An Architecture for an Intelligent Assistant System for use in Software Project Planning

Rory V O'Connor

Submitted for Examination for Doctor of Philosophy

Department of Computing
City University
London

April 2000

TABLE OF CONTENTS

TABLE OF FIGURES

ACKNOWLEDGMENTS

Only one name appears on the cover of this thesis, but a great many people have been indirectly involved in its production.

Firstly, I would like to express my sincere gratitude to my supervisor, Dr. J.O.Jenkins, for his assistance and careful guidance. My research has benefited from his insight and innumerable suggestions.

I would also like to thank all my colleagues at Dublin City University for making it such a pleasant place to conduct this research. In particular, I would like to acknowledge the significant input of Prof. J.A.Moynihan, whose encouragement and assistance were invaluable.

A word of thanks to all my P3 project partners - Annie, Brian, Chantal, Christophe, Herve, Ioannis, Mark, Martine, Marty, Philippe, Robert, Tristan and Vassilis - who helped in so many ways. A special thanks to Eamon Gaffney for all his help.

I would also like to thank my family, Kevin, Teresa and Tracy, for their love and support throughout my prolonged existence as a student.

To my girlfriend Margaret, who was always with me when I needed support, shared my worries and problems, and provided my mind with a fail safe mechanism. I would like to thank her for her support and encouragement, but most of all her faith in me.

Finally, a message of thanks to my parents who bought a BBC home computer for me in 1983 and encouraged me to use it. Looking back, their inspired decision was the first step on the road that has culminated in this thesis. Thanks.

After some time they crossed the Water, west of Hobbiton, by a narrow plank-bridge. The stream there was no more than a winding black ribbon, bordered with leaning alder trees. A mile or two further south they hastily crossed the great road from the Brandywine Bridge; they were now in the Tookland and bending south-eastwards they made for the Green Hill Country. As they began to climb its first slopes they looked back and saw the lamps in Hobbiton far off twinkling in the gentle valley of the Water. Soon it disappeared in the folds of the darkened land, and was followed by Bywater beside its grey pool. When the light of the last farm was far behind, peeping among the trees, Frodo turned and waved a hand in farewell. 'I wonder if I shall ever look down in that valley again', he said quietly.

The Lord of the Rings
J. R. R. Tolkien

DECLARATION

I grant powers of discretion to the University Librarian to allow this thesis to be copied in whole or in part without further reference to me. This permission covers only single copies made for study purposes, subject to normal conditions of acknowledgment.

ABSTRACT

It is the proposition of this research that there are a number of weaknesses in the current approaches being taken in the provision of software project planning tools and that there is significant scope to improve on existing systems by the development of an intelligent assistant system which will provide decision support for the software project planner in the creation of plans for a software development project.

This research has devised a framework and architecture based on a fusion of a number of techniques within a multi-agent framework which aims to improve the quality of the decision making process of software project planners. This framework incorporates the information gathering and analysis techniques of a Decision Support System with the ability of an Expert System to propose possible solutions using expert knowledge and best practices and the power of Blackboard to exchange information between components. This novel approach enables the inter-working of a variety of well understood techniques within a single underlying framework - that of the agent-orientated paradigm.

To assist with validating the proposed architecture, a prototype application was developed and a series of user trials conducted. The conclusion of these trials was that the prototype system demonstrated that the notion of an intelligent assistant system for software project planning was a viable concept, worthy of further investigation. Further, it demonstrated that the proposed architecture provided a viable framework for supporting the software project planners decision making process and has the potential to be of use in a commercial setting.

CHAPTER 1 INTRODUCTION

1.1 Background

This thesis describes a multi-agent based architecture for an intelligent assistant system for use in software project planning. The research explored the role of artificial intelligence techniques for automated tool support as applied to software project planning and in particular addressed the issues of knowledge capture and reuse in such a tool environment. This research also examined the issues surrounding the emerging requirements for support systems for distributed multi-platform software development projects. In addressing these aims, this research devised a framework and architecture for use as the basis for the design and construction of an intelligent assistant system and implemented a prototype system for use by software project managers in the planning of a distributed multi-platform software development project.

The following sections describe the motivation and background of the work, both from a user-level and system-level perspective, leading to a discussion of issues concerned with a new architecture for intelligent assistant systems. The statement of the aims and objectives of the research and an outline plan for the rest of this thesis complete the chapter.

1.2 Software Project Management

The Project Management Institute defines project management as [PMI, 96];

> *"The applications of knowledge, skills, tools and techniques to project activities in order to meet or exceed stakeholders needs and expectations from a project".*

Project management is an integrative endeavor - an action, or failure to take action, in one area will usually affect other areas. The interactions may be straightforward and well understood, or they may be subtle and uncertain. For example, a scope change will almost always affect project costs, but it may not affect team morale or product quality. These interactions often require trade-offs among project objectives - performance in one area may be enhanced only by sacrificing performance in another. Successful project management requires actively managing these interactions.

Many techniques of general project management are applicable to software project management, but Brooks [Brooks, 87] pointed out that the processes and products of software projects have certain characteristics that make them different. One way of perceiving software project management is as the process of making visible that which is invisible [Hughes and Cotterall, 99]:

- **Invisibility** - when a physical artifact such as a road is being built the progress can actually be seen. With software, progress is not immediately visible.
- **Complexity** - Per dollar, pound or euro spent, software products contain more complexity than other engineering artifacts.
- **Flexibility** - The ease with which software can be changed is usually seen as one of its strengths. However this means that where the software system interfaces with a physical or organisational system, it is expected that, where necessary, the software will change to accommodate the other components rather than vice versa. This means the software systems are likely to be subject to a high degree of change.
- **Standard Process** - in other engineering disciplines with a long history the processes are tried and tested. Our understanding of software processes has developed significantly over the past few years, however we still cannot predict with complete certainty when a particular software process is likely to cause development problems.

Software project managers are responsible for planning and scheduling software development. They supervise the work to ensure that it is carried out to the required standards and monitor progress to check the development is on time and within budget.

1.3 Software Project Planning

Software project planning is an integral part of the software project management activity. Its objectives are to provide a framework that enables the project manager to make reasonable estimates of resources, costs, and schedule [Pressman, 97]. These estimates are made within a limited time frame at the beginning of a project and should be revised regularly as that project progresses. In addition, estimates should attempt to define 'best-case' and 'worst-case' scenarios so that project outcomes can be bounded.

Effective management of a software project requires thorough planning of its progress. The project manager must anticipate problems which may arise and prepare tentative solutions to those problems. A plan drawn up at the start of a project, should be used as a driver for the project. The initial plan is not static but must be modified as the project progresses and new information becomes available. Project planning is probably the activity that takes most management time [Sommerville, 95]. The planning process starts with an assessment of the constraints affecting the project. The progress milestones and deliverables are then defined and a schedule drawn up. Project managers revise their assumptions about the project as more information becomes available.

1.4 Difficulties Facing Software Project Managers

Due to the growing complexity of products and commercial systems, large projects are facing more constraining production objectives in terms of time, cost, quality and risk. This evolution in the nature of projects being undertaken by software organisations has resulted in increased difficulties associated with planning, managing and executing software development projects.

One of the key issues is decision making. Software project managers make many decisions every day, ranging from the relatively inconsequential to the significant.

Such decisions are based on a combination of judgement and information from staff, clients, research literature and current market forces, as well as knowledge gained from previous projects. Ideally, all relevant information should be brought together before judgement is exercised. The quality of a decision depends on the adequacy of the available information, the quality of the information, the number of options available at the time of the decision and the ability of the people involved to interpret this information.

Software projects often fail because the project managers lack knowledge of good practices and effective processes which can reduce risk and increase the likelihood of success. Managers of projects need to know how to establish a set of processes which are tailored to a project's requirements in terms of time, cost, quality and their associated risks [Ould, 90]. A desired outcome of this research is a planning tool which will increase the likelihood of success by helping the project manager who has to make decisions on these issues. Such a tool will encapsulate expert knowledge and make it available to all users. Some of the potential benefits of this approach as applied to the decision-making process in the domain of software project planning are:

- Suggestions are made which help the user balance cost, quality and time in making decisions about the use of project resources.
- Knowledge is shared about different lifecycle models and why one or another may be more suitable for the users projects.
- Measurements are suggested which will enable the user to see how well the project is reaching greater organisational goals and re-plan the ways to reach these goals, if necessary.

Even the most experienced project manager may have difficulty knowing the best planning options, even if the critical input parameters of resources, constraints and requirements are known.

1.5 Intelligent Assistance for Software Project Planning

The notion of an intelligent assistant is not new. Indeed, as far back as 399 BC Socrates claimed to have an intelligent assistant, although not in the strictest sense of course. But Socrates did claim to have a non-human companion, which he called a Daemon. Intelligent and always ready to offer good advice, Socrate's daemon could be trusted to act without prompting. Real, hard-coded, linguistic and symbolic links abound between Socrates daemon and today's notion of an intelligent assistant [Leonard, 98].

[Boy, 91] offers the following characterisation of an intelligent assistant system:

> *"In an aircraft cockpit, a human copilot shares the work, but not the ultimate responsibility, with the captain. The captain is the master on board: he may consult the copilot at any stage but will take the ultimate decisions. If the captain delegates a part of his responsibility to the copilot, then the copilot will take this delegation as a task to be executed. In addition, the captain may at any time choose to stop the execution of a task by the copilot, if he judges it to be necessary. However, a copilot may have personal initiatives, for example, testing parameters, keeping current with the evolving situation, predicting deducible faults, etc. A copilot may process the knowledge included in the operation manual on his own initiative or at the request of the captain. He should be capable of explaining, in an appropriate amount of detail, the results of his processing."*

Weld [Weld, 95] suggests that a software system designed to act as a team member could help in the planning and execution of a project. Such an intelligent project assistant could help to preserve knowledge about tasks, to record the reasons for decisions and retrieve information relevant to new problems. They could function as co-workers, assisting and collaborating with the design or operations team for complex systems. They could also supply institutional memory. They could recall the rationale of previous decisions and, in times of crisis, explain the methods and reasoning previously used to handle that situation.

In software development projects, an intelligent project assistant could keep track of specifications, design proposals, and implementations for a software project throughout its life cycle. It can record the design decisions of a constantly changing team and also be a repository of solutions for new projects. Reasoning techniques can be used to track the (mis)match between specifications and implementations, while analogy techniques can be used to look for existing specifications, components or implementations that match some new requirement.

An intelligent project assistant can additionally be of benefit when training new personnel. For many tasks, on-the-job training is extremely effective, providing the trainee with the chance to make real, on-the-spot decisions and see the consequences. On-the-job training is impossible, however, when a bad decision can be disastrous - as in the planning of a large complex software development project. Simulations of the project planning process, would enable the development of training systems for such situations [Grosz and Davis, 94]. These same simulation capabilities are also important when the cost of assembling large groups of people for training is prohibitive.

1.6 Aims and Objectives

One aim of this research was to understand the complex decision making process associated with planning a software development project. Additionally, the development of an intelligent assistant system to assist project managers in their decision-making process was planned. A prototype system was constructed to test the proposed architecture and feedback from trials by commercial tool users evaluated to assess the usefulness of such a system.

This research started from the standpoint that there is significant scope to improve on existing software project planning systems and in particular to provide capabilities such as;

- The provision of advice to assist project managers in the decision making processes associated with the formulation of project plans.
- The ability to reason about a project's plans, analyse alternatives and select the most suitable course of action.
- The ability to assimilate knowledge and best practice.
- The ability to assist the project manager in adherence to standards, industry best practices and implementation of company policy.
- The ability to dynamically update the knowledge base.
- The ability to cope with new and evolving standards and best practices.
- The ability to manage and analyse large amounts of project data.

1.7 The P3 Project

The implementation and testing of the prototype system was conducted within the scope of the P3 (Project and Process Prompter) project [O'Connor et al., 97a].

The P3 project was funded by the fourth framework programme of the European Commission as ESPRIT project 22241 (cf. Appendix A). The two main deliverables of this project are a "Handbook and Training Guide" and a pre-commercial prototype decision support tool "Prompter".

The Handbook and Training Guide [Catalyst, 99] is designed as a standalone document which requires no other documents or tools to be useful and has two main components:

- **Volume 1** considers process planning as it relates to anyone starting a project, i.e. the basic processes that a project manager needs to know.

Part I contains a high-level view of some of the general challenges of project management. Part II takes the project manager from setting up to closing down the project. Parts III, IV, V, and VI are the technical details of project management, and Part VII examines the pros and cons of following some well-known standards.

- **Volume 2** includes models and decision processes to assist the project manager which were incorporated into the Prompter tool.

The prototype of the Prompter tool has implemented the decision models above to assist project managers in the planning of a software development project. Its aims are to provide project managers with a greater understanding of options available during planning and why one choice should be made over another. Prompter gives the project planner the opportunity to input project goals and certain project-specific variables, match them against a generic model to create a specific project model, then analyse a set of options which may be used to organise the project so that it will meet its goals.

This researcher's role in the P3 project was that of project manager in charge of the overall architectural design (as described in chapter 6) and implementation of the knowledge base (agents) for the Prompter tool [O'Connor and Renault, 98]. As, such this project provided an ideal framework within which to implement the proposed architecture, utilising the knowledge from the Handbook and Training Guide.

1.8 Layout of Thesis

In this chapter the motivation and objectives of the work have been explained. Chapter 2 describes the domain of software project planning in more detail in order to understand its unique characteristics and assess what special considerations are necessary when developing intelligent assistance systems. The findings of a survey of tool users are also presented and analysed in conjunction with a discussion on existing project support tools to assess the need for, and usefulness of, the integration of intelligent assistance in software project planning tools. Chapter 3 provides a review and discussion of approaches to supporting intelligent assistant systems, leading to a proposed architecture for applying intelligent assistance to the domain of software project planning. Chapter 4 presents a critical review of intelligent assistant systems, from both a user and system architecture perspective. Chapter 5 discusses issues relating to the design and development of a knowledge base, including knowledge representation and acquisition. Chapter 6 details the proposed architecture for an intelligent assistant system based on the system proposed in Chapter 2 and the issues discussed in subsequent chapters. Chapter 7 describes the design and construction of a prototype implementation of the system. Chapter 8 provides an overview of research methodologies and describes the approach taken to the validation process. Chapter 9 presents a strategy for trial usage by a group of commercial users and discusses the lessons learned from user feedback gained from these trials. Finally, Chapter 10 contains the conclusions and the recommendations which describe the advances made in this research.

CHAPTER 2 SOFTWARE PROJECT PLANNING

2.1 Introduction

This chapter describes the domain of software project planning in order to understand its characteristics and assess what considerations are necessary when developing an intelligent assistant system for this domain. It also describes the current states of art and practice in the software industry with regard to the usage of software project support tools, thus highlighting the potential benefits of incorporating intelligent assistance into software project planning tools.

2.2 Software Project Planning

Whatever the size of the project, good planning is essential if it is to succeed. The software project planning process [Fairclough, 96] contains five major activities (figure 2.1), which can be applied to a whole project or to a phase of a project. Each activity may be repeated several times to make a feasible plan. In principle, every activity can be linked to the other activities by feedback loops, in which information gained at a later stage in planning is used to revise earlier planning decisions.

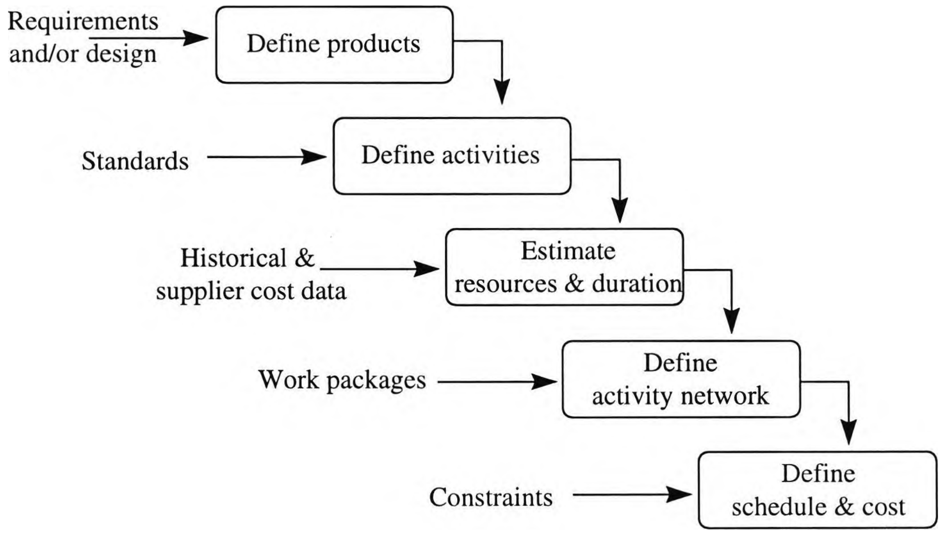


Figure 2.1 - Planning Process

During the life of a project, the management emphasis will shift from initiation planning to implementation planning according to different time zones, each with its own time horizon. Usually these time zones cover the immediate, intermediate and future periods [Procter and Bouchier, 94]. At the initiation stage (immediate time zone), the plan (and its alternatives) will be expressed at a high level, because the detail is not available and uncertainty will be high. The plan is a 'theoretical' model of the potential project. The system does not issue instructions nor record and respond to feedback. It does not need to communicate with the 'real' world it attempts to represent and control.

The work scheduling and monitoring in the 'intermediate' zone may create the most serious practical problems. At this stage the planner is concerned with the current position of the project and its immediate future. Tools are better at reporting on the current status (based on the inputted data). They can report on what has been done, what the plans say should be done next and what can be done with resources available. The danger is that reports may be incomplete, out of date or inappropriate for the recipient. In this intermediate time zone, of potentially significant benefit to the project manager is the ability to reason about project plans, analyse alternative strategies in the approach to problems, seek advice from lessons learned during previous projects (the knowledge base) and consult organisational and international standards.

Beyond the intermediate horizon, the future has both 'micro' and 'macro' planning modes [Procter and Bouchier, 94], neither of which are well supported by tools. In the 'micro' environment of the project, there is little to assist the predictive function necessary for detailed forward planning, nor is there much support for 'macro' strategic planning, which sees the project in relation to the wider environment it seeks to serve.

2.3 Software Project Support Systems

To support project managers, organisations have sought to develop tools to assist with various aspects of the management of their software processes. As well as general purpose software project planning tools that support activity definition, PERT and scheduling, specialised project planning tools are available for constructing process models and estimating software project costs. Project planning tools normally support [Fairclough, 96]:

- The definition of work packages and their duration
- The definition of resources and resource availability
- the allocation of resources to work packages
- The highlighting of resource conflicts, or over utilisation
- The construction of activity networks
- The definition of the critical path
- The definition of the schedule

Most projects will benefit from initiation planning with even the simplest tool. However, constructing, maintaining and extending large complex software systems pose the problems of managing all the people, systems and agencies involved. Although many project management systems are readily available, the enormous scope and complexity of software systems means moving beyond the current state of the art, as such systems do little to support the 'average' project manager. For example [O'Connell, 96] suggests;

> *"...if you are a poor project manager, then using Microsoft Project™ will probably make you worse".*

Most project planning tools are successful at showing the outline plan - phases, stages, etc. - identifying critical path and overall duration. Given appropriate basic cost and time data, they may be able to do basic 'what-if' analysis.

It is the proposition of this research that what would be of greater benefit to the project manager is advice on basic strategic alternatives such as selection of a lifecycle process, and reuse of knowledge gained from previous projects undertaken in the organisation.

We have isolated two main areas in which tool support is weakest: First is the creation of plans. While support in some areas has significantly improved, few tools yet offer automatic creation of technical and management plans; the user has still to directly input the plan data with little support for the accuracy, completeness and quality of the plan. Of use to the project manager would be the automatic creation of an outline plan from specified (pre-defined) types of projects, which could be further refined to the particular project under consideration.

Second is decision support. Most of these systems fall short of supporting the project manager in the decision making process and do not offer assistance in representing knowledge about plans and designs, or provide mechanisms for reasoning about plans and designs in flexible ways. Although most tools offer 'what if' analysis in response to changing parameters, few offer direct 'recommendations' for action given a certain situation. Less still allow for the simulation of a possible future plan, given a key parameter change, yet such information and decision support would be of great benefit to the project manager in the immediate and intermediate time zones.

A further aspect to supporting the software project manager which is not addressed by today's support systems is the distributed and cross-platform nature of systems development. The massive reduction in the cost of Personal Computers coupled with enhanced communications technology, has lead to an increased trend towards the development of heterogeneous client-server systems. At the system level we have witnessed the widespread acceptance of distributed middleware technology such as CORBA (Common Object Request Broker Architecture) [OMG, 96] and at the programming level, the Java programming language is poised to bring platform-independent languages to a new level. However, for the software developer there is nothing by way of support systems which are orientated towards these new trends in systems development.

One of the main reasons for this pattern of strengths and weaknesses is that 'standard' software is being offered for the management of 'unique' projects. It follows that the software will concentrate on the common functions, such as calculating the overall schedule from the set of activities. This is concerned with the logical relationship between tasks, which can be represented by standard symbols, rather than the technical content of those tasks. In contrast, short term control and decision support (where tools are not strong) are more dependent on the specific technical detail of the way things are actually done. This tends to be industry, organisation, or project specific.

Users of existing software project planning systems could benefit greatly from the inclusion of intelligent assistance techniques in such tools. In addition, such new support systems should provide for the distributed cross-platform nature of modern client-server development.

2.4 Software Project Management Tools

In the 1980's there was a marked increased in the number of organisations using software engineering methodologies and tools to assist in the planning, control and execution of software development projects. In 1984 70% of organisations said they used no recognised methods or tools. This figure reduced to 30% by 1988 and a survey in 1992 suggested this has fallen to less than 15% [Mair, 92]. It is clear that the number of tools in the market place and the number of organisations using them will continue to increase, thus leading to increased demands from users for more sophisticated tools [Hughes and Cotterall, 99]. Recent studies of project management software trends indicate that the worldwide project management software market has grown to US$750 million and it will continue to grow, exceeding US$1.2 billion by the year 2000 [Hodges and Rogers, 97].

Software project management tools are available at different levels of sophistication and can typically be categorised by the types of project managers who use them [Hampton, 97]:

1. The multi-project manager - Some organisations have a need to track multiple projects simultaneously. It requires software that can identify conflicting demands on the same resources as well as allow the project manager to set priorities among the projects that require the same resource.
2. Mid-range project managers - These users manage large projects - up to 2000 tasks. They may have a couple of projects going at the same time, but the emphasis is not on multiple projects and they are typically interested in planning, scheduling, tracking and the production of reports.
3. Low-end project managers - Are typically used by project managers who want to automate the process of laying out plans, prepare occasional status reports and produce simple Gantt and PERT charts. Such users may be in charge of small development projects and thus require limited functionality from a project management tool.

Table 2.1 shows a classification [Jones, 94] and description of the main types of features found in project management tools.

Feature	Description
Input methods	A key task for users is the entry into the system of a task breakdown, task details and dependencies between tasks. If this is to be done with the minimum of errors it is essential that input can be performed in an intuitive manner.
Applying dependencies	The range of dependency types which a product can support is a key factor in its usefulness in particular situations. Among the key types are; start to finish, finish to start, lags and inter-project logic.
Scheduling	Scheduling is a key activity for project managers and the sophistication of the algorithm affects the usefulness of the product. Aspects which should be taken into account include; task priorities, multiple schedules, fixed date, as soon as possible and as late as possible.
Data import / export	Users may wish to import or export data to other packages.

Resource control	Initial and ongoing control of the resources applied to a project is a key element of project management. Typically tools assist with allocating and monitoring resources.
Cost monitoring	Information regarding actual and estimated costs should be captured, such as; timesheets, committed costs, cash flows, borrowing needs, etc.
Progress tracking	There are a wide variety of metrics for the progress of a project against its plans. Products normally support a variety of these types such as; % completion for time, cost or work, estimation of end date or cost and baseline comparison for time or work effort.
Reporting features	A varied reporting mechanism is essential and should include a variety of reports such as; milestone report, variance reports, status per task/team member, etc.
Multiple projects	A project manager may be responsible for many projects and will require support to handle issues such as; prioritisation between projects, staff / resource sharing and viewing consolidated information.
Charts	A variety of charting mechanisms is desirable, such as; Gantt, Pert, work breakdown structure, resource, etc.
What-if capabilities	A common requirement for project managers is to be able to investigate the effects of potential changes in the situation of a project. They may need to see the effects of adding or withdrawing a particular resource.
Help facilities	There are a number of aspects to help including; online tutorials, internet support, on-screen context sensitive help.
Networking	Organisations may require packages to operate in a network environment and allow for multiple simultaneous users.
System parameters	The limitations of the tool may be an important factor. Among the principal limits are; maximum number of projects, tasks, resources and levels of granularity.

Table 2.1 - Classification of project management tool features

The following section will briefly contrast some project management tools which are representative of the current state of the market, from the perspective of the provision of intelligent assistance. For a definitive review of project management tools the reader is directed to [Budd, 98].

2.4.1 Commercial Examples

A market leader for type 1 (multi-project managers) projects discussed above would be Primavera Project Planner (supplied by Primavera Systems Inc.), which is primarily aimed at the high end of the multi-project market [Heck and Mitchell, 96]. It provides all the standard project management functionality outlined above and in addition provides extra functionality specifically orientated towards the needs of managers of large-scale multi-project organisations. It also provides for developing what-if scenarios but little in terms of intelligent assistance as discussed in section 1.5 and does not assist with analysing the suggested alternatives and selecting the most suitable course of action.

PE/Project Manager (supplied by LBMS Corporation) is primarily orientated towards organisations with a well defined software process [Humphrey, 89], with its typical users being in charge of type 2 (mid-range project managers) projects. PE/Project Manager provides much of the standard project management functionality outlined in table 2.1 and in addition provides extra functionality specifically orientated towards software process management. Unlike Primavera, PE/Project Manager claims to have some intelligent assistance capabilities. These are mainly concerned with the selection of the most appropriate process model (from a supplied set) and providing assistance in guiding the process of converting it into an outline project plan. However, much of the information that is used in the procedure is gained from pre-existing process models which are supplied with the tool. It is therefore rather general and offers nothing by way of assisting the project manager in reasoning about the selection process. Although it does provide a facility to build and reuse process models, it does not provide any features for capturing knowledge gained during the execution of a process, or the rationale behind the choices made during the execution of the project/process.

MS-Project (supplied by Microsoft) is the popular choice of type 3 (Low-end project managers) project managers [Heck and Mitchell, 96]. It provides a number of the project management functionality outlined in table 2.1, and is primarily orientated towards basic project planning and scheduling. Project does advertise an "*intelligent assistant to provide guidance while you work*". However, this is actually a 'Microsoft style Wizard', referred to as a Microsoft Agent [Microsoft, 97]. It appears as a 'friendly' face or icon on the screen which provides advice on how to achieve certain tasks using the Project tool. A number of intelligent agent projects including Microsoft Bob [Miller, 97] have explored the concept of giving human-like attributes to agents by visually representing them in the form of cartoon-like animated faces [Maes, 97]. However, in Project, this assistant uses a predefined mechanism to provide the user with tool assistance and does not provide any features for capturing knowledge or assisting the user in reasoning about a project.

IntraPlan (supplied by Intra2000) is an Internet groupware project management application orientated towards organisations involved in Internet/Intranet development [Stone, 97], with a typical client being a type 3 (Low-end project manager) project manager. IntraPlan does advertise some 'intelligent' capabilities. However, these are also a 'Microsoft style' Wizard [Microsoft, 97] which guides the user through a series of predefined questions in an attempt to identify a set of suggested solutions.

A number of commercial tools claim to have 'intelligent' features. However, when these features are further investigated, it can be seen that they primarily refer to efficient algorithms, task automation, or other clever labour and time saving facilities. Some of these tools feature Wizards or graphical assistants which assist the user in interacting with the tool or in following a predetermined recommended series of actions. These however, only react to predetermined situations and do not have any capabilities for analysing the current situation or for providing advice unique to that situation. Most of these tools do provide a 'What if' method of generating scenarios to allow the project manager hypothesise about the impact of possible future decisions.

These features simply use algorithmic techniques to generate project plans, etc. based on altering the value of given project variables. They do not analyse the impact of a potential decision on the project or the organisation as a whole, or attempt to give advice to the project manager on best practices in a given situation.

None of these tools provide the intelligent assistance features discussed in chapter 1, such as:

- Assisting the user in assimilating knowledge and best practices, with regard to decision making.
- Providing the capability for project managers to reason about a project's plans, analyse alternatives and select the most suitable course of action.
- Providing a facility to capture knowledge gained during a project, and reuse this knowledge as an aid to future project decision-making.
- Assisting the project manager in adherence to standards, industry best practices and implementation of company policy.

2.5 Tool Users

In this section, the results of a tool user survey are presented. The purpose of the survey was to obtain an appreciation of the type of tools that are being used by project managers and to get a better understanding of the actual state-of-practice regarding these tools, i.e. what do the project managers actually use these tools for and is this consistent with the tool vendors intended usage. In addition, users were asked to consider the aspects of intelligent assistance and comment on the possible benefits of incorporating these into a project management support system.

It was not the purpose of this survey to provide a comprehensive in-depth study of tools users. In order to guarantee a wider range of opinion and thus be further assured of the representative nature of the survey results, a more comprehensive survey of project managers would be required. However, this small scale survey provides enough data to obtain an appreciation of the general trend of user opinion.

A group of six project managers from three European countries took part in this study. These project managers represent a variety of software development organisations, from small project teams of 2-3 developers, to large multi-national organisations with over one hundred developers. The project managers themselves varied from novice (first time), to experienced (greater than 10 years) senior managers. The projects they manage vary from small projects of 2-3 months duration (2-6 person months of effort), to large scale complex projects in excess of 2 years duration.

Each interview lasted approximately two hours and was tape recorded to assist the interviewer in writing a report after the meeting. Each project manager was asked a series of questions (cf. Appendix B), ranging from the highly specific - aimed at investigating a particular area - to the more generic, to allow the project manager to further consider and develop their own opinions. The questions asked in the survey were divided into the following categories:

- General background questions regarding the person and employer.
- Questions to assertain the type of projects normally undertaken.
- Questions about methodologies, standards and development tools used.
- Questions about the use of project management tools.
- Questions regarding the potential benefits of an intelligent assistant system.

In relation to project management tools, each project manager was asked a series of questions aimed at finding out what (if any) tools were being used for project management, the manner in which they were being used and the usefulness of certain types of features. Each manager was also asked to consider the proposal of an intelligent assistant in the context of a software project planning tool and offer an opinion on the proposed features.

The following six sections present a synopsis of the interview with each of the six project managers, in order of the size of the organisation. A profile of each of the project managers and their working environment is presented, as well as summary of the interview under the main heading above. In order to respect the privacy the project managers, they will be referred to as subjects ‘A’ to ‘F’.

2.5.1 Project Manager A

Heading	Description
Organisation	Software Competency Centre, Group Schneider Electric (France).
Organisation profile	Specialist software development division (100 software engineers) of a large multi-national specialising in industrial control and automation products. Within the organisation there is a strong emphasis on adherence to agreed processes, standards and quality procedures.
Personal profile	15 years in software business, 8 of them as a project manager.
Tools available	There are a range of high-end and low-end tools, with the choice usually dependent on the individual preferences of the project manager.
Tool usage	There is a strong bias towards the use of the Artemis (high-end enterprise tool). Low-end tools such as Microsoft Project are also used for planning small projects and also as an aid to 'sketch out' new projects.
Recurring project management difficulties	The main areas of recurring difficulties in project management for subject are estimation and scheduling of priorities between tasks. She suggested that existing project management tools are not of much help in these areas.
Benefits of intelligent assistant	She considered that an intelligent assistant system could prove useful - in particular the notion of a tool which a manager could 'bounce' ideas of, in terms of project plans. She also believed a tool which could assist a project manager in ensuring the implementation of company policy (in respect of quality, organisational processes, etc.) would be advantageous.

Table 2.2 - Profile of project manager A

2.5.2 Project Manager B

Heading	Description
Organisation	Development Programmes Department, Intracom (Greece).
Organisation profile	Research and development division (50 software developers) of long established telecommunications organisation.
Personal profile	15 years in research and development, 5 as a project manager.
Tools available	A number of different tools are available. The choice of tool usually depends on the scale of the project or the individual preference of the project manager.
Tool usage	Microsoft Project is frequently used in pre-project planning stages, to help build possible views of the project and in trying to get a 'feel for' it. This information is often used to assist in developing tentative schedules and calculating potential resource requirements for the project. For the actual management of an individual project, a number of mid-range tools such as the LBMS tool set are often used.
Recurring project management difficulties	Scheduling of multiple projects and ensuring that all relevant steps were taken into account in the planning stages. This issue is often compounded by the fact that the developers on some projects are geographically dispersed throughout a number of locations and often with access to different computer platforms.
Benefits of intelligent assistant	He had a number of suggestions for improved tool support: a tool which could be used to centrally manage project plans for a multi-location project (distributed heterogeneous network); the ability to analyse the impact (knock-on effect) on other projects of a decision taken in one project; assistance with ensuring traceability is kept between requirements and product functionality. He also suggested that a tool which could 'coach along' a project manager in aspects of improving control and quality would be valuable.

Table 2.3 - Profile of project manager B

2.5.3 Project Manager C

Heading	Description
Organisation	Siemens Business Services (Ireland).
Organisation profile	Software development and consulting division, with approximately 30 software engineers.
Personal profile	17 years in IT business, 6 as a senior project manager.
Tools available	Microsoft Project, some use of LBMS tools for certain clients.
Tool usage	Although subject C is a senior project manager, with 3 project managers reporting to her, the organisation has a very low usage of specialist project management software. Spreadsheets are widely used to create and update schedules. Microsoft Project is used on occasion for scheduling and other reporting to clients.
Recurring project management difficulties	The main areas of recurring difficulties in project management for subject C are that of scheduling and estimation, as well as coping with new technologies. However, in general, C stated her projects usually ran on time and within budget, but that there is scope for a more useful tool (than Microsoft Project) to assist in routine work.
Benefits of intelligent assistant	Among her suggestions were; a tool facility which could prompt her about the various phases of a project - to act as a reminder of what management activities should be carried out when; assistance/advice on how to build concepts such as quality into particular projects; a facility in which to record the lessons learned from a project and provide easy access to it when a similar situation occurred again in another project.

Table 2.4 - Profile of project manager C

2.5.4 Project Manager D

Heading	Description
Organisation	Irish Distillers, member of Pernod Ricard group.
Organisation profile	IT department, with 7 software developers and numerous support staff. IT consultants are regularly used.
Personal profile	21 years in IT, 15 of them as a project manager.
Tools available	Microsoft Project.
Tool usage	Pen and paper charts and occasionally Microsoft Excel are used to sketch out required resources for a particular project. The only time a project management tool is used is on larger new projects, when Microsoft Project is used to schedule high level tasks.
Recurring project management difficulties	Slippages in project deadlines, mostly due to a lack of prior notification of new projects which have to be started in parallel with existing projects.
Benefits of intelligent assistant	D was very articulate about what he would require from a project management tool and responded to the notion of intelligent assistant capabilities in a project management tool with several suggestions: The tool should be 'very easy to use and not require loads of data input'; He felt that tools should be able to capture information regarding people's skills and abilities, and assist with using this information when assigning people to tasks and scheduling those tasks; The ability to insert data about tasks and priorities and to 'bounce schedule ideas' of a tool which could develop possible schedule scenarios.

Table 2.5 - Profile of project manager D

2.5.5 Project Manager E

Heading	Description
Organisation	BTT Systems (Ireland).
Organisation profile	Software house, with 10 software engineers.
Personal profile	Owner-director, 30 years experience in IT, 10 in project management.
Tools available	Microsoft Project.
Tool usage	Microsoft Project, which is used to develop schedule reports which are shown to customers. However, E admits that "*these schedules are not even looked at by developers*", they are produced "*to keep the customer happy*". All actual management aspects of projects are done manually using paper based plans and charts.
Recurring project management difficulties	Problems are primarily due to the rapidly changing pace of technology, particularly in the area of network software and the multi-vendor situation they work in. They have constant difficulties in keeping up to date with new technology/products and capturing the associated knowledge gained from using this technology.
Benefits of intelligent assistant	He believed the availability of an intelligent assistant would be of great benefit to him, if it incorporated a knowledge base which would keep track of the "Eureka Factor", (i.e. specialist knowledge gained during projects), which could then be searched when similar problems occur again. Other additions he suggested would be appropriate for such a system included; an ability to assist the project manager in identifying complexity in a project; help with formulating steps to deal with customers, especially during the requirements phase; and a tool which would assist with, and enforce a procedure for documenting knowledge learned on projects.

Table 2.6 - Profile of project manager E

2.5.6 Project Manager F

Heading	Description
Organisation	Softworks Computing (Ireland).
Organisation profile	Recently established software house, specialising in human resources management software. 5 software developers.
Personal profile	Owner-director, 6 years experience in software development.
Tools available	None.
Tool usage	All plans and schedules are "*scribbled on bits of paper*" and when a "*reasonable looking plan*" emerges, it is translated onto a whiteboard which is altered in an ad-hoc manner over time to reflect the current state of all tasks and people assigned to them in all current projects.
Recurring project management difficulties	They have had requirements management problems due to clients who regularly change their requirements, thus causing a knock-on effect throughout the project. Like project manager E, he would like a tool which could assist in the requirements engineering process.
Benefits of intelligent assistant	F has previously conducted postgraduate research in Expert Systems and thus had a good understanding of the proposal of intelligent assistance in project management tools. He considered the proposed intelligent assistant to be very useful and had potential to be of use in other aspects of management, not just for software development. He warned of potential user problems, which could be overcome with careful user interface design and subtle direction by the tool. He also suggested that a tool for capturing knowledge about projects, and the reasons behind design and other decisions would be most useful. In addition, the notion of being able to automatically create possible alternative project plans and receive advice on which may be the most appropriate 'text book' approach, would also be useful.

Table 2.7 - Profile of project manager F

2.5.7 Survey Results

The project managers who participated in the study represented a range of organisations, both in terms of number of software developers and types of project management tools used. However, it is interesting to note that the majority of project managers had similar problems, particularly in respect to estimation and scheduling. Additionally, they identified similar difficulties with existing project management tools. The project managers surveyed provided a number of suggestions for enhancements to such tools, and a number of these suggestions serve to reinforce the proposition previously discussed. The project managers considered an intelligent project assistant to be a useful addition to the existing range of features in project management tools. In particular, they supported the concept of a tool which could intelligently manage project knowledge and capture knowledge and lessons learned about projects into a project knowledge base. Apart from the intelligent assistant aspects of this research, the problems associated with organisations having distributed project teams coupled with multiple hardware platforms was identified by all the project managers surveyed, thus highlighting the need for a distributed platform tool.

The main purpose of the tool user survey was to validate the proposition of an intelligent assistant system and obtain user feedback on the notion of incorporating intelligent assistance in a project planning tool. Having analysed both the software project management tools discussed above and the survey of tool users, it is considered that the initial premise of this thesis has been validated.

To obtain further feedback from potential tool users and from the wider software engineering community, the results of this survey coupled with details of the proposed intelligent assistant system were presented at the 9th European Software Control and Metrics (ESCOM) conference [O'Connor and Jenkins, 98]. A number of positive comments were made about the need for, and potential usefulness of, such a tool. The main concern that was expressed was in relation to aspects of knowledge elicitation and knowledge representation. These issues will be dealt with in chapter 5.

Further validation was obtained by the implementation of a prototype system, which was demonstrated to the tool users previously surveyed, in conjunction with a wider study. This exercise allowed users to more easily appreciate and consider the proposed system, and in addition, a prototype system provided an easier mechanism to demonstrate the system to a larger audience.

2.6 Summary

This chapter contains a description of software project planning and its characteristics in order to better understand what is required from an intelligent assistant system for software project managers. This has also presented a discussion on the functionality of software project planning tools and a survey of software project planning tool users in an attempt to validate the premise of this research.

Chapter 3 will examine intelligent assistant systems in other domains with an emphasis on their underlying architecture. The issues surrounding the architectural concerns of client-server and distributed systems will also be examined to set the context for a discussion on the architecture for the proposed system.

CHAPTER 3 APPROACHES TO INTELLIGENT ASSISTANCE

3.1 Introduction

This chapter describes several commonly used approaches to intelligent assistance. Following this discussion, a strategy for implementing intelligent assistance within the scope of a software project planning tool is proposed.

3.2 Intelligent Assistance Approaches

Many solutions have been proposed to the notion of intelligent assistance (in different domains) over the years. These fall under several main categories: Decision Support Systems, Expert Systems, Expert Critiquing Systems, Intelligent Tutoring Systems, Blackboard Systems and Intelligent Agents.

The following sections review each of the above approaches and attempt to assess the potential application of each to the development of more powerful and useful software project management tools. Finally, in section 3.3 a proposal for the incorporation of intelligent assistance within a software project planning tool will be presented.

3.2.1 Decision Support Systems

Decision Support Systems are interactive computer based systems which help decision makers utilise data and models to identify and solve problems and make decisions. [Bonczek et al., 81] offers the following description of a DSS:

> *"The system must aid a decision maker in solving unprogrammed, unstructured (or semistructured) problems".*

Essentially DSS are computer-based systems that bring together information from a variety of sources, assist in the organisation and analysis of this information and facilitate the evaluation of underlying assumptions [Mallach, 94]. They help managers/decision makers use and manipulate data; apply checklists and heuristics; and build and use mathematical models. The availability of DSS provides the opportunity to improve data collection and analysis processes associated with decision making, and further, DSS provide opportunities to improve the quality and responsiveness of decision making and hence the opportunity to improve the management of projects. According to Turban [Turban, 95], a DSS has four major characteristics: DSS incorporate both data and models; they are designed to assist managers in their decision processes in semistructured (or unstructured) tasks; they support, rather than replace, managerial judgment; their objective is to improve the effectiveness of the decisions, not the efficiency with which decisions are being made.

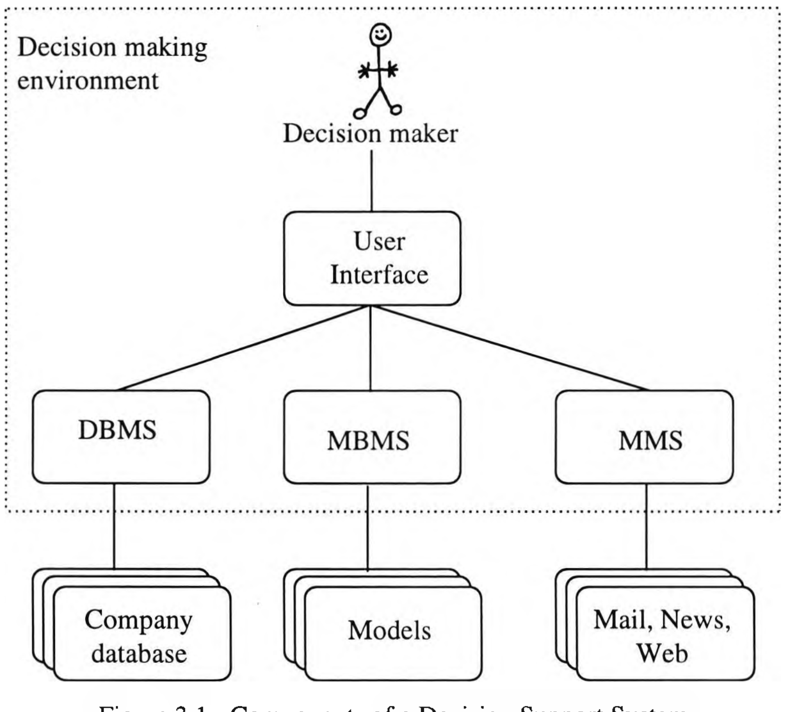


Figure 3.1 - Components of a Decision Support System

A DSS is typically composed of four components (figure 3.1); DBMS (Database Management System) - providing access to data and control programs to get the data into appropriate forms for analysis; MBMS (Model-Base Management System) keeps track of all models running during an analysis and provides the user with a facility to question the assumptions of models; A user interface provides the mechanism whereby information is presented to the user; and recently a new component - the MMS (Mail Management System) has emerged which incorporates mail and other on-line data into the decision support models.

DSS have been successfully applied in a number of fields. For example the DESSERT project [Tierney and Davison, 95], a European Commission funded RACE II project which examined the use of DSS technology to help users (suppliers of telecommunications products) cope with increasingly complex engineering problems associated with telecommunications service provision. The project implemented the GSAC (Generation and Selection of Alternative Configurations) prototype DSS which took customers technical requirements as input and generating all feasible telecommunications access configurations and established the best (optimal) solution that meets both the customer and service providers requirements. The group's conclusions, supported by prototype trials by British Telecom and SEMA Group Telecom, suggested that DSS technology was an important element in making key parts of service management more efficient.

3.2.2 Expert Systems

An Expert System (ES) is a computer program that represents and reasons with knowledge of some specialist subject with a view to solving problems or giving advice [Jackson, 90]. An ES may completely fill a function that normally requires human expertise, or it may play the role of assistant to a human decision maker. The symbolic reasoning of an ES enables it not only to draw conclusions, through a process similar to the one used by human experts, but also enables them to provide explanations concerning their estimations. ES technology is based on the domain knowledge of the problem being addressed. A problem domain defines the objects, properties, tasks and events within which a human expert works and also the heuristics that trained professionals have learned to use in order to perform better [Klein and Methlie, 95].

The DENDRAL project [Lederberg, 87] initiated by Ed Feigenbaum and others at Stanford University in the mid 1960s was the first system to demonstrate that it was possible for a computer program to rival domain experts in a specialised field. In the case of DENDRAL the domain was the identification of complex compounds in a molecular structure.

DENDRAL was an early example of the basic structure of an expert system: problem solving behavior and formalized domain specific knowledge (in the form of a rule-based system). It had the ability to explore and abandon potential goal paths and is considered one of the earliest successes in expert systems. Many systems were spawned from DENDRAL. Two of the most noteworthy are Meta-DENDRAL and GENOA [Buchanan and Feigenbaum, 78].

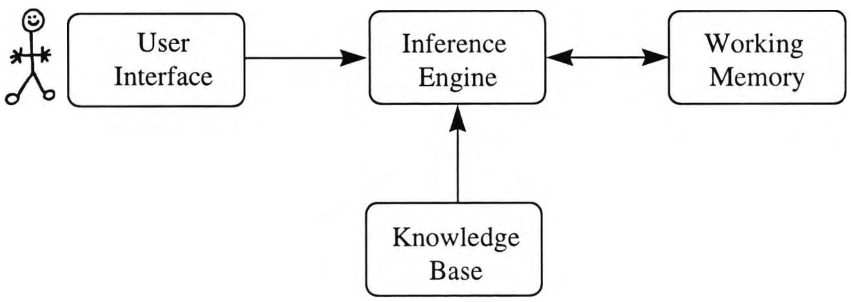


Figure 3.2 - Components of a Expert System

The heart of every expert system consists of two principal parts (see figure 3.2): the knowledge base; and the reasoning, or inference, engine. The knowledge base of expert systems contains both factual and heuristic knowledge. Factual knowledge is that knowledge of the task domain that is widely shared, typically found in textbooks or journals, and commonly agreed upon by those knowledgeable in the particular field. Heuristic knowledge is the less rigorous, more experiential, more judgmental knowledge of performance. In contrast to factual knowledge, heuristic knowledge is rarely discussed, and is largely individualistic. It is the knowledge of good practice, good judgment, and plausible reasoning in the field. It is the knowledge that underlies the “art of good guessing”. Other system components include: Working memory, which is used to store the current ‘facts’ or state of the domain - and a user interface to handle user input and output.

Primarily, the benefits of ES's to end users include [Engelmore , 93]:

- Improved quality of decision making.
- A speed-up of human professional or semi-professional work.
- Preservation of scarce expertise. ESs are used to preserve scarce know-how in organizations, to capture the expertise of individuals who are retiring, and to preserve corporate know-how so that it can be widely distributed to other factories, offices or plants of the company.
- Introduction of new products. For example, a pathology expert advisor sold to clinical pathologists to assist in the diagnosis of diseased tissue.

A more recent example of ES technology applied to a technical domain would be the EXPIDER system [Shen et al., 97] which was applied to solving channel routing problems in VLSI (Very Large Scale Integration) design. The system was built using CLIPS (C Language Integrated Production System) [Giarratano and Riley, 94].

3.2.3 Expert Critiquing Systems

Expert Critiquing Systems (ECS) [Silverman, 92] are a class of programs that receive as input the statement of the problem and the user-proposed solution. They produce as output a critique of the users judgement in terms of what the program considers is wrong with the user-proposed solution. Simple, yet widely used examples of critiquing system are spell checkers.

Critiquing programs may be user invoked (passive) critics or active critics. They may work in 'batch mode' - process the entire solution after the user has finished it; or incremental mode - interrupt the user during their problem-solving task. They do not necessarily solve problems for the user. Their core task is to recognise and communicate debatable issues concerning a product. Critiquing systems point out errors and suboptimal conditions that might otherwise remain undetected. They also advise users on how to improve the product and explain their reasoning, thus helping users avoid problems and learn different views and opinions.

An appealing characteristic of ECS is that they can be employed in a wide variety of situations with a broad range of supportive functions. They can, for example, complement problem-solving systems with or without full (i.e. global) knowledge of the problem space by being able to comment on particular sections of the problem/solution at hand. They can also function in situations where no near-optimal or optimal solution can be given at all and still be able to provide useful comments.

The critiquing process is illustrated in figure 3.3. The user initiates the task using some task support software. For a given task, the users input to the system consists of:

1. A description of the problem (e.g. design requirements). The problem may also be one the computer is displaying to the user, as perhaps in a process control application.
2. The proposed solution to the problem, such as the final design or completed diagnosis.

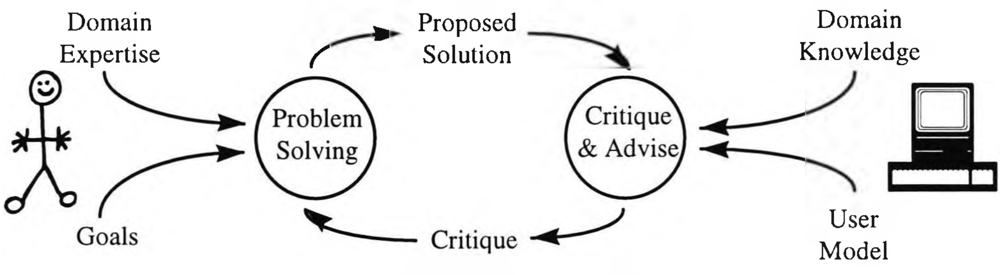


Figure 3.3 - The Critiquing Process

The critic applies its domain knowledge to the user problem and attempts an analysis of the user supplied solutions. It produces a critique and advises on possible flaws in the solution and on possible unforeseen implications of the proposed solution. This information is then used by the user to apply in the next iteration of this process.

Critiquing systems have appeared in fields such as LISP programming environments [Fischer, 87] and CAD (Computer Aided Design) [Gupta et al., 96]. They have also been applied in the field of software project support systems - the IMPW project (Integrated Management Process Workbench) [Jenkins et al., 87] a European Commission funded ESPRIT project, designed a workbench of tools to aid project managers organise and control projects. One of the tools developed was RISKMAN [Verbruggen et al., 89] which enabled project managers ‘walk around’ a proposed project and help him/her anticipate any major risks. This project was taken further with the development of the RISKMAN2 tool [Moynihan et al., 94].

3.2.4 Intelligent Tutoring Systems

Intelligent Tutoring Systems (ITS) [Polson and Richardson, 88] allow the emulation of a human teacher in the sense that an ITS can know what to teach (domain content), how to teach it (instructional strategies), and learn certain teaching-relevant information about the student being taught. This requires the representation of a domain expert's knowledge (Expert Model), an instructor's knowledge (Instructional Model) and the particular student that is being taught (Student Model).

The expert model represents information specific to the subject being taught, the student model portrays the current student understanding or misunderstanding of the subject and the instructor model contains knowledge required of teachers to select meaningful lessons for their students. Through the interaction of these models, intelligent tutoring systems are able to make judgments about what the student knows and how well the student is progressing. Instruction can then be automatically tailored by the Instructional Model to the student's needs, without the intervention of a human instructor. The ITS acts as the student's private tutor, while the human trainer or teacher is then free to focus on more complex and individualised student needs.

The design of most ITS is closely linked to how the ITS presents the domain to a student: knowledge from the simulation, knowledge packaged as rules, frames, or scripts, or intuitive interfaces. Figure 3.4 illustrates the dimensions of device or operational simulation, the domain expert and the interface.

A large number of ITS’s have been constructed, although few are in widespread use [Frasson and Gauthier, 86]. ITS have been applied in many domains such as: IDE (Instructional Design Environments) [Russell et al., 89] which has been used in foreign language instruction; ISD Expert [Merrill, 87] for instructional developers; teaching concepts in debugging electronic circuits [Brown et al., 82] and EDUC [Cox, 94] in the domain of engineering education.

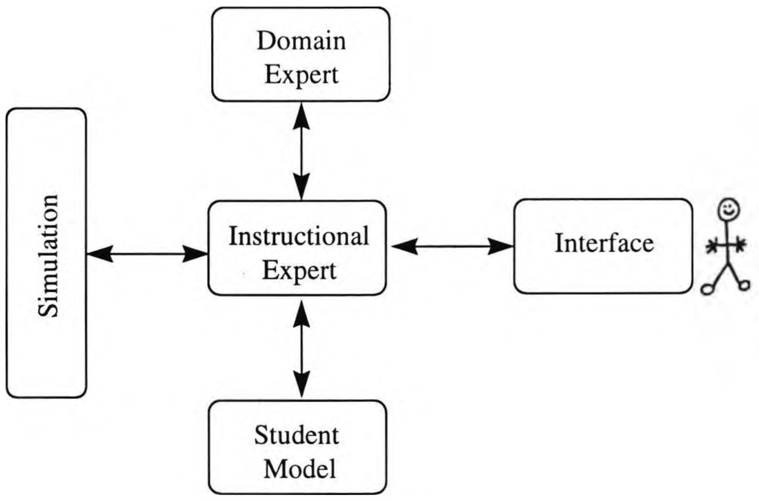


Figure 3.4 - Domain Knowledge Architecture

3.2.5 Blackboard Systems

The Blackboard [Hayes-Roth, 83] is a problem solving model prescribing the organisation of knowledge, data and the problem solving behavior within the overall organisation. The blackboard paradigm is best described using the following analogy [Englemore and Morgan, 88]:

> *"Imagine a number of people in a room trying to solve a jigsaw puzzle together. The room has a large blackboard and around it there is a group of people each holding pieces of the puzzle. Some volunteer to put their most 'promising' pieces on the blackboard and the others look at the pieces in place on the blackboard and then examine their own pieces to see if any fit. Those who have pieces that fit then go to the blackboard and fit their pieces. This process continues until all the pieces have been placed on the blackboard and the problem is solved."*

Essentially blackboard systems use multiple independent knowledge sources to analyse different aspects of a problem. Each knowledge source contributes its information to the common working memory (the blackboard). Blackboards compose solutions from component subsolutions, each of which may be generated or modified by its own knowledge sources.

Blackboards consist of three major components (figure 3.5):

1. **Knowledge sources** - (KSs). Like the human experts, each KS provides specific expertise needed by the application.
2. **The blackboard -** a shared repository of problems, goals, partial solutions, suggestions and contributed information. The blackboard can be viewed as a dynamic library of requests and contributions that have been recently published by other KSs.
3. **The control shell** - which controls the flow of activity in the application. Just as eager human specialists sometimes need a moderator to prevent them from trampling in a mad dash to grab the chalk, KSs need a mechanism to organize their use in the most effective and coherent fashion.

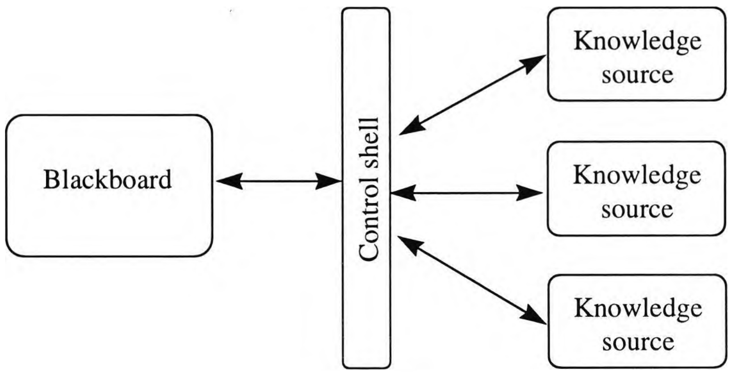


Figure 3.5 - Blackboard Components

The blackboard paradigm offers a number of important advantages, including:

- *Modularity* - KSs can be developed independently (for example, by programming teams) or even developed long before the application itself is developed.

- *Integration* - KSs can be implemented using widely differing approaches and programming languages. They may run remotely, on diverse hardware.
- *Extensibility* - New KSs can be added easily, and existing KSs can be replaced with enhanced versions.
- *Reusability* - KSs that provide expertise to one application can be redeployed in new applications.
- *Strategic Control* - Strategic control knowledge can be used to determine where the application expends its computational resources. Effective control is important when the number of KSs grows, when KSs have overlapping capabilities and when the best approach to solving today's problem differs from the approach used in yesterday's problem.

Blackboard technology has been successfully applied in a number of problem domains. The RISKMAN2 expert critiquing system project discussed in section 3.2.3 used a blackboard approach to pool knowledge from multiple knowledge sources. More recently USA based BBTech have developed Generic Blackboard Builder [Corkhill, 97] a generic blackboard product which has been used by the Ford Motor Company in their engineering design process and the US Army for logistics planning.

3.2.6 Intelligent Agents

The area of Intelligent Agents has emerged in recent times as a 'hot topic' in several branches of computing research from artificial intelligence to information systems. Among the agent community there is no commonly agreed definition of a software agent, indeed Carl Hewitt remarked[1]:

> *"...the question what is an agent? is embarrassing for the agent-based computing community in just the same way that the question what is intelligence? is embarrassing for the mainstream AI community".*

[1] At the 13th International Workshop on Distributed Artificial Intelligence, Washington, USA, 1994.

The problem is that although the term is widely used by many people working in closely related areas, it defies attempts to produce a single universally accepted definition. With this in mind it is possible to distinguish two general usages of the term 'agent': the first is weak, and relatively uncontentious; the second is stronger, and potentially more contentious [Wooldridge and Jennings, 95].

The weaker, and perhaps the most general way in which the term agent is used, refers to a kind of UNIX-like software process which exhibits the following properties:

- **Autonomy** - agents operate without the direct intervention of humans and have some kind of control over their actions and internal state.
- **Social ability** - agents interact with other agents (and possibly humans) via some kind of agent-communication language.
- **Reactivity** - agents perceive their environment and respond in a timely fashion to changes which occur in it.
- **Pro-activeness** - agents do not simply act in response to their environment, they are able to exhibit goal-directed behaviour by taking the initiative.

This weak notion of agency is basically that used in the emerging discipline of agent-based software engineering [Genesereth and Ketchpel, 94] and describes agents as being able to "communicate with their peers by exchanging messages in an expressive agent communication language".

For some researchers - particularly those working in AI - the term 'agent' has a stronger and more specific meaning than that sketched out above. These researchers generally mean an agent to be a computer system that, in addition to having the properties identified above, is either conceptualised or implemented using concepts that are more usually applied to humans. For example, it is quite common in AI to characterise an agent using mentalistic notions such as knowledge, belief, intention, and obligation [Shoham, 93]. Some AI researchers have gone further and considered emotional agents [Bates et al., 92]. Another way that agents have been given human-like attributes is by visual representation in the form of an animated face [Maes, 97].

Agent technologies also offer models of social cooperation. Using agent-based approaches, economists have constructed informative (if not completely predictive) models of economic markets. Agent technologies have exerted an increased influence on the design of distributed computing systems, the construction of internet search tools and implementation of cooperative work environments.

From an architectural viewpoint, [Luger and Stubblefield, 98] argue that a single architecture cannot account for all intelligent behavior. Instead, intelligence results from the cooperation of highly specialised agents. [Minsky, 85] outlines such a model in which the mind consists of a collection of specialised agents. Each agent contributes a particular ability to such tasks as understanding data or high-level problem solving. Other proponents of this multi-agent view of the mind include Hebb [Hebb, 49], Selfridge [Selfridge, 59], Fodor [Fodor, 83], Brooks [Brooks, 89] and Dennett [Dennett, 91].

When compared with the technologies discussed in the previous sections, agents are a young research area. However, a number of agent based systems have emerged; for example the ARCHON (ARchitecture for Cooperative Heterogeneous ON-line systems) project [Jennings et al., 96a], a European Commission funded ESPRIT II project aimed at developing an architecture, software framework and methodology for multi-agent systems for industrial applications in the area of power system control supervision. Another example is the ADEPT (Advanced Decision Environment for Process Tasks) project [Alty et al., 94], which provides an agent-based infrastructure for managing business processes. Both of these systems will be examined in further detail from an architectural perspective in Chapter 4.

3.2.7 Review of approaches

In this section, the six approaches to intelligent assistance previously discussed are contrasted to assess their comparative strengths and weaknesses in order to appraise their suitability as approaches to implementing intelligent assistance in software project planning tools.

Expert Critiquing Systems are different to Expert Systems in that ES only accept the problem statement as input and provide their machine generated solution as output, by applying rules in its knowledge base to the problem under consideration. ECS take as input a user proposed solution and a statement of the problem and attempt to offer an opinion on this solution. ECS do not necessarily solve problems for the user, they simply point out errors and sub-optimal conditions in the user-proposed solution that otherwise may remain undetected. ECS are particularly well suited to complex problem domains (such as project planning) as they do not always have an optimal solution and the problem cannot be precisely specified before attempting a solution [Fischer et al., 91]. In contrast to ES, ECS can function with only a partial understanding of the task, as they can provide support by applying generic domain knowledge, whereas ES are inadequate in situations where it is difficult to capture sufficient domain knowledge, because they leave the human out of the decision process and all 'intelligent' decisions are made by the computer.

Both ES and ECS have been applied in various problem domains. For example, the EXPIDER system [Shen et al., 97] successfully applied ES technology to the domain of VLSI design and the RISKMAN2 tool [Moynihan et al., 94] implemented an ECS in the domain of software project risk. Both of these technologies have proved to be of benefit to the user and can be seen to have a complementary approach, where ES assist the user in arriving at a solution and ECS enhance solutions by providing a critique of them. This dual approach to problem solving can potentially overcome the difficulties of operating in a situation where domain knowledge may be incomplete or inconsistent. However, it is worth noting that most implementations of ES's have been for well understood domains, which lend themselves more easily to capturing all necessary decision making data in the form of rules or other similar representation.

In contrast to ES and ECS, where domain knowledge in problem solving is embedded in a knowledge base, Decision Support Systems provide a framework for users in which models of the domain may be built, data gathered and decisions arrived at through informed analysis. Traditionally DSS do not have ES style prescriptive knowledge bases, rather they assist the user in analysing and evaluating assumptions underlying a problem by using models, but do not propose or critique a solution. In the complex ‘data rich’ domain of large scale software development projects, DSS could prove useful in the gathering and analysis of project data, in an attempt to evaluate models of potential solutions. The results of the DESSERT project [Tierney and Davison, 95] in the telecommunications domain affirm this proposition. However, DSS may be more useful to the software project manager if coupled with the advisory and critiquing capabilities of both ES and ECS.

A view of an ITS would be that a persons problem solving skills would be represented by a set of production rules. Errors in problem solving efforts can be explained by the absence, incorrectness, or misuse of one of these rules. Intelligent computer-aided instruction seeks to identify the missing or incorrect rule and then teach the learner that skill or rule. ITS ‘reconstruct’ a problem solving process in a data driven way whereas an ES ‘executes’ a problem solving process in a goal driven manner. [Kamsteeg and Blerman, 89] illustrate this by the example of using an ES as the domain knowledge component of an ITS, which results in the writing of a new ES specifically designed to use the domain knowledge component of an ITS. This is due to having to add knowledge (e.g. incorrect knowledge, more levels of detail) and change knowledge representation to make it more explicit. In the context of providing training support for software project managers, the ITS approach would be a primary candidate. However the ITS approach would be less useful in the context of tool support for project managers, as this is removed from the notion of training support.

The Blackboard concept is a general model of problem solving and is particularly useful where there exists a number of (independent) knowledge sources. It provides a framework in which these knowledge sources may work cooperatively to assist the user in arriving at a solution. In common with DSS, the blackboard approach does not propose or critique a solution. It provides a framework for arriving at a solution state. The blackboard concept has been successfully applied within a number of approaches: DSS blackboards have been used in a number of systems, including DESSERT project mentioned earlier. In ES projects such as ESHELL blackboards were used to enhance working memory of the ES; and in ECS such as RISKMAN2, blackboards are used as a framework for each critic (or knowledge source) to add its advice and inspect the state of the developing solution. It is clear from previous research that blackboards are both a viable and enhancing technology around which to base a future intelligent assistant systems.

Intelligent Agents are a relatively new and potentially exciting aspect to intelligent assistance systems. The agent properties of autonomy, reactivity and pro-activeness which fit naturally with the characteristics of the technologies discussed above. Blackboards are a natural candidate for use by agents, as they can represent ('autonomous') knowledge sources each of which cooperates via a blackboard to arrive at a solution state. Agents may also function as critics (as in the 'reactive' nature of ECS) or expert advisors (where an agent represents a 'pro-active' mini ES), where each agent is a specialist in some aspect (or sub-domain) of the greater problem domain being addressed. The former (critic mode) of agents was explored in conjunction with the blackboard model in the RISKMAN2 project and the latter (expert mode) was tested during the ADEPT project. From the research outlined previously and other current work in the area of agents it is apparent that agent based technology has a significant role to play in the development of future intelligent assistant systems.

3.3 Proposal for a new Intelligent Assistant

It is the proposition of this research that, in the complex domain of software project planning, a useful framework to support the project manager in the decision making process is a hybrid of a number of techniques including DSS, ES, ECS and the blackboard model. It has therefore been proposed [O'Connor et al., 97b] [O'Connor and Renault, 98] to incorporate the information gathering and analysis techniques of DSS, with the ability of ES to propose possible solutions using expert knowledge and best practices and the power of ECS to critique the possible solutions, thus providing the project manager with every facility to make an informed and quality decision.

It is considered that an agent based system will provide for an approach which enables the inter-working of a variety of well understood techniques within a single underlying framework - that of an agent-orientated system as illustrated in figure 3.6.

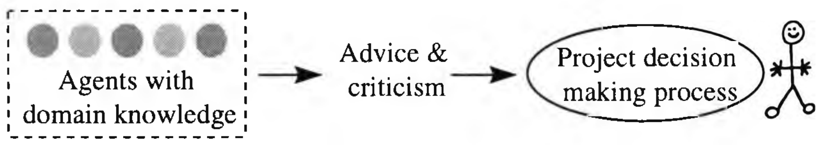


Figure 3.6 - Decision Making Process

The system proposed therefore is composed of a library of intelligent software agents - where each agent would play the role of a 'mini-expert system' or 'mini-critiquing system', each with an associated knowledge base. These agents would utilise the blackboard model of problem solving to communicate and thus converge on possible solution states and examine those states to assess their suitability given current conditions. This agent-orientated system would operate within the overall framework of a Decision Support System, which would provide for the gathering and analysis of data regarding a project and the development of models of the project with the aid and critique of the agents, as illustrated figure 3.7.

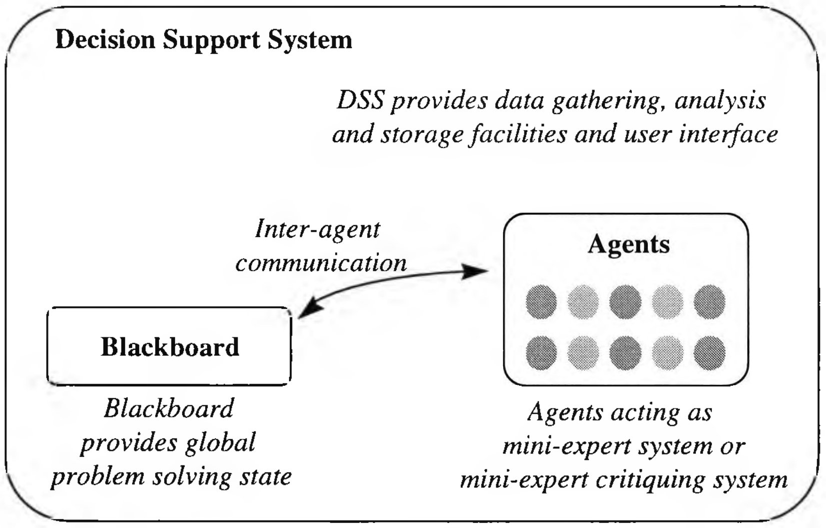

Decision Support System
DSS provides data gathering, analysis and storage facilities and user interface
Inter-agent communication
Agents
Blackboard
Blackboard provides global problem solving state
Agents acting as mini-expert system or mini-expert critiquing system

Figure 3.7 - Decision Making Framework

The major perceived benefits of this approach are the facilitation and improvement of the quality of decision making by a software project manager by reducing information overload and augmenting the cognitive limitations and bounds of the decision maker. This hybrid method of assistance, coupled with the architectural properties of intelligent agents (dynamic and distributed objects), present an ideal strategy to implement intelligent assistance system for use in software project planning.

In addition to the properties outlined above, an agent-orientated architecture is a natural choice to address the issue of heterogeneous client-server systems development. Recent research in Java-based agents [Watson, 97] [Caglayan and Harrison, 97] and mobile Java-based agents [Lange and Oshima, 97] has concluded that they are a viable technology on which to establish a platform independent agent-orientated architecture. To address the client-server issue, Orfali [Orfali and Harkey, 98] has successfully demonstrated the use of CORBA (Common Object Request Broker Architecture) as a basis for developing platform independent client-server systems, including agent-orientated systems.

To provide the necessary flexibility for the proposed system and to tackle the issues above, it is considered that both Java and CORBA provide an appropriate framework on which to base the system.

3.4 Summary

This chapter has described several commonly used approaches to intelligent assistance. They are contrasted to gain a better understanding of how to approach intelligent assistance in the domain of software project planning. Finally, a proposal for an agent-orientated hybrid approach to implementing an intelligent assistance tool for software project planners was presented.

Chapter 4 contains a review of the architectural aspects of intelligent systems and the trend towards distributed client-server platform independent systems. A number of intelligent assistant systems are investigated and the concepts of agent-based architectures for intelligent assistant presented.

CHAPTER 4 ARCHITECTURAL PERSPECTIVES

4.1 Introduction

This chapter describes architectures of intelligent assistant systems. A number of systems and research projects which fall under the heading of 'intelligent systems' and which are related to the concept of an intelligent assistant which has been proposed, are examined in conjunction with general aspects of software architecture. This chapter also describes an investigation into the trend towards distributed client-server platform-independent systems and its implications for the development of an architecture for the proposed intelligent assistant system.

4.2 Software Architectures

Recently, software architecture has emerged as an important field of study for software engineering researchers and practitioners [Bass et al., 98]. Software architecture can be defined as [Shaw and Garlan, 96];

> *"Structural issues concerning the organisation of a system as a composition of components; global control structures; the protocols for communication, synchronization and data access; the assignment of functionality to design elements; the composition of design elements; physical distribution; scaling and performance; dimensions of evolution; and selection among design alternatives".*

Abstractly, software architecture involves the description of elements from which systems are built, interactions among those elements, patterns that guide their composition, and constraints on these patterns.

Good architectural design has always been a major factor in determining the success of a software system. However, while there are many useful architectural paradigms (such as pipelines, layered systems, client-server), they are typically understood only in an pragmatic way and applied in a non standard fashion. Consequently, software systems designers have been unable to exploit commonalties in system architectures, make principled choices among design alternatives or specialize general paradigms to specific domains.

A sound basis for software architecture promises benefits for both development and maintenance. For development, it is increasingly clear that effective software engineers require proficiency in architectural software design. First, it is important to be able to recognise common paradigms, so that high level relationships among systems can be understood and so new systems can be built as variations of old systems. Second, getting the right architecture is often crucial to the success of a software systems design. Third, detailed understanding of software architectures allows the engineer to make principled choices among design alternatives. Fourth, an architectural system representation is often essential to the analysis and description of the high level properties of a complex system. Fifth, fluency in the use of notations for describing architectural paradigms allows the software designer to communicate new system designs to others.

One of the difficulties in working with software architecture is that different designers may interpret an architectural paradigm in different ways. For example, although two designers may both claim that their design is built around a client-server paradigm, they may mean quite different things by that term [Berson, 92]. A related problem is that several systems may be designed with similar architectural structures, but the designers never recognise that the similarities exist, consequently they overlook opportunities to capitalise on the experience of other designers.

Choosing the most appropriate architecture for a given situation remains an open problem [Bass et al., 98]. The rules of the style usually determine how to package components, for example, as procedures, objects, or filters. As a result, components cannot usually be interchanged across styles, for example code, may not be reusable because its interface makes it incompatible.

4.3 Intelligent System Architectures

Given the above discussion, it follows that in order to fully understand the architectural requirements for an intelligent assistant system, an examination of existing architectures must take place. Examining these architectures in conjunction with the characteristics of the problem domain will lead to an improved understanding of the nature of architectures for intelligent assistant systems and assist with the specification of an architecture for the proposed system.

In the following sections, three assistant systems will be analysed from an architectural perspective. These systems all operate in the area of management decision making and have diverse architectures which are based around the agent-orientated paradigm. For each system a brief description of its origin will be given, followed by an examination of its architecture and implementation as well as a discussion about its impact on the decision makers who use the system.

4.3.1 ADEPT

The ADEPT (Advanced Decision Environment for Process Tasks) project [Alty et al., 94] had as it main aim the implementation of an agent-based approach to managing business processes. The system was expected to:

- Allow decision makers access to relevant information wherever it is situated and request and obtain information from other departments within the organisation and outside the organisation.
- Provide timely and relevant information to decision makers which may not have been asked for.
- Inform decision makers of changes made elsewhere which impinge on current decision processes.
- Identify parties which may be interested in the outcome and results of the decision making activity.

The ADEPT system involves the transformation of some business process descriptions into a number of agencies, each with a connection to some common communication medium. An agency is recursively defined: an agency consists of a single responsible (or controlling) agent, a set of tasks the agent can perform and a possible set of sub-agencies. This relationship between sub-agency and responsible agent can be viewed as a type of social commitment and provides a mechanism for the encapsulation and abstraction of services. The responsible agent provides a specification of the services that it can and is willing to provide to its peers, even though some of the activity will require the assistance of its sub-agents.

Essentially each agent is able to perform one or more 'services', where a service is a unit of problem solving activity. The simplest service (called a task) represents an atomic unit of problem solving endeavor. These atomic units can be combined to form 'complex services' by addition of ordering constraints (e.g. two tasks can run in parallel, must run in parallel, or must run in sequence) and conditional control. The nesting of services can be arbitrarily complex and at the topmost level the entire business process can be viewed as a service.

Services are associated with one or more agents which are responsible for managing and executing them. Each service is managed by one agent, although it may involve execution of sub-services by a number of other agents. Since agents are autonomous, there are no control dependencies between them; therefore, if an agent requires a service which is managed by another agent, it cannot simply instruct it to start the service. Rather, the agents must come to an agreement about the terms and conditions under which the service will be performed (called a service level agreement).

In ADEPT, all agents have the same architecture (figure 4.1) which involves a 'responsible agent' that interacts with its peers and the 'subsidiary agencies' and tasks within its agency. An agents agency represents its domain problem solving resources. The responsible agent has a number of functional components concerned with each of its main activities - communication, service execution, situation assessment and interaction management.

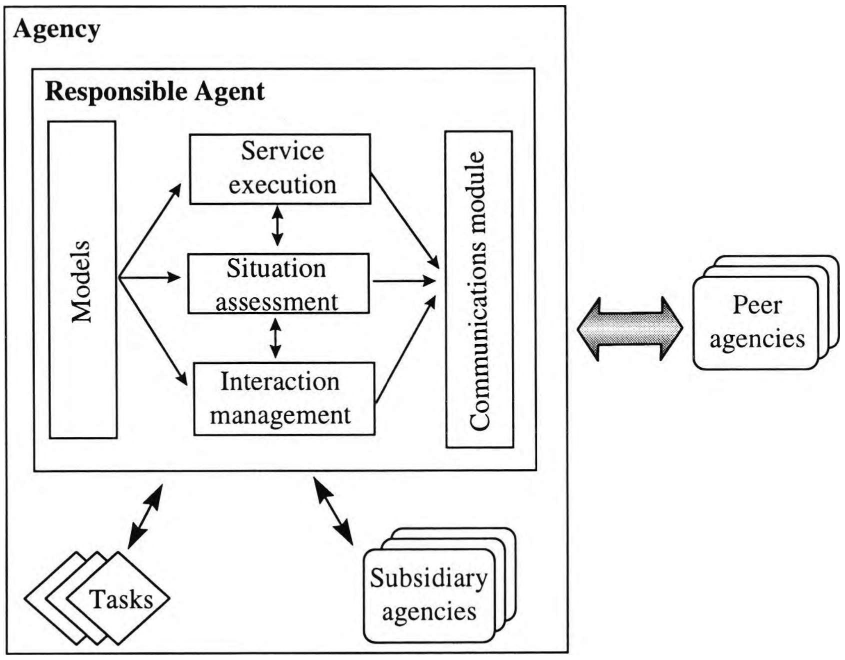


Figure 4.1 - ADEPT Agent Architecture

- The **communications module** routes messages between an agent and its agency and between peer agents.
- The **interaction management module** (IMM) provisions services through negotiation. It generates initial proposals, evaluates incoming proposals, produces counterproposals and finally accepts or rejects proposals. If a proposal is accepted then it generates a new SLA (service level agreement).
- The **situation assessment module** (SAM) is responsible for assessing and monitoring the agents ability to meet the SLAs it has and the potential SLAs which it may agree in the future. For example, if a service is delayed then the SAM may decide to reschedule it, to renegotiate its SLA, or to terminate it altogether.
- The **service execution module** is responsible for managing services throughout their execution. This involves execution management, information management and exception handling.
- The **models** are the primary storage site for SLAs to which agents are committed, descriptions of the services the agent can provide and generic domain information.

The agent knowledge base is represented as a set of strategies and a mechanism for selecting between them, which is implemented using a modified version of the CLIPS system. For agents to participate in the ADEPT environment, it is necessary for them to communicate using a common expressive language. This common language KIF [Genesereth and Fikes, 92], consists of a protocol and a syntax for expressing information and allows agents to interpret other agents intentions.

The overall ADEPT framework contains three basic components (or layers): The application layer is at the level of user interaction, where service descriptions and other information is gathered. The management layer contains all the agents and their sub-agents and the convergence layer presents a standard interface between the agent system and the underlying platform. The agents themselves are implemented on top of a CORBA compliant distributed platform.

The ADEPT system was tested by British Telecom, who used the system to support a number of concurrent business processes, such as the generation of customer quotations for designing a network to provide particular services to a customer. This would involve up to six parties: the sales department, the customer service division, the legal department, the design division, the surveyor department and the provider of an out-sourced service for vetting customers. For representing this business process, a full description of the process was translated into an agent system, where (at an abstract level) each agent represented a distinct department involved in the process.

ADEPT agents themselves do not carry out the totality of a business process; much of the work is ultimately carried out by humans or other software (often legacy) systems that are externally interfaced to ADEPT agents. ADEPT agents do need a certain amount of domain knowledge (or meta knowledge) and in most cases this domain knowledge can be of considerable size and complexity. However, this type of information can be extracted during a business process re-engineering exercise, which many efficient organisations conduct.

4.3.2 ARCHON

The ARCHON (ARchitecture for Cooperative Heterogeneous ON-line systems) project [Jennings et al., 96a] had as its aim the development of a general purpose architecture which would allow pre-existing expert systems, dealing with different aspects of decision making of a given complex environment, to cooperate in a mutually beneficial way. The main design feature of ARCHON was that it put an emphasis on loose coupling as a means to increased cooperation between a set of systems. Thus, systems that participate in mutual cooperation are conceived of as autonomous and capable of completing their allocated tasks without much reliance on other systems in the community, but benefiting from each others activities through cooperation mechanism.

ARCHONs design objectives [Wittig et al., 94] were for the interworking of semi-autonomous agents. It can complement integration architectures that provide for tight coupling of systems, such as client-server architectures in which a client would demand a service and a server is mandated to provide that service. ARCHON agents may enter into a client-server relationship with each other for a contracted set of tasks, but are never designated (pre-destined) to perform one or the other of those roles. ARCHON agents can pass unsolicited information to their acquaintances, leaving it to the recipient to decide what to do with it.

Figure 4.2 illustrates the modules which form the ARCHON architecture and shows the interface to the intelligent system.

- The architecture needs a communications facility, which is called the **High Level Communications Module** (HLCM). It is the ‘high level’ since it not only provides communications facilities, but also addressing and filtering. For example, if the domain system produces a result that may be relevant for other agents, the Planning and Coordination Module (PCM) just asks the HLCM to send it to all interested agents without specifying them.

- The **Agent Information Module** (AIM) provides an object orientated information management model and a query language to define and manipulate the information.
- The **Agent Acquaintance Models** (AAM) contain representations of other agents in the community in terms of their skill, interests, current status of workload, etc. Agents will not actually maintain models of all agents in the community, simply a subset based on similar interests/capabilities.
- The **Self Model** (SM) is an abstract representation of an agents domain system. It primarily contains information about the current state of this system i.e. its workload, or what tasks are being executed, but also embodies the precompiled plans (behaviors).
- The **Monitor** is responsible for the control of the intelligent system and for passing information to and from it.
- The **Planning and Coordination Module** (PCM) represents the main reflective part of ARCHON. If an exception occurs, it is the task of the PCM to reason about it and find a way out.

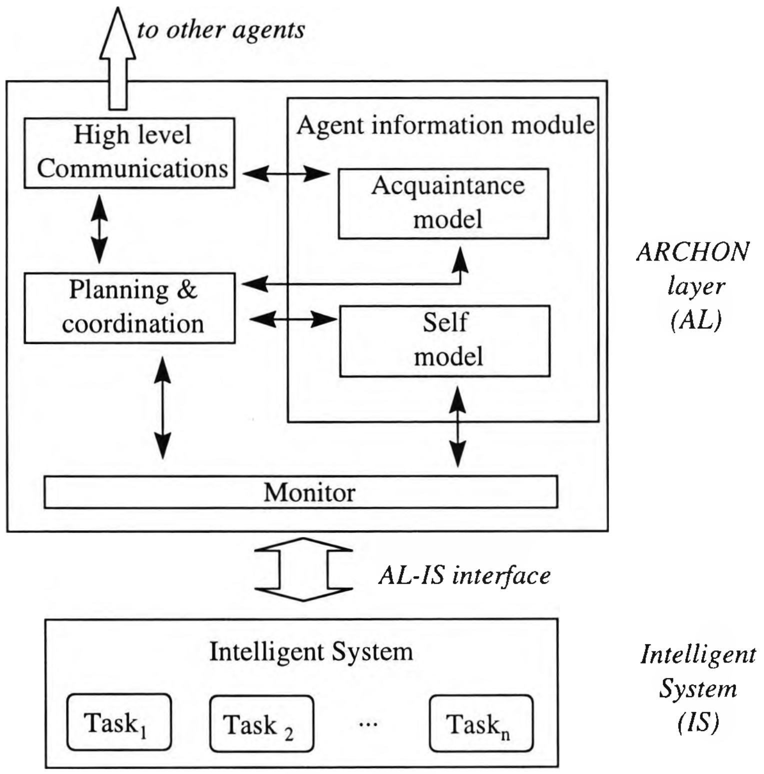


Figure 4.2 - ARCHON Agent Architecture

In an ARCHON agent community there is no centrally located authority, each agent controls its own IS (Intelligent System) and mediates its own interactions with other agents. The systems overall objectives are expressed in separate local goals of each community member. Because the agents goals are often inter-related, social interactions are required to meet the global constraints and to provide the necessary services and information. Such interactions are controlled by the agents AL (ARCHON Layer), for example: asking for information from other agents, requesting processing services from them, or volunteering information to other agents. Essentially an agents AL needs to control tasks within its local IS and decide when to interact with other agents.

The ARCHON architecture concentrates upon loose coupling of semi-autonomous agents. If an organisation has a collection of pre-existing systems, each dealing with a separate aspect of the same domain, then this architecture presents an opportunity for bringing these together into a useful co-operative framework. However, ARCHON was not only designed for pre-existing systems, but for providing cooperation between any set of semi-autonomous systems, which had not been adequately explored. Before this can be achieved, one must consider that ARCHON, on its own, restricts an integration approach to formulating a solution only in terms of loosely coupled, semi-autonomous agents.

The ARCHON project's principal test environment was in the domain of alarm analysis in the electricity supply network of Iberdrola in Bilbao, Spain. The increased automation and complexity of the automatic controllers has brought the electricity utility to the point where human intervention is scarcely needed, but whenever it does occur, the responsibility on the decision maker is even greater than ever before. This increase in automation also produced an increase in the amount, reliability and complexity of information received. In order to help human operators during the monitoring of the network at Iberdrola, several expert systems were developed over the years, such as; alarm analysis system and black-out identifier system - which were modelled using ARCHON agents. When one agent (expert system) identified a problem (e.g. alarm message caused by a fault), it could analyse the situation and (voluntarily) inform other agents which it considered needed to know. This inter-agent communication reduced the need for the operator to input information from one system to another, allowed other systems (agents) to have more timely information about a potential situation and cumulate more efficient and precise information communication to the human operator (decision maker).

Although several applications were evaluated using the ARCHON approach, all had some pre-existing systems. The project itself did not evaluate how the ARCHON approach might complement an existing client-server or distributed type integration environment. In addition, the ARCHON project has not been extended to systems with many (hundreds of) agents and it is likely that in such a situation the designer may need to consider if some of the smaller agents need to be coalesced again.

4.3.3 RISKMAN2

The RISKMAN2 project [Moynihan et al., 94] aimed to develop a critiquing system to support risk analysis for software development projects. This project built upon the lessons learned from Integrated Management Process Workbench project [Jenkins et al., 87], which developed the RISKMAN tool [Verbruggen et al., 89].

The goal of the RISKMAN2 project was to build a tool which would enable project managers 'walk around' a proposed project and help them anticipate any major risks to which the project might be exposed. The major inputs to the system are 'Risk Drivers', where each risk driver is seen as contributing in a linear, additive fashion to the risk measure (i.e. value) for one or more areas of risk management. The values of the risk drivers are elicited from the user (a project manager) by a Project Definition Tool. The major output of the tool is a risk report, which is split in two sections. The first section provides the computed risk measures, one for each of the risk management areas identified. The second section, consists of advice paragraphs, where each paragraph consists of text offering comments and advice to the user relating to the management of risk on the project. The user is provided with the ability to query the risk report to determine the basis upon which the risk measures and the advice paragraphs were arrived at.

The outline architecture of RISKMAN2 [Power, 94][Henry, 94] is illustrated in figure 4.3 and contains five major components: Generic Project Model (GPM), the software risk taxonomy, the Risk Analysis Daemon Library (a collection of risk analysis agents, each of which specialises in a different element of the risk taxonomy), the Blackboard and the Reporter. The user begins a session by instantiating the GPM for the project under consideration. Then the risk analysis daemons (agents) inspect the instantiated GPM, compute their results, and write these to the blackboard. In computing their results, daemons have access to results which other daemons may have written to the blackboard. When the daemons have completed their analysis, the user can invoke the reporter to retrieve the daemons results from the blackboard and display the results in different ways.

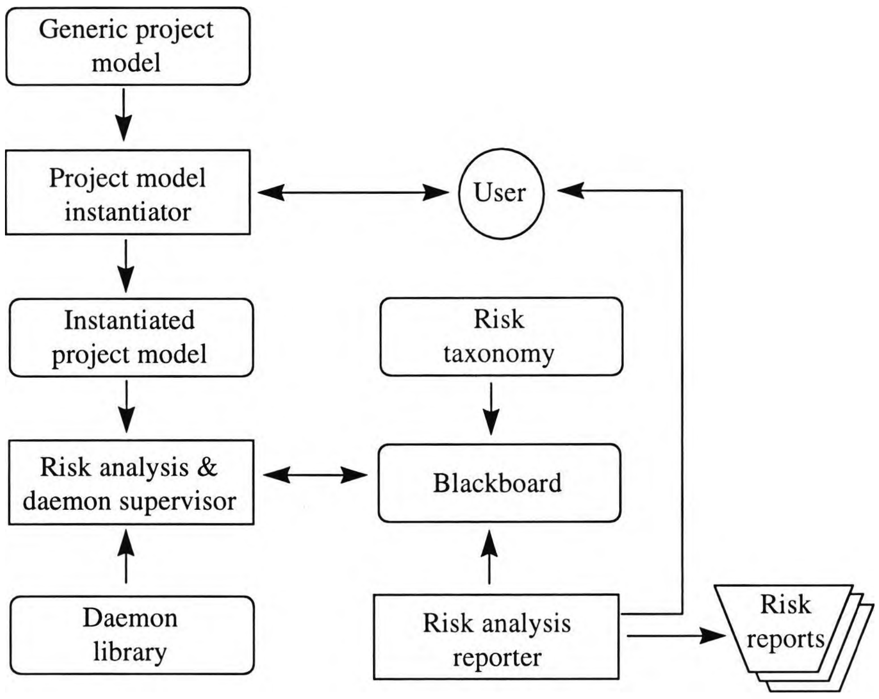


Figure 4.3 - RISKMAN2 Architecture

The RISKMAN2 components are:

- The **Generic Project Model** (GPM) is a very general, customisable model of a software project. The GPM takes the form of an object-model and uses the notation and semantics of the OMT modelling technique.

- The project **Risk Taxonomy** is a description of the types of risk to which a software development project can be exposed. For example, the risk of a product not meeting its performance requirement. The taxonomy takes the form of a classification in which risk types are broken down into sub-types and is based on the US AIR Force risk management taxonomy [USAF, 88].
- The Generic **Project Model Instantiator** (PMI) through dialogue with the user, customises the GPM to produce a model of the particular project under consideration. The PMI is driven by the GPM in the sense that the PMIs goal is to instantiate as many of its elements as possible. The PMI does this by asking the project manager a series of multiple-choice questions, to get values for the GPM elements.
- The **Risk Analysis Daemons** are mini-experts in some aspect of risk management, one for each area in the risk taxonomy. Each daemon (or agent) is structured as a set of small rule-based production systems, with variables in the conditional rule-set corresponding to elements in the GPM.
- The **Daemon Library** acts as a storage point for a collection of daemons.
- The RISKMAN2 **Blackboard** is a generic blackboard. When a daemon has performed its risk calculations, it must write its results to the appropriate place on the blackboard. In this way every daemon can access the results of any previously executed daemon.
- The function of the **Risk Analyser Reporter** is to deliver a structured report to the user. It does this by accessing the conclusions written to the blackboard by the daemons.

All daemons are controlled by a 'daemon supervisor'. A daemons begins its task by attempting to assign values to the variable in the conditional part of its rule-set. If it finds the data it needs, it proceeds with its analysis (executes its production rules). If it cannot find all the data it needs, the daemon terminates and returns 'failure' to the supervisor, in which case the supervisor will schedule that daemon to execute at a later time. Eventually, assuming data becomes available, the daemon writes its conclusions to the blackboard, from which they can be read by other daemons and the risk reporter. A daemons conclusions are in three parts: A **risk metric** which is a measure representing the degree of risk which the daemon considers is present in the project; **why text** which is hard-wired into the daemon and explains how the daemon worked and reached its conclusions; and **advice text** which includes suggestions as to how the particular risk may be reduced or otherwise managed.

The RISKMAN2 architecture represents an open and flexible architecture which facilitates, over time, the ability to improve the system by adding extra daemons, enhancing the GPM, adding or replacing the risk taxonomy and the blackboard approach allows for a degree of modularity and dynamic control. However, a number of weaknesses exist in relation to the daemons; The daemons are organised like an army in the sense that each daemon must write to a single place on the blackboard and 'higher up' daemons must only take as their input the output of lower daemons in the taxonomy. Also, no mechanism exists for handling disagreement or conflict between daemons, or to ensure the system can still function if the user does not fully instantiate the GPM. In addition, the variables in the conditional part of a daemons rule-set are constrained to be elements of the GPM - in other words, a daemon may only view the project through the lense of a GPM. This excludes the possibility of building daemons which 'see' projects in a richer or different way.

4.3.4 Review of Architectures

In this section, the three different architectural approaches to implementing intelligent assistance systems for management decision making are contrasted to assess their comparative strengths and weaknesses in order to appraise the suitability of agent-based architectures to support management decision making and the possibility of adapting these approach to the domain of software project planning.

The ADEPT system encapsulates business processes in agents, where each agent can provide a 'service' (or task) to other agents and enters into 'service agreements' with other agents to provide a service to them. In the ADEPT hierarchy, agents at a high level may only enter into such service agreements with their peer agents, although these agents usually require the services of subordinate agents (those lower in the hierarchy) to provide such services. Such service provisioning requires complex inter-agent communication, as well as the ability for service negotiation and a knowledge of the agent hierarchy. Therefore each agent has additional structures to cope with such communications, which represents an implementation overhead for each agent in addition to the overall system overhead of continuous complex inter-agent communication traffic. Further, all knowledge in the ADEPT system is embedded in 'models' within its agents, which does not allow for the dynamic updating of the ADEPT knowledge base - which may only be achieved by manually replacing agents. A positive aspect of the ADEPT implementation is the support for multiple platforms. Agents are implemented on top of a 'convergence layer' which represents a communications interface to the operating system. In the case of ADEPT, the CORBA broker DAIS was used in implementation however further expansion was envisaged (although not tested) using OLE (Object Link and Embedding - now ActiveX).

It is worth mentioning that members of the ADEPT project consortium considered the project successful and provided a useful step forward in investigating agent-based support systems [Alty, 97]. However, they recognized that the implementation of the user-orientated section of the tool (application layer), through which the user interacted with the system, was poorly constructed and diminished the end users productivity.

While there are many interesting architectural lessons to be learned from the ADEPT approach, it does not provide a suitable basis upon which to implement an intelligent assistant system for software project planning. The ADEPT approach to intelligent assistance is closely related to software process enaction environments [Finkelstein et al., 94] as the system simply automated existing (business) processes. ADEPT agents are not adaptable to a situation where there is no pre-defined process, nor can they provide a facility to define additional processes (such as a software project plan). In addition, much of the knowledge required by ADEPT agents is restricted to low-level design guidelines and ADEPT agents have an inherent lack of expressive power, with explicit support only for top-down design processes [Jennings et al., 96b].

The ARCHON system essentially provides an agent-based shell or wrapper within which a pre-existing expert system may operate and exchange information with other subsystems via an inter-agent communication mechanism. Unlike ADEPT agents, ARCHON agents do not themselves directly contain any problem solving data, but contain a model of other agents in the system and the ability to communicate with them. Like ADEPT, all ARCHON agents must have structures to cope with complex communications, which represents an implementation overhead for each agent, in addition to the overall system overhead of continuous complex inter-agent communication traffic. It should be noted that in this agent community there is no centrally located authority - the system is simply made up of a community of loosely coupled agents, each of which represents a subsystem capable of completing some task. Further, the ARCHON approach was designed only with a small number of pre-existing systems in mind. To expand the ARCHON system requires the addition of new agents, however, these agents represent pre-existing subsystems which must be in place before the agent can be deployed - thus adding an extra layer of complexity and inhibiting the possibility of dynamically updating the system. Further, the ARCHON, unlike ADEPT implementation does not provide any support for multiple platforms as it is tied directly to the underlying operating system.

The ARCHON system does not directly provide a suitable framework for a software project planning intelligent assistant system. Overall, ARCHON can be characterised as approximating the functionality of a DSS, as it provides for a number of pre-existing systems to operate in a more timely manner and provide the user with more accurate and timely information upon which to base a decision. ARCHON agents themselves do not conduct any problem solving or interact with the user to solve their problems. ARCHON agents were designed to interact with other ARCHON agents and their associated intelligent systems and, as such, are not readily adaptable to directly represent an area of problem solving activity for a given domain (such as software project planning). However, the ARCHON approach could be used to co-ordinate multiple (discrete) intelligent systems for software project management where ARCHON agents represent a software project planning system, software project risk analysis system, etc.

The RISKMAN2 approach provides a framework in which a community of agents can operate and exchange information. Unlike ADEPT or ARCHON agents, RISKMAN2 agents use a simple form of communication, that of a blackboard structure, and in addition have a controlling entity which directs the agent community, thus reducing the overheads associated with inter-agent communication. In addition, this approach allows agents to be pro-active, in that all agents can access the blackboard and may respond in an opportunistic manner, unlike ADEPT or ARCHON agents which rely on explicit message passing. Another advantage the RISKMAN2 approach has over the others is the ability to dynamically update the knowledge base via the 'Daemon Writer Kit' [Power, 94], which provides a template for the construction of agents. As every agent is autonomous and fully self-contained, any agent may be removed or updated without affecting the rest of the system. However the RISKMAN2 approach also has a number of problems: Agents are only capable of viewing a project through a predefined project model (the GPM) and agents may only take input from agents which are lower in the hierarchy than themselves. Like ARCHON, the RISKMAN2 implementation does not provide support for multiple platforms and is tied directly to the underlying operating system. Further, RISKMAN2 suffered from a number of implementation problems and, unlike ADEPT and ARCHON, the end system was not sufficiently tested in real world situations.

The RISKMAN2 approach has potential to be of use in implementing an intelligent assistant system for software project planning. Unlike both ADEPT and ARCHON, the RISKMAN2 system was not simply a DSS. It also incorporated features of both an expert system - in that it could diagnose the existence of a given situation (e.g. high level of risk in a certain area), and an expert critiquing system - in that it could provide advice on how to deal with that situation (e.g. advice on mitigating an identified risk in a particular area). However, the GPM view of the users project was very restrictive as was the hierarchical approach to agent communications. The RISKMAN2 system was designed with a narrow view of a software project in mind - that of risk identification and associated advice generation. Therefore the structures of the Risk Taxonomy, GPM, Agent and Blackboard are not readily extendible.

4.4 Desirable Architectural Characteristics

The architectural approaches discussed above have demonstrated the suitability of an agent-based approach to the support of decision making. There are a number of interesting lessons to be learned from the above discussion. Specifically the list below comprises the desirable architectural characteristics which should be provided by candidate architectures for an intelligent assistant system for the domain of software project planning.

1. **Specialism** - A multi-agent approach - that of having a community of cooperating agents - provides flexibility in the design of a system. Each agent can be responsible for, or, an expert in, one particular area.
2. **Information exchange** - The multi-agent approach also provides for enhanced information sharing, as agents within the community will exchange information and volunteer information to other interested agents.
3. **Collaboration** - To fully support a multi-agent community, a complex inter-agent communication mechanism must be in place. This represents an overhead in both complexity and system performance.
4. **Blackboard** - The blackboard approach to agent communication reduces the level of complexity and volume of communications traffic.
5. **Supervision** - In an agent community, there does not have to be a single controlling agent. However, the indications are that the existence of such a supervisory agent leads to more organised communications in the community.
6. **Hierarchy** - It seems natural to develop a hierarchy when modelling a multi-agent community. However, the placing of limitations on the operation of agents according to rank within such a hierarchy appears to diminish flexibility, therefore agents should be allowed to operate regardless of rank.
7. **Knowledge evolution** - Using an agent approach to modelling a knowledge base provides the ability to dynamically update the knowledge base by adding, updating and deleting agents. However, the exact internal

architecture of the agents is a factor in this flexibility and as such, they must be designed with this in mind.

8. **Data separation** - The separation of data and agents leads to a more open and flexible system. This allows alterations in the data storage mechanisms and the agents themselves to be carried out independently of each other.
9. **Platform independence** - Agents provide an ideal basis on which to develop a platform independent system. ADEPT demonstrated that agents can be successfully implemented in a CORBA distributed environment.
10. **Multiple inference strategies** - If agents in the community are autonomous, they may be implemented in disparate manners and therefore each agent may be implemented in the most suitable fashion for their given purpose.
11. **Component separation** - In separating the agents (which represent the 'intelligence' of the system) from the user-orientated framework (which represents the 'application') the result is a flexible architecture, where either application or intelligent components may be altered independently.
12. **Multiple paradigms** - An agent approach allows the development of a number of different supporting frameworks such as decision support or expert systems. It may also be possible to develop a hybrid framework within which a community of agents could function.

The remainder of this chapter will be devoted to a discussion of the trend towards distributed client-server platform-independent systems and its implications for the development of an architecture for the proposed intelligent assistant system. However, the above characteristics will be further considered in chapter 6 in terms of the proposed architecture.

4.5 Architectural Trends

With the increase in globalization, distributed information systems are rapidly becoming the norm. As the need to both compete and cooperate using information systems becomes more clear, system designers are becoming increasingly aware of issues associated with distributed information systems. These systems cross geographical and often organisational boundaries and are typically broadly heterogeneous in both hardware and software terms. Indeed it has been remarked [Orfali et al., 99] that the IT industry stands at a new threshold brought on by; 1) the exponential increase of low-cost bandwidth on Wide Area Networks (WAN), such as the Internet; and 2) a new generation of network-enabled, multithreaded desktop operating systems, such as Windows NT. This new threshold may mark the beginning of a transition from small (LAN-based) client-server systems, to larger (WAN-based) client-server system that will result in the irrelevance of proximity.

This trend in global distributed information systems and the associated issues of project management is not new. Indeed it has been investigated by projects such as GOAL [Goal, 95], an ESPRIT funded project which addressed the definition of a tool set supporting multi-organisational project management. However, this trend has been intensified by the technological advancements outlined above.

The design and implementation of software is a difficult and expensive activity even where this can be done on a single stable platform using a single operating system and a single programming language. A considerable amount of the software being written now faces additional complexities:

- It must run on a network of machines with the overall functionality distributed among those machines.
- The machines on the system may run different operating systems.
- The components of the system must be integrated easily into new systems, perhaps systems that are not yet planned.
- Legacy systems must be integrated into the system, either to allow the new software to access legacy data or to request the legacy code to carry out some processing.
- It may be necessary to use different programming languages for the components, perhaps because of the use of legacy systems or because a particular programming language is a good choice for some subset of the components.

There are a number of candidate solution technologies for addressing the concerns above, which include; CORBA, DCOM, RMI and sockets. Of these, the two strongest candidates are OMG's CORBA and Microsoft's DCOM, on the basis of both industry popularity and the power of the organisation(s) behind them. However, it is the opinion of many industry observers that CORBA provides a level of openness, flexibility, and industry standardization that brings it ahead of its competitors and as such provides an ideal basis for addressing the needs outlined above in the context of this research. For an in-depth discussion of these technologies the reader is directed to [Orfali and Harkey, 98], which provides a detailed comparison and critique.

The following sections provide an introduction to the technologies of CORBA and Java. CORBA is investigated as a potential candidate to cater for the needs of distributed client-server systems and Java is introduced as an ideal partner to CORBA to provide the extended flexibility of platform independence.

4.5.1 CORBA

CORBA [OMG, 96] (Common Object Request Broker Architecture) has two aims; Firstly, it makes it easier to implement new applications that must place components on different hosts on a network or use different programming languages. Secondly, it encourages the writing of open applications, ones that can be used as components of larger systems. The ORB (Object Request Broker) is the middleware that establishes the client-server relationships between objects. Using an ORB, a client can transparently invoke a method on a server object, which can be on the same machine or across a network. The ORB intercepts the call and is responsible for finding an object that can implement the request, pass it the parameters, invoke its method, and return the results. The client does not have to be aware of where the object is located, its programming language, its operating system, or any other system aspects that are not part of an object's interface. In so doing, the ORB provides interoperability between applications on different machines in heterogeneous distributed environments and seamlessly interconnects multiple object systems.

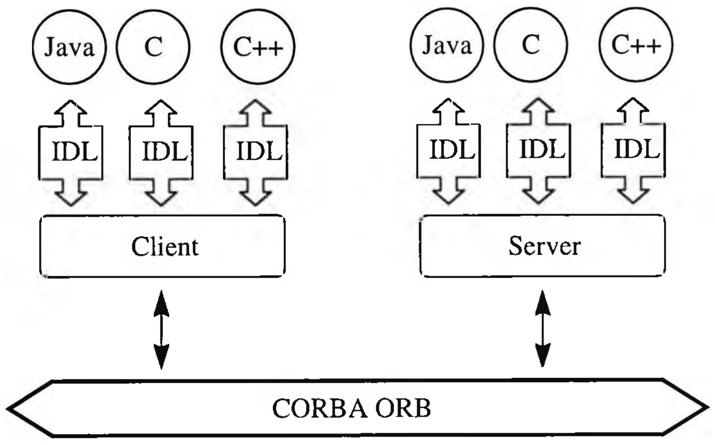


Figure 4.4 - CORBA Interoperability

Within the CORBA framework, all objects are written in a neutral Interface Definition Language (IDL) that defines a components boundaries - that is, the services (or methods) the object provides to potential clients. As IDL is completely language independent, objects specified using IDL are portable across languages, tools, operating systems and networks. IDL is used to specify a components attributes, the parent classes it inherits from, the exceptions it raises, the methods it supports - including the input and output parameters and their datatypes. IDL specified interfaces can be written in any programming language for which there exists a CORBA binding. Essentially IDL allows client and server objects written in different languages to interoperate, as illustrated in figure 4.4.

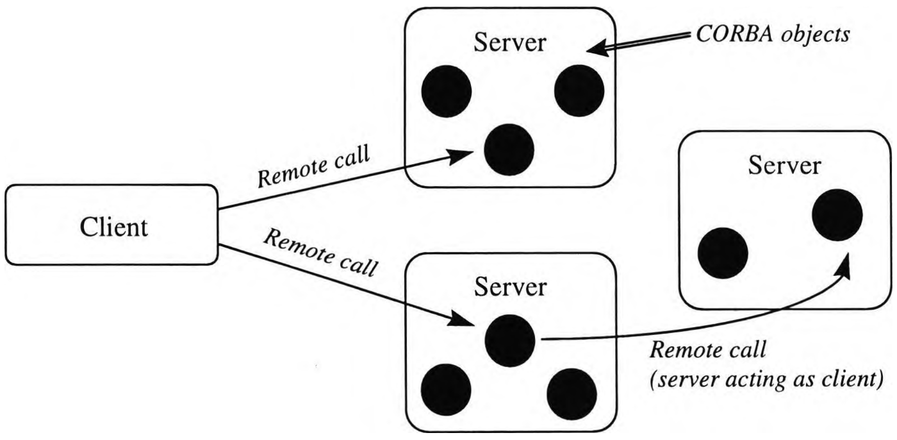


Figure 4.5 - CORBA Client-Server Relationship

Even though CORBA uses the term *client* and *server*, this does not mean that the system must have a star-shaped architecture, where a set of client machines uses a single server. Instead, the objects in one server may use the objects in other servers. This is very useful when decomposing a system into components, because it allows a client to invoke an object in a server, and for that object to invoke on others in order to fulfill the clients request. A server that makes a call to an object remote to it (in another server, on the same or a different machine) is acting as a client for the duration of that call. This relationship is illustrated in figure 4.5

4.5.2 The Java Language

The Java programming language was developed by Sun Microsystems in 1990 and since then has created a large amount of hype and discussion in the computer industry. The purpose of the following sections is to show that Java is a platform independent language which is suitable for implementing CORBA based agent applications.

Sun Microsystems define Java as [Gosling and McGilton, 96];

> *"A simple, object-orientated, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded and dynamic language".*

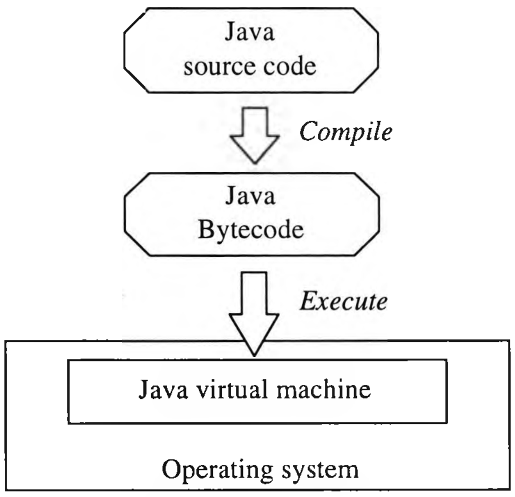


Figure 4.6 - Java Virtual Machine Architecture

Much of the excitement about Java comes from the self-contained, virtual machine environment in which Java applications run. Java compilers generate machine independent bytecode - an architecture neutral intermediate format designed to transport code efficiently to multiple hardware and software platforms. The interpreted nature of Java solves both the binary distribution problem and the version problem; the same Java language byte codes will run on any platform. These bytecodes are then interpreted by a Java Virtual Machine (JVM) - a layer of software on a machine, which takes the Java bytecodes and executes them. This results in code that executes the same no matter what its underlying architecture is. Thus Java bytecodes can be shipped over the net and are guaranteed to function the same on all platforms. This relationship is illustrated in figure 4.6.

4.5.3 Java as an Agent Language

Since Java is a mobile code system it is considered by many as an ideal language for the implementation of mobile systems and mobile agents [Srinivas et al., 97]. Mobile code refers to the ability to transfer executable binaries to wherever they are needed to be executed. Though the basic Java support for bytecode migration implies Java code mobility this capability is not completely viable as all Java objects reside on a single host and Java lacks mechanisms for transmitting arguments from one host to another. In contrast, the fundamental premise of CORBA, is that an object on one host can invoke a method of an object on another host, with CORBA employing a referencing model to avoid the issues of object migration.

In distributed systems, the idea of an agent is seen as a natural metaphor, and by some as a development of the concurrent object programming paradigm [Agha et al., 93]. Much of the attention that currently surrounds agent-based technology is related to the phenomenal growth of the Internet. In particular there is great interest in 'mobile agents', that can move around a (local or wide area) network of machines on a users behalf. Many well publicised mobile agents fall into the category of Internet search and filtering agents such as BarginFinder, ShopBot and CyberYenta [Lesnick and Moore, 97].

Recently a large number of languages for agent-orientated systems have been proposed by both the research community and commercial organisations [Muller et al., 99]. Languages such as General Magic's 'Telescript' [White, 94] - which is closely related to the functionality provided by languages such as Java - have failed to gain widespread acceptance. Due to its unique characteristics and overwhelming industrial support, Java is seen by many as providing the ideal candidate for implementing agent-based frameworks and also agents themselves.

4.5.4 Java as a CORBA Object Language

Java language bindings for IDL provide an application programmer with CORBAs high-level distributed object paradigm. With this in mind, CORBA can be said to bring a number of benefits to Java, as its extends Java with a distributed object infrastructure [Orfali and Harkey, 98]:

- **CORBA provides a scalable server-to-server infrastructure** - Pools of server objects can communicate using the CORBA ORB, and these objects can run on multiple servers, thus providing load-balancing for incoming client requests. The ORB can dispatch the request to the first available object and add more objects as demand increases.
- **CORBA extends Java with a distributed object infrastructure** - CORBA allows Java to communicate with other objects written in different languages across address spaces and networks. In addition, CORBA provides a rich set of distributed object services that augment Java - including transactions, security, trader and persistence.

The Java infrastructure starts where CORBA ends. CORBA provides a distributed object infrastructure that lets applications extend their reach across networks, languages and operating systems. Java provides a portable object infrastructure that works on every major operating system. CORBA deals with network transparency, while Java deals with implementation transparency. Java offers a number of attractions for implementing CORBA objects:

- **Java allows CORBA to move code around** - Java's mobile code facility allows code to dynamically move across the CORBA infrastructure to where it is needed, thus allowing clients and servers to dynamically gain behavior.
- **Java simplifies code distribution in CORBA systems** - Java code can be deployed and managed centrally from the server. This means that code on the server is updated once and clients can receive it when they need it.
- **Java complements CORBA's agenting infrastructure** - CORBA defines a framework for distributed objects which lets 'roaming objects' move from node to node. Java bytecodes are ideal for shipping such behavior around.

Orfali [Orfali and Harkey, 98] states that "*Java is almost certainly the ideal language for writing both client and server objects*". Java's built-in multi-threading, garbage collection and error management makes it easier to write robust network objects. Also, Java's object model complements CORBA's, as they both use the concept of interfaces to separate an objects definition from its implementation.

4.6 Summary

This chapter has presented the architectural aspects of intelligent systems and discussed the trend towards distributed client-server platform independent systems. A number of intelligent assistant systems were investigated and the concept of agent-based architectures for intelligent assistance presented. In addition, CORBA and Java were presented as a solution technology to the issues previously identified.

Chapter 5 will outline the issues surrounding the knowledge base which forms the basis of the intelligent agents for the proposed system. There will also be an examination of the selection of a suitable method for acquiring and representing knowledge.

CHAPTER 5 KNOWLEDGE BASE ISSUES

5.1 Introduction

This chapter describes the issues surrounding the knowledge base, which forms the foundation for the intelligent agents for the proposed system. The subject areas of knowledge engineering and knowledge representation are introduced, and the issues of choosing a suitable method for acquiring knowledge as well as the choice of agent implementation language are investigated.

5.2 Knowledge Engineering

The term Knowledge Engineering is now commonly used to describe the process of Knowledge Based System (KBS) development. Knowledge engineering includes [Smith, 96]:

- Acquiring from experts the knowledge that is to be used in the system.
- Choosing an appropriate method for representing the knowledge
- Implementation in an appropriate computer language.

Just as with traditional computer systems development, the notion of a systems development lifecycle has been applied to the development of KBS's. The following are the main activities in the KBS development lifecycle [Bochsler, 88]:

- **Planning** - produces a formal workplan for the KBS development.
- **Knowledge definition** - defines the knowledge requirements of the KBS.
- **Knowledge design** - produces a detailed design for the KBS.
- **Code** - produces the actual code implementation of the KBS.
- **Knowledge verification** - determines the correctness, completeness and consistency of the system.

A number of KBS development methodologies have evolved such as - GEMINI (General Expert systems Methodology INItiative) [Montgomery, 88] RUDE [Bander et al., 88] and KADS (Knowledge Acquisition and Design Process) [Schreiber et al., 93]. However surveys [Smith, 96] [DTI, 92] indicate that a prototyping approach is most popular among KBS developers - perhaps because KBS development relies heavily on the involvement of the systems users (those with the appropriate expertise). In addition, prototyping is well suited to the types of problems often encountered in KBS development, where the initial requirements are typically ill-defined and the problem itself often ill-structured. The concept of intelligent agents provides an ideal basis on which to pursue a prototyping approach to the development of an intelligent assistant system. Individual agents can be rapidly prototyped, as each agent can be relatively small and in addition be an expert in just one small area of the problem domain, thus allowing a basic agent to be developed quickly and used in the testing and exploration process discussed above.

5.3 Knowledge Representation Systems

Knowledge representation is the formalising and storing of knowledge in some suitable structure to allow for subsequent computer processing. To represent knowledge in a meaningful way it is important to relate facts in a formal representation scheme to facts in the real world. The formal representation will be manipulated using a computer program, with new facts concluded. Therefore the main issue in knowledge representation is to find a suitable knowledge representation language in which domains of knowledge can be described.

A knowledge representation language should allow complex facts to be represented in a clear and precise way, and in a way that easily allows the deduction of new facts from existing knowledge. The main requirements of a knowledge representation language are as follows [Ringland and Duce, 88] [Cawsey, 98]:

- **Representation adequacy** - It should allow the representation of all the knowledge that is needed to reason with.

- **Inferential adequacy** - It should allow new knowledge to be inferred from a basic set of facts.
- **Inferential efficiency** - Inferences should be made efficiently.
- **Clear syntax and semantics** - It should be known what the allowable expressions of the language are and what they mean.
- **Naturalness** - The language should be natural and easy to use (in a linguistic sense).

However, no one representation language satisfies all these requirements (perfectly). In practice the choice of representation language depends on the reasoning task - just as the choice of programming language depends on the problem. Given a particular task it will generally be necessary to choose an appropriate language for the particular requirements of the application.

The remainder of this section will briefly compare three commonly used knowledge representation schemes - Production Rules, Semantic Nets and Predicate Logic - in order to give an appreciation of the types of schemes that may be employed. Section 5.4 will then outline the main agent languages that can be used to implement knowledge representation in an agent-orientated framework.

A classical way to represent human knowledge is the use of production rules, where the satisfaction of the rule antecedents gives rise to the execution of the consequence [Ringland and Duce, 88]. Production systems represent knowledge in terms of a set of IF-THEN rules, a set of facts normally representing things that are currently held to be true, and some interpreter (inference engine) controlling the applications of the rules, given the facts. Production systems exhibit useful modularity, in that rules are independent of each other and of the rest of the system, as each rule encodes a ‘chunk’ of independent domain knowledge. Further, the straightforward if-then form of a rule often maps well into English for the purposes of explanation. Due to the independence of the rules from each other and from the control strategy, it is very difficult to rigorously determine the properties of the systems behavior by static analysis. Further, rules have no intrinsic structure, which makes management of large knowledge bases difficult.

Semantic Networks were first developed as a way of representing human memory and language understanding [Quillian, 68], but since then have been applied to knowledge representation. In a semantic net, knowledge is represented as a graph, where the nodes represent objects and the links represent relations between objects, where an object can be any physical item such as a book or a person. Nodes can also be used to represent concepts, events or actions. The nodes in a semantic net are interconnected by links or arcs which show the relationships between objects. Semantic nets allow knowledge to be represented about objects and relationships between objects in a simple and fairly intuitive way, with a graph notation that allows for easy knowledge organisation. However, semantic nets have certain limitations, such as the lack of link and node name standards and the combinatorial explosion of searching nodes.

Prolog is a programming language that is used for solving problems that involve objects and the relationships between objects [Clocksin and Mellish, 81] and as such is considered to be a highly suitable implementation language for semantic nets. Prolog is a declarative programming language that allows the programmer to express 'what' a problem is, without having to worry about 'how' a solution to that problem is to be obtained. Essentially a Prolog program takes the form of a database of knowledge about a particular problem domain, with a facility to query that database. Essentially Prolog can be thought of as a 'question and answer' environment which is continually waiting to answer queries about knowledge in its database. Prolog (and predicate logic) provides a powerful mechanism to represent and reason with knowledge. However, some things are difficult to represent using Prolog, particularly facts that involve uncertainty and beliefs.

5.4 Agent Representation Languages

Various research projects have investigated languages for implementing intelligent agents in recent years [Muller et al., 99]. In the early stages, agents were local to individual projects and their languages were mostly idiosyncratic. As a result there are a large number of representation languages, each with their own particular characteristics, which do not have inter-agent communication capabilities. An obvious solution is to have a lingua franca, where ideally all agents that implement the same lingua franca would be mutually intelligible [Huhns and Sing, 97]. However, the agent community is still a long way from attaining this goal.

The following sections discuss three commonly used agent languages that can be used to implement knowledge representation in an agent-orientated framework, in order to give an appreciation of the types of languages that may be employed.

5.4.1 KQML

KQML (Knowledge Query and Manipulation Language) [Finin et al., 97] is a language and protocol for exchanging information and knowledge. It is part of a larger effort, the ARPA Knowledge Sharing Effort [Bradshaw, 97] which is aimed at developing techniques and methodology for building large-scale knowledge bases which are shareable and reusable. KQML is both a message format and a message-handling protocol to support run-time knowledge sharing among agents. KQML can be used as a language for an application program to interact with an intelligent system, or for two or more intelligent systems to share knowledge in support of cooperative problem solving.

Communication takes place on several levels. The content of the message is only part of the communication. Locating and engaging the attention of another agent with which an agent wishes to communicate is part of the process, packaging a message in a way that makes it clear the purpose of an agents communication is another. KQML assumes the message transport is reliable and preserves the order of messages, but does not guarantee delivery times. For this reason, the underlying paradigm of communication is asynchronous. At the application level, synchronous communication is achieved by tagging messages to relate to each other. For example, responses to queries may be linked. In this way, KQML supports some elementary interaction protocols, although more sophisticated protocols must be built externally to KQML.

When using KQML, a software agent transmits content messages, composed in a language of its own choice, wrapped inside a KQML message. The syntax of KQML is based on a balanced-parenthesis list. The initial element of the list is the performative; the remaining elements are the performatives arguments as keyword/value pairs. For example, a message representing a query about the price of a compiler might be encoded as follows:

```
(ask-one
    : content (PRICE VisualJava ?price)
    : receiver query-server
    : language standard _prolog )
```

In this message, the KQML performative is "ask-one", the content is "(PRICE VisualJava ?price)", the receiver of the message is to be a server identified as "query-server" and the query is written in standard form of Prolog.

KQML still suffers from poorly defined semantics and as a result many KQML implementations seem unique. [Cohen and Levesque, 97] discuss difficulties with the current specification of KQML. In particular they have identified ambiguity and vagueness in standard performatives, misidentified performatives and missing performatives. In addition, the issues of security and authentication are only beginning to be addressed by the KQML community.

KQML has been successfully used to implement a variety of information systems using different software architectures, and include projects such as: Magenta (Stanford University Logic Group), KATS (Lockheed Corporation) [Mayfield et al., 95] and COBALT [Benech and Desprats, 97] (Toulouse University and HP - an agent communication toolkit based on KQML and CORBA. COBALT is coded in both Java and C++ languages and uses HP ORBPlus / JORBPlus ORBs.

5.4.2 Telescript / Odyssey

Telescript [White, 1994] is an object-oriented mobile agent language developed by General Magic Inc. and is arguably the first commercial agent language. Telescript technology has been overtaken by the commercial success of the Internet in general and Java. This, coupled with Telescript being a proprietary language, lead General Magic to reimplement the paradigm in Java and launch a new technology, Odyssey [White et al., 97]. This system effectively implements the Telescript concepts in the form of Java classes.

There are two key concepts in Telescript / Odyssey: Places and Agents. Places are virtual locations that are occupied by Agents. Agents are providers and consumers of goods in the electronic market place. Agents are software processes and are mobile: they are able to move from one place to another, in which case their program and state are encoded and transmitted across a network to another place where execution recommences. Agents are able to communicate with one another - if they occupy different places then they can connect across a network in much the standard way. If they occupy the same location, then they can meet one another.

Four components have been developed to support Telescript / Odyssey. Firstly the Telescript language which is designed for carrying out communication tasks. Secondly the Telescript engine acts as an interpreter for the Telescript language, maintains places, schedules agent execution and provides an interface with other applications. The third component is the protocols set, which deals primarily with the encoding and decoding of agents, to support transport between places. The final component is a set of software tools (and API's) to support the development of Telescript / Odyssey applications.

Although Telescript / Odyssey is technically a very sophisticated mobile agent system it is also costly. The engine requires significant computer resources to run and financially it is an expensive item of software which inhibits its acceptance as a general mobile agent system among nearly all mobile agent developers.

5.4.3 JESS / CLIPS

CLIPS (C Language Integrated Production System) [Giarratano and Riley, 94] is a multiparadigm rule based programming language, based on the OPS5 production rule language [Brownston et al., 85]. CLIPS was originally developed by NASA in 1984 for internal use, but after modifications, version 3 of CLIPS was made available to groups outside of NASA in 1986. Since then CLIPS has undergone continuous improvement. Version 5 was released in 1991, introducing two new programming paradigms: procedural programming and object-oriented programming. Version 6.0, released in the Spring of 1993, added fully integrated object/rule pattern matching and support features for rule based software engineering. Version 6.1 was released in 1998, adding C++ compatibility and functions for profiling performance. Because of its portability, extensibility, capabilities and low-cost, CLIPS has received widespread acceptance throughout industry and academia and has a user base in excess of 5,000 developers.

JESS (Java Expert System Shell) [Friedman-Hill, 99] is a clone of the core of the CLIPS expert system shell. JESS was originally developed by Sandia National Laboratories and contains the main features of CLIPS and is downward compatible with CLIPS, in that every valid JESS script is a valid CLIPS script. The primary representation methodology in both CLIPS and JESS is a forward chaining production rule language based on the Rete algorithm [Forgy, 82].

CLIPS / JESS production systems consist of both conditional statements known as rules stored in production memory and a global database known as working memory. When every condition in a rule is satisfied by matching data in working memory, the rule is placed in the conflict set. Conflict resolution is performed on the conflict set to determine which actions in the production rules are eligible to be executed. CLIPS uses a LISP-style syntax where expressions are enclosed in parentheses and use a prefix functional form. The following is an excerpt from a deftemplate statement which is used to define a working memory structure and a simplified example of a rule to print all Java programmers.

```
( deftemplate employee
        ( slot last_name )
        ( slot job_description )
        ( slot favoured_language
                ( default Java ) ) )

(  defrule print_programmer
        ( employee
                ( last_name ?last )
                ( job_description ?job)
                ( favoured_language ?language ) )
        =>
                ( printout "Likes Java:" ? last, ? job ) )
```

JESS can be used in two overlapping ways - as rule engine for an expert system and as a general purpose programming language. JESS can directly access all Java classes and libraries, and for this reason is also frequently used as a dynamic scripting or rapid application development environment [Friedman-Hill, 99]. While Java code generally must be compiled before it can be run, a line of JESS code is executed immediately upon being typed. This allows developers to experiment with Java APIs interactively and build up large programs incrementally. It is also relatively easy to extend the JESS language with new commands written in Java or in JESS itself, and so the JESS language can be customised for specific applications. JESS is therefore useful in a wide range of situations.

5.4.4 Language choice

The previous sections have outlined some of the commonly used knowledge representation schemes and agent languages which are widely available. Chapter 4 outlined the suitability of both Java and CORBA as an ideal framework on which to base the proposed assistant system. Given this, the choice of scheme / language used to implement agents should complement the chosen framework.

Currently, only a small number of Java based products for the development of expert system tools exist in the market place. [Hall, 97a] [Hall, 97b] reviews the five commercially available tools: Advisor/J (Neuron Data Inc.), Ilog Rules for Java (Ilog Inc.), CruXpert (Crux Inc.), Selectica SRx Selection Engines (Selectica Inc.) and JESS (Sandia National Laboratories). JESS, unlike previous Java based expert system tools, is based on CLIPS which is an open, mature and portable knowledge representation language. A further pragmatic issue in the context of this research is that of cost - as tools such as Selectica and Advisor/J cost large amounts per developer license, whereas JESS is provided free for research purposes.

It is put forward that the use of production rules implemented under JESS provide an appropriate language scheme for the implementation of the agent rules for the proposed system. JESS provides a powerful knowledge representation scheme - in the form of production rules - and an open and flexible Java component which is suited for implementation in a CORBA environment. The choice of JESS may be seen as both a technical and pragmatic one.

5.5 Knowledge Acquisition

Knowledge acquisition is defined as [Buchanan et al., 83]:

> *"The transfer and transformation of potential problem-solving expertise from some knowledge source to a program".*

Essentially knowledge acquisition is the process of acquiring knowledge from a human expert (or group of experts). Knowledge acquisition involves elicitation of data from the expert, interpretation of the data to deduce the underlying knowledge and the creation of a model of the experts domain knowledge in terms of the most appropriate knowledge representation mechanism. For this, knowledge engineers must familiarise themselves with the domain of the expert and represent the knowledge in a form that can be computerised.

Although there have been several moves to use software tools for knowledge elicitation, a UK survey [Smith et al., 94] revealed that 77% of KBS had used some form of unstructured interview to obtain information. Most of these started out with informal discussions to explain the project and to gather preliminary information, followed by more formal structured interviews.

There are a number of commonly used knowledge elicitation methods as follows:

- **Printed / Electronic Sources** - The simplest form of knowledge acquisition is from printed sources. This includes searching through documents, books and other items of printed material to find the knowledge necessary to build a knowledge base.
- **Interviews** - This is the most widely used method. Informal interviews consist of the knowledge engineer asking spontaneous questions, where there has been little or no planning prior to the interview. By contrast, formal interviews use a variety of techniques exist such as: tutorial interviews, trigger-question interviews, introspective (or 'think aloud') interviews and retrospective interviews.
- **Questionnaires** - are a less frequently used technique and are best suited to a well understood domain.
- **Observational Studies** - are carried out at the same time as actual problem solving. That is, the knowledge engineer is present at the exact time that the experts apply their knowledge. Consequently, this allows for the observation of the experts behavior.

For the purpose of this study, the particular method of knowledge acquisition employed is not a primary concern. The chosen method should elicit a sufficient quantity of useful data which can in turn be represented by JESS in the proposed system, thereby demonstrating the usefulness of the intelligent assistant system. Further consideration should be given to the appropriateness of any given method after the demonstration of a prototype system. For this reason, a combined choice of printed / electronic sources and interviews will be used for the purposes of this study.

Opinions about the use of interviews vary widely. [Kawaguchi et al., 91] consider they are "*essential in eliciting new knowledge from domain experts*", while [Cooke and McDonald, 86] refer to them as "*a less than optimal knowledge acquisition technique*". However the interview process remains the most frequently used method for obtaining domain knowledge from human experts. This, coupled with printed / electronic sources, should provide an adequate mechanism to elicit knowledge for the proposed system.

5.6 Summary

This chapter presented the key issues surrounding the development of the knowledge base for the proposed assistant system. In particular, three commonly used knowledge representation schemes were outlined and three commonly used agent languages were presented, which concluded with the choice of a production rule representation based on JESS. In addition, the issues of choosing a suitable method for acquiring knowledge was investigated.

Chapter 6 describes the architectural implications for the proposed intelligent assistant system in light of the architectural presentation in chapter 4 and the knowledge base issues in this chapter. An architecture for the proposed system will be described and its component modules presented.

CHAPTER 6 SYSTEM ARCHITECTURE

6.1 Introduction

This chapter describes a high level view of the architecture of the proposed intelligent assistant system, its component modules and the interfaces between these modules.

6.2 Architectural Issues

The design of the system architecture has been motivated by a number of issues previously discussed. This section will summarise these choices, prior to description of the system architecture.

In chapter 3, a number of candidate solution technologies were discussed - Decision Support Systems, Expert Systems, Expert Critiquing Systems, Blackboards and Intelligent Agents - with the proposal of a hybrid approach being set forth [O'Connor, 98]. This hybrid approach consists of a library of intelligent agents (multi-agent system), where each expert advisory agent is a specialist in an individual area of software project planning within the framework, on an overall tool which represents the Decision Support System. Communication amongst agents is facilitated by a Blackboard structure in which agents may write their conclusions or interim findings and the rest of the multi-agent community may read, thus allowing for effective inter-agent communication.

The uses of Java and CORBA have been described in chapter 4 as solution technologies to the issues of building a platform independent, distributed, client-server system, with the use of JESS being put forward in chapter 5 as an agent implementation language. The choices allow for the creation of a Java-based DSS system containing a JESS-based multi-agent community, in which the various components communicate via CORBA [O'Connor and Moynihan, 98].

6.3 Architecture Components

Figure 6.1 illustrates a high-level view of the system architecture being proposed to implement the intelligent assistant system. This consists of a user interface, the decision support system itself and the underlying knowledge base which contains the expertise and knowledge (ie. agents).

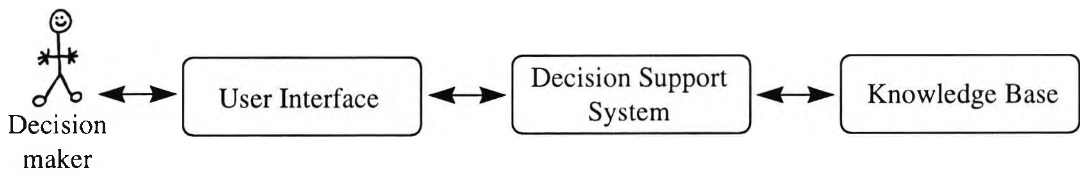


Figure 6.1 - High-level View of Architecture

These architectural components can be further subdivided into their constituent modules which represent CORBA (client and server) components. This view of the system architecture is illustrated in figure 6.2, followed by a description of the component modules.

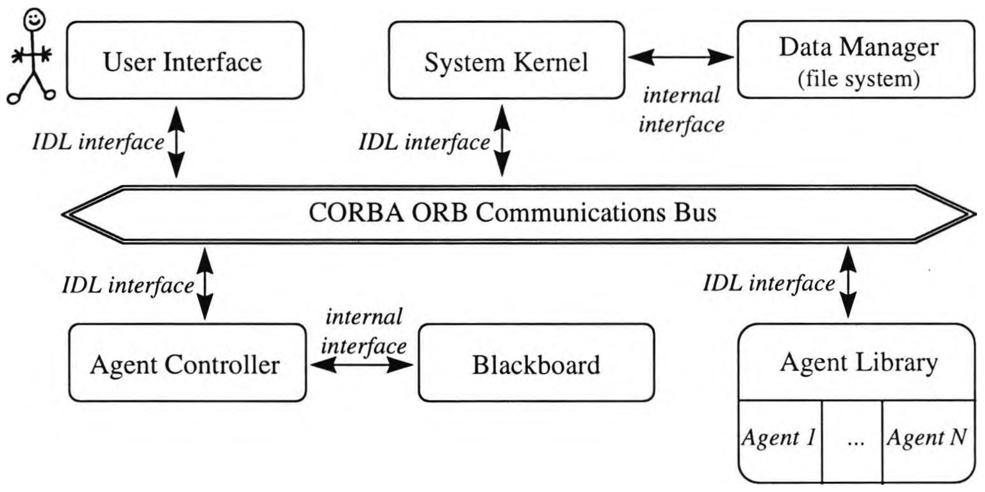


Figure 6.2 - Component Architecture

- **User Interface** - This component handles the management of all the screen elements (menus, dialog boxes, etc.), validates data entered by the user and passes on clear functional messages to the rest of the system.

- **System Kernel** - This is the core component of the system and handles all the processing and storage of user entered data. It manages all aspects of project plans and channels advice from the agents to the user.
- **Data Manager** - This manages all aspects of the mapping from the logical view of data to its physical storage. It is under the control of the System Kernel, through which all requests for data must be channelled.
- **Agent Controller** - This module acts as a controller (or supervisor unit) over the agent community and manages the scheduling and execution of agents, as well as governing write access to the Blackboard.
- **Blackboard** - This represents the global problem solving state of the system. Over time, agents produce changes to the Blackboard which lead incrementally to advice on the project under consideration. The Blackboard is under the control of the Agent Controller and all requests for data read / write must be channelled through the Agent Controller.
- **Agent Library** - All agents are contained in an Agent Library, but remain under the control of the Agent Controller. The purpose of the Agent Library is to manage the physical agents themselves and to service requests for agent interactions from the Agent Controller.

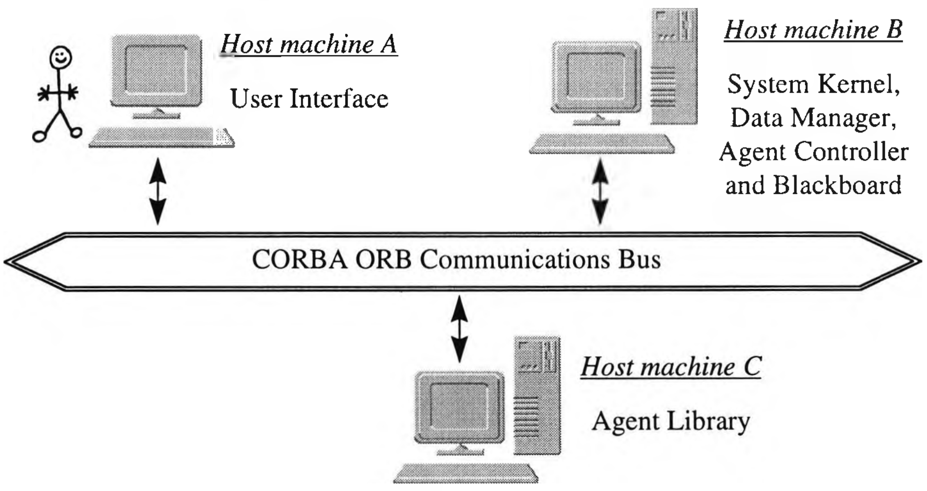

Host machine A
User Interface
Host machine B
System Kernel, Data Manager, Agent Controller and Blackboard
CORBA ORB Communications Bus
Host machine C
Agent Library

Figure 6.3 - Component Configuration

This CORBA based architecture allows for several possible implementation configurations in terms of what host machine a particular component may be installed on (in a network environment). Figure 6.3 illustrates one such configuration in which the User Interface is installed on host machine A (say a typical desktop PC), the main DSS components (ie. System Kernel, Data Manager, Agent Controller and Blackboard) are on host machine B (say a standard network fileserver) and the Agent Library is installed on host machine C (say a backend server).

6.3.1 Component Interfaces

The systems component modules communicate via interfaces (or API - Application Programmer Interface) and are of two types: public CORBA IDL interfaces and internal interfaces, which are not implemented in IDL.

IDL interfaces essentially define each components boundaries - that is, the services (or methods) the objects in that component can provide to other components. As IDL is completely language independent, objects specified using IDL are portable across languages, operating systems and networks. Thus the component modules with IDL interfaces may be implemented in different languages and reside on different machines (in a network or Internet environment). For example, in an extreme case, the four main modules (GUI, Kernel, Agent Controller and Agent Library) could reside on four separate host machines in a network, with all inter-module communication being automatically handled by the CORBA ORB.

The use of internal (or private) interfaces between modules provides a control mechanism over the global data structures of the Blackboard and file system and also provides for encapsulation of data with its appropriate control structure. This is achieved by allowing access to both the Blackboard and Data Manager only via calling an IDL method in the appropriate controlling module, which can then choose to pass on the request or modify it. In such a configuration, these two (non-IDL interface modules) must have their interfaces implemented using standard programming language constructs, thus both modules must reside on the same machine as their controlling module.

CORBA IDL is a declarative language used to define object APIs. When an IDL file (i.e. an object API specified in the IDL language) is compiled (by an IDL compiler), it produces two sets of code files: sub code to create proxy objects that a client can use for making invocations on object references of the interface types defined in the IDL file, and skeleton code for access to objects that support those interfaces. The role of the stub and skeleton are illustrated in figure 6.4. If the client and the target object are in the same address space, no extra code is required to communicate between them. If they are in different address spaces, on the same or different hosts, then extra code is required in the client to send the request to the server side, and extra code is required at the server side to accept the request and pass it to the target object.

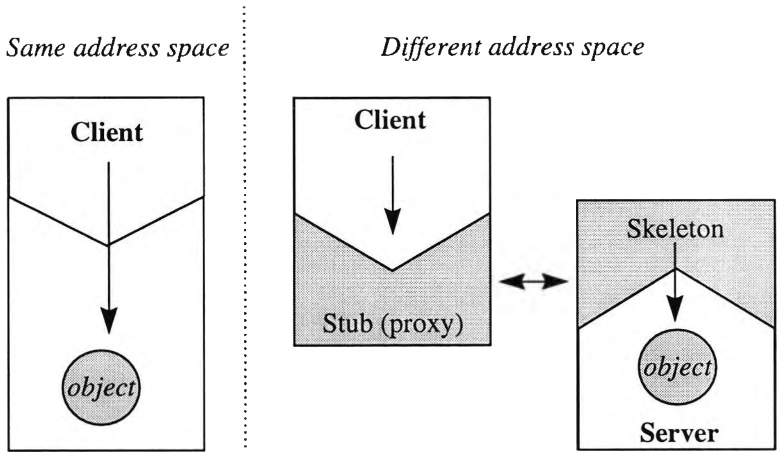


Figure 6.4 - Stub and Skeleton Code (Proxy Objects)

The following sections will described each of the component modules in further detail, including a description of the modules functionality and its interface.

6.3.2 User Interface

The user interface (or GUI) has two tasks within the system; GUI management, which includes managing the screen elements, validating data inputs and passing on commands to the system kernel; and the management / control of the overall structure of screens and dialog boxes. This functionality is represented (at an architectural level) by three components, as illustrated in figure 6.5:

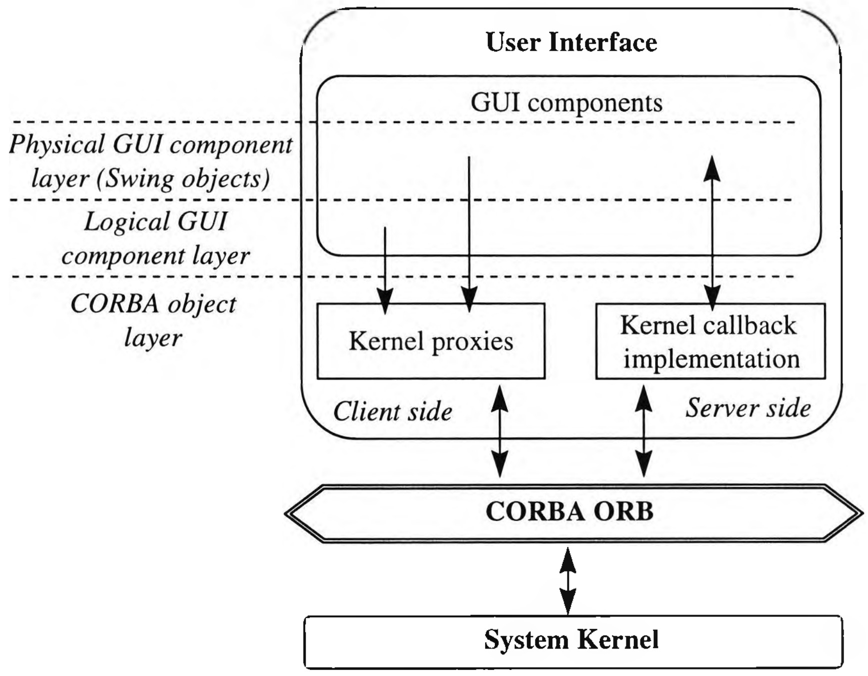


Figure 6.5 - User Interface Component Architecture

- **GUI components** - are divided into two layers: The first layer maps the proxy objects (CORBA object layer) to the logical component layer, which allows for the provision of a model of user interface components to be developed. The second layer (physical component layer) contains the actual GUI objects based on the Sun Swing toolkit. This layer (or separation) approach is based on the user interface management paradigms of *Model-View-Controller* and *Presentation-Abstraction-Control* model [Coutaz, 87].
- **Kernel proxies** - are Java objects which are created by compiling the IDL interface specification and are used to communicate with the Kernel.
- **Kernel callback implementation**[2] - handles service requests from the System Kernel, and represent objects which implement the interface as specified by IDL specification. This component is necessary to allow the Kernel get the attention of the GUI when, for example, an agent has produced some advice and it wishes it to be presented to the user.

[2] In CORBA, a callback is a method call from a server to a client, which reverses the client-server roles. They extend a clients architecture to include messages from their servers and are generally invoked when a server has something important to send to a client. The implementation code of the callback method exists on the client side, with a callback proxy method on the server side.

GUI management is an abstract layer of the user interface which exists between the actual GUI functions (as implemented using the Sun Swing JavaBeans components) and the rest of the modules. It's main areas of functionality are:

- **Data input checking** - Receive data inputted by the user and check it for validity. This may involve ensuring that an integer or real number is within certain bounds, checking that a date is valid and checking that text strings are correct when possible.
- **Data pre-processing** - Some types of data need to undergo an element of pre-processing prior to transmission via a CORBA ORB. For example, a date in the format DD/MM/YYYY would needed to be converted to a long[3] number of the format NNNNNNNN.
- **Data request batching** - In many cases it is desirable (and more economical from a CORBA data transmission point of view) to batch groups of related data for subsequent processing. For example, many of the agents will require a series of data inputs simultaneously in order to execute - this user-entered data could be batched in small groups for communication to the rest of the system. Likewise, when advice is received from agents, it may be more desirable to batch the incoming advice data and present it in small groups to the user, instead of interrupting the user on a frequent basis with individual items of advice.
- **General screen house-keeping** - While most screen house-keeping functions (such as updating menus, windows management etc.) can be performed by the physical GUI components themselves, some functions require more complex processing. For example, whenever the GUI needs to process an item of data which is managed by the kernel, the GUI must first negotiate with the kernel to send/receive that data.

[3] In the CORBA IDL mapping scheme, the Java datatype int maps to the IDL datatype long.

6.3.3 System Kernel

The System Kernel provides the main interface between the User Interface component, the knowledge (advice) generator component provided by the Agents and the Data Manager. The Kernel represents the main data processing / management aspects of the system and acts as the overall controlling component of the system. It is represented (at an architectural level) by five main components and five further sub-components, as illustrated in figure 6.6:

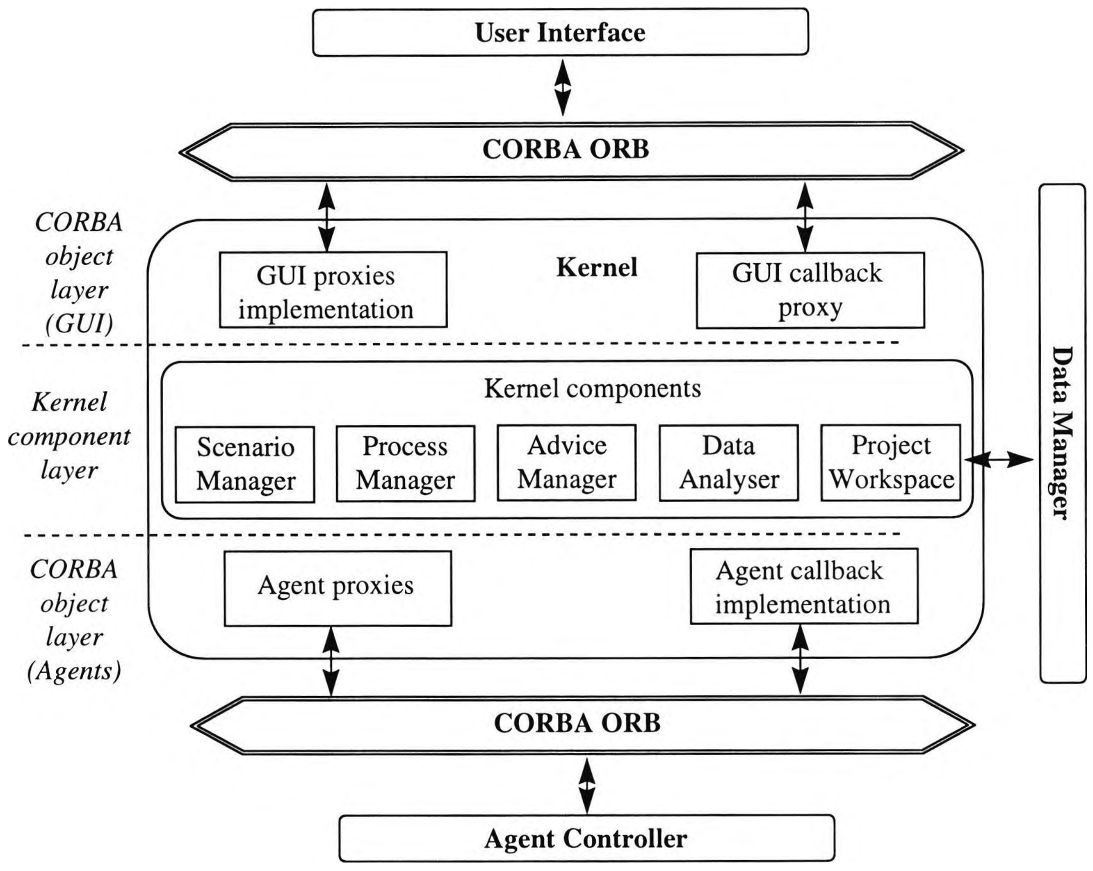


Figure 6.6 - System Kernel Component Architecture

- **GUI proxies implementation** - are the implementation of the Kernel Proxies as seen by the GUI. These objects implement the interface as specified by IDL specification and are used by GUI components to communicate with the Kernel to send or receive data.

- **GUI callback proxy** - are Java objects which are created by compiling the IDL interface specification. They handle service requests which are generated by the Kernel and destined for the User Interface and are represented on the User Interface side by the Kernel Callback Implementation.
- **Kernel components** - The main kernel components are as follows:
  - **Process Manager** - is concerned with assisting the user to select a suitable process template from the process repository (via data manager) to form the basis of a project definition.
  - **Scenario Manager** - Projects are categorized by scenarios, which are different views (work breakdown structures) of the same project. Two scenarios may be practically the same except for differences in two or three small pieces of information. For example, the duration of a sub-task of one scenario may be longer than another, thus enabling the user to 'experiment' with small variations in a project and possibly receive different advice from agents.
  - **Advice Manager** - packages advice received from the agents for subsequent delivery to the user interface. As advice is generated in an asynchronous manner by agents, it is the job of the Advice Manager to ensure advice is batched according to scenario under examination, etc.. The Advice Manager also ensures that advice is in the appropriate format (e.g. HTML) before being sent to the User Interface.
  - **Data Analyser** - undertakes any calculations required on project data, particularly in the creation and analysