defined heuristics could potentially overcome the strong dependence of current research in enterprise integration on human, expert definitions of heuristics and other semantic information. Only the anchor points are briefly mentioned (e.g., Section 5.4.2, 5.9 and 5.10.3). The field is outside the scope of this research though it is closely related to conflict detection and resolution. Furthermore, future research could be directed at providing sound guidelines and methods for the definition and maintenance of the Agent Knowledge and the Enterprise Model. A step into this direction is, for example presented for the CYC knowledge-base by Lenat [LEN90].

Future research could be directed at negotiation schemes for information agents to exchange incomplete, possibly inconsistent Agent Knowledge. Schemes for such negotiation have been presented in the area of Distributed Artificial Intelligence (Section 3.4 and Appendix A). For example, the scheme by Laärsi *et al.* [LAÄ92] is based on the phases proposal, critique, explanation and possible resolution heuristics. Such a scheme could be adapted to enable agents to propose Agent Knowledge to other agents. These might then be able to critique the proposal if they detect that the proposal is inconsistent to their own Agent Knowledge.

6.7 Conclusion on Evaluation

Section 6.1.2 outlined the aim of this evaluation. Accordingly the evaluation has two parts:

1. The implementation of the integration environment and
2. The case study.

The first part showed how information in heterogeneous information sources can be inconsistent within an information sources and across systems with a demonstration of the implementation of the Agent Knowledge. Finally, the Demonstrator showed how the conflict detection and resolution mechanism, outlined in Chapter 5, could be implemented. It follows that the case study illustrated that a rational conflict detection and resolution mechanism is functional in enterprise integration environments.

Furthermore, it was made explicit why domain level resolution should be applied before the application of a more general mechanism. The implementation demonstrated that any rational strategy could be incorporated in this general scheme. The case study showed some example cases that were managed within the conflict detection and resolution mechanism in a way that is potentially rational to any client of the integration environment.

It may thus be concluded that the aims of this evaluation outlined in the opening Section 6.1.2 on the methodology of this evaluation, have been met.

6.8 Chapter Summary

First the method of evaluation was sketched. It is divided into the two sections (i) introduction of the integration environment with an example implementation, and (ii) a case study to demonstrate and evaluate the conflict detection and resolution mechanism.

The integration environment includes an object-oriented database, a relational database, a rule-based system, and other standard software systems such as a production control system or a production management system. Furthermore, a small simulation of an enterprise model is implemented in the environment. Each of these systems is integrated into the environment via a uniform Agent View. Figure 14 has a brief overview of the implementation.

The fictitious 'Cafeteria' example is implemented in this environment in order to show its functionality. It demonstrates the complexity of enterprise integration and modelling. Furthermore, the environment implements true autonomous sources that are accessed via an Agent View on these sources. This makes it explicit that inconsistencies and incompleteness on the schema level as well as on the data level are unavoidable. It also shows that assuming consistency in enterprise integration is unrealistic.

Moreover, an agent model is implemented in order to show that all the Schema Knowledge, Resources Knowledge and Organisational Knowledge introduced in Section 2.7 can be accessed and managed by an information agent.

The case study is based on this agent model within the 'Cafeteria' example. It demonstrates and evaluates the conflict detection and resolution mechanism by example cases. This is an illustration of the approach as well as a proof of concept. The shortcomings of previous research are clearly elaborated. Finally, the steps of the conflict resolution phase are illustrated and critically evaluated. In particular, the inherent complexity, and in may cases inadequacy, of judgement based on credibility assessments is outlined. The case study illustrates the importance of making conflict detection explicit. Much previous research has neglected this stage.

7. Concluding Remarks

7.1 Summary

The first two chapters include a brief introduction to Distributed Collaborative Environments for Enterprise Integration (DCEEI). They cover the basic terminology, architectural issues, and the problem of integrating inconsistent information from autonomous, heterogeneous but semantically related information sources. Chapter 2 also incorporates a survey of all the information that is potentially available to an information agent in DCEEIs.

Existing research into enterprise integration environments has been surveyed in Chapter 3. Most research fails to provide conflict detection and resolution mechanisms and assumes internal, and inter (cross) system consistency. However, it is shown that this assumption is unrealistic for open distributed information systems. Furthermore, those research approaches that explicitly address conflict management fail to ensure that this is rational to any user, or client of the integration environment.

In summary, no rational scheme for conflict detection and resolution exists that is specifically suitable for DCEEI. Only a general structure for gathering candidates in conflict detection is provided by research in uncertainty management. Information agents lack domain expertise and, hence, lack domain-specific resolution procedures. However, no rational framework exists that guides information agents on the selection and application of existing resolution strategies to conflicts in the integration environment. In other words, conflict resolution approaches have been analysed in DCEEI, in the wider field of distributed artificial intelligence, and uncertainty management. It follows from this analysis that a framework is required with which agents can detect and rationally resolve conflicts.

A theoretical framework for conflict detection and resolution has been developed in Chapter 4 based on evidence law. This theory is applied to conflict detection and resolution in DCEEI. Strategies from the field of distributed planning and enterprise modelling are intertwined with this scheme.

Chapter 5 describes the mechanism. However, conflict detection requires a novel structure for object identity suitable for integration environments. It is an extension to existing object structures as described in Khoshafian and Copeland [KHO90]. Furthermore, a uniform formal representation of results from heterogeneous information sources (candidates), and evidence that warrants or refutes these candidates, is introduced. This formal specification enables the information agent to assess, represent

and classify, any known conflicts in enterprise integration. Syntactic and semantic conflict detection is outlined. This includes ways of strengthening the correspondence assumption made in the case of explicit and most implicit conflicts.

A framework for rational conflict resolution and judgement on conflicting results is thus designed. This includes a mechanism for identifying:

a. Domain-specific resolution procedures (Domain-Specific Problem-Solving);

b. The application of general resolution heuristics to domain-specific information, which is in the form of candidates and their evidence (Scientific, Domain-Specific Heuristics);

c. Judgement on the reliability of conflicting candidates (Domain-Independent Evaluation).

The latter includes a scheme for ranking the reliability of the candidates. Furthermore, new alternative rankings and reliability estimates are developed and negotiated.

Chapter 6 evaluates the approach by first describing the implementation of an enterprise integration environment. This environment integrates autonomous, heterogeneous information sources including an object-oriented database, two relational databases, rule-based systems, and various standard software systems such as production management and control systems. An enterprise model that acts as a reference model for the information agents is also implemented. Each source is integrated by an information agent, which acts as a client to the autonomous source. A C++ program called Demons is the implementation of a model of the information agent. In particular it contains the Agent Knowledge (all information an agent has about itself, its integrated source, and the other agents) and the conflict management program. A fictitious example is realised in this environment. The implementation clearly illustrates the complexity of a DCEEI. This makes explicit the inappropriateness of assuming consistency of the data, or the schema level of the distributed environment.

A case study is conducted in this environment. An information agent and its conflict detection and resolution mechanism are implemented in a Demonstrator program that is part of the agent model. The case study shows the functionality of the rational (called Principle Rational) conflict detection and resolution approach. It demonstrates that the mechanism is complete in respect to all known conflicts, and that it can incorporate all known resolution strategies. The evaluation illustrates the advantages over existing approaches, the limitations, the importance of thorough conflict detection in DCEEI, and proves that domain level resolution should be applied first to any conflict in DCEEI.

7.2 Results and Contributions

The following list briefly summarises the main contributions of this research.

- A theoretical framework for conflict detection and resolution in Distributed Collaborative Environments for Enterprise Integration (DCEEI) has been designed. It forms the basis for the structure of the conflict detection and resolution mechanism.

- A novel object identifier has been developed, which is an extension to existing object structures typically used in DCEEI based on Khoshafian and Copeland [KHO90]. It enables the information agent to assert and represent the identity of objects from heterogeneous information sources. The identifier incorporates the heterogeneous notions of identity as they exist in the integrated information sources. On this basis object correspondence can be defined accurately by the information agent. Furthermore, the agents can investigate weak notions of identity and object correspondence. Explicit (and some implicit) conflicts are based on the assumption that the candidates correspond to same thing (individual). If the information agent can identify that the objects (candidates) do not correspond to the same thing, there can be no conflict.

- A concept for syntactic and semantic conflict detection is designed. It is based on rehearsing enterprise modelling and schema integration phases. This makes conflict detection suitable for open enterprise integration environments that may not be modelled and integrated such that all-time consistency can be assumed.

- Admissibility is introduced to DCEEI. Results that are generally not worth considering are rejected in the Admissibility phase.

- This research has outlined multiple resolution strategies. However, no concept for information agents exists that enables them to apply these to conflict resolution in enterprise integration. It has been shown for conflict management in the field of planning (Section 3.4.4), that domain level resolution should be applied before more general strategies. A rational scheme for conflict resolution is proposed based on this premise. The adequacy of this principle has been shown in the evaluation.

- It is shown how information agents can apply general resolution heuristics to domain-specific information.

- Judgement on the reliability of conflicting candidates, with the aim of solving data conflicts in DCEEI, is briefly introduced. Synthesising existing research, it includes three steps: Ranking credibility; Defining new alternative solutions; and Negotiating a compromise.

The presented design of a conflict detection and resolution mechanism overcomes deficiencies of existing research:

- It provides a rational scheme for conflict detection and resolution, and does not just assume that enterprise integration environments contain only consistent information.

- Conflict detection is based on a complete scheme to detect conflicts that are merely syntactic, or a mere semantic mismatch. Existing approaches typically lack a detection scheme. No existing mechanism investigates the correspondence assumptions made about conflicting candidates. Furthermore, existing research fails to address concept correspondence in conflict detection in DCEEI.

- The demonstrator shows the inadequacy of conflict resolution based only on the reliability of conflicting candidates. Judgement based on the resolution of conflicting candidates can be one strategy to resolve conflicts, but it does not replace thorough conflict detection, and may not be as well suited as more domain-specific resolution strategies.

This research has also produced an outcome that is important to any conflict detection and resolution mechanism in DCEEI:

- The implementation of the integration environment makes the complexity and potential risk of inconsistencies obvious. These may occur within the integrated sources without the knowledge of the integrating agent, across information systems, and on the schema and integration level (meta information level).

- The notion of rationality was defined for information agents such that it can be Principle Rational and Application rational (Section 2.6).

- The evaluation illustrated the importance of thorough conflict detection in conflict management.

One result of this research is the elaboration of the limitations to this approach, and of conflict detection and resolution in enterprise integration in general. These are listed in the following section.

7.3 Limitations

The completeness of the conflict detection and resolution mechanism is constrained in two aspects (Section 3.2):

- Conflict detection is limited to conflicts between those results that the information agent receives from the information retrieval process. In other words, information retrieval necessarily precedes conflict detection. A detection mechanism can only detect conflicts among those results that are provided by information retrieval.

- Within the results that are presented to the mechanism, all known explicit and implicit conflicts can be detected. The information agent is, therefore, limited to those implicit conflicts that it 'knows'. In other words, the detection is constrained by the agent's principle of coherence. For example, implicit conflicts between various objects' different attributes require very specific domain knowledge. Such an implicit conflict may exist between 'Peter's blue shoes' and 'Mark's blond hair' based on the notion of coherence: 'Either Peter has blue shoes, or Mark has blond hair'. Typically, conflict detection is concerned with explicit conflicts or, alternatively, implicit conflicts of objects that are concerned with the same object (e.g. 'Peter') and the same attribute (e.g. 'colour of shoes'), but different properties (e.g. 'Colour is 'blue'' and 'colour is 'red'') (Section 3.2.1 and 5.2.4.3). These implicit conflicts require that the agent knows that the properties are exclusive, e.g. 'Peter's shoes only have one colour'.

All conflict detection and resolution steps are limited to the information that an agent has about the integration environment and its integrated sources (Agent Knowledge). For example:

- Syntactic conflict detection is limited to those syntactic conflicts that the agent can investigate with its Organisational Knowledge. For example, an agent can only investigate the semantic correctness of the information retrieval, translation, and matching if it has extensive knowledge of the communication protocol and modalities.

- The information agent can only systematically apply all available resolution strategies as guided by the rational scheme demonstrated in this research. It does not provide all resolution strategies. Not all 'known' resolution strategies can be applied to all conflicts. For example, if an agent lacks the information on a Domain-Specific

Problem-Solving strategy that exists for a given conflict, then it cannot apply this strategy. However, this is a limitation of the Agent Knowledge, and not of this scheme itself. It is complete in as far as it can incorporate all resolution strategies that the information agent knows about, and can apply them to conflict resolution in enterprise integration.

Judgement based on the reliability of conflicting information sources is very complex and only possible in those cases where relations between heterogeneous certainty assessments can be established. It has been established that a potential problem in enterprise integration, and in uncertainty management, is that the same certainty estimates may have semantically different meanings in multiple heterogeneous, autonomous sources. In addition, different certainty estimates, which may be inconsistent, may be used across information systems.

Furthermore, certainty estimates may be derived from roles, expertise or authority of e.g. information sources. These certainty measures, therefore, depend very much on a specific form of certainty or reliability. Comparing these different forms of certainty may not always be possible and semantically correct. Moreover, the scheme for judgement on certainty estimates is limited to those cases where judgement is Principle Rational (Section 2.6) so that meaningful results are produced for any clients of the integration environment.

A potential limitation of the approach, as described in the mechanism and implementation in the Demonstrator, are pragmatic assumptions about consistent Agent Knowledge. However, this limitation can be overcome by providing the agent with heuristics that let it manage conflicts within its knowledge. For example, it may have heuristics that let it rank conflicting Admissibility Heuristics, or guide the agent on how to decide when such conflicts arise (Section 5.5 and 6.5.5). Provided a candidate is declared admissible by one heuristic, and inadmissible by another, then a resolution that the candidate is provisionally admissible is possible (i.e. the result of conflict detection and resolution will have to include this provisional status).

Security issues are not addressed in this research. They are not taken into account in the description of the mechanism and the functionality of the detection and resolution mechanism. The scope of this work does not permit the in-depth discussion of these issues. It is worth noting that they are also not discussed by many fundamental DCEEI research works, including the KQML language [CHA92]. It is difficult to incorporate security issues when all other components of the DCEEI are not concerned with security.

Computational issues of conflict detection and resolution are not discussed and the effort and time it will take agents to perform the specific steps of the resolution mechanism are not calculated. However, this mechanism assumes autonomous information sources. It is, therefore, not necessary to closely integrate in a computationally expensive way. For example, no tight global schema and management system is required, with which local sources need to cohere (e.g. Master Model approaches in Section 3.3.1.1). Hence, the concept of this scheme emphasises modularisation and local autonomy, which potentially reduces and distributes computational costs.

Many issues in enterprise integration are very closely related to conflict detection and resolution. These include information retrieval, information exchange between information agents, communication between information agents, and specifically learning and adaptation. Future research in these areas should improve conflict detection and resolution. This future work is described further in the next section.

7.4 Future Work

Learning and adaptation are very closely related to conflict detection and resolution. For example, the Negotiation phase concludes with a resolution proposal to clients of the integration environment, such as human decision makers. In principle, this proposal can be accepted or rejected, for example, by a human decision maker, and so provide a good basis for agent learning. However, the agent could similarly improve its Resource Knowledge and heuristics (Organisational Knowledge) by direct interrogation of domain experts. Furthermore, it could install and change Comments (Resource Knowledge), for example, on the reliability of a source based on the agent's past experience.

Adaptation of the information agent to changes in the integrated information sources is crucial in autonomous, heterogeneous integration environments. Some research proposes to monitor integrated information systems, as for example described in Section 3.4.3. These mechanisms should be extended to support the information agents described in this research.

Furthermore, this research proposes a number of new concepts for enterprise modelling. For example, the novel object identifier and the object sameness predicates are concepts of potential use in the design of integration environments. A concept for Versioning of object identifiers has been mentioned in Section 5.2.1.4. The agent model described in this research also provides concepts for the modelling of information agents in enterprise integration. For example, Section 2.7 and the implementation (Section 6.2.3) provide a structure for the Agent Knowledge, which includes information about the integration environment, other agents and the integrated sources. The latter may entail the source's reasoning, processes, data, problem-solving strategies, etc.

The conflict detection and resolution mechanism is based on the quality and reliability of the Agent Knowledge. Hence, guidelines on the implementation and gathering of this Agent Knowledge would improve and guarantee the functionality of the integration environment, which includes conflict management.

> "Enterprise integration is concerned with how to improve the performance of distributed organisations and markets. It focuses on the communication of information and the coordination and optimisation of enterprise decisions and processes in order to achieve higher levels of productivity, flexibility and quality" [[FOX93]p.425].

This integration is, for example, provided by cooperative work and decision support systems (Section 2.6). The conflict detection and resolution mechanism is one step in providing information agents with intelligent integration capabilities. The next step is to facilitate the applications, or decision makers that use information from the DCEEI, to make best use of all the available information.

Scientific Heuristics are applied in the conflict resolution mechanism to make rational judgements on the accuracy of results. A concept for these judgements has been adopted from earlier research, such as the CYCCESS project [GUH94]. Research in qualitative uncertainty management, e.g. Argumentation [CLA90a], provides further means of judging the correctness of conflicting results. Further research is needed to tailor these approaches so that they can best provide services to rational conflict resolution in enterprise integration.

The integration of Domain-Specific Problem-Solving systems and strategies in the sharing environment is a complex problem. Integrating them into the agents' conflict management activities is typically limited to human expertise. Only a few examples have been presented by this research. Future research will further the integration of (intelligent), non-human expert knowledge such as advisory systems (Section 3.5.3.4) [BRO89] [HEW91]. This closer integration will then potentially provide more domain-specific resolution strategies to conflict management.

Conflict detection and resolution is only one of a number of processes that mediate between information retrieval and information use (by clients). These processes should be closely linked. For example, this research can provide concepts useful to information validation and uncertainty estimation of results provided by the integration environment.

For example, this research has illustrated a mechanism to investigate object correspondence in enterprise integration environments. This mechanism is used to compare the correspondence of candidates or results. However, it could potentially be used to improve information retrieval by comparing the correspondence of the results to the information actually requested.

The collaboration in the DCEEI demonstrator is limited to the exchange of results. Nevertheless, Sections 2.3, 2.7, 3.3.1 and 3.3.2 have shown how information agents in enterprise integration environments exchange Agent Knowledge (Schema Knowledge, Resources Knowledge and Organisational Knowledge). However, multiple information agents may incorporate inconsistencies among their Schema, Resource, or Organisational

Knowledge. Future work is needed to develop a concept for the sharing of inconsistent meta knowledge.

Furthermore, information agents could critique their conflict resolution strategies. For example, an agent could develop a new alternative and then request critiques from the other information agents. This principle has been well established in the area of distributed artificial intelligence but needs to be implemented in the conflict resolution mechanism.

The last point is of particular interest in the case of heterogeneous information agents. The present research is based on homogeneous information agents. Each agent uses the same knowledge representation, internal problem-solving strategies, inter agent communication language, etc. Furthermore, the agents benevolently exchange all their Agent Knowledge. However, DCEEI should also integrate heterogeneous information agents that may not necessarily be as benevolent. For example, information agents from multiple organisations may communicate directly and form an integration environment. The conflict detection and resolution mechanisms are applicable to heterogeneous agent environments. It would, however, be necessary to extend the Agent Knowledge and available resolution strategies for the detection and resolution framework. For example, syntactic conflict detection becomes more complex in heterogeneous agent worlds.

Appendix A: Distributed Artificial Intelligence

A.1 Example Approaches in Distributed Artificial Intelligence Research

This appendix provides a brief description of the research projects described in Table 3. Section 3.4.3 discusses some elements of these approaches that are important to conflict management by information agents in enterprise integration. The introduction to these approaches, however, has been placed in an Appendix to keep the relevant research section as brief as possible.

Kind of Conflict	Contract Net	Conflict Type	Resolution Strategy	Paper Reference
Task Assignment	Yes	coop.	Negotiation consists of proposal, critique, explanation and resolution heuristics;	[LÄA92]
Goals in DPS	No	coop.	Compromise based on case-based reasoning and preference analysis, persuasion to change agents goals;	[SYC89]
Goals in Design Systems	No	coop.	Compromise by alternatives and domain-dependent heuristic decision-making if compromise fails;	[WER91]
Goals in DPS	No	coop.	Compromise by redefining goals and integrative negotiation (most important goals of all parties are integrated);	[LAN89]
Goals in DPS	No	coop.	Fit to the problem and time/cost optimisation of resolution mechanism, human decision for essential conflicts;	[STE90]
Goals in DPS	No	coop.	Game theory in adverbial situations, including a rating of 'worth';	[EPH91] [ZLO91]
Human / Computer	No	coop.	Human user makes a decision;	[STO91]
Learning in Co-ordination of Agents	No	coop.	Manager agent decides with the help of a learning system which is building in form of a case-base for use in coordination (similar to SYC89);	[VIT91]

Table 3: Conflict Detection in DAI (also in Section 3.4.3)

Läasri *et al.* [LÄA92] have developed a generic model for cooperative problem-solving. First, a complex task is decomposed into multiple hierarchically structured tasks and the goals they need to fulfil. Negotiation can have the two forms of :

1. "Exchange and evaluation of possibly conflicting partial results generated during local problem-solving....
2. Agents interact to resolve conflicts only after independently completing their local problem-solving."[[LÄA92]p.301].

In the first case agents interact in a similar way to partial global planning [DUR91a] (Section 3.4.2) where agents exchange partial results to co-ordinate their activities at run

time. In the latter case, the agents resolve a conflict that has occurred as a result of the action they have already taken.

Negotiation is blackboard [NII86] based in that an agent makes a proposal, other agents critique the proposal, and the proposing agent may want to explain its result again. In addition, agents are given 'meta-information' which is information about how particular conflicts can be solved fastest. The meta information speeds up the process of conflict resolution.

Conflict detection is concerned with detecting explicit or implicit conflicts. However, these terms are differently defined than in this present research in Section 3.2.1. For **Läarsi** *et al.* [LÄA92] explicit conflicts occur when an agent critiques another agents proposal. Implicit conflicts are those that stem from concurrently developed proposals that are not consistent.

Negotiation systems can be very different from the blackboard based style described above. For example, the Persuador is a case based negotiation system by **Sycara** based on third party mediation.

> "The negotiation process consists of three main tasks: Generation of a proposals; Generation of counterproposals based on feedback from dissenting party and; Persuasive argumentation"[[SYC89]p.121].

An initial proposal is generated by the mediator and sent to the parties. The mediator gets an acceptance or an evaluation from each party's perspective on the proposal. Based on this the mediator generates an initial compromise. It may then again get an acceptance from both parties or decide to generate further compromises. If compromise is possible then the mediator can try to change the conflicting parties' views. This step is called persuasion.

The mediator has a case-base that is formed from previous experience. "The cases are organised hierarchically in memory around important domain concepts" [[SYC89]p.124]. It allows the agent to propose compromises based on the successes and failures experienced in the past. If no example cases are available then the agent analyses the preferences of the conflicting parties. This enables the mediator to search for areas where the parties are willing to make compromises.

Persuasion is based on argumentation by the mediator.

> "The task of a Persuador can be viewed as finding the most effective argument that will increase the agent's payoff."[[SYC89]p.129].

The payoff can be effected by changing the importance or the value a persuadee attached to the issue in conflict. For example, changing the importance that an persuadee attached to the goal 'get new resource A', may be achieved by proposing compensation of 'resources A' to be provided by the other conflicting party.

Similar to the previous approach, **Werkmann** [WER91] presents a negation and coordination mechanism for multi-agent systems. It is also based on mediation by an arbitrator (third party) agent. A conflict is detected if agents have common issues, plans or actions that are explicitly conflicting. If any one agent in the system wants to change its action then it investigates which other agents have a common issue. These agents are then asked to comment on this change. If they do not agree then a conflict has arisen.

A multi-step resolution mechanism is initiated by the arbitrator agent in the case of conflict. It reviews the dialogue between agents that has lead to the conflict and investigates the relations between the agents. It then proposes viable alternatives and asks the agents to rank these. If no agreement can be reached by this mediation then the arbitrator has to make a decision based on domain-specific heuristics.

A problem with the approach by Sycara [SYC89] (which is also true for Werkman's approach) has been described by Lander and Lesser [LAN89] as:

> Mediation is generally useful when the conflicting parties are antagonistic. But the persuasion by mediator requires that the mediator has access to the goals and the values which the conflicting parties attach to their goals. If no domain-specific heuristics are available as in [WER91] then direct negotiation is necessary.

Lander and Lesser's [LAN89] agents directly negotiate a 'compromise' or try to 'integrate' their goals. In other words, if two agents such as a buyer and a seller have a conflict over finding a price for a product then they can negotiate a compromise. The compromise is a price that is acceptable for both parties. An integrative resolution is one in which both parties crystallise their most important goals. A solution is then sought that satisfies these most important goals of both parties. In principle, compromise is a sort of fine-tuning negotiation. Integrative negotiation is applicable when completely new / alternative solutions are needed.

Agents exchange information benevolently among each other in the coordination environment by **Steiner** *et al.* [STE90]. All agents have knowledge about themselves and other agents capabilities. A request is processed by one managing agent. For any request there may be either only one or multiple agents or partners that can provide the result.

> "If the agent knows about matching partners, requests are send out to these partners. If more than one partner responds, the initiating agent applies conflict resolution strategies to chose the best partner. Factors in such strategies are the degree of match to the problem at hand, the time/load constrains of the partners, the cost of communication, and other less important factors" [[STE90]p.119].

Finally, conflicts that cannot be solved by the system are notified to a human user for further consideration.

In other words, a conflict is detected if the responses from all sources that can possibly provide results, are not consistent (matching). In the case of conflict the resolution strategy aims at finding the sources that can do the task most completely. For example, two sources A, B and C may be able to solve parts t1, t2, t3 of the task T. If A can solve t1 and t2, B can provide t2, and C can solve t3 then A and C are selected. This is so because A can solve more of the task T (t1 and t2) than B, which can only provide the sub-task t2. It is not considered if A or B can produce correct solutions t_2. Other factors are directed at optimising the conflict resolution costs by evaluating the lowest utility of the agents (time/load constrains), or the cost of communicating to particular nodes.

Negotiation between agents can be based on selfish agents that cooperate under adversarial situations. **Zlotkin and Rosenschein** describe a mechanism in which agents have to define how much a goal is worth to them:

> "[Worth] captures the importance of achieving a given goal. A useful way of redefining utility was to use worth as the baseline of the utility measurement (i.e. an agent's utility is the worth of his achieved goal minus the cost of his part of the joint plan)" [[ZLO93]p.175].

Similarly, Ephraty and Rosenschein [EPH91] developed a mechanism for agents in adverse agent scenarios to define their true preferences. This is done based on a voting mechanism by agents to specify preferences in respect to all possible outcomes. It includes an incentive mechanism for truth-telling based on a tax system. Oversimplified, agents receive compensation (distributed tax) for not having their optimal solution chosen by the group. Agents are therefore not inclined to overestimate their true costs by having to accept something other than its optimal solution. Game theory is used to evaluate the group choice.

Game theory is suitable for situations where all possible payoffs of all solutions can be calculated. It is therefore a typical example of a mathematical rather than a semantic approach. This, however, is a potential weakness of game theory in complex problem-solving tasks [LÄA92], e.g. enterprise integration. Furthermore, game theoretic decision situations typically "do not count as group decisions because each agent chooses an action with the aim of furthering individual goals, not mutual goals"[[WON94]p.409].

Stolze and Gutknecht [STO91] describe a human centred information system called IKEA. It focuses on a close integration of computer and human agents in one system.

> "Problem-solving does not only take place inside the agents but also directly on the interface where the human and the automated agents cooperate" [[STO91]p.106].

This system addresses cooperative problems that can be solved by shifting the detection and resolution task to a human user. In other words, the system is limited in that the problem space must be suited to the user's expertise. Furthermore, this approach is applicable when no automated problem-solving or conflict detection and resolution is necessary, e.g. in environments with large numbers of agents.

Finally, a learning system in the field of DAI will be introduced. Some of the above systems, e.g. by [SYC89], have implemented case-based reasoning and, hence, are based on previous experience (cases). An investigation of learning for conflict resolution in 'intelligent' systems, however, has been undertaken by **Vittal** *et al.* [VIT91] [VIT92].

> "Learning and adaptation are keys to the intelligent processing of information, be it distributed or localised" [[VIT92]p.348].

This research group developed the integrated learning system which is based on a central learning agent that co-ordinates individual learning agents. Each agent may have different learning mechanisms and storage of the learned facts. In other words, there are numerous learning mechanisms such as knowledge-based learning, inductive learning, search-based learning or feature learning to name a few discussed in [VIT92]. Each of these mechanisms has a different way to encode both general and also newly learned information, e.g., in a knowledge-base, in simple data structures or a domain model.

Cooperative learning starts with collecting recommendations from any agent in the system. Each recommending agent evaluates its own contribution by a vote in the range of 1 to 5. In a second stage the agents critique each other's proposals. The information that is finally learned by all agents is that which has the most votes.

Some problems were experienced with learning. For example, newly learned information is difficult to evaluate if its applied to frequently changing domains. In other words, if an agent has learned information and applies this latest information then it assumes that any changes in the environment are due to this new information. This may lead to situations where agents constantly change / learn without actually improving the system. Furthermore, it is difficult to "know whether a successful action was the best of all the suggested actions" [[VIT91]p.177].

Learning typically depends on the authority of the agent that supposes a fact to be learned. But "rather than discovering that one agent is generally more reliable than another, it is more useful to discover the reliability of agent/situation pairs" [[VIT92]p.359]. In other words, when information is proposed than the reliability of a particular agent in that particular situation is taken into account in order to judge on its recommendation.

Multiple agents in the previous system learn about particular problem-solving situations and about each other's reliability. However, Gasser and Ishida propose further changes to the agent system including "adjusting inter-agent relationships, the knowledge agents have about one another, the size of the agent population, and the resources allocated to each agent" [[GAS91]p.185].

Learning and adoption are central to any problem-solving mechanism including conflict detection and resolution. In principle, agents can either change the problem to resolve a conflict or they have to change themselves to fit the problem.

> "A well-known AI [Artificial Intelligence] approach to adoptive problem-solving systems has been to use a fixed problem-solving architecture which responds to environmental change by restructuring problems (e.g. by relaxing problem constrains, abstract search spaces, or changing decision criteria dynamically.....) or by long term adoption of problem-solving knowledge (learning)" [[GAS91]p.185].

Thus a fixed conflict detection and resolution mechanism should be based on an agent's changing heuristics and information about the environment

A conclusion of described research approaches can be found in Section 3.4.3. This appendix only provides some further descriptions of the approaches.

A.2 Conflict Detection and Resolution in Distributed Knowledge-Based Systems

Agents can be rule-based systems. Collaborating systems of such agents have been termed 'Collaborating Knowledge-Bases' [STE90], 'Federated Expert Systems' [KIR91a], 'Cooperative Knowledge-Based Systems' [CARL91] or 'Distributed Real-time Knowledge-Based Systems' [DAI93]. These kinds of systems have been briefly introduced as mainstream DAI (Section 3.4.3).

Within distributed artificial intelligence some systems are particularly concerned with the integration of multiple, traditional rule-based systems, such as knowledge-bases or expert systems. These systems are not agents that cooperate and / or collaborate but merged rule-based systems. Example include 'Distributed Knowledge-Based Systems' [CARL89][CARL91], 'Integrated Knowledge Derivation Systems' [SU 91], or 'Cooperating Knowledge-Based Systems' [HUA92].

A typical example is presented by Carlson and Ram [CARL91].

> "Distributed Knowledge Based System (DKBS) would consist of several Knowledge Based Systems (KBS) that are logically and physically distinct from one another. The KBSs are linked together and managed by a single Distributed Knowledge Based Management Systems (DKBMS) which coordinates global inferencing to resolve goals that span multiple KBS sites in the network" [[CARL91]p.11].

In other words, the knowledge-based systems are centrally co-ordinated. A global knowledge-base is installed and managed for integrating all the component systems. This typical approach, hence, installs a very tightly coupled distributed system of knowledge-bases.

> "The global knowledge base is responsible for managing conflicting responses or null responses from queried KBSs and for determining the best reply to the original query" [[CARL91]p16].

The 'best reply' can be evaluated, for example, by the degree of match to the problem at hand, the time/load constraint of the KBS, the cost of communication.

In principle, this system detects conflicts based on domain knowledge such as a truth maintenance system (Section 3.5.3.1). In other words, the central system is 'responsible for managing conflicting responses' including detecting them. Conflict resolution,

however, assumes only non-essential conflicts where the central systems can choose between mutually acceptable solutions that are all correct.

However, the previous system does not take into account the explicitly defined reasoning structure of rule-based systems. Su and Park [SU 91][SU 90], for example, have developed the following framework:

> "It integrates heterogeneous rule-based systems, whose data and knowledge are in the form of rules and constraints. To embody this approach, a new knowledge representation scheme was developed to capture the dynamic aspects of knowledge" [[SU 91]p.243].

In principle, all rules are merged into a distributed model. The model also incorporates the information interactions between systems, e.g. particular output from one system is used as input by another. This may lead to cyclic relations of rules from multiple systems passing around results.

> "Special processing is required for parallel paths [parallel results from multiple sources], which can derive alternative values for the same data item ... Some special query optimisation is performed to resolve conflicts among such competing values" [[SU 91]p.236].

Optimisation may be achieved by heuristics such as 'select the highest result', 'select the average outcome over all results' or 'first come first serve'. Su's 'special processing' includes that queries are sent many times to the KBSs or expert systems. A result is then, for example, averaged over the obtained responses, or the responses may automatically converge to a joint result. In other words, it investigates whether the results change over time as multiple systems constantly exchange improved results. This approach does not produce a snapshot result that ignores the dynamic behaviour of information from KBSs or expert systems.

Kind of Conflict	Contract Net	Conflict Type	Resolution Strategy	Paper Reference
Integrating Rule-Based Systems	YES	coor.	Coordination by a central system; Resolution is based on heuristic selection among mutually acceptable solutions (e.g. best fit to the problem);	[CARL91]
Integrating Rule-Based Systems	YES	coor.	Detection includes analysing the dynamics of knowledge; Resolution based on heuristics e.g. 'highest value', 'average', 'first come first serve';	[SU 91]

Table 8 : Distributed Knowledge-Based Systems

In summary, typical distributed knowledge-bases as described by [CARL91] are tightly integrated systems that allow conflicts to occur and be detected. However, conflict detection in these systems is only necessary in the initial integration phase and when new beliefs are encountered. Thus, the integration requires complete exploration of the integrated sources, abolishes the autonomy of the integrated systems, and detects only data-value conflicts. It is not applicable to conflict detection in enterprise integration environments but only in closed subsets.

The approach by Su establishes consistency among traditional rule-based systems to some degree an ad hoc (real-time) manner. In other words, conflicts are detected as multiple results. However, they are resolved by pre-defined heuristics. No investigation of the kind of conflict or the accuracy of the heuristic resolution is made. However, this approach has illustrated dynamic behaviour of 'knowledge', or information, from derivation systems.

Appendix B: Object Identity

B.1 Introduction

The following Sections present a brief summary of different notions of identity. First, standalone information systems are analysed. Section B3 will briefly introduce 'external' notions of identity. Section B4 briefly investigates some ways in which these heterogeneous notions of identity can be integrated.

B.2 Identity in Information Systems

Different ways exist to implement a notion of identity in information systems. Table 9 provides an overview of the most commonly used notions in information systems as, for example, outlined by Khoshafian and Copeland [KHO90]. The implementation techniques are classified in respect to their location, data and structure independence. These concepts have been introduced in Section 5.2.1.1.

Number	Identity Implementation	Independence of Location	Data	Structure	Examples
1.a	address based	no	yes	no	PASCAL, C,
1.b	address based according to identifier keys	no	yes	no	Hierarchical and Network Databases
2.	indirection	yes	yes	no	Smalltalk-80, KBZ [OXB88]
3.	structured identifiers	no	yes	no	LOCUS
4.	identifier keys	yes	no	no	Relational Databases
5.	tuple identifiers	yes	yes	no	INGRES, RM/T [COD79]
6.	surrogates	yes	yes	yes	ORION-2, C++, Object-Oriented Systems based on e.g. [KHO90]

Table 9: Major Notions of Identity in Information Systems

Surrogate based identification can clearly be proved to be the most adequate form of object identification in information systems [PAT88] [KIM91a] [KEN91] [KHO90]. Object-oriented databases use an object-oriented data model that is not set oriented and offers the potential for clear logical identifiers among objects. An instance of an object in an object-oriented database is typically referred to as an individual. Most object-oriented databases, however, mix the notion of identity with the notion of addressability [UNL90]. To establish a concept of identity based on physical addresses is to mix the

logical identity of an object with its physical location. An objects identity in surrogate based systems should, however, be independent of its physical address.

> "Surrogates are system-generated, globally unique identifiers, completely independent of any physical location. They are associated with each object of any type at the instant the object is created. They cannot be changed; i.e. they represent the identity of the object throughout the lifetime of the object" [[UNL90]p.167].

Surrogates or unique identifiers (UID) are typically referred to as 'strong notions of identity'. Khoshafian and Copeland [KHO90] characterise this 'strong' notion as:

- A notion of identity that has built in identifiers which are provided by the system or language;
- In this notion identity is preserved between transactions, which makes the identity of an object consistent in a temporal dimension.

The closest match of surrogate identifiers in relational database management systems are called '**tuple identifiers**' [KHO90]. They are system defined keys which are added to each tuple in a relational table. These identifiers are generated or defined by the system and integrated into the internal layer of the database management system. Codd [COD79], for example, has called these 'E-Attributes', which are defined in a central table called 'E-Domain'. For example, the table with the fields 'Employee, Name, Address,' may have an additional field which holds the surrogate E-Attribute. If two tuples share a common identity then they have to be explicitly defined by a 'coalescing' command. This command will tell the system that two tuples need to have the same identifier (E-Attribute) because they are concerned with the same object.

System generated identifier keys are data and location independent. In Codd's RM/T, surrogates are "unique within the entire database" [[COD79]p.410]. The attributes within a relation, however, are not identifiable because only tuples are identifiable. In other words, the individual attributes, e.g. 'Peter' for the attribute Name, are not identified by a surrogate as in object-oriented databases. This makes the notion of identity weaker than surrogate identifiers in object-oriented systems because the identifier becomes structure dependent.

One problem with surrogates is, however, that they are rather awkward to use:

> "Value based matching is a straightforward and transparent technique for expressing relationships, it provides no implicit support for referential integrity

> and is a potential source of update anomalies" [[PAT88]p280] "A system which supports objects with unique keys is therefore less expressive than one which supports object identification as defined above [by value based matching or keys]. ... It is our belief that for most applications, the benefits offered by associating objects with a key more than compensate for the corresponding loss of expressiveness" [[PAT88]p.285].

However, surrogate identifiers are expensive to store because they require the surrogate to be stored alongside each object [PAT88]. In addition, many information systems cannot provide a system generated identifier so that most database management systems are based on user defined identifier keys [KHO90]. Hence, a more commonly used notion of identity is "the concept of the table 'key' [that] is tied to a specific table within a relational database. There is no formal, or informal for that matter, concept of identity across tables in the relational model" [[KHO90]p.59]. Such **identifier keys** are, for example, implemented in relational databases such as INGRESS [STO76] or System R [AST76].

Identifiers across tables need to be defined in the retrievals or updates in the form of 'joins' [COD90]. In other words, a query needs to hold the information on how to select an individual in one table to match with another in another table. There are no fixed path expressions established between relations that could permanently link objects across relations.

User defined identifiers are location independent because the location of a tuple is not important for the identification. They are data dependent as they use the values or attributes in a tuple to identify an object. Naturally, the identification depends strongly on the structure of the relational table. In particular value dependence, as described above, makes user defined keys a much weaker form of identification than system generated surrogates or tuple identifiers.

Structured identifiers are structured because part of the identifier captures the location of the object. For example, a file in the MS-DOS hierarchical file system can be identified based on its hierarchical location such as 'C:\dos\' (disk C directory 'dos'). The file name may, e.g., be 'doskey.com' and the identifier is therefore 'C:\dos\doskey.com'. It is questionable, however, in how far a notion of identity is implemented. For example, the same file 'doskey.com' may be replicated in another directory such as 'c:\utility\' (disk C directory 'utility') and there is no possibility to identify the two files as being the same. The problem with this weak notion of identification is that it does not provide structure or full location independence. In other words, the identification allows the same file to be

moved within the directory. But if it would be moved between directories it would have to change its identifier despite the fact that it is actually still the same file.

Identity based on **indirection** is, e.g., implemented in Smalltalk-80 [KAE83] via object-oriented pointers [KHO90]. Each object is linked to an entry in the object table with a pointer. The object table holds an entry for every object that exists. If the system implements indirect addressing so that the address of an object is read via its entry in the object table then the identifier is fully location independent.

Indirection is a predecessor to surrogate based identification because it shares the idea of system managed identification. It therefore provides full data independence. However, in contrast to surrogates, the identification is not structure independent because the key to object identification is the object table. Any changes to this table can corrupt the whole identification. The object itself has no identification attached to it but only a pointer to the object table.

> "Perhaps the simplest implementation of the identity of an object is the physical address of the object" [[KHO90]p.43].

For example PASCAL implements object identification via the address of an object. A virtual address is composed of one part pointing at an address space and a second part, which allows identification of the object within this space. However, with physical addresses as identifiers no location independence is provided. Virtual addresses can provide location independence only within the address space. This kind of location dependence automatically means that the identification is not structure independent. Implementing address based identification, however, provides full data independence.

The most basic way to store data is in **sequential files**. These can be composed of records of an equal structure, or records of different structures.

> "A record generally contains an identifying field ... The identifying field of a record is called its key field" [[LOO89]p.46].

Typically, a record is regarded as an individual such as a tuple in relational databases [DAT90], and an object in object-oriented databases [HUH92].

Sequential files that hold records without an identifying field have no notion of identity implemented in them. Typically, however, sequential files have records with an identifying field. This can be the address of that record in which case the identification is

purely address based. In general, however, the key field is a data value and identification is therefore identifier-key based.

Sequential files do not allow direct access but only sequential reading. This in general does not effect identification. But it is inherently more difficult to ensure consistent identification when there is no direct access to all existing files. For example, often the creation of a new record writes the record at the end of the file. This assumes that the record does not already exist. Redundancies are very difficult to detect if there is not direct access via, e.g., an index. Eliminating redundancies in sequential files would, for example, mean that the new record is compared with all existing records in that file.

A more effective way to organise a file with direct access to individual records is **relative organisation**. It is based on a relation between the key and the physical location of the object:

> "When a record is to be written into a relative file, the mapping function R is used to translate the record's key value to an address, which indicates where the record is to be stored. When it is necessary to retrieve the record with a particular key value, the mapping function R is applied to that key value, translating it to the address where the record can be found" [[LOO89]p.324].

Relative organisation can be implemented as absolute addressing where R points directly to the address of a record. Relative addressing implements R to map from the key value to the virtual address, e.g. an order number, within the file which is then linked to an address. Many more address calculation techniques have been proposed such as those found in, e.g., [LOO89].

Initially relative organisation was used for file organisation on persistent storage media such as hard disks and floppy disks by IBM. Other examples include COBOL, FORTRAN and PL/I. In these languages the programmer supplies for the system a record's relative address when the record is stored and when it is retrieve. "It is the programmer's responsibility to perform the key-to-address transformation; this mapping is not done automatically by the system" [[LOO89]p.352]. These are examples of purely address-based identification.

Relative organisation can be improved by employing an **index** to map between the address and key values. Such an index has to be based on a key identifier of each record. An index can be implemented into the storage medium and map each key value into an address-based pointer. Example mechanisms are the B-Tree, or B*Tree. A program that

has such a file organisation can implement value based identification based on identifier keys.

If, however, the storage system itself does not provide key based identification or an index then the whole COBOL or FORTRAN program may have a key based identification system programmed into the application. Strictly speaking, the program has implemented address based identification within the file but provides value based identification within the program.

> "An effective way to organise a collection of records when there is the need both to access the records sequentially by some key value and also to access the records individually by that same key is **indexed sequential file organisation**" [[LOO89]p.398].

For example, records can be linked sequentially by pointers and, in addition, an index can be implemented allowing sequential and direct access. Most programming languages provide utilities to implement index sequential files such as COBOL. Programming languages that do not provide these mechanisms, such as Pascal, may be used to manually program an index in addition to the sequential data access into the application.

Multi key file organisation is implemented in most database management systems. It is an extension of the sequential indexing by allowing the use of not only one primary key but multiple keys. This access may be implemented by installing multiple indexes.

> "A key's inversion index contains all the values that the key presently has in the records of the data file. Each key-value entry in the inversion index points to all of the data records that have the corresponding value" [[LOO89]p.425].

Many database systems, including most relational databases, are based on inverted indexes.

Another way of implementing multiple key access is the 'multi-list' approach. The first key is used as a primary key for identification purposes. This key value in the index points to the address of the according record. Any further, secondary key can be determined by pointers between the records with the value in the secondary key field. The secondary key is therefore implemented as a linked list.

> "The multi-list approach ... has been the basis for physical database structures in many of the commercially available network and hierarchical database

management systems, including the CODASYL family of systems, Cincom's Total and IBM's IMS" [[LOO89]p.430].

Hierarchical databases such as IMS are based on the hierarchical data model. The data in these systems is structured in trees starting from one initial root. The tree structure can have multiple subtrees, which in turn can have subtrees, etc. All subtrees are connected to their parent tree by one root. Every tree can have multiple occurrences. These occurrences can have values for the number of fields that are part of that tree. For more detail see, for example, Date [DAT90].

Identification is made by pointers. All the occurrences within one tree are linked by pointers. Each occurrence has pointers connecting each of its fields. An individual is referred to as one occurrence record [HUH92]. In IMS terminology this is called a segment [DAT90]. An occurrence record is identified by the pointer linking it to its predecessor record in the next higher tree level called a parent record. A parent record has zero to many children which are related to it by pointers. When a child record is created then it is linked to the parent via a key. This concept is similar to the foreign key in relational databases. However, in contrast to relational databases, the parent child relation is implemented by a pointer.

Furthermore, the relation from one child to the next with the same name but a different second key is also implemented by a pointer. The identification in hierarchical databases is, however, based on identifier keys or key fields. The implementation based on pointered records is not location independent. In other words, multi key sequential indexing provides value based identification but no location independence. It is therefore a weaker notion of identity than location independent identification, e.g., in relational databases.

The network structure of the CODASYL data model [COD71] as, for example, implemented in IDMS, consists of a number of record types. Each record has a name (e.g. variable name and variable identifier), and may take several values or occurrences [OLL78]. The member items or occurrences of a record are connected by pointers. Records can be of the type parent or child similar to hierarchical databases. In contrast to hierarchical data structures, every record can have multiple parent records. The relations are therefore called owner or member relations.

Records in **network databases** are typically identified by keys [DAT90]. The key values can in general be :

1. Stored alongside the record in the database;

2. Stored as a separate 'dummy' record type or index; or
3. Held externally in the application program [[OLL78]p.33].

In the first two cases an index is available allowing key based identification. If the key is located in the application system and is not part of the database then the database itself has an address based identification mechanism. The key in that case is part of the external program and the notion of identity within the database consists of the address based pointers assigned to the records.

In addition, all records are linked by pointers to their owners just as fields of a record are linked. In principle, the same index sequential organisation can be found as in hierarchical databases. In the CODASYL type databases several different ways of retrieving any individual record exist in a database [OLL78]. These include index access via the primary key. In addition, pointer addresses may be used directly. Sequential reading of records of the same type is typically possible in network databases. Furthermore, rooting through the database is possible by finding owner or member records. The secondary indexes, however, are implemented by pointers and not by an external index. The secondary index is only accessible via address based pointers of the primary key index.

Network databases have a value based notion of identity. The secondary key, however, is value based but only accessible by pointers. Furthermore, the network structure is not location independent, pointers can be used to identify objects. Network databases implement address based identification and value based identification which is not value nor location independent. Hierarchical and network databases implement a weaker notion than purely identifier key based relational databases.

B.3 External Notions of Identity

Many information systems that have only weak notions of identity such as file systems, or heterogeneous sources in integration environments may refer to external sources for object identification. For example, a sequential file system may have virtually no notion of identity. Data from this system may have to be identified by the system, or human user, that receives this data. For example, the C++ programs in the Demons program described in the implementation (Chapter 6) stores data in sequential files. The data from these files is read in by the C++ program sequentially inserted into a data structure. The Demons program identifies records that have been stored in these sequential files. Hence, the notion of identity that is implicitly implemented in data of the sequential file is essentially externally implemented, in the C++ application program.

Enterprise integration environments use another form of external identification. Objects are matched based on data values to their counterparts in a global model, e.g. a common knowledge-base such as CYC [LEN90], or enterprise models such as MKS [PAN91a], TOVE [FOX92], *et al.* The CYC approach will be further analysed in Section B4.

Closely related to enterprise models is the use of essentialistic concepts.

> An essentialistic approach: "enables one to divide the properties which characterise an individual into those that do so essentially and those that do so merely de fact, accidental " [[RES75]p.15].

For example, an employee could have the essential properties 'Name, Address, Employee Number' and the accidental properties 'Names of Children, Name of Wife/Husband'. Humans typically use essentialistic models to identify objects. For example, someone could identify an object as a table based on the essential characteristics:

> Object has at least three legs, a table top and no back (which would make it a chair).

In the same way, an information agent with knowledge of the essential properties of a concept, could use this knowledge to identify objects.

All approaches to externally identify objects carry a certain risk of error. No ultimate essentialistic model is possible other then reality (existence) and existence is a metaphysical questions which lies far out of the scope of distributed information systems [KEN91].

The use of essential characteristics to identify objects is not only shared by enterprise models, application programs, or humans. Essentialism shares with user defined keys essential, or key attributes of individuals. In contrast, however, essentialsm is based on essential properties of one particular object. For example, a table may be said to have the essential properties '4 legs', and 'a table top'. Identifier keys in relational databases are defined for a whole relation or set of objects. For example, a relational table on furniture must have an identifier key that is valid for all the tuples (furniture) that are stored in this table. Normalising a relational database is concerned with finding a data structure that conforms with the reality, including identifiers that are suitable for all tuples of a relation [COD90].

In summary, external object identification is common in information sources, and enterprise integration environments based on enterprise models. However, external identification carries some risk of becoming inadequate because of possible divergence between the underlying model (enterprise or essentialistic model) and the real world.

B.4 Integration of Heterogeneous Notions of Identity

In the previous Sections different notions of identity have been described. Enterprise integration environments integrate heterogeneous information sources that may have different, heterogeneous notions of identity. The integration of objects with heterogeneous notions of identity is concerned with object sameness (Are two objects from heterogeneous sources concerned with the same object?). Section 5.2.4.3 has introduced some basic sameness predicates. This Section will provide further background information on existing approaches to integrate heterogeneous notions of identity.

Approach	Notion of Identity	Relationships between Global Objects	Form of Integration of notions of identity
[COL91] CARNOT	Every object references a concept in the global base CYC (value matching);	Objects have a common counterpart in CYC, or they inherit the relations that their counterparts have in CYC;	Based on generalisations;
[PAN91a] MKS	Referencing of every object to a strong object structure of the whole enterprise called MKS;	Identical based on same surrogate;	Remodelling of every object globally;;
[AHM91] Pegasus	Strong object structure;	Same surrogate identifier;	Integration into 'home system' based on generalisation; External objects (results) are integrated at hoc;
[OXB90] KBZ	Object table;	Same reference to object table;	Mapping based on generalisations;
Eliassen and Karlsen [ELI91]	Strong object structure	Accessibility relations between a strong, global notion of identity with three levels of identity including immutable, value based and session identities;	Hardwired generalisations so that the local autonomy of a source is weakened;
Masunaga [MAS90]	Strong object structure;	Trivial-equal, Referential-equal, Arbitrary-equal	Manual real-world to conceptual level mapping;
Heiler and Blaustein [HEI89]	Strong object structure with functions for external objects;	Same identifier same object; Coercion functions can be installed for other objects that are identical;	Integration via coercion functions;

Table 10: Overview Integration of Heterogeneous Notions of Identity

Existing research on object identity in distributed, heterogeneous information systems falls into two categories:

- Those that map the different notions of identity into one central notion of identity and sameness;
- Systems that operate with multiple, coexisting notions of sameness.

The first group covers most heterogeneous information systems and only a few approaches exist that allow objects to keep their heterogeneous notions of sameness in the global level. Examples of both groups are included in Table 10. Only the last three approaches belong to the second group. The table describes the approaches (identified by the project name or first author of the publication) by summarising the notion of identity they implement, the way relationships between objects in the global level can be defined, and the form in which information systems are integrated.

The CARNOT [COL91] project is based on the CYC global knowledge-base [LEN90], which has been described as a master model schema for integrating heterogeneous information systems in Section 3.3.1.1. It is frame based and therefore implements identification by names and characteristics. For example, the person with the family name 'Guha' is identified by its name in the following two slot entries in the global knowledge-base such as :

> **"SeeUnitFor-Guha∈ residents Texas**
>
> instanceOf:: (SlotEntryDetailTypeOfSeeUnit)
> modifies Unit: (Texas)
> modifies Slot: (residents)
> modifiesEntry: (Guha)
> BecameTrueIn: (1987)
> surpisingTo: (Guha Lenat)
> MoreLikelyThan: (SeeUnitFor-PickupTruck∈ ownsA Mary)
>
> and that unit would be pointed to by the Guha entry in the residents slot of the Texas unit:
>
> **Texas**
>
> capitol: (Austin)
> residents: (Doug Guha Mary)
> stateOf: (UnitedStatesOfAmerica)" [[LEN90]p.39].

Information systems with heterogeneous notions of identity are integrated by linking local data concepts to the common knowledge-base and then generalising these matches in the formal specification called articulation axioms (described in Section 5.2.1.3). Objects can be identified in the global knowledge-base provided an object has been identified locally on the following levels [COL91]:

Data Model	Individual Object
Object-oriented data model	Object instance level,
Relational data model	Relational tuple level,
Hierarchical data model	Hierarchical record level,
CODASYL data model	CODASYL record level,
Entity-relationship model	No individual level

Table 11: Local Level Object Identification in CARNOT

The MKS master model and system by Pan and Tenenbaum [PAN91a] has been briefly described in Section 3.3.1.2. However, in contrast to the CYC knowledge-base, all processes, entities, objects, etc., in the enterprise are modelled in an object-oriented data model, which uses surrogate based identifiers. In other words, every object or process in the enterprise is given a strong notion of identity, based on a system generated identifier within the global model.

A third kind of integration is implemented in the Pegasus project by Ahmed *et al.* [AHM91] that is based on a homogeneous, object-oriented home system to which heterogeneous sources are linked via gateways. In other words, the home system has a common, strong notion of identity. Heterogeneous sources are integrated, similar to the generalisations described above for the CARNOT system, into the global, or 'home', system. For example, a relational database may hold a person with the name 'Peter'. This person may be identified by the user defined key Name in the relational database. To import the data for this person into the home system, the data on 'Peter' is retrieved from the local source, and the result is then given a unique identifier (surrogate) by the home system. This result becomes a member of the 'home' system and can be integrated within this object-oriented (database) system.

In Table 10 also the KBZ approach by Oxborrow is listed which only varies from Pan and Tenenbaum's approach in not being based on an enterprise model but simply a table in which every objects is represented. The KBZ approach is, however, directed at integrating databases, and not at integrating large numbers of information sources in an enterprise.

Most distributed information systems use a strong notion of identity (surrogates) on the global level for the same reasons it has been used on the local level, for example:

- The merging of databases and programming languages (software components) requires a persistent identification of objects.

- Objects with a strong and therefore persistent notion of identity can be shared by multiple programs, and can be replicated over multiple locations.
- Mechanism to modify the identity of objects can only operate if there is a concise way of identification, etc.

Section 5.2.1.2 concluded that strong notions that provide location, value, structure and session independence are best suited to represent objects on the global level of an enterprise integration environment. It was further demonstrated that a strong notion of identity on the global level should be implemented by weakening local autonomy. Local autonomy is weakened by enforcing stronger notions on local sources so that these cohere with a strong global notion of identity [ELI91]. Hence, a novel structure for object identification and sameness is introduced in Sections 5.2.1.4 and 5.2.4.3. This novel structure draws on the following research.

Eliassen and Karlsen [ELI91] have identified the following three support levels for object identity:

1. Immutable object identifiers;
2. Value based object identifiers;
3. Session based object identifiers.

Immutable identifiers are persistent and do not change during the lifetime of an object such as surrogate identifiers. Value based identifiers, such as user defined keys in relational data models, can change during the lifetime of the same object. A person, that is identified by the Name 'Smith' can have their name changed to 'Horton', which changes its identifier while the object is still the same. Finally, session based identifiers only survive a particular transaction or process. These are, for example, found in programming languages that use variable names or memory addresses (e.g. C++ pointers) for objects which are only valid for a particular process.

Eliassen and Karlsen show how these levels can be mapped into a strong (ID-based) global notion of identification if the local sources have restricted update-autonomy. A possible worlds contexts is used to integrate the heterogeneous information systems. Each information source is a possible world which is linked to other information sources / worlds. Mappings from an information source (local world) to the global system (global world) are generalisations between these worlds.

A formalism is introduced of the form :

(id, T(Tj,ej)(Ti, ei)...)

- A global identifier id is of the type immutable, session, or value based;
- T is the generalisation used for the mapping between local and global level for the particular object j;
- The object j is identified (filtered) by the requirement e. This requirement e can be an id number, a value for key attributes, or a session identifier. If objects are replicated then multiple T generalisations can be defined for the same object to multiple sources.

The approach is essentially based on strengthening the weaker notions of identity in the integrated sources. For example, by restricting the autonomy of a relational database to change any user defined keys of objects that are represented in the global system, such that the value based identification becomes 'stronger' [KHO90] or more persistent.

It has been discussed above (Section 3.3.1.4) that such tight integration is difficult to implement in large, heterogeneous information systems spanning the whole enterprise. A 'global registration' is not possible for weak notions of identity. Hence, the conflict detection and resolution mechanism designed in this research could not be built by assuming this type of tight integration.

A second deficiency is that other types of sameness than the 'identical' predicate exist in information systems. For example, in Khoshafian and Copeland [KHO90] and Masunaga [MAS90] different notions of sameness have been described (Section 5.2.4.3).

Less restrictive on the local autonomy is, for example, the approach by Heiler and Blaustein [HEI89] who have developed "a mechanism for assigning and manipulating object surrogates and identifiers in an Object Management System [(OMS)] that integrates heterogeneous systems. The mechanism preserves external-assigned identifiers for use by clients and application programs and provides a uniform scheme for accessing objects that make transparent their origins and location, without requiring that each object be registered with the OMS" [[HEI89]p.236].

The OMS manages the identity of objects in a home system by generating surrogate identifiers. External objects, from external information systems are integrated in a way that preserves their identifiers in the original sources. Functions are used to import or retrieve information from an information source similar to that described by Eliassen and Karlsen [ELI91], and Kent *et al.*[KEN93]. Multiple objects can be imported by one function. These objects are specified by a common type.

"The OID [object identifier] actually consists of the pair <type identifier, identifier within type>" [[HEI89]p.241].

The type identifier specifies the function that is used to integrate the object. The 'identifier within the type' is the actual, external identifier such as 'Peter' for a relational table in which a row is identified by the key 'Name' of the object. The identification is specific in that it supports type specific type structures. In other words, every 'identifier' has to be individually specified within the 'type'. Hence, the approach lacks generality and cannot be conducted automatically by investigation, e.g., of an information agent in existing enterprise integration environments.

Sameness of objects is based on the same object identifier. For example, two external objects that have the same 'identifier within the type' are identical. Furthermore, objects can be defined as identical by a hand-crafted function called 'coercion function'. In other words, objects are defined as 'identical' by manual mappings (coercion functions) or on the basis of 'same identifier same object'. No other sameness predicates, e.g. described by Khoshafian and Copeland [KHO90] and Masunaga [MAS90], are defined.

The approach adopted by Heiler and Blaustein [HEI89] is one of many that apply functions to integrate objects. All these approaches integrate objects, called foreign objects, into a home system conceptually similar to Ahmed's approach described above. A major trade off is the inflexibility of implementing functions. Hence, Section 5.2.1.4 will present a synthesis of the described approaches: It uses concepts similar to functions, but it is more general similar to the approach described by Eliassen and Karlsen [ELI91]. In other words, the present research designs an identifier that is tailored to the information available to information agents through the use of Identifier_Classes (Section 5.2.1.4).

B.5 Summary and Conclusion

Different notions of identity have been described in Appendix B. Furthermore, external identification of objects has been outlined. Approaches to integrate these heterogeneous notions of identity are typically based on weakening local autonomy. However, some research, e.g. by Eliassen and Karlsen [ELI91], or Heiler and Blaustein [HEI89], provides ways to integrate heterogeneous notions of object identity in a more flexible way.

The novel approach described in Section 5.2.1.4 and 5.2.3.4 extends and combines the last two approaches with Khoshafian and Copeland's object structure [KHO90]. In particular, it automates the functional approach by Heiler and Blaustein by applying the concept of Identifier_Classes from Eliassen and Karlsen. This allows for the integration of principally any notion of object identity, e.g. such as those described in Section B2. Furthermore, the novel approach is embedded in an object structure that is typically used in enterprise integration [KHO90]. This provides a complete set of sameness predicates for the global system.

Appendix C: The Lewis System of Counterparts

Counterpart theory is based on the following four predicates :

W_x (x is a possible world)
I_{xy} (x is in possible world y)
A_x (x is actual)
C_{xy} (x is a counterpart of y)

The four 'possible worlds' primitives are based on the following eight postulates as in [[LEW68]p.114]:

P1: $\forall x \forall y\ (I_{xy} \supset W_y)$
(Nothing is in anything except a world)

P2: $\forall x \forall y \forall z\ (I_{xy} \,\&\, I_{xz} .\supset y = z)$
(Nothing is in two worlds)

P3: $\forall x \forall y\ (C_{xy} \supset \exists z I_{xz})$
(Whatever is a counterpart is in a world)

P4: $\forall x \forall y\ (C_{xy} \supset \exists z I_{yz})$
(Whatever has a counterpart is in a world)

P5: $\forall x \forall y \forall z\ (I_{xy} \,\&\, I_{zy} \,\&\, C_{xz} .\supset x = z)$
(Nothing is a counterpart of anything else in its world)

P6: $\forall x \forall y\ (I_{xy} \supset C_{xx})$
(Nothing in a world is a counterpart of itself)

P7: $\exists x\ (W_x \,\&\, \forall y\ (I_{xy} \equiv A_y))$
(Some world contains all and only actual things)

P8: $\exists x Ax$
(Something is actual)

A world in distributed information systems is typically one information source. Section 5.2.1.4 defines a world as a system with a common notion of identity. Hence, a world may also be part of an information source, or a group of sources. An example of the latter case are homogeneous distributed databases like ORION-2 by Kim *et al.* [KIM91a]. It is necessary, however, to ensure that 'nothing is a counterpart of nothing else in its world' (P5).

The actual world is equivalent to the real world or a model of the world that is taken for the real world such as enterprise models. Accordingly Lewis [LEW68] specifies the

actual world as 'some world contains all and only actual things' (P7); 'Nothing is in two Worlds' (P2); and 'Something is actual' (P8).

'Possible worlds' are any other worlds, typically the information sources in the integration environment. For the definition of the possible worlds framework to model distributed information systems see, for example, Duong and Hiller [DUO93a].

The system described by Lewis [LEW68] does not imply the following three constraints and it is therefore possible that:

1. Objects can have more than one counterpart in another world such that those objects are identical and both are counterparts;
2. Multiple objects in one world can have a common counterpart in some other world;
3. Objects may not have any counterpart in a particular world, or in any world,

Cases one and two cover compression and dispersion [MAI91]. Two objects in the real world can be represented as one in any possible world (compression), and two objects in any possible world can be equivalent to any one actual object (dispersion).

Counterparts are not transitive so that equivalence or counterpart relations could be inherited. For example, if O_1 is a counterpart of O_2 and O_2 is a counterpart of O_3 then that does not imply that O_1 is also a counterpart of O_3. Counterpart relations differ in this respect from accessibility relations between worlds [HIN62], and any centralised identification mechanism such as object tables in, for example, the KBZ model by Oxborrow and Ismail [OXB88].

Appendix D: Information Systems in the Integration Environment 'Cafeteria'

D.1 Object-Oriented Database 'ProductionDB'

The 'production database' ('ProductionDB') is implemented in the object-oriented database management system POET [POET]. The production related data is stored in three major object classes called Production, Employee and Robot.

In addition, a number of classes are installed by the database management system to provide management functions (e.g. PtCluster, PtRight, PtUser, etc.). An overview of all the classes in the database is provided by the POET DEVELOPER (Figure 18). This tool allows browsing of all classes and the inspection of individual objects.

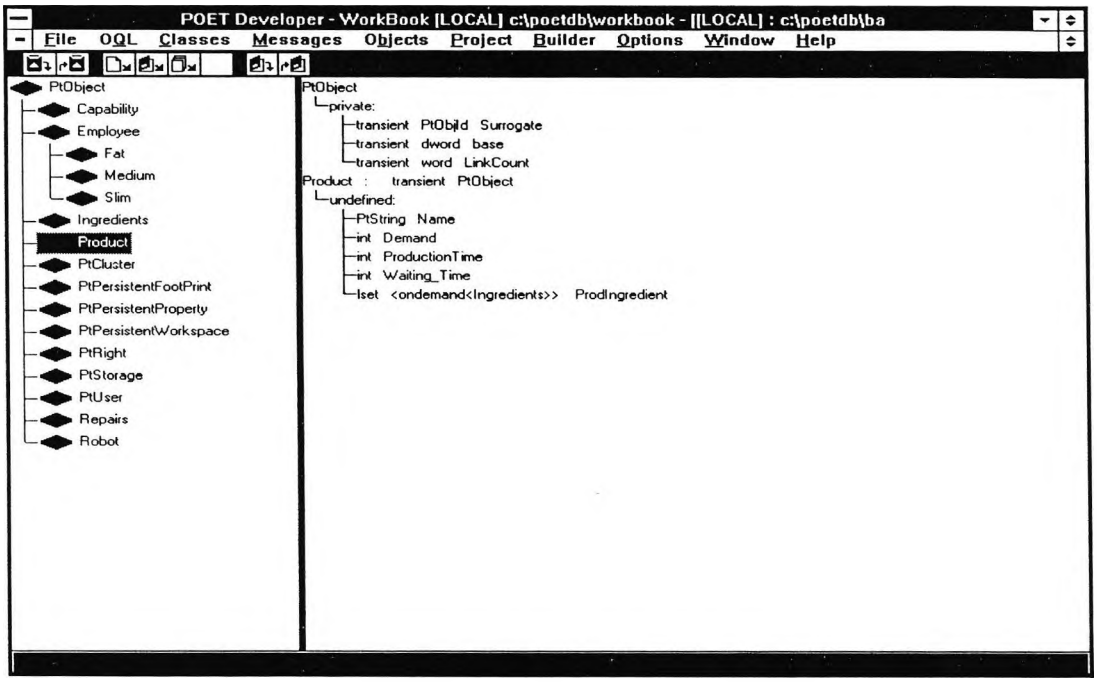

Figure 18: POET Developer - Class Overview

All classes are derived from the class PtObject, which provides every object in the database with its unique surrogate, specifies the database in which this object is located, and provides information on the current links to this object. For example, an employee object called 'Peter' automatically is an instance of PtObject, where it receives its Surrogate from, and in addition is a member of the class Employee, where the information on the employee's name (e.g. 'Peter') is stored.

A customised interface has been programmed for the database called PoetDB. It is implemented in Microsoft (MS) Visual C++ (Quickwin) [MSV]. It enables the input of data to the database, and the retrieval of all the implemented data. The interface runs in a program called 'PoetDB.exe' and has two windows. The main window contains a menu-

driven interface and the other window provides an overview of the classes of this application (Figure 19). The interface could be an agent's window (or local view) on the object-oriented database ProductionDB.

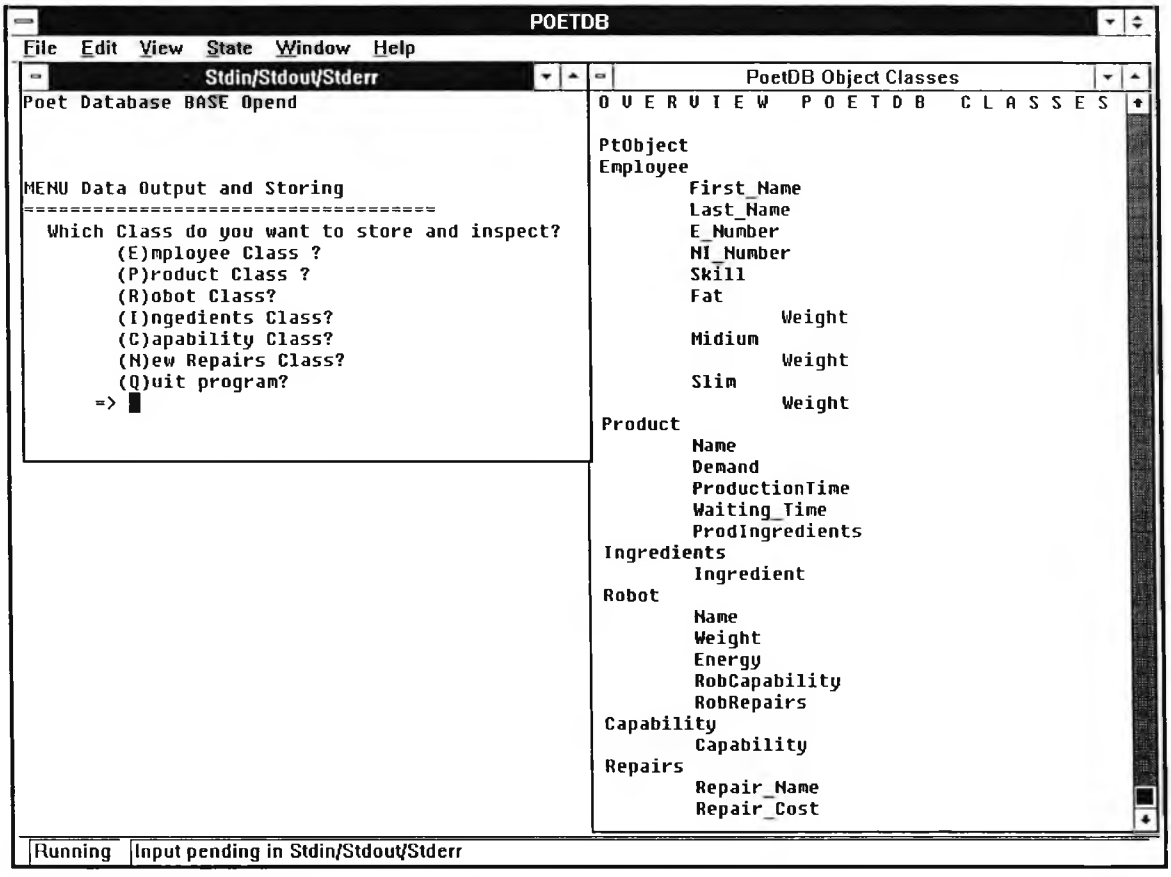

Figure 19 PoetDB Application Interface

Figure 20 provides an overview of the implementation of the ProductionDB with the hierarchical relations between complex objects, and between complex objects and their attributes. The modelling technique is used in research on mainstream object-oriented databases such as by Kim [KIM93]. For example, the Employee class (complex object) has the attributes First_Name, Last_Name, an employee number (E_Number), and the employee's National Insurance Number (NI-Number). In addition, the specific skill that this employee may have is listed (Skill), such as 'Selling', or 'Cooking'. Finally, every employee's body weight is stored. The Weight attribute provides the basis to specify the Type of employee as Fat, Medium, and Slim. This information is important because the company's marketing department has found out that fat people sell more diet products, and that moderate (Medium) people sell more hamburgers. In this object-oriented database the class Employee has three derived classes Fat, Medium, and Slim, which inherit all properties of the Employee Class (First_Name, Last_Name, etc.) and also specify the employee's weight in stone.

The ProductionDB has information on every Product, including its Name, the expected Demand of this product per hour, and the foreseen time to produce one unit of this product (Production_Time). For every product, a company policy has defined a specific maximum time that a customer has to expect to wait for this product (Waiting_Time). Each product requires a number of ingredients that are necessary for its production (ProdIngredients). Because there are multiple ingredients per product, the attribute ProdIngredients has references to one-to-many instances of the class Ingredients. This class has only one attribute, which stores the ingredient's name (Ingredient).

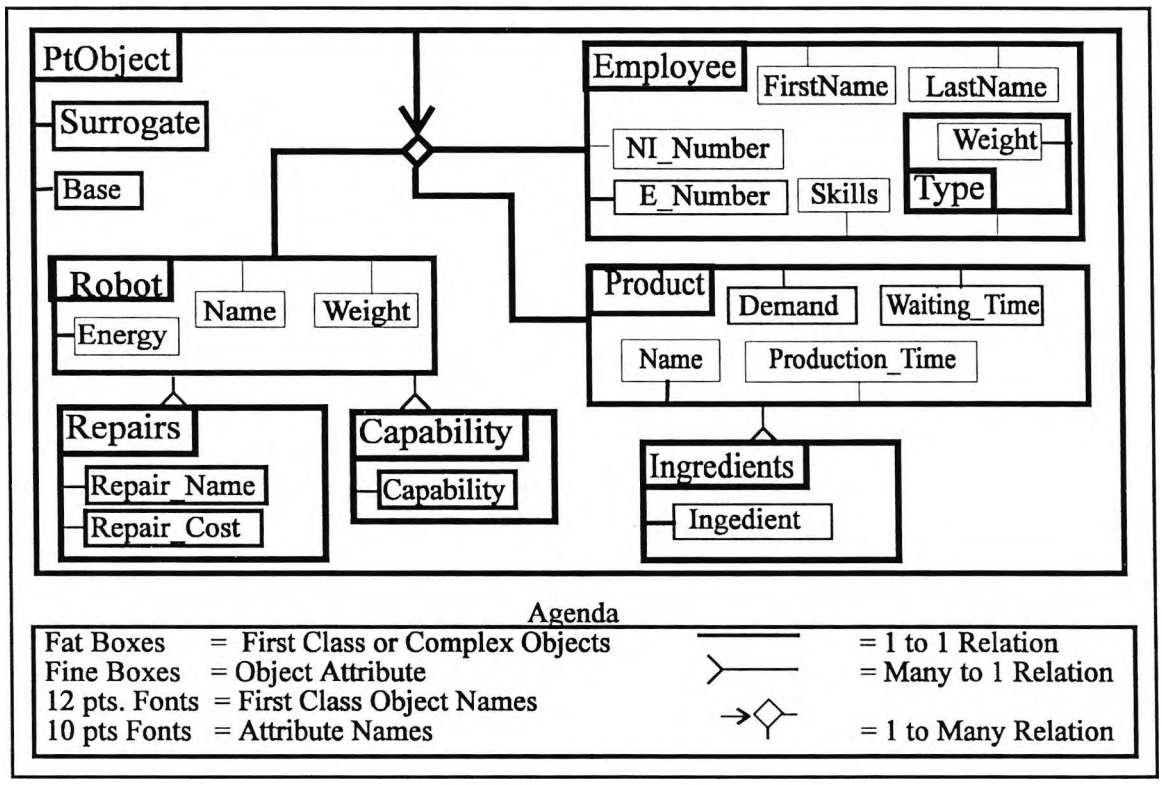

Figure 20: Overview ProductionDB

Information on each Robot on the shop floor is stored in the Robot class. This class has the attributes Name, Weight (for the Weight of the Robot), and the Energy (in kilowatts per hour) that the robot typically needs when in operation. Furthermore, each Robot can provide one or many services such as produce a number of products. This information is stored in the class Capability of which each instance can be a member of the robot's set RobCapability. The class Capability only has a name of the capability (attribute Capability). In addition any repairs that have been carried out on this Robot are specified in the class Repairs with their Name (Repair_Name) and Costs (Repair_Cost). The robots attribute RobRepairs references one or multiple Repair instances.

D.2 Relational Databases 'BookkeepingDB' and 'MaterialDB'

A bookkeeping database (BookkeepingDB) for the finance department, and a small database for information on product materials (MaterialDB) have been implemented in the relational database management system SQLBASE and application development tool SQLWindows (by Gupta [GUP]). An interface to each database includes an Overview window, which lists all the tables, the names of their columns, and the key of each table (in curly brackets). A second window provides a menu driven interface to the BookkeepingDB and one to the MaterialDB. Figures 21 and 22 show the interfaces to the BookkeepingDB and the MaterialDB.

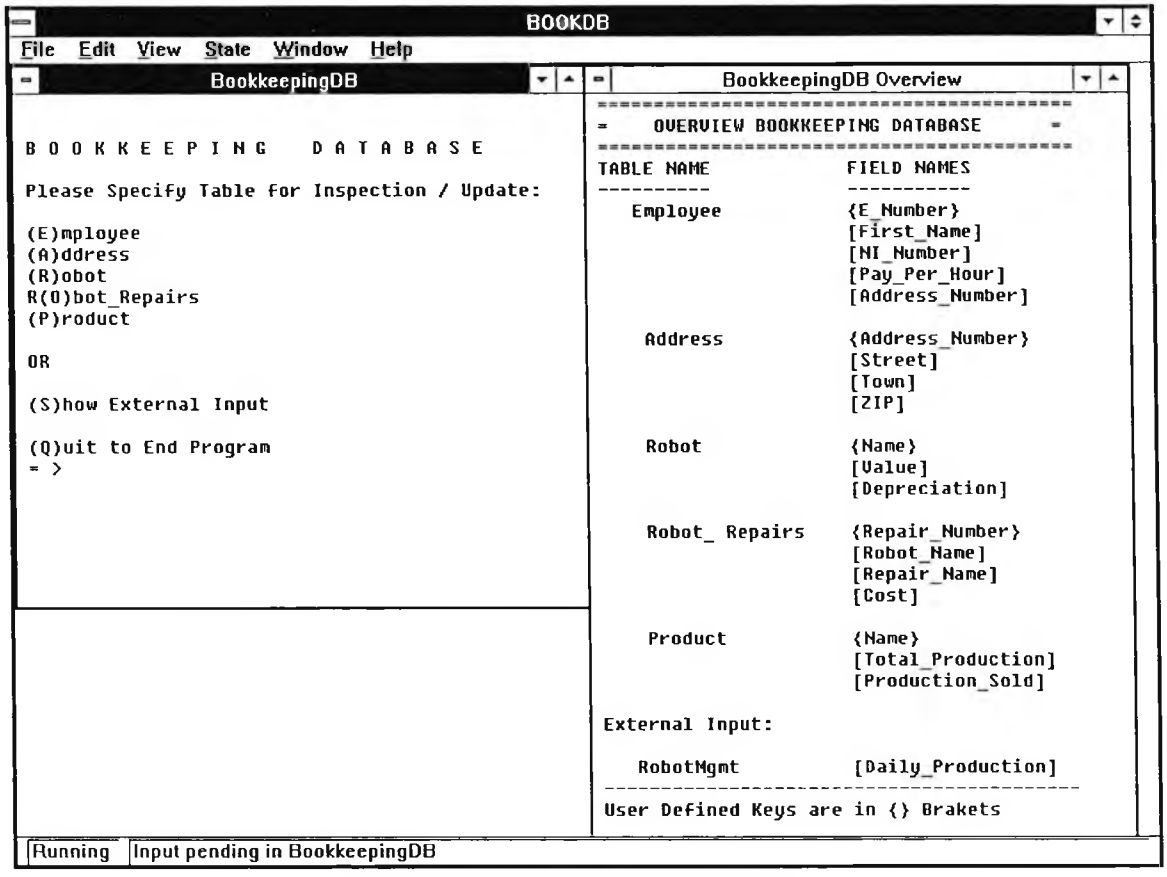

Figure 21: BookkeepingDB Interface

The SQLWindows application development tool has been used to program graphical, windows-based interfaces to the SQL databases. In other words, selecting a table from the main menu in the interface opens a second window with one, or multiple related tables (e.g. Figure 23 for the tables Employee and Address). 'Select', 'update', 'insert', and 'delete' operations can then be performed on the selected table via the options in the menu bar. For example, the results of a 'select' operation are presented in this empty table. The information can also be printed in a report.

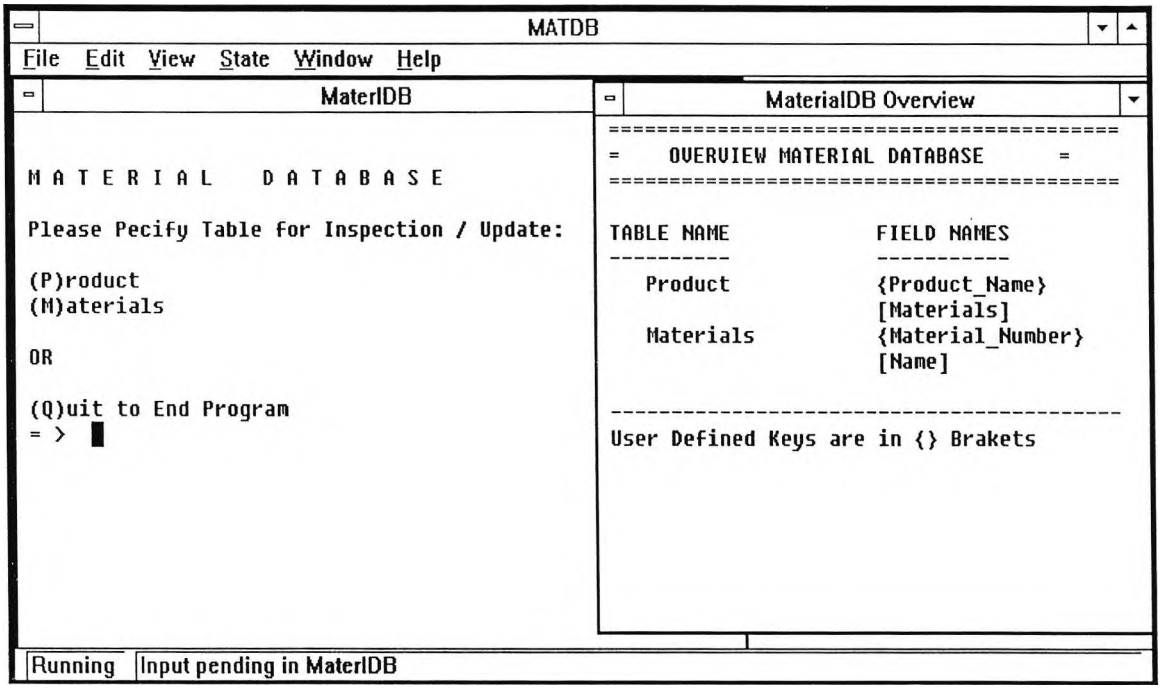

Figure 22: MaterialDB Interface

In addition to the graphical interfaces, the database can be accessed directly via the SQL-Talk utility. This enables a direct view on all the information that is available in the database, and not just the information presented in the interface. Hence, the latter presents an agent's view on an autonomous database. SQL-Talk runs with standard ANSI SQL [SQL].

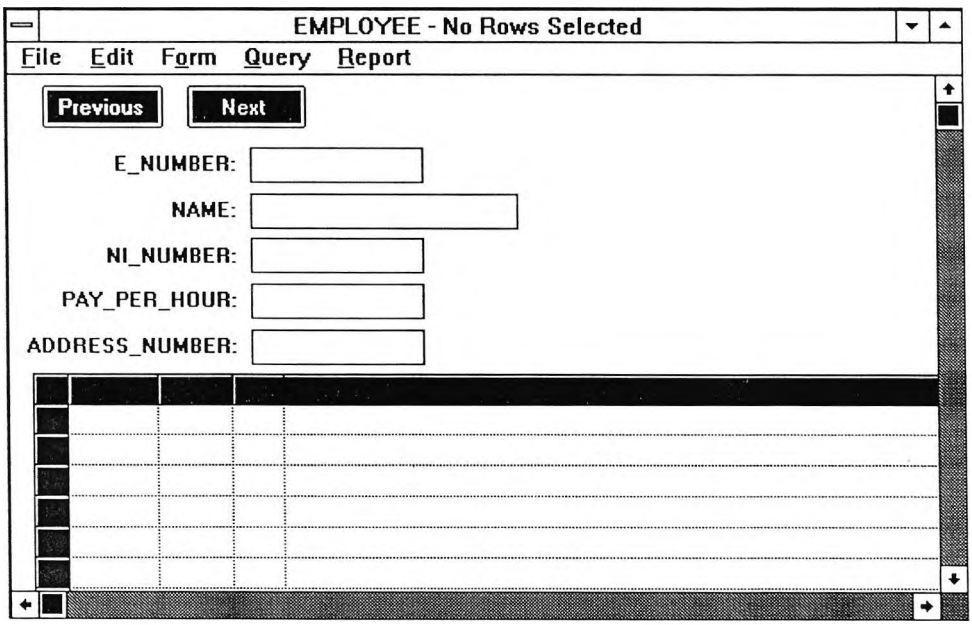

Figure 23: Table Employee and Address Interface

Figure 21 shows the information available from BookkeepingDB in the overview window within the interface to the BookkeepingDB. It includes a table Employee with

the key employee number (E_Number), and the attributes First_Name, the National Insurance Number (NI_Number), the hourly rate that an employee earns (Pay_Per_Hour), and the employee's address (Address_Number). All addresses that are known to the system are stored in the table Address with the attributes Street, Town, and Zip (Zip-Code). Each employee may have one address that is linked to the Employee table via the attribute Address_Number (also the key). Hence, when an employee entity is manipulated or inserted, its address dependencies are automatically shown in the interface Employee and Address (Figure 23). For example, the employee 'Peter' may have the address number '12', and this address number, with its Street, Town, and Zip specifications, is shown whenever the employee 'Peter' is selected.

The table Robot has the key Name. In addition each Robot has the attributes Value and Depreciation. The table Robot_Repair with the key attribute Repair_Number stores information on every repair that has been carried out on a robot. In other words, each repair job has a Robot_Name for the name of the robot that has been repaired. The Repair_Name specifies what has been repaired. Furthermore, the Cost of this repair is specified. In contrast to the repair information in the ProductionDB (Section D.1), the information in this table is inserted when the bill is being processed in the book-keeping department (and not when the repairs are actually carried out on the robots).

Finally, the BookkeepingDB has a table Product with the key Name. The column Total_Production specifies the quantity of this product that has been produced. The daily production is concurrently updated with the latest production data from the RobotMgmt (Section D.5). In other words, a report (in the file 'bookdb.a') is produced by the RobotMgmt whenever any products are produced in the kitchen. This report can be read in via the main menu of the BookkeepingDB interface. In addition the number of pieces that have actually been sold is read periodically from the staff on the cash registrars into table Production attribute field Production_Sold. The number of products sold (Production_Sold) is typically lower than the produced goods (Total_Production). It is easy to imagine for the following reasons:

- High class fast food products can only be stored a few minutes and cannot be re-heated so that they have to be thrown away after a specific time ;
- Products may be stolen by staff or students.

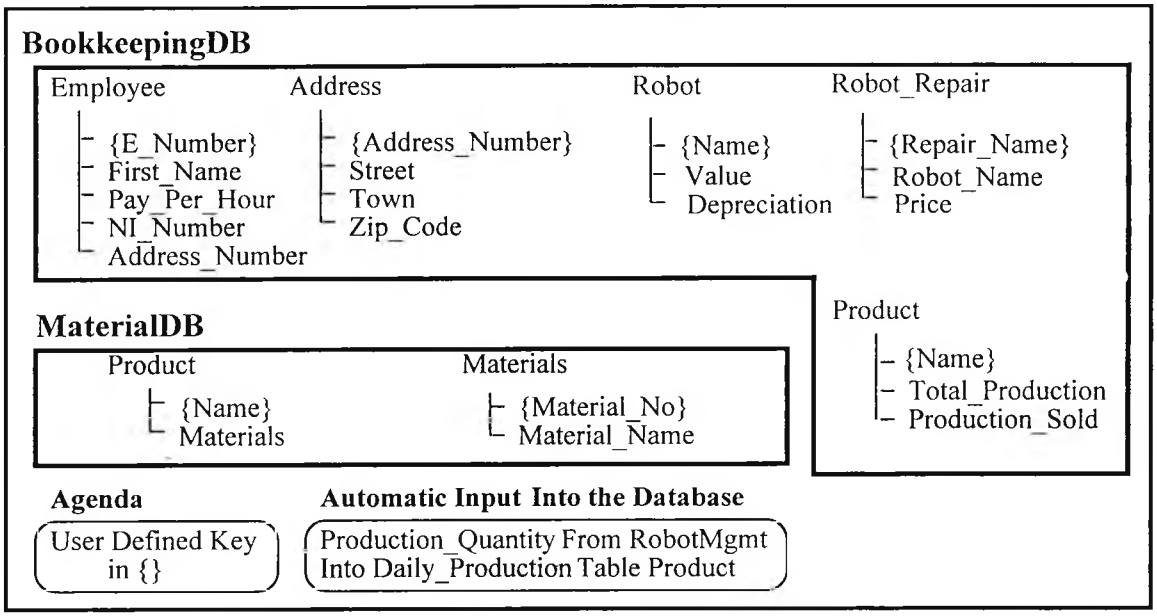

Figure 24: Overview BookkeepingDB and MaterialDB

The database MaterialDB has only got two tables: Product and Materials. Every Product has a Name (key), and a Material_Number. This number is used in the second table Materials to link via the Material_Number to multiple Material_Names (all the materials that are ingredients of this product). Manipulating and investigating either table (e.g. Product) shows the dependencies to the other table (the Product's Materials).

D.3 Expert System 'MarketingEXP'

The Marketing Expert, called 'MarketingEXP', is a small, one-rule, knowledge-based system that is implemented in MS Visual C++ [MSV]. The system contains two windows, an interactive interface to the knowledge based system, and an overview window that provides a brief description of the systems input and output (Figure 25).

The Marketing Expert system is part of the marketing department, which develops strategic plans for the production management. A brief data flow diagram provides a basic overview in Figure 26. The system has a knowledge-base that is stored in a file called 'Impio.a'. For every product it holds the current sales (Current_Sales) and demand figures (Current_Demand). The current sales figure is a mean over the actual sales, which may, for example, be calculated for the marketing department by the shop floor management. The demand is an estimate within a marketing strategy for the marketing department. The figures in the knowledge base can be inspected and updated in the MarketingEXP interface.

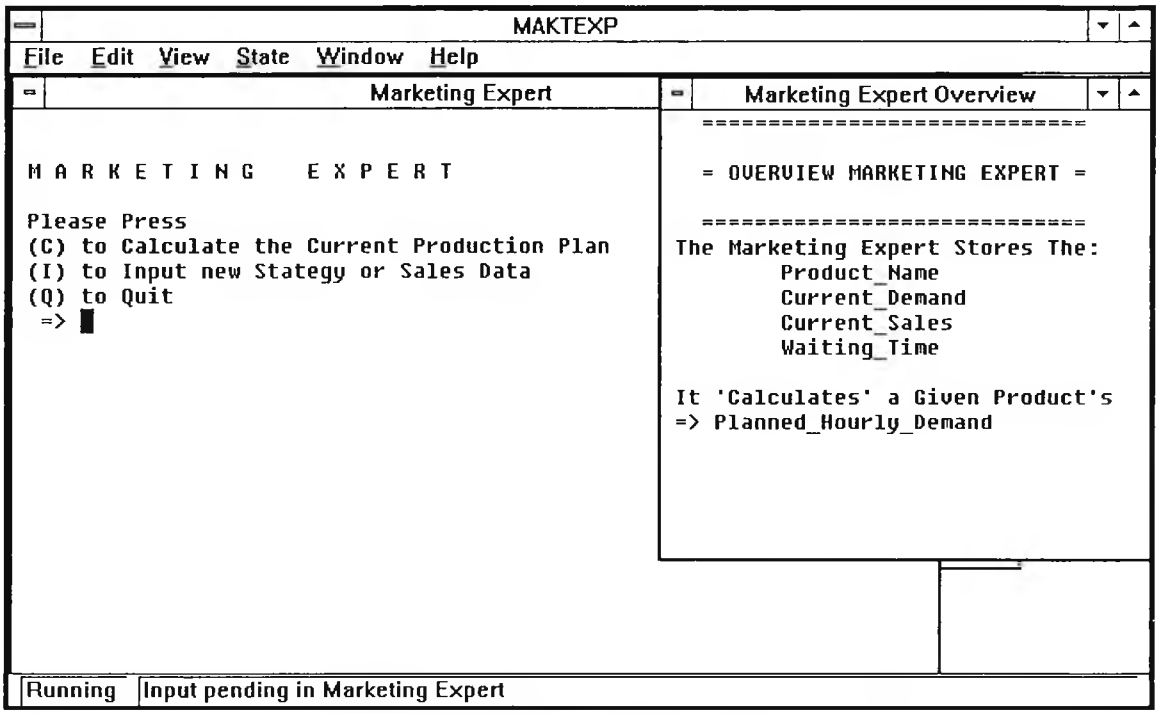

Figure 25: MarketingEXP Interface

The expertise of this system is limited to the following rule, that calculates the current production plan expressed by the Planned_Hourly_Demand for every product:

If the Current_Sales are lower than the Current_Demand per hour;

Then that demand figure (Current_Demand) is still the targeted Planned_Hourly_Demand.

Else the Current sales are equal or higher than the Current_Demand (Current_Sales >= Current_Hourly_Demand) and the targeted Planned_Hourly_Demand is the arithmetic mean between the Current_Sales and the Current_Hourly_Demand.

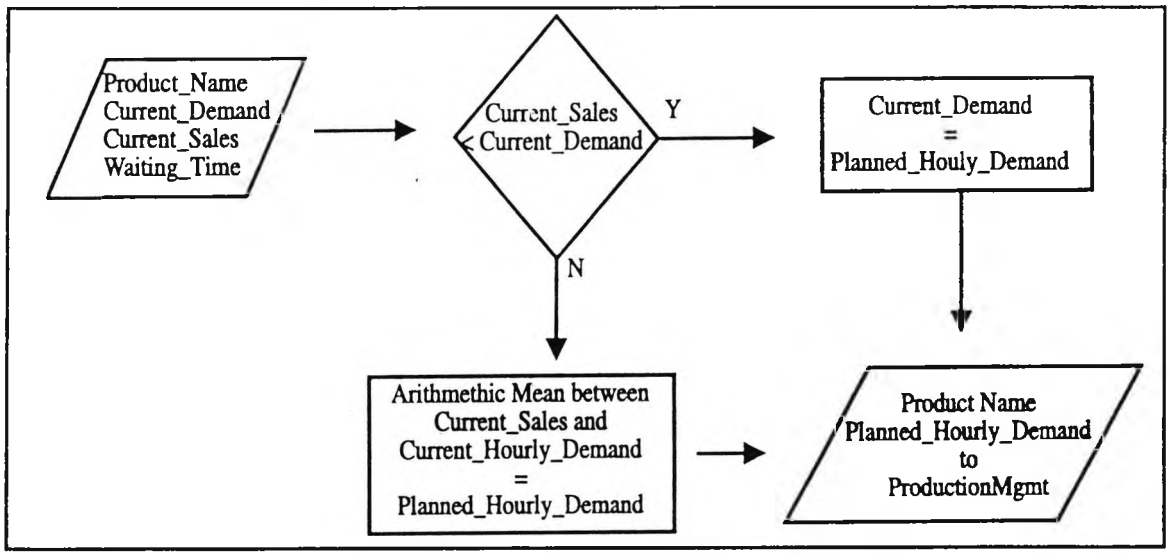

Figure 26: Data Flow Diagram Marketing Expert

The Planned_Hourly_Demand from the Marketing Expert system is proposed as a guideline to the ProductionMgmt system (Section D.4). The output format is the product name (Product_Name) and its planned demand (Planned_Hourly_Demand). The file 'mktplan.a' contains a list of each product's planned demand figure.

D.4 Standard Software System 'ProductionMgmt'

The production in the fictitious cafeteria is managed by a co-ordination software system called 'ProductionMgmt'. A brief overview is provided in the 'Overview' window of the ProductionMgmt interface that has been programmed in Visual C++ [MSV] (Figure 27). The interactive window to the production management system provides an interface to the system's data files and monitors the calculation of the demand for the management of the shop floor production (Figure 28).

For every product, the data file (ProFile.a) within the ProductionMgmt system stores its name (Product_Name), its demand (Current_Hourly_Demand), and its current sales (Current_Sales). The Current_Sales figures are provided to the management system by the staff working the cash registers. In other words, the staff enter the information in the form Product_Name and Current_Sales in the ProductionMgmt interface, which then updates its internal file system accordingly. The Current_Demand figure is constantly updated with the latest demand calculated by the ProductionMgmt system through the following process.

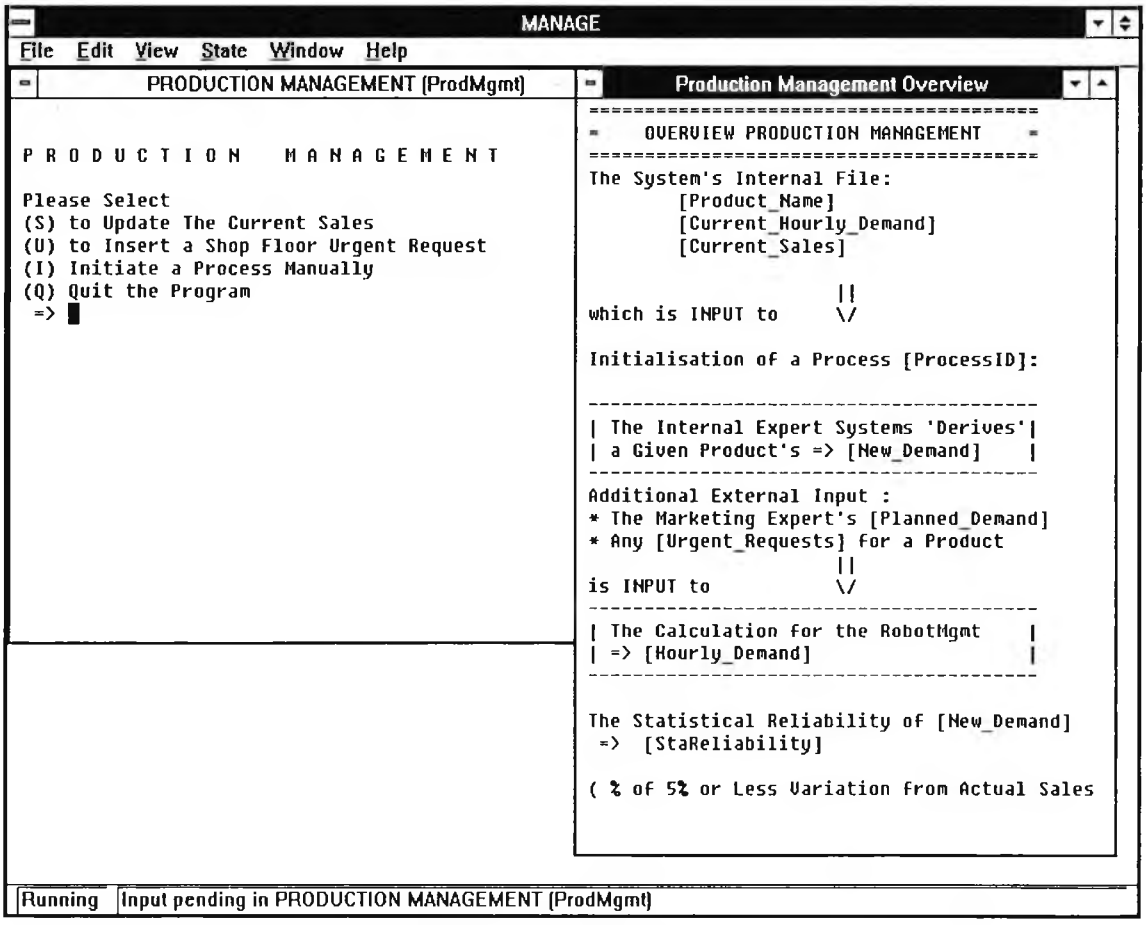

Figure 27: ProductionMgmt Interface

Within the ProductionMgmt is a very small expert system that calculates the new demand for a given product. In other words, the ProductionMgmt waits for an update of the current sales (Current_Sales) figure of a given product X by staff on the cash registrars. This change in the file systems initiates the ProductionMgmt to select the Current_Demand and Current_Sales variables of the product X. This initialisation can be manually interrupted, and may also be manually initiated (without having to change the Current_Sales figure).

At the beginning of a process in the ProductionMgmt, a process identification number (ProcessID) is assigned to these two figures. This process is then inserted into the internal expert system, which calculates the New_Demand by the following rule:

If	the Current_Sales are equal or higher than the Current_Demand,
then	New_Demand is set to Current_Sales.
If	the Current_Demand minus five is bigger than Current_Sales,
then	New_Demand is set to Current_Demand minus five.
Else	New_Demand is equal to the Current_Demand minus 1.

The result from this calculation is a product X's new demand figure (New_Demand). Furthermore, this rule is also the basis for justifying the calculated New_Demand figure (rule-based justification). Hence, the application of this rule provides evidence (Sections 2.8 and 4.3) for a calculated New_Demand result.

The next step is to calculate how many pieces of this product are going to be produced in the kitchen. The kitchen is managed by the RobotMgmt system (Section D.5), which simply produces the quantity of a product per hour as the ProductionMgmt's demand figure orders. This order is defined by the Hourly_Demand figure. It is calculated from the process's New_Demand, Product_Name and the Planned_Demand figure from the MarketingEXP. In other words, this calculation takes into account the projected or planned demand for that product from the marketing department. This is the demand that the marketing department expects for a given product (Section D.3). It can be read from the file 'Mktplan.a' in the marketing manager's directory.

In addition, the staff on the shop floor can send an urgent request for a particular product to the ProductionMgmt. For example, a big group of first degree students may have just entered the cafeteria, and ordered twenty hamburgers and fries. This would then be send as an urgent request (Urgent_Request is '20', Product_Name is 'Hamburger') to the ProductionMgmt, which will ensure that this request is satisfied.

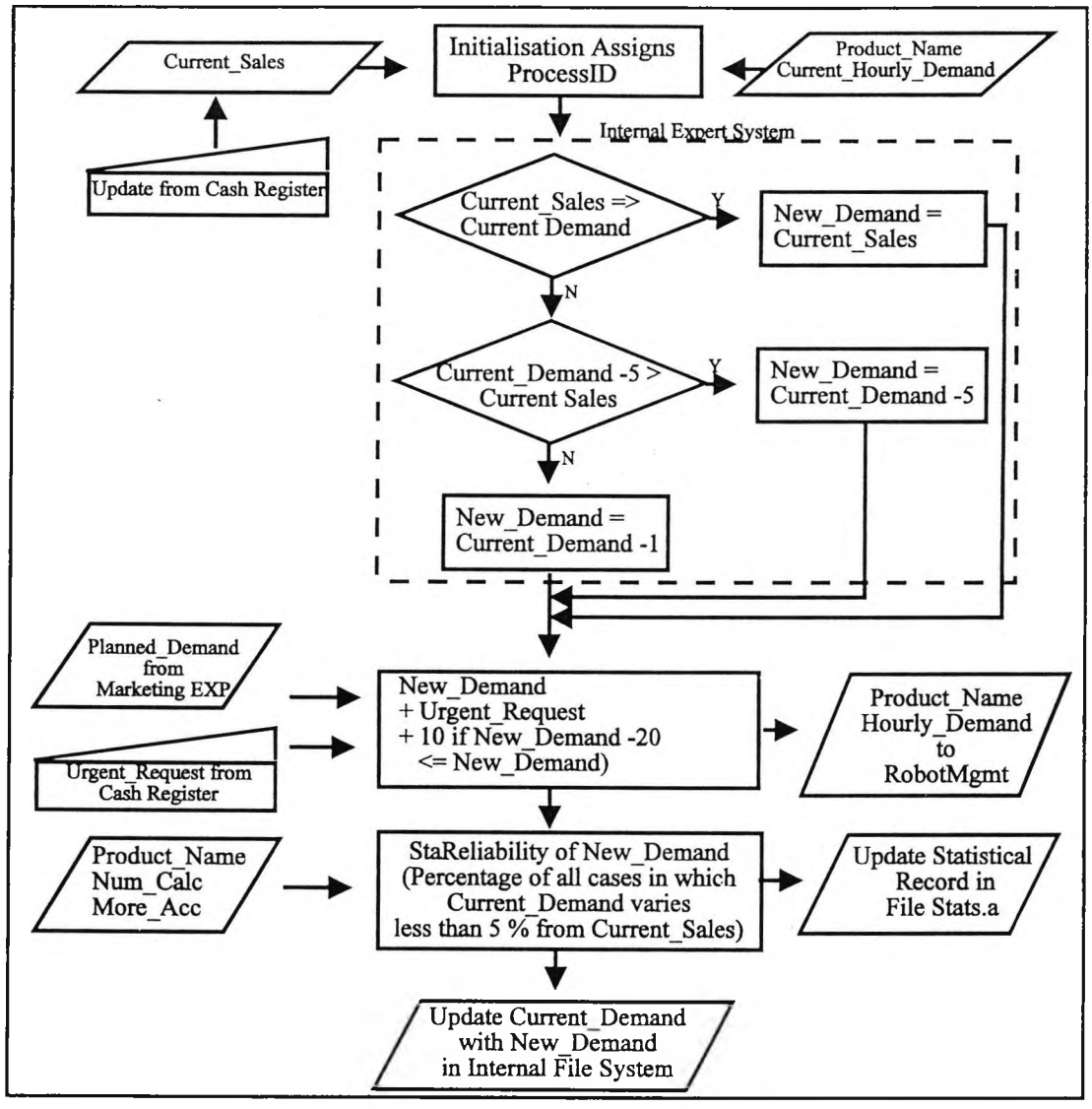

Figure 28: Data Flow Diagram ProductionMgmt

On the basis of all this information (New_Demand, Planned_Demand, and Urgent_Request) the ProductionMgmt calculates the Hourly_Demand. In principle, the New_Demand figure is taken to be the Hourly_Demand (New_Demand = Hourly_Demand). However, if the New_Demand is more than twenty units smaller than the Planned_Demand variable from the MarketingEXP, then a further ten units are added to the Hourly_Demand figure. Furthermore, the figure from the 'Urgent_Request' variable is added to the Hourly_Demand.

In the final stage of a process the ProductionMgmt places an order for the RobotMgmt system. It stores this information in a file called 'Order.a'. For each order the process identification (ProcessID), the product name (Product_Name), and the hourly demand (Hourly_Demand) is inserted.

In addition, the ProductionMgmt computes the statistical correctness of the New_Demand figure that has been estimated by the internal expert system. In other words, the ProductionMgmt system takes the old Current_Demand figure and compares it to the Current_Sales. Account is taken whether the variation is less than five per cent. An average is calculated of the cases that have had less than five per cent variation out of all recorded cases. The result of this calculation is stored in the file 'Stats.a', which holds for every product:

- The product's name (Product_Name);
- The number of statistical calculations (Num_Calc), which is increased by one for every calculation;
- The number of times that the expert system's Current_Demand varied less than five percent from the Current_Sales (More_Acc).

At the end of the statistical calculation the old Current_Demand variable in the internal file system is updated with the value of the New_Demand as calculated by the process.

The expert system and the rest of the ProductionMgmt are closely connected, so that the process of calculating a New_Demand has the same identification number in the ProductionMgmt system as in the expert system. However, multiple concurrent processes may exist in the ProductionMgmt system. The interface window, therefore, shows with every result the according process number used within ProductionMgmt.

A.5 Co-ordination System 'RobotMgmt'

The RobotMgmt is a software system implemented in Visual C++ [MSV] that co-ordinates the 'production' in the cafeteria's kitchen. In other words, food sold in the cafeteria is cooked by a number of robots (Cooking Robots). The interface to the RobotMgmt (Figure 29) system has an Overview window that provides a basic description of the data available from the system. The interactive window can be used to monitor the process within the co-ordination system. The system can be run in a 'step-by-step' mode where the user can investigate the changing variables after each processing step.

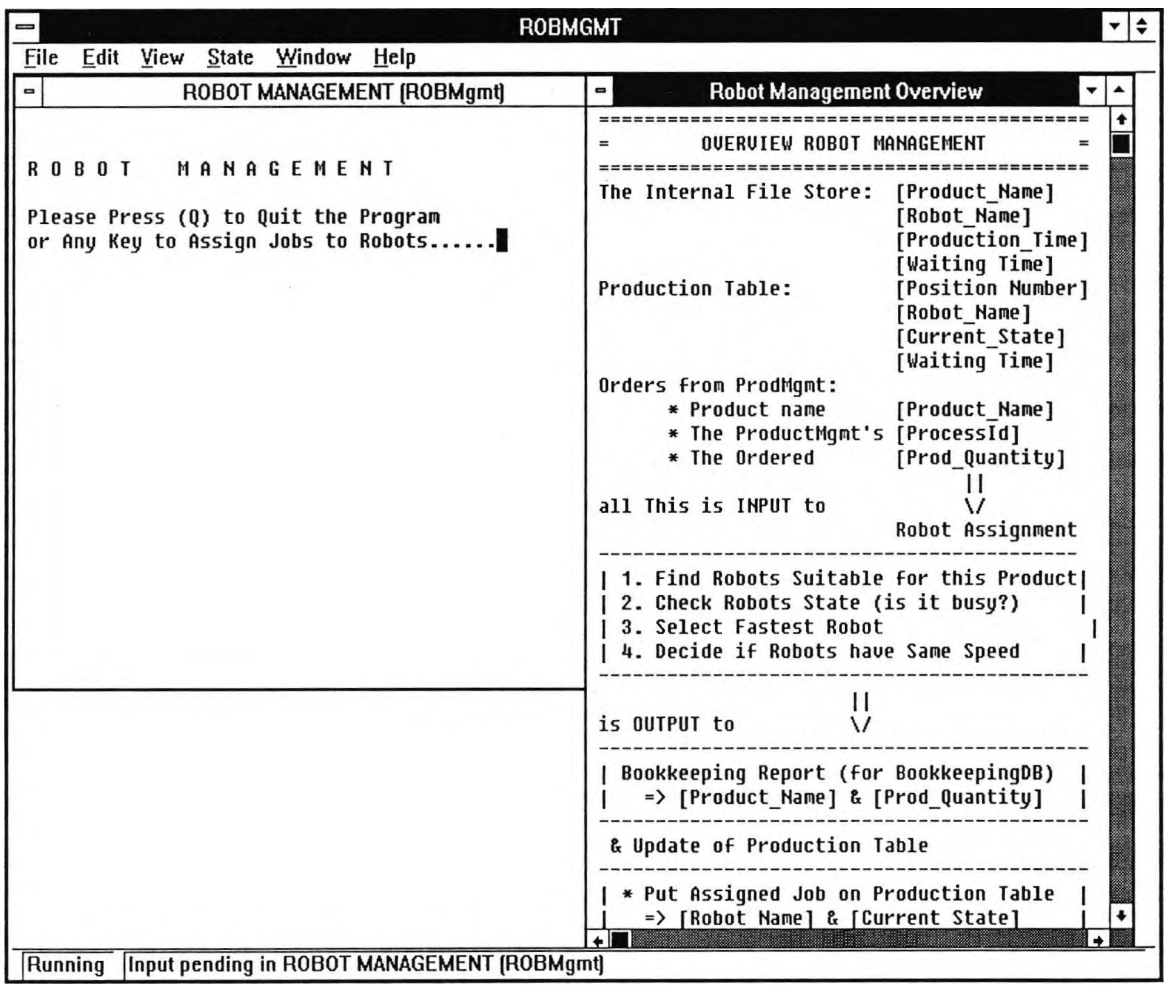

Figure 29: Interface RobotMgmt

The RobotMgmt system receives an order from the ProductionMgmt system in the form Hourly_Demand, the specification of the product's name (Product_Name), and the process identification (processed) for this order. This information is stored in the file 'Order.a'. The main task of the RobotMgmt is to assign a robot on the shop floor to the

task of producing the product according to the demand specification (Data Flow diagram Figure 30).

The co-ordination system reads orders from the ProductionMgmt system sequentially row-by-row, so that each row provides one order. For each order, the co-ordinateion system first reads the Product_Name and identifies, which Robots can produce this product. The system has an internal file system ('Robot.a'); each row contains a product name (Product_Name), and a name of a Robot (Robot_Name), which can produce this product. Furthermore, the time that this Robot will require to produce one unit of the product is specified (Production_Time). It may be the case that multiple robots can produce the same product but with different speed.

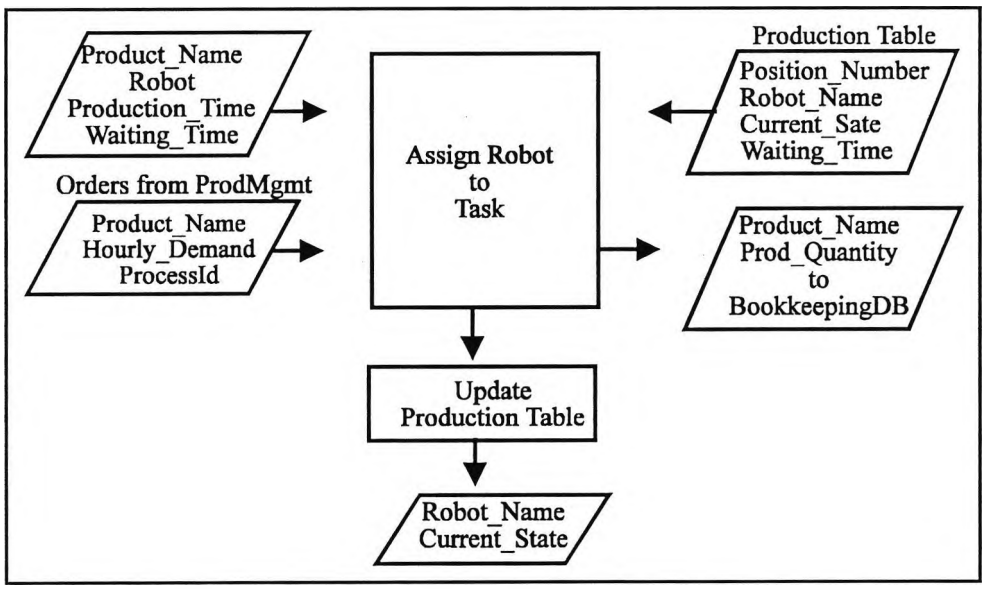

Figure 30: Data Flow Diagram RobotMgmt

The second step is to assign one robot to this specific task from the list of robots that can produce the product. It can be imagined that the robots on the shop floor are a problem-solving community in that they directly interact with each other in order to co-ordinate their activities. However, these robots are eager to have jobs assigned to them. All robots that are currently not busy bid for assignment to the job. Hence, the RobotMgmt system searches the list 'ProductionTable', which lists all the robots (by their Robot_Name) that are currently busy working on the job specified in ProductionTable in the Current_State variable. The assignment of a robot to the task at hand has the following steps (Figure 30):

1. The co-ordination system searches for Robots that have been identified in the internal file 'Robot.a' as being able to produce the current order.

2. If more than one robot can produce the product, then those that are currently not busy are identified (no entry in the ProductionTable).
3. If multiple robots are 'not busy', or all robots are busy, then the one that can produce the product fastest (smallest Production_Time) is selected. In case robots are equally fast, then the task is assigned to the robot that is listed first in the system file 'Robot.a'.

Once a job is assigned to a robot it is put on the ProductionTable automatically. The table can be updated whenever a new job has been assigned. For example, an update is necessary when a robot has finished a job. A hard copy of this list is saved in file 'ProdTab.a'. Every job is internally identified by its position in the ProductionTable. This position is shown for every job in the interface.

The data output of the RobotMgmt system is a 'Production Report' to the database of the bookkeeping department (BookkeepingDB). When ever the RobotMgmt system has assigned a job to a robot it stores the product's name (Product_Name) and the quantity that has been produced (Production_Quantity) in the file 'RobOut.a', which can be read into BookkeepingDB.

D.6 Enterprise Model

In Section 3.3.1 it has been briefly outlined that much research in the field of Enterprise Integration and Modelling has described enterprise models. In the present research generalisations such as $(G(ist(G\Phi) \Leftrightarrow (ist(C_i\Psi))$ [HUH93], can be defined between an information sources and the global context or enterprise model. Counterparts have been introduced as a more exact extension to generalisations in Section 5.2.1.4. They facilitate the assignment of individual objects from information sources, e.g. the object 'Peter', as counterparts of global objects in the enterprise model. Generalisations and counterpart relations to objects in the enterprise model are stored in an information system's external representation, which is the information agent's local view and Agent Knowledge (Section 6.2.3).

The enterprise model has been limited to a few schema objects and concepts that form a partial reference model of the enterprise. The sole purpose of this model is to prove its functionality in the conflict detection and resolution mechanism. It is not a complete model that provides useful assistance for enterprise modelling.

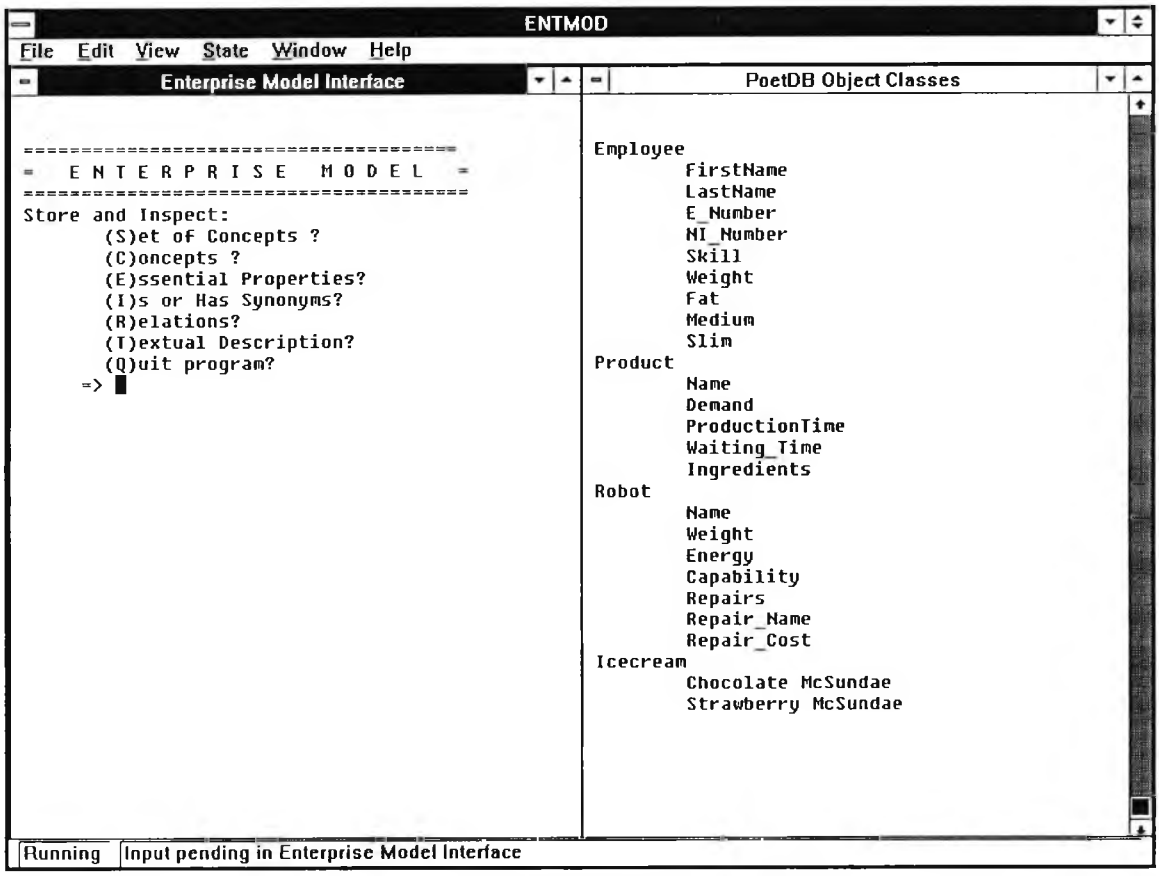

Figure 31: Enterprise Model Interface

The model is implemented in a POET database (Section D.1) with an interface programmed in MS Visual C++ [MSV] in the program 'EntMod.exe'. The interface has an interactive window to investigate the model, and an Overview window that provides a list of the concepts that are stored in this enterprise model (Figure 31).

The enterprise model includes schema objects as concepts, and concepts that are not schema objects. In other words, schema objects are descriptor objects for abstract objects that are stored in the integrated information systems. For example, the schema object Employee may described the abstract objects 'Peter', 'Mark', 'Fred,' and other employees that are stored in the relational database BookkeepingDB. Schema objects from different information sources may be related by generalisations (Sections 3.3.1.1.) to the schema objects in the enterprise model.

Every schema object is a concept. For example, Employee is a schema object and a concept for people that work in a company. However, there may be concepts that are not schema objects. For example, Technician is a concept for an employee that repairs things but it is not a schema object in any information system. Multiple, sources may use different names for the same concepts. For example, what is a Technician in one company (or information source) may be Technical Support Staff in another. Furthermore, multiple sources may have the same name for different concepts. For example, a technician may be an electrical engineer for repairing robots in one system, and a computer specialist in another. The enterprise model therefore has multiple concepts so that information systems throughout the enterprise can relate their own use of concepts to these centrally-defined concepts. This allows multiple heterogeneous systems to relate their concepts via the central enterprise model.

Hierarchical relations between concepts are implemented by sets of concepts and ordinary concepts. A set of concepts (Set Concept) is a supertype that may have multiple concepts (Concept) as its subtypes. For example, the Set Ice-cream may have the subtypes Chocolate McSundae and Strawberry McSundae. The Set Employee, for example, includes the concepts First_Name, Last_Name, E_Number, etc. This hierarchical relation can be described in the form '(Set of Concepts).Concept', e.g. 'Employee.E_Number'.

Any Set or ordinary Concept may have the following information attached to it:

- Information as to whether the Set or Concept 'Has' or 'Is' an **essential property**. For example, the concept employee number (E-Number) 'Is_a' essential property of the set Employee.

- **Synonyms** relating any (set of) concepts in an 'Is a' relation to its synonym, for example, 'Technician Is_a Technical Support Staff'.
- **Relations** are used to described links between (sets of) concepts. For example, the Set Employee has the types 'Fat', 'Medium', and 'Slim'. The concept 'Fat' may have the relation: 'An Object is a member of the concept Fat if its Concept 'Weight' is lager than 15 stone' (Employe.Weight ≥ 15 stone).
- A **textual description** is an informal text string that is used by the system administrators or designers as a description of the concept.

A Concept automatically inherits the characteristics of its supertype. For example, if the Set of Concepts Employee has the synonym 'Hired hand' then that is inherited by all its concepts. The concept E_Number, hence, inherits the synonym 'Hired hand'. If information on the concept E_Number of the set Employee is requested, then the synonym 'Employee Is_a Hired hand' is included.

Appendix E: Overview Implementation Concept

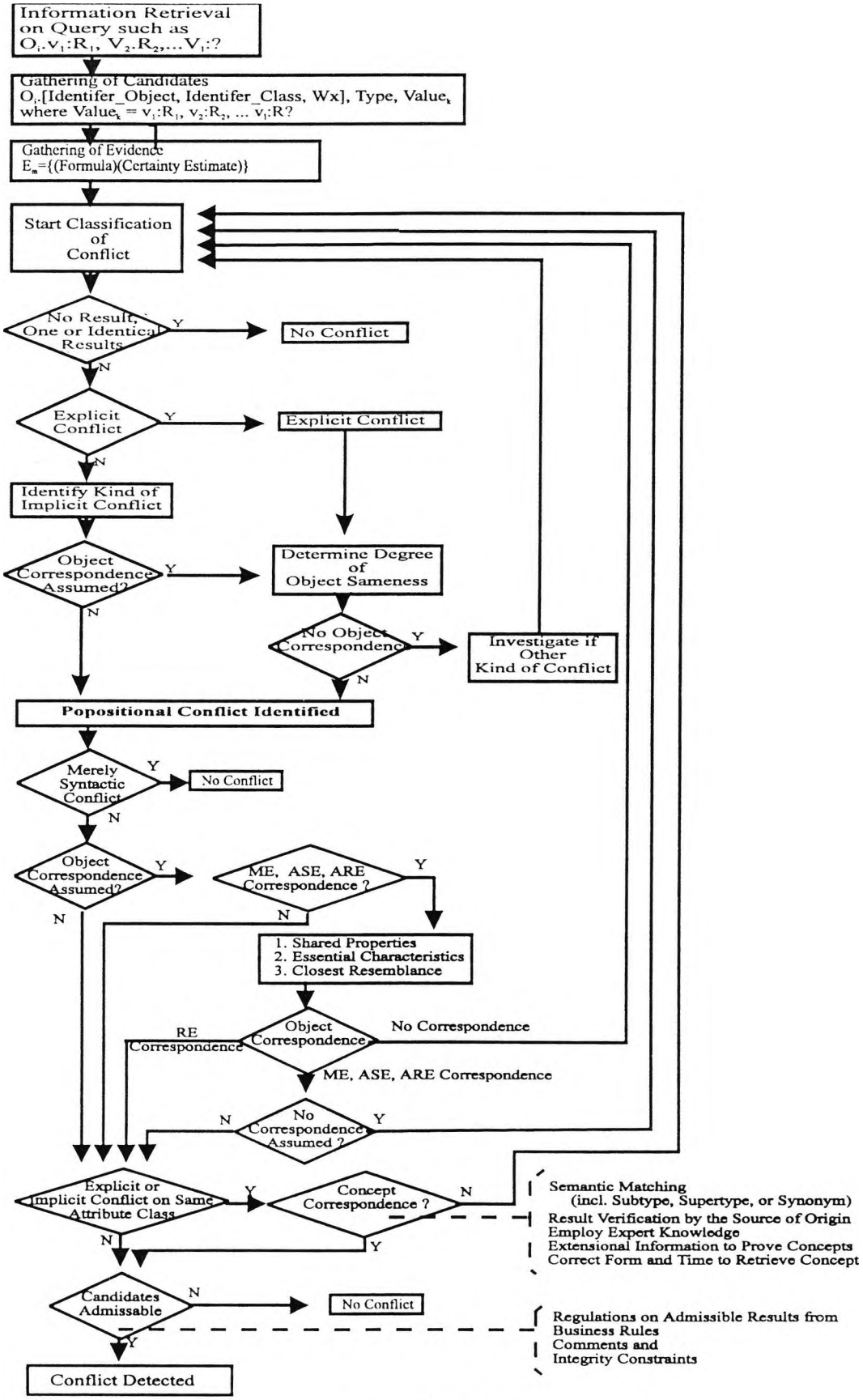

Figure 32: Conflict Detection

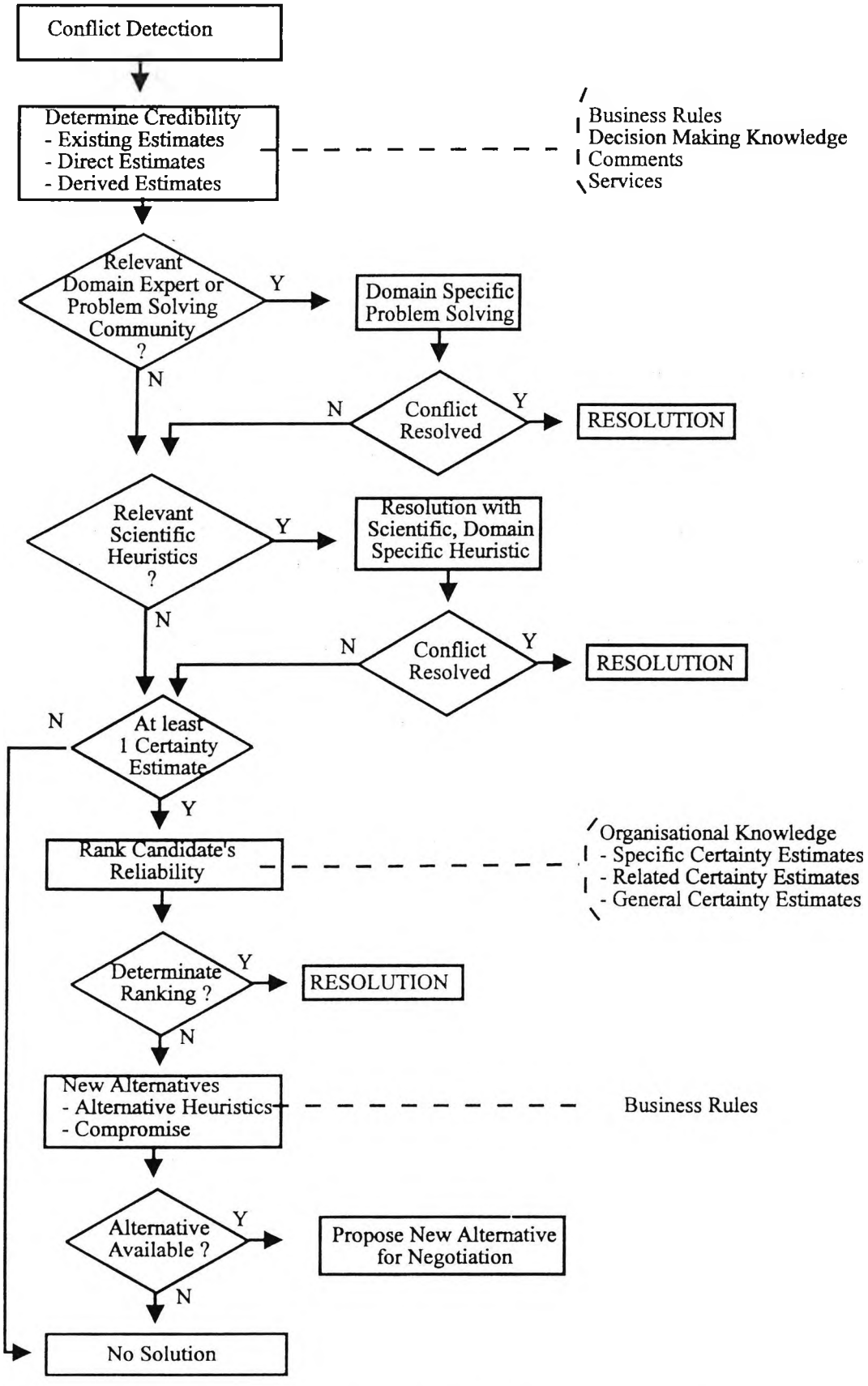

Figure 11: Conflict Resolution (as on page 203)

References

[ADL89] Adler, M. R. Davis, A. B. Weihmayer, R. Worrest, R. W. (1989). Conflict-Resolution Strategies for Nonhierachical Distributed Agents. *In:* 8th. Workshop on Distributed Artificial Intelligence, ed. by Gasser, L. Huhns, M. (London, UK: Pitman).

[ADL92] Adler, M. Durfee, E. Huhns, M. Punch, W. Simoudis, E. (1992). AAAI Workshop Notes on Cooperation Among Heterogeneous Intelligent Agents. *AI Magazine 13* (2): 39-42.

[AHM91] Ahmed, R. De Smedt, P. Du, W. Kent, W. Ketabchi, M. A. Litwin, W. A. Rafii, A. Shan, M.-C. (1991). The Pegasus Heterogeneous Multidatabase System.*IEEE Computer 24* (12): 19-27.

[AN 93a] An, Z. Bell, D.A. Hughes, J.G. (1993). RES - A Logic for Relative Evidential Support. *International Journal of Approximate Reasoning 8*: 205-230.

[AN 93b] An, Z. Bell, D.A. Hughes, J.G. (1993). Relation-based Evidential Reasoning.*International Journal of Approximate Reasoning 8*: 231-251.

[ANS75] ANSI/X3/SPARC Study Group on Database Management Systems. (1975). *INTERIM Report ACM-SIGMOD* 7 (2).

[ARE93] Arens, Y. Chee, C.Y. Hsu, C.-N. Knoblock, C.A. (1993). Retrieving and Integrating Information From Multiple Information Sources. *International Journal of Intelligent and Cooperative Information System 2* (2): 127-158.

[AST76] Astrahan, M.M. Blasgen, M.W. Chamberlin, D.D. Eswaran, K.P. Gray, J.N. Griffiths, P.P. King, W.F. Lorie, R.A. McJones, P.R. Mehl, W. Putzolu G.R. Traiger I.L. Wade, B.W. Watson, V. (1976). System R: Relational Approach to Database Management. *ACM Transactions on Database Systems 1* (2): 97-137.

[BAK92] Baker, K. Evans, M. Anderson, J. (1992). Measuring Autonomy in Heterogeneous Cooperative Systems. Paper presented at the Annual Conference of the American Association for Artificial Intelligence (AAAI), Workshop on Cooperation Among Heterogeneous Intelligent Systems, San Jose, CA.

[BAR94a] Barbuceanu, M. Fox, M. (1994). The Information Agent: An Infrastructure for Collaboration in the Integrated Enterprise. *In: 2. International Working Conference on Cooperating Knowledge Based Systems. Proceedings.* Keele, UK: DAKE Centre, University of Keele, pp. 257 - 294.

[BAR94b] Barbuceanu, M. Fox, M. (1994). The Information Agent: An Infrastructure for Collaboration in the Integrated Enterprise. Paper presented at 2nd. *International Working Conference on Cooperating Knowledge Based Systems as in the*Draft Proceedings, Keele, UK, pp. 145-159.

[BAR94c] Barbuceanu, M. Fox, M.S. (1994). Conflict management with an Authority / Deniability Model. *In:* Workshop on Models of Conflict Management in Cooperative Problem Solving,

ed. by Lander, S. (Menlo Park, CA: *American Association for Artificial Intelligence*, AAAI Technical Report 360).

[BEL87] Bell, D.A. Grimson, J.B. Ling D.H.O. (1987). EDDS - A System to Harmonize Access to Heterogeneous Databases on Distributed Micros and Mainframes. *Information and Software Technology 29* (7): *362-370.*

[BEL89] Bell, D.A. Grimson, J.B. Ling, D.H.O. (1989). Implementation of an Integrated Multidatabase-Prolog System. *Information and Software Technology* 31 (1): 29-38.

[BEL90] Bell, D.A. Shao, J. Hull, M.E.C. (1990). Integrated Deductive Database System Implementation: A Systematic Study. *The Computer Journal 33* (1): 40-48.

[BEL92] Bell, D.A. Grimson, J.B. (1992). Distributed Database Systems. (Wokingham, UK: Addison-Wesley Publishing Company).

[BEL93] Bell, D.A. (1993). From Data Properties to Evidence. *IEEE Transactions on Knowledge and Data Engineering* 5 (6): 965-969.

[BER84] Bercovitch, J. (1984). Social Conflicts and Third Parties - Strategies of Conflict Resolution. (Westview Press, Boulder, Colorado).

[BES91] Besnard, P. (1991). Default Logics. *In: Lecture Notes in Computer Science* Vol. 548, ed. by Kruse, R. Siegel, P. (Springer Verlag).

[BOR89] Borgida, A. Brachmann, R.J. McGuiness, D.L. Resnick, L.A. (1989). CLASSIC: A Structural Data Model. *In: ACM SIGMOD Conference on the Management of Data. Proceedings.* ACM Press, *pp.* 59-67.

[BRO89] Brodie, M.L. (1989). Future Intelligent Information Systems: AI and Database Technologies Working Together. *In:* Readings in Artificial Intelligence and Databases, ed. by Mylopoulos, J. Brodie, M.L. (Morgan Kaufmann).

[BUß92] Bußler, C. (1992). Capability Based Modelling *In: 1st. International Conference on Enterprise Integration. Proceedings.* Cambridge, MA: MIT Press,.pp. 389-398.

[CAR91] Carver, N. Cvetanovic, Z. Lesser, V. (1991). Sophisticated Cooperation in FA/C Distributed Problem Solving Systems. *In: Annual Conference of the American Association for Artificial Intelligence (AAAI). Proceedings.* Menlo Park, CA: AAAI, pp. 191-198.

[CARL89] Carlson, D.A. Ram, S. (1989). An Object-oriented Design for Distributed Knowledge-Based Systems. *In: 22nd. Hawaii International Conference. on Systems Science. Proceedings.* CA, Los Alamitos: IEEE Computer Society Press pp. 55-63.

[CARL91] Carlson, D. Ram, S. (1991). An Architecture for Distributed Knowledge-Based Systems. *Data Base Winter/Spring:* 11-21.

[CAW92a] Cawsey, A. Galliers, J. Reece, S. Sparck Jones, K. (1992). Conflict and Cooperation in a Heterogeneous System. Paper presented at the Annual Conference of the American Association for Artificial Intelligence (AAAI), Workshop on Cooperation Among Heterogeneous Intelligent Systems, San Jose, CA.

[CAW92b] Cawsey, A. Galliers, J. Reece, S. Sparck Jones, K. (1992). A Comparison of Architectures for Autonomous Multi-Agent Communication. *In: 10th International Conference on Artificial Intelligence (ECAI'92). Proceedings.* Vienna: John Wiley and Sons, pp. 249-251.

[CAW92c] Cawsey, A. Galliers, J. Reece, S. Sparck Jones, K. (1992). Automating the Librarian: Belief Revision as a Base for System Action and Communication with the User. *The Computer Journal 35* (3): pp. 221-232.

[CER84] Ceri, S. Pelagatti, G. (1984). Distributed Databases Principals and Systems. (NY: McGraw-Hill Book Company).

[CHA90] ChaibDraa, B. Millot, P. (1990). A Framework for Cooperative Work: An Approach Based on the Intetionality. *Artificial Intelligence in Engineering 5* (4): 199-205

[CHA92] Chalupsky, H. Finin, T. Fritzson, R. McKay, D. Shapiro, S. Wiederhold, G. (1992). An Overview of KQML - DRAFT - DARPA Knowledge Sharing Effort. *Available From:* University of Maryland, Baltimore MD.

[CLA90a] Clark, D. Fox, J. Glowinski, A.J. O'Neil, M.J. (1990). Symbolic Reasoning for Decision Making *In:* Contemporary Issues in Decision Making, ed. by Borcherding, K. Larichev, O.I. Messick, D.M. (Holland: Elsevier Science Publ. B.V.).

[CLA90b] Clark, D. (1990). Numeric and Symbolic Approaches to Uncertainty Management in AI. *Artificial Intelligence Review* 4: 109-146.

[COE92] Coenen, F.P. Finch, I. Bench-Capon T.J.M. Shave M.J.R. Barlow J.A. (1992). Task Scripting for an Intelligent Aide de Camp System. *In: Expert Systems Applications and AI (EXSYS'92). Proceedings.* Gournay sur Marne, France: IITT-International, pp. 191-196.

[COE93a] Coenen, F.P. Finch, I. Bench-Capon T.J.M. Shave M.J.R. Barlow J.A. (1993). Autonomous Communication Using Aide de Camp. *In: Joint Framework for Information Technology (JFIT) Technical Conference Digest. Proceedings.* Great Britain: Department of Trade and Industry, Science and Engineering Research Council, pp. 297-305.

[COE93b] Coenen, F.P. Finch, I. Bench-Capon T.J.M. Shave M.J.R. (1993). Cooperating Knowledge Based Systems in Aide de Camp. *In: Expert Systems Applications and AI (EXSYS'93). Proceedings.* Gournay sur Marne, France: IITT-International, pp. 33-38.

[COD71] Conference on Data Systems Languages (CODASYL), Database Task Group Report (1971), NY: ACM.

[COD79] Cod, E.F. (1979). Extending the Database Relational Model. *ACM Transactions on Database Systems* 4 (4): 397-434.

[COD90] Cod. E.F. (1990) The Relational Model For Database Management Version 2. (Addison-Wesley Publ. Comp. MA.).

[COH85] Cohen, P. (1985). Heuristic Reasoning About Uncertainty: An Artificial Intelligence Approach. (London: Otmann).

[COH87] Cohen, P.R. Day, D. Lisio, de J. Greenberg, M. Kjeldsen, R. Suthers, D. Berman, P. (1987). Management of Uncertainty in Medicine. *International Journal of Approximate Reasoning 1*: 103-116.

[COL91] Collet, C. Huhns, M.N. Shen W.-M. (1991). Resource Integration Using a Large Knowledge Base in Carnot. *IEEE Computer 24* (12): 55-62.

[CON88] Connors, T. Lyngbaek, P. (1988). Providing Uniform Access to Heterogeneous Information Bases. *In:* Advances in Object-Oriented Database Systems - Lecture Notes in Computer Science Vol. 334, ed. by Dittrich, K. (Springer Verlag).

[COX46] Cox, R. (1946). Probability, Frequency and Reasonable Expectation'.*American Journal of Physics 14*: 1-13.

[CZE92] Czejdo, B. Taylor, M.C. (1992) Integration of Information Systems Using an Object-Oriented Approach. *The Computer Journal 35* (5): 501-513.

[DAI93] Dai, H. Hughes, J. G. Bell, D.A. (1993). A Distributed Real-Time Knowledge-Based System and its Implementation using Object-Oriented Techniques. *In: International Conference on Intelligent and Cooperative Information Systems ICICIS'93. Proceedings.* Los Alamitos, CA: IEEE Computer Society Press, pp. 23-30.

[DAT90] Date, C.J. (1990). An Introduction to Database Systems Volume 1. (MA: Addison-Wesley Publ. Comp. 5th Edition).

[DEC89] Decker, K.S. Durfee, E.H. Lesser, V.R. (1989). Evaluating Research in Cooperative Distributed Problem Solving. *In:* 8th. Workshop on Distributed Artificial Intelligence Volume 2, ed. by Gasser, L. Huhns, M. (London, UK: Pitman).

[DEC92] Decker, K. Lesser, V. (1992). Generalizing The Partial Global Planning Algorithm. *International Journal of Intelligent and Cooperative Information Systems 1* (2): 319-346.

[DEE93] Deen, M.S. (1993). A General Framework for Coherence in a CKBS. *Journal of Information System 2*: 83-107.

[DEM67] Dempster, A.D. (1967). Upper and Lower Level Probabilities Induced by Multivalued Mapping. *Americ. Math. Stat. 38*:325-329.

[DOY79] Doyle, J. (1979). Truth Maintenance Systems. *Artificial Intelligence 12*: 213-272.

[DOY92] Doyle, J. (1992). Reason Maintenance and Belief Revision. *In:* Belief Revision, ed. by Gärdenfors, P. (UK: Cambridge University Press.).

[DUB91] Dubois, D. Lang, J. Prade, H. (1991). A Brief Overview of Possibilistic Logic.*In:* Symbolic and Quantitative Approaches to Uncertainty - Lecture Notes in Computer Science 548, ed. by Kruse, R. Siegel, P. (Springer Verlag).

[DUB92] Dubois, D. Lang, J. Prade, H. (1992). Dealing with Multi-source Information in Possibilistic Logic. *In: European Conference on Artificial Intelligence. Proceedings.* Vienna: John Wiley and Sons, pp. 38-42.

[DUO93a] Duong, T. Hiller, J. (1993). Modelling the Real World by Multi-World Data Model. *International Conference on Intelligent and Cooperative Information Systems ICICIS'93. Proceedings.* Los Alamitos, CA.: IEEE Computer Society Press, pp. 279- 290.

[DUO93b] Duong, T. Hiller, J. Ngu, A.H.H. (1993). A Framework of Flexible and Dynamic Integration for Mulitdatabases. *In: International Conference on Intelligent and Cooperative Information Systems ICICIS'93. Proceedings.* CA, Los Alamitos: IEEE Computer Society Press, pp. 43-54.

[DUR87] Durfee, E.H. Lesser, V.R. Corkill, D.D. (1987). Cooperation Through Communication in a Distributed Problem Solving Network. *In:* Distributed Artificial Intelligence, ed. by Huhns, M. (London, UK: Pitman).

[DUR89] Durfee, E.H. Lesser, V.R. (1989). Negotiating Task Decomposition and Allocation Using Partial Global Planning. *In:* 8th Workshop on Distributed Artificial Intelligence Volume 2, ed. by Gasser, L. Huhns, M. (London, UK: Pitman).

[DUR91a] Durfee, E.H. Lesser, V.R. (1991). Partial Global Planning: A Coordination Framework for Distributed Hypothesis Formation. *IEEE Transactions on Systems, Man, and Cybernetics21* (5): 1167-1183.

[DUR91b] Durfee, E. (1991). The Distributed Artificial Intelligence Melting Pot. *IEEE Transactions on Systems, Man, and Cybernetics 21* (5): 1301-1305.

[DUR92] Durfee, E.H. (1992). What Your Computer Really Needs to Know, You Learned in Kindergarten. *In: Annual Conference of the American Association for Artificial Intelligence (AAAI'92). Proceedings.* Menlo Park, CA: AAAI, pp. 858-864.

[DUR95] Durfee, E. (1995). Rational Agents, Limited Knowledge, and Nash Equilibria.*In: AAAI-95 Fall Symposium Series Rational Agency: Concepts, Theories, Models and Applications.* To appear as a Technical Report of the American Association for Artificial Intelligence (AAAI), Menlo Park, CA, pp. 38-42.

[EDM95] Edmonds, B. Moss, S. (1995). Modelling the Bounded Rationality of Economic Agents by Modelling Limited Incremental Search. *In: AAAI-95 Fall Symposium Series Rational Agency: Concepts, Theories, Models and Applications.* To appear as a Technical Report of the American Association for Artificial Intelligence (AAAI), Menlo Park, CA, pp. 43-47.

[ELI91] Eliassen, F. Karlsen R. (1991). Interoperability and Object Identity. *SIGMOD Record 20* (4): 25-29.

[ELL87] Elliot, D.W. (1987). Elliot and Phipsen Manual of the Law of Evidence. (London, UK: Sweet and Maxwell).

[ENC92] Encyclopaedia of Ethics Volume I., ed. by Becker, L.C. Becker, C.B., (Chicago and London: St. James Press).

[EPH91] Ephrati, E. Rosenschein, J. (1991). The Clark Tax as a Consensus Mechanism Among Automated Agents. *In: Annual Conference of the American Association for Artificial Intelligence (AAAI'91). Proceedings.* Menlo Park, CA: AAAI, pp. 173-178.

[EPH92] Ephrati, E. Rosenschein, J.S. (1992). Reaching Agreement through Partial Revelation of Preferences. *In: 10th European Conference on Artificial Intelligence. Proceedings.* Vienna: John Wiley and Sons, pp. 229-233.

[EPS95] Epsetin, S.L. (1995). Collaboration and Interdependence Among Limitedly Rational Agents. *In: AAAI-95 Fall Symposium Series Rational Agency: Concepts, Theories, Models and Applications.* To appear as a Technical Report of the American Association for Artificial Intelligence (AAAI), Menlo Park, CA, pp. 51-55.

[FAC91] Facione. P.A. Scherer, D. Attig, T. (1991). Ethics in Society. (NJ: Prentice Hall).

[FAG87] Fagin, R. Halpern, J.Y. (1987). Belief, Awareness, and Limited Reasoning: Preliminary Report. *In: International Joint Conference on Artificial Intelligence. Proceedings.* San Mateo, CA: Morgan Kaufmann, pp. 491-501.

[FIN93] Finch, I. Coenen, F.P. Bench-Capon T.J.M. Shave M.J.R. Barlow J.A. (1993). Applying CKBS Techniques to Electronic Mail. *In: Workshop of the Special Interest Group on Cooperating Knowledge Based Systems, Subgroup of the British Computer Society. Proceedings.* University of Keele: DARK Centre, University of Keele, pp. 101-118.

[FIN94a] Finin, T. Fritzson, R. McKay, D. McEntire, R. (1994). KQLM - A Language and Protocol for Knowledge and Information Exchange. *In:* 13th. International Workshop on Distributed Artificial Intelligence, ed. by Klein, M. (Menlo Park, CA: AAAI, Technical Report WS-94-02).

[FIN94b] Finin, T. Fritzson, R. McKay, D. McEntire, R. (1994). KQLM as an Agent Communication Language. *In: Third International Conference on Information and Knowledge Management. Proceedings. ACM-Press.* Available from www@umbc.edu.

[FOX91] Fox, J. Krause, P. Dohnal, M. (1991). An Extended Logic Language For Representing Belief. *In:* Symbolic and Quantitative Approaches to Uncertainty - Lecture Notes in Computer Science Vol. 548, ed. Kruse, R. Siegel, P. (Springer Verlag).

[FOX92] Fox M.S. (1992). The TOVE Project Towards a Common-Sense Model of the Enterprise. *In: 1st. International Conference on Enterprise Integration and Modelling. Proceedings.* London, UK: MIT Press, pp. 310-319.

[FOX92b] Fox, J. Krause P, Ambler, S. (1992). Arguments, Contradictions and Practical Reasoning. *In: 10th International Conference on Artificial Intelligence (ECAI'92). Proceedings.* Vienna: John Wiley and Sons, pp. 623-627.

[FOX93] Fox, M.S. Chionglo, J.F. Fadel, F. (1993). A Common-Sense Model of the Enterprise. *In: 2nd. Industrial Engineering Research Conference. Proceedings.* Norcross, GA: Institute for Industrial Engineers, pp. 425-429.

[FRO92] Froidevaux, C. Mengin, J. (1992). A Framework for Default Logics. *In:* Logics in AI - Lecture Notes in Computer Science Vol. 633, ed. by Pearce, D. and Eagner, G. (Springer Verlag).

[GAL90a] Galliers, J.R. (1990). The Positive Role of Conflict in Cooperative Multiagent Systems.*In:* Decentralised Artificial Intelligence, ed. by Demazeau, Y. and Müller, Y.-P. (Holland: Elsevier Science Publishers B.V.).

[GAL90b] Galliers, J.R. (1990). Cooperative Interaction as Strategic Belief Revision.*In: International Working Conference on Cooperative Knowledge Based Systems. Proceedings.* London: Springer Verlag, pp. 148-163.

[GAL92] Galliers, J. (1992). Autonomous Belief Revision and Communication.*In:* Belief Revisions, ed. by Gärdenfors, P. (UK: Cambridge University Press).

[GÄR88] Gärdenfors, P. (1988). Knowledge in Flux - Modelling the Dynamics of Epistemic States. (Cambridge, MA: Bradford Book, MIT Press.).

[GÄR92] Gärdenfors,P. (1992). Belief Revision: An Introduction. *In:* Belief Revision, ed. by Gärdenfors, P. (UK: Cambridge University Press).

[GAS91] Gasser, L. Ishida, T. (1991). A Dynamic Organisational Architecture for Adaptive Problem Solving. *In: Annual Conference of the American Association for Artificial Intelligence (AAAI'91). Proceedings.* Menlo Park, CA: AAAI, pp. 185-190.

[GIN91] Ginsberg, M.L. (1991). Knowledge Interchange Format: The KIF of Death.*AI Magazine 3*: 57-63.

[GOT92] Gottinger, H.W. Weimann, P. (1992). Intelligent Decision Support Systems, *Decision Support Systems 8*: 317-332.

[GRA92] Grashoff, H. Long, J.A. (1992). Cooperative Intelligent Information Systems: Extending Global Schemata. *In:* 3. IEE Colloquium on Distributed Databases (London: Institute of Electrical Engineers (IEE), Colloquium Digest).

[GRA93] Grashoff, H. Long, A.J. (1993). Cooperative Intelligent Information Systems Integrating Knowledge Based Systems. *In: 6th. Florida Artificial Intelligence Research Symposium. Proceedings.* Menlo Park, CA: AAAI Press, pp 305-309.

[GRA94a] Grashoff, H. Long, A.J. (1993). Conflict Detection and Resolution in Intelligent Cooperative Information Systems. Paper presented at the 2. International Working Conference on Cooperating Knowledge Based Systems. Draft Proceedings. Keele, UK: DAKE Centre, University Keele, pp. 293-312.

[GRA94b] Grashoff, H. Long, A.J. (1993). Intelligent Cooperative Information Systems Conflict Detection and Resolution. *In: 13th. International Symposium on Automotive Technology and Automation, Conference on Mechatronics. Proceedings.* Aachen, Germany and Croydon, UK: Automotive Automation Ltd., pp. 265-272.

[GRA95a] Grashoff, H. Long, A.J. (1995). Object Identity and Sameness in Enterprise Integration Environments. *In: 2nd. International Conference on Concurrent Engineering, Research and Applications. Proceedings.* Washington DCConcurrent Technologies Corporation, pp. 315-326.

[GRA95b] Grashoff, H. (1995). Rationality and Information Agents *In: AAAI-95 Fall Symposium Series Rational Agency: Concepts, Theories, Models and Applications.* To appear as a Technical Report of the American Association for Artificial Intelligence (AAAI), Menlo Park, CA, pp. 75-79.

[GUH94] Guha, R.V. Lenat, D.B. (1994) Enabling Agents to Work Together.*Communications of the ACM, Special Issue on Intelligent Agents 37* (7): 127-142.

[GUA92] Guan, J. Bell, D.A. Lesser, V.R. (1992). Evidential Reasoning and Rule Strength in Expert Systems. *In: 3rd. Irish Conference on Artificial Intelligence and Cognitive Science. Proceedings.* NY: Springer Verlag in collaboration with the British Computer Society, pp. 378-390.

[GUP] Gupta Technologies Inc. Application Development Tool SQL Windows, Version 3.1.

[HAL85] Halpern, J.Y. Moses, Y. (1985). A Guide to the Modal Logics of Knowledge and Belief: Preliminary Draft. *In: International Joint Conference on Artificial Intelligence (IJCAI'85). Proceedings.* San Mateo, CA: Morgan Kaufmann, pp. 480-490.

[HAL87] Hall, R. King, R. (1987). Semantic Database Modelling: Survey, Applications, and Research Issues. ACM Computing Surveys 19 (3): 201-260.

[HAL91] Halpern, J.Y. Moses, Y. (1991). Knowledge and Common Knowledge in a Distributed Environment. *Journal of the ACM 37* (3): 549-587.

[HAL92a] Halpern, J.Y. Fagin, R. (1992). Two Views of Belief: Belief as Generalized Probability and Belief as Evidence. *Artificial Intelligence 54* (3): 275-317.

[HAL92b] Halpern, J.Y. Moses, Y. (1992). A Guide to the Modal Logics of Knowledge and Belief. *Artificial Intelligence 54* (3): 319-379.

[HAM93] Hammer, J. McLeod, D. (1993). An Approach to Resolving Semantic Heterogeneity in a Federation of Autonomous, Heterogeneous Databases Systems. *International Journal of Intelligent and Cooperative Information System 2* (1): 51-83.

[HAR86] Harmann, G. (1986). Change in View - Principles in Reasoning. (Cambridge, MA: Bradford Book, MIT Press).

[HEI85] Heimbinger, D. McLeod, D. (1985). A Federated Architecture for Information Management. *ACM Transaction on Office Information Systems 3* (3): 253-278.

[HEI89] Heiler, S. Blaustein, B. (1989). Generating and Manipulating Identifiers for Heterogeneous, Distributed Objects. *In: 3rd. International Workshop on Persistent Object Systems. Proceedings.* Springer Verlag, pp. 235-247.

[HEW91] Hewitt, C. Inman, J. (1991). DAI Betwixt and Between: From Intelligent Agents to Open Systems Science. *IEEE Transactions Systems, Man and Cybernetics 21* (6): 1409-1419.

[HIN62] Hintikka, J. (1962). Knowledge and Belief - An Introduction to the Logic of the Two Notions. (Ithaca, NY: Cornell University Press).

[HSI92] Hsiao, D.K. (1992). Federated Databases and Systems: Part I - A tutorial on Their Data Sharing. *VLDB Journal 1*: 127-179.

[HSU91] Hsu, C. Bouziane, M. Rattner, L. Yee, L. (1991). Information Resource Management in Heterogeneous, Distributed Environments: A Metadatabase Approach. *IEEE Transactions on Software Engineering 17* (6): 604-609.

[HUA92] Huang, G.Q. Brandon, J.A. (1992). AGENTS: Object-oriented Prolog Systems for Cooperating Knowledge-Based Systems. *Knowledge-Based Systems 5* (2): 125-136.

[HUH90] Huhns, M. Bridgeland, D.M. (1990). Distributed Truth Maintenance. *In: International Working Conference on Cooperative Knowledge Based Systems. Proceedings.* London: Springer Verlag, pp. 133-145.

[HUH91] Huhns, M. Bridgeland, D.M. (1991). Multiagent Truth Maintenance. *IEEE Transactions on Systems, Man and Cybernetics 21* (6): 1437-1445.

[HUH92] Huhns, M.N. Singh, M.P. (1992). The Semantic Integration of Information Models. Paper presented at the Annual Conference of the American Association for Artificial Intelligence (AAAI), Workshop on Cooperation Among Heterogeneous Intelligent Systems, San Jose, CA.

[HUH93] Huhns, M.N. Jacobs, N. Ksiezyk, T. Shen, W.-M. Singh, M.P. Cannata, P.E. (1993). Integrating Enterprise Information Models in Carnot. *In: International Conference on Intelligent and Cooperative Information Systems ICICIS'93. Proceedings.* CA, Los Alamitos, CA: IEEE Computer Society Press, pp. 32-42.

[HUH94] Huhns, M.N. Singh, M.P. Ksiezyk, T. Jacobs, N. (1994). Global Information Management via Local Autonomous Agents. *In:* 13th. International Workshop on Distributed Artificial Intelligence, *ed. by Klein, M.* (Menlo Park, CA: AAAI Press Technical Report WS-94-02).

[JAG92] Jagannathan, V. Karinthi, R. Raman, R. Montan, V. Petro, J. (1992). A System for Integrating Heterogeneous Information Repositories. Paper presented at the Annual Conference of the American Association for Artificial Intelligence (AAAI), Workshop on Cooperation Among Heterogeneous Intelligent Systems, San Jose, CA.

[JAG94] Jagannathan, V. Karinthi, R. Raman, R. Almasi, G. (1994). Strategies for Wide-area Information Sharing. *In: 1st. International Conference on Concurrent Engineering: Research and Applications. Proceedings.* PA: Society for Computer-Aided Engeneering, Concurrent Engeneering Research Center, pp. 15-21.

[JEN92] Jennings, N.R. Wittig, T. (1992) ARCHON: Theory and Practice. In: Distributed Artificial Intelligence Theory and Praxis, ed. by Avouris, N. and Gasser, L. (Kluwer Academic Press).

[JOR95] Jorkinen, K. (1995). Rationality in Constructive Dialogue Management. *In: AAAI-95 Fall Symposium Series Rational Agency: Concepts, Theories, Models and Applications.* To appear as a Technical Report of the American Association for Artificial Intelligence (AAAI), Menlo Park, CA, pp. 89-93.

[KAE83] Kaehler, T. Krasner, G. (1983). LOOM-Large Object-Oriented Memory for Smalltalk-80 Systems. *In:* Smalltalk-80: Bits of History, Words of Advice (Reading, MA: Addison-Wesley Publ. Co.).

[KEN91] Kent, W. (1991). A Rigorous Model of Object Reference, Identity, and Existence.*Journal of Object-Oriented Programming 4* (3): 28-36.

[KEN93] Kent, W. Ahmed, R. Albert, J. Ketabchi, M. Shan, M.-C. (1993). Object Identification in Multidatabase Systems. *Transactions A of the International Federation for Information Processing IFIP A-25 (Netherlands: Computer Science Technology)*: 313-30.

[KEY21] Keyns, J.M. (1921). A Treatise on Probability. (London: Macmillan).

[KHO90] Khoshafian, S.N. Copeland, G.P. (1990). Object Identity. *In:* Readings in Object-Oriented Database Systems, ed. by Zdonik, S.B. and Maier, D. (Morgan Kaufmann Publishers). And (1988). *In: Object Oriented Programming Systems, Languages, and Applications (OOPSLA'88). Proceedings.* ACM Press, pp. 406-416. .

[KIM90] Kim, W. (1990). Introduction to Object-oriented Database Systems. (Cambridge, MA: MIT Press).

[KIM91a] Kim, W. Ballou, N. Garza, J.F. Woelk, D. (1991). A Distributed Object-oriented Database System Supporting Shared and Private Databases. *ACM Transactions on Information Systems 9* (1): 31-51.

[KIM91b] Kim, W. Seo, J. (1991). Classifying Schematic and Data Heterogeneity in Multidatabase Systems. *IEEE Computer 24* (12): pp. 12-18.

[KIM93] Kim, W. (1993). On Object-Oriented Database Technology. Proceedings. 19th. Conference on Very Large Databases (VLDB), Morgan-Kaufman, pp. 676-687.

[KIR91a] Kirn, S. Schlageter, G. (1991). Intelligent Agents in Federative Expert Systems - Concepts and Implementation. *In: International Working Conference on Cooperative Knowledge Based Systems. Proceedings.* London: Springer Verlag, pp. 53-75.

[KIR91b] Kirn, S. Scherer, A. Schlageter, G. (1991). Problem Solving in Federative Environments: The FRSECO Concept of Cooperative Agents. In: The Next Generation of Information Systems - From Intelligence to Distribution and Cooperation, ed. by Papazoglou, M.P. and Zeleznikow, J. (Sprinter Verlag).

[KLE89] Klein, M. Lu, S.C.-Y. (1989). Conflict Resolution in Cooperative Design.*AI in Engineering 4* (4): 168-180.

[KLE91] Klein, M. (1991). Supporting Conflict Resolution in Cooperative Design Systems.*IEEE Transactions on Systems, Man, and Cybernetics 21* (5): 1379-1390.

[KNE49] Kneale, W. (1949). Probability and Induction. (Oxford, UK: Oxford University Press).

[KON88] Konolige, K. (1988). Hierarchical Autoepistemic Theories for Nonmonotonic Reasoning. *In: 7th Annual Conference of the American Association for Artificial Intelligence. Proceedings.* Menlo Park, CA: AAAI-Press, pp. 439-443.

[KRA93] Krause, P. Clark, D. (1993). Representing Uncertain Knowledge - An Artificial Intelligence Approach. (The Netherlands: Kluwer Academic Press).

[KRI63] Kripke, S. (1963). A Semantic Analysis of Modal Logic I: Normal Modal Propositional Calculi. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematick 9*: 67-96.

[KRU91] Kruse, R. Schwecke, E. (1991). On the Combination of Information Sources. Lecture Notes in Computer Science - Vol. 521. ed. by Bouchon-Mequier, B. Yager, R.R. Zadeh, L.A. (Springer Verlag).

[LÄA92] Läasri, B. Läasri, H. Lander, S. Lesser, V. (1992). A Generic Model for Intelligent Negotiating Agents. *International Journal of Intelligent and Cooperative Information Systems 1* (2): 291-317.

[LAN89] Lander, S. Lesser, V. (1989). A Framework for the Integration of Cooperative Knowledge-Based Systems. *In: 4th International Symposium on Intelligent Control. Proceedings.* Los Alamitos, CA: IEEE Computer Society Press, pp. 472-477.

[LEM67] Lemmon, E.J. (1967). If I Know, Do I Know that I know? *In:* Epistemology, ed. by Stroll, A. (NY: Harper and Row).

[LEN90] Lenat, D.B. Guha, R.V. (1990). Building Large Knowledge-Based Systems - Representation and Inference in the CYC Project. (Reading, MA: Addison-Wesley Pub. Comp. Inc.).

[LES91] Lesser, V. (1991). A Retrospective View of FA/C Distributed Problem Solving. *IEEE Transactions Systems Man and Cybernetics 21* (6): 1347-1362.

[LEV84] Levesque, H.J. (1984). A Logic of Explicit and Implicit Belief. *In: Annual Conference of the American Association for Artificial Intelligence AAAI'84. Proceedings.* Menlo Park, CA: AAAI, pp. 198-202.

[LEV89] Levesque, H.J. (1989). Knowledge Representation and Reasoning. *In:* Readings in Artificial Intelligence and Databases ed. by Mylopoulos, J. Brodie, M. (Morgan Kaufmann).

[LEW68] Lewis, D.K. (1968). Counterpart Theory for Quantified Modal Logic. *Journal of Philosophy 65*: 113-126.

[LIT82] Litwin, W. Boudenant, J. Esculier, C. Ferrier, A. Glorieux, A. La Chimia, J. Kabbaj, K. Moulinoux, C. Rolin, P. Stangret, C. (1982). SIRIUS System for Distributed Data Management. *In:* Distributed Data Bases, ed. by Schneider, H.-J. (The Netherlands: North Holland Publishing).

[LIT90] Litwin, W. Mark, L. Roussopoulos, N. (1990). Interpretability of Multiple Autonomous Databases. *ACM Computing Survey 22* (3): 267-293.

[LOO89] Loomis, M.E.S. (1989). Data Management and File Structures. 2nd. Edition. (NJ: Prentice Hall).

[MAC91] MacGregor, (1991). Using a Description Classifier to Enhance Deductive Inference. *In: 7th. IEEE Conference of Artificial Intelligence Applications. Proceedings.* Los Alamitos, CA: IEEE Computer Society Press, pp. 141-147.

[MAI91] Maida, A.S. (1991). Maintaining Mental Models of Agents Who Have Existential Misconception. *Artificial Intelligence 50*: 331-383.

[MAL94] Malheiro, B. Jennings, N.R. Oliveira, E. (1994). Belief Revision in Multi-Agent Systems. *In: 11th. European Conference on Artificial Intelligence (ECAI'94). Proceedings.* Vienna: John Wiley and Sons, pp. 294-298.

[MAN87] Manola, F. (1987). Applications of Object-Oriented Database Technology in Knowledge-Based Integrated Information Systems. *In: 2nd. Symposium Knowledge-Based Integrated Information Systems Engineering. Proceedings.* Cambridge, MA: MIT Press, pp. 126-134.

[MAN92] Manold, F. Heiler, S. (1992). Distributed Object Management. *International Journal of Intelligent and Cooperative Information Systems 1* (1): 5-42.

[MAR91] Marinos, L. (1991). A Corporate Architecture and Object Oriented Modelling Substrate for Distributed Heterogeneous Information Systems. (München: Gesellschaft für Mathematik und Datenverarbeitung, Bericht Nr. 191, R. Oldenbourg Verlag).

[MAS89] Mason, C.L. Johnson, R.R. (1989). DATMS: A Framework for Distributed Assumption Based Reasoning. *In:* Distributed Artificial Intelligence II, ed. by Gasser, L. Huhns, M. (London: Pitman).

[MAS90] Masunaga, Y. (1990). Object Identity, Equality and Relational Concept. *In: 1st. International Conference on Deductive and Object-Oriented Databases (DOOD'89). Proceedings.* The Netherlands: Elsevier Science Publishers B.V., pp. 185-202.

[McC86] McCarthy, J. (1986). Applications of Circumscription to Formalising Common-Sense Knowledge. *Artificial Intelligence 28*: 89-116.

[McL92] McLean, D.R. Page, B. Tuchman, A. Kispert, A. Yern, W. Potter, W. (1992). Emphasizing Conflict Resolution Versus Conflict Avoidance During Schedule Generation. *Expert Systems With Applications 5*: 441-446.

[MOO85] Moore, R. (1985). Semantical Considerations on Non-monotonic Logic. *Artificial Intelligence 25*: 75-94.

[MUR86] Murphy, P. Barnard, D. (1986). Evidence and Advocacy. 2. Edition. (London: Financial Training Publ.).

[MUR90] Murphy, P. (1990). A Practical Approach to Evidence. 2. Edition. (London: Financial Training Publ.).

[MSV] Microsoft Corporation, Microsoft Visual C++, Version 1.5.1.

[MSW] Microsoft Corporation, Microsocft Windows, Version 3.1.

[NEC91] Neches, R. Fikes, R. Finin, T. Gruber, T. Patil, R. Senator, T. Swartout, W. (1991). Enabling Technology for Knowledge. *AI Magazine 12 (3): 36.56.*

[NII86] Nii, P. (1986). Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architecture. *AI Magazine 2*: 38-53.

[NOR94] Norman, D.A. (1994). How Might People Interact with Agents. *Communications of the ACM 37* (7): 68-71.

[OAT94] Oates, T. Nagendra Prasad, M.V. Lesser, V.R. (1994). Cooperative Information Gathering: A Distributed Problem Solving Approach. UMass Computer Science Technical Report 94-66, University of Massachusetts.

[OHO90] Ohori, A. (1990). Representing Object Identity in a Pure Functional Language. *In: 3rd. International Conference on Database Theory (ICDT'90). Proceedings.* Springer Verlag, pp. 41-55.

[OLL78] Olle, T.W. (1978). The CODASYL Approach to Data Base Management. (NY: John Wiley and Sons).

[OXB88] Oxborrow, E. Ismail, H. (1988). KBZ - an Object-oriented Approach to the Specification and Management of Knowledge Bases. *In: 6th. British National Conference on Databases (BNCOD 6). Proceedings.* Cambridge, UK: Cambridge University Press, pp. 21-46.

[OXB90] Oxborrow, E. Ismail, H.M. (1990). An Object-oriented Approach to Distributed Database Management Systems. *Database Technology 3* (1): 13-26.

[OXF75] The Oxford Illustrated Dictionary. (1975). 2nd. ed., ed. by Coulson, J. Petter, D. Eagel, D. Hawkins, J. (Oxford: Clarendon Press).

[ÖZU90] Özsu, T. Barker, K. (1990). Architectural Classification and Transaction Execution Models of Multidatabase Systems. *In:* Lecture Notes on Computer Science - Vol. 468, ed. by Goos, G. Hartmanis, J. (Springer Verlag).

[PAL92] Palananiappan, M. Yankelovich, N. Fitzmaurice, G. Loomis, A., Haan, B. Coombs, J. Meyrowitz, N. (1992). The Envoy Framework: An Open Architecture for Agents. *ACM Transactions on Information Systems 10* (3): 233-264.

[PAN89] Pan, J.Y.-C. Tenenbaum, J.M. Glicksman, J. (1989). A Framework for Knowledge-Based Computer Integrated Manufacturing. *IEE Transactions on Semiconductor Manufacturing 2* (2): pp. 33-46.

[PAN91a] Pan, J.Y.-C. Tenenbaum, J.M. (1991). Towards an Intelligent Agent Framework for Enterprise Integration. *In: Annual Conference of the American Association for Artificial Intelligence (AAAI 91). Proceedings.* Menlo Park, CA: AAAI, pp. 206-212.

[PAN91b] Pan J.Y.-C. Tenenbaum, J.M. (1991). An Intelligent Agent Framework for Enterprise Integration. *IEEE Transactions on Systems, Man and Cybernetics, Special Issue on Distributed Artificial Intelligence 21* (6): 223-250.

[PAP90] Papazoglou, M.P. Marinos, L. Bourbakis, N.G. (1990). Distributed Heterogeneous Information Systems and Schema Transformation. *In: International Conference on Databases, Parallel Architecture and Their Applications (PARABASE). Proceedings.* Los Alamitos, CA: IEEE Computer Society Press, pp. 388-549.

[PAP91] Papazoglou, M.P. Bobbie, P.O. Hoffman, C. (1991). Active Information Systems in Support of Decision Making. *In: 24th. Hawaii International Conference. on Systems Science. Proceedings.* Los Alamitos, CA: IEEE Computer Society Press, pp. 407-416.

[PAP92a] Papazoglou, M. Laufmann, S. Sellis, T.K. (1992). An Organisational Framework for Cooperating Intelligent Information Systems. *International Journal of Intelligent and Cooperative Information Systems 1* (1): 169-202.

[PAP92b] Papazoglou, M.P. Blum, B.I. Hughes, J.G. (1992). An Expert System-like Architecture for Integrated Disparate Information Sources. *In: 25th. Hawaii International Conference on Systems Science. Proceedings.* Los Alamitos, CA: IEEE Computer Society Press, pp. 600-610.

[PAT88] Paton, N.W. Gray, P.M.D. (1988). Identification of Database Objects by Key. *In: 2nd. International Workshop on Advances in Object-Oriented Database Systems. Proceedings.* Springer-Verlag, pp. 280-285.

[PEA88] Pearl, J. (1988). Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. (San Metro, CA: Morgan Kaufmann).

[PEA93] Pearl, J. (1993). Belief Networks Revised. *Artificial Intelligence 59*: pp. 49-56.

[PET92] Petri, C. (1992). A Minimalistic Model for Cooperation. In: *1st. International Conference on Enterprise Integration. Proceedings.* Cambridge, MA: MIT Press, pp. 409-415.

[POET] Poet Software Corporation, San Mateo, CA, The Object Database for C++, Release 10. January 1995, Version 3.0.

[POL70] Pollock, J.L. (1970). The Structure of Epistemic Justification. In: *American Philosophical Quarterly 4*: pp. 62-78.

[POL74] Pollock, J.L. (1974). Knowledge and Justification. (Princton, NJ: Princton University Press).

[POL92] Pollock, J.L. (1992. How to Reason Defeasibly. *Artificial Intelligence 57*: 1-42.

[POL94] Pollock, J.L. (1994). Justification and Defeat. *Artificial Intelligence 67*: 377-407.

[POL95] Pollock. J.L. (1995). Rational Agency in OSCAR. *In: AAAI-95 Fall Symposium Series Rational Agency: Concepts, Theories, Models and Applications.* To appear as a Technical Report of the American Association for Artificial Intelligence (AAAI),Menlo Park, CA, pp. 112-116.

[POLA92] Polat, F. Güvenir, H. A. (1992). A Conflict Resolution Based Cooperative Distributed Problem Solving Model. Paper presented at the Annual Conference of the American Association for Artificial Intelligence (AAAI), Workshop on Cooperation Among Heterogeneous Intelligent Systems, San Jose, CA.

[PRE92] Preece, A.D. Shinghal, R. (1992). Verifying Knowledge Bases by Anomaly Detection: An Experience Report. *In: 10th. European Conference on Artificial Intelligence (ECAI-92). Proceedings.* Vienna: John Wiley and Sons, pp. 835-839.

[PRI67] Price H.H. (1967). Some Considerations About Belief. *In:* Knowledge and Belief, ed. by Griffiths. (Oxford, UK: Oxford University Press).

[PRI69] Price, H.H. (1969). Belief. *In:* The Gilford Lecture Delivered at the University of Aberdeen in 1960. (London, UK: George Allen and Unwin Ltd.).

[QUT92] Qutaishat, M.A. Fiddian, N.J. Gray, W.A. (1992). A Meta-Integration System for Heterogeneous Object-Oriented Database Environment - Implementation in Prolog.*In: 1st. International Conference on Practical Applications of Prolog. Proceedings.* London, UK: Institute of Civil Engineering, pp. 1-16.

[RAZ86] Raz, J. (1986).The Morality of Freedom. (Oxford, UK: Oxford University Press).

[RES75] Rescher, N. (1975). A Theory of Possibility. (Pittsburgh: University of Pittsburgh Press).

[SAD90] Sadreddini, M.H. Bell, D.A. McClean, S. (1990). Architectural Considerations for Providing Statistical Analysis of Distributed Data. Information and Software Technology 32 (7): 459-469.

[SEL93] Seligman, L. Kerschberg, L. (1993). An Active Database Approach to Consistency Management in Data and Knowledge-Based Systems. *International Journal of Intelligent and Cooperative Information System 2* (2): 187-200.

[SEU92] Seung, T.K. Bonevac, D. (1992). Plural Values and Indeterminate Rankings. Ethic 102 (4): 799-813.

[SHA75] Shave, M.J.R. (1975). Data Structures. (London, UK: McGraw-Hill Book Company Ltd.).

[SHA82] Shave, M.J.R. Bhaskar, K.N. (1982). Computer Science Applied to Business Systems. (London, UK: Addison-Wesley Publishers Ltd., International Computer Science Series).

[SHA76] Shafer, G. (1976). A Mathematical Theory of Evidence. (Princton: Princton University Press).

[SHA93] Shaw, M.J. Fox, M.S. (1993). Distributed Artificial Intelligence for Group Decision Support. *Decision Support Systems 9*: 349-367.

[SHE90] Sheth, A. Larson, J. (1990). Federated Database Systems for Managing Distributed Heterogeneous, and Autonomous Databases. *ACM Computing Surveys 22* (3): 183-235.

[SHE93] Sheth, A.P. Gala, S.K. Navathe, S.B. (1993). On Automated Reasoning for Schema Integration. *International Journal of Intelligent and Cooperative Information System 2* (1): 23-50.

[SMI80] Smith, R. (1980). The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *IEEE Transactions on Computers C-29* (12): 1104-1113.

[SMI81] Smith, G.R. Davis, R. (1981). Frameworks for Cooperation in Distributed Problem Solving. *IEEE Transactions on Systems, Man, and Cybernetics 11* (1): 61-70.

[SMI95] Smit, R.A. Verhagen H.J.E. (1995). On Being Social: Degrees of Sociality and Models of Rationality in Relation to Multiagent Systems. *In: AAAI-95 Fall Symposium Series Rational Agency: Concepts, Theories, Models and Applications.* To appear as a Technical Report of the American Association for Artificial Intelligence (AAAI), Menlo Park, CA, pp. 131-135.

[SOL95] Solomon, M. (1995). Social Empiricism. *In: AAAI-95 Fall Symposium Series Rational Agency: Concepts, Theories, Models and Applications.* To appear as a Technical Report of the American Association for Artificial Intelligence (AAAI),Menlo Park, CA, pp. 141-144.

[SPA91] Spaccapietra, S. Parent, C. (1991). Conflicts and Correspondence Assertions in Interoperable Databases. *Sigmod Record 20* (4): 49-54.

[SPA92] Spaccapietra, S. Parent, C. Dupon, Y. (1992). Model Independent Assertions for Integration of Heterogeneous Schemas. *VLDB Journal 1*: 81-126.

[SQL] American National Standard for Information Systems (ANSI) Database Language SQL (Structured Query Language) ANSI X3.135. (1986). NY: ANSI.

[STE90] Steiner, D. Mahling, D. Haugeneder, H. (1990). Collaboration of Knowledge Bases via Knowledge Based Coordination. *In: International Working Conference on Cooperative Knowledge Based Systems. Proceedings.* London: Springer Verlag, pp. 113-129.

[STO76] Stonebraker, M. Wong, E. Kreps, P. (1976). The Design and Implementation of INGRES. *ACM Transactions on Database Systems 1* (3): 189-222.

[STO84] Stocker, P.M. Atkinson, M.P. Oxborrow, E.W. (1984). PROTEUS: A Heterogeneous Distributed Database Project. *In:* Database - Role and Structure, ed. by Gray, P. Stocker, P.M. Atkinson, M.P. (Cambridge, UK: Cambridge University Press).

[STO91] Stolze, M. Gutknecht, M. (1991). Building Human-Centred Intelligent Cooperative Information Systems with IKEA. *In:* The Next Generation of Information Systems - From Intelligence to Distribution and Cooperation, ed. by Papazoglou, M.P. and Zeleznikow, J. (London: Sprinter Verlag).

[SU 90] Su, S. Park, J.H. (1990). A Knowledge Representation Scheme and a Knowledge Derivation Mechanism for Achieving Rule Sharing Among Heterogeneous Expert Systems. *In: Conference on Database and Expert Systems Applications (DEXA 90). Proceedings.* pp. 359- 366,

[SU 91] Su, S. Park, J.H. (1991). An Integrated System for Knowledge Sharing Among Heterogeneous Knowledge Derivation Systems. *Journal of Applied Intelligence 1*: 223-245.

[SUB90] Subramanyan, S. Park, J.H. Su, S.Y.W. (1990). An Integrated Heterogeneous Expert System: Design and Implementation. *In:* PROCIEM 90.Proceedings. Tampa, Fl

[SYC89] Sycara, K. (1989). Multiagent Compromise via Negotiation. *In:* 8th. Workshop on Distributed Artificial Intelligence Volume 2, ed. by Gasser, L. Huhns, M. (London, UK: Pitman).

[THO90] Thomas, G. Thompson, G. Chung, C.-W. Barkmeyer, E. Carter, F. Templeton, M. Fox, S. Hartman, B. (1990). Heterogeneous Distributed Database Systems for Production Use. *ACM Computing Surveys 22* (3): 138- 265.

[TOY94] Toy, G. Cutkosky, M.R. Leifer, L.J. Tenenbaum, M.J. Glicksman, J. (1994). SHARE: A Methodology and Environment for Collaborative Product Development. *International Journal of Intelligent and Cooperative Information System 3* (2): 129-153.

[TRU87] Trusted, J. (1987). Moral Principles and Social Values. (London, UK: Routledge and Kegan Paul).

[UNL90] Unland, R. Schlageter, G. (1990). Object-Oriented Database Systems: Concepts and Perspectives. Lecture Notes in Computer Science Vol. 466. ed. by Blaser, A. (Springer Verlag).

[VIT91] Vittal, J. Silver, B. Frawley, W. Iba, G. Fawcett, T. Dusseault, S. Doleac, J. (1991). A Framework for Cooperative Adaptable Information Systems. *In:* The Next Generation of Information Systems - From Intelligence to Distribution and Cooperation, ed. by Papazoglou, M.P. and Zeleznikow, J. (Springer Verlag).

[VIT92] Vittal, J. Silver, B. Frawley, W. Iba, G. Fawcett, T. Dusseault, S. Doleac, J. (1992). Intelligent and Cooperative Information Systems Meet Machine Learning. *International Journal of Intelligent and Cooperative Information Systems 1* (2): 347-361.

[WER91] Werkman, K. (1991). Using Negotiation and Coordination in Multiagent Intelligent Cooperative Information Systems. *In:* The Next Generation of Information Systems - From Intelligence to Distribution and Cooperation, ed. by Papazoglou, M.P. and Zeleznikow, J. (Sprinter Verlag).

[WIE90] Wiedehold, G. (1990). Future Architectures for Information Processing Systems. *International Conference on Databases, Parallel Architectures, and Their Applications (PARABASE-90). Proceedings.* Los Alamitos, CA: IEEE Computer Society Press, pp. 160-176.

[WIE91] Wiederhold, G. (1991). The Roles of Artificial Intelligence in Information Systems. *In:* Lecture Nodes in Computer Science. 6th. Methodologies for Intelligent Systems. (Springer Verlag).

[WIE92] Wiedehold, G. (1992). Mediators in the Architecture of Future Information Systems.*IEEE Computer 3:* 38-49.

[WIL82] Williams, R. Daniels, D. Haas, L. Lapis, G. Lindsay, B. Ng, P. Obermarck, R. Selinger, P. Walker, A. Wilms, P. Yost, R. (1982). R*: An Overview of the Architecture.*In:* Improving Database Usability and Responsiveness, ed. by Scheuermann, P. (Orlando, FL: Academic Press). *And. In:* 2nd. International Conference on Databases: Improving Responsiveness. Proceedings. Jerusalem, Israel: Hebrew University Jerusalem.

[WIT93] Witteveen, C. Brewka, G. (1993). Sceptical Reason Maintenance and Belief Revision. *Artificial Intelligence 61*: 1-36.

[WON94] Wong, S.T.C. (1994). Preference Based Decision Making. *ACM Transactions on Information Systems 12* (4): 407-435.

[WOO92] Woo, C.C. Lochovsky, F.H. (1992). Knowledge Communication in Intelligent Information Systems. *International Journal of Intelligent and Cooperative Information Systems 1* (1): 203-228.

[ZAN90] Zaniolo, C. (1990). Object Identity and Inheritance in Deductive Databases - An Evolutionary Approach. *In: 1st. International Conference on Deductive and Object-*

Oriented Databases (DOOD'90). Proceedings. The Netherlands: Elsevier Science Publishers B.V. pp. 7-24.

[ZLA92] Zlatareva, N.P. (1992). Truth Maintenance Systems and Their Application for Verifying Expert Systems Knowledge Bases. *Artificial Intelligence Review 6*: 67-110.

[ZLO91] Zlotkin, G. Rosenschein, J. (1991). Cooperation and Conflict Resolution via Negotiation Among Autonomous Agents in Noncooperative Domains. *IEEE Transactions on Systems, Man and Cybernetics 21* (6): 1317-1324.

[ZLO93] Zlotkin, G. Rosenschein, J.S. (1993). Negotiation with Incomplete Information about Worth: Strict versus Tolerant Mechanism. *In: International Conference on Intelligent and Cooperative Information Systems (ICICIS 93). Proceedings.* Los Alamitos, CA: IEEE Computer Society Press, pp. 175-184.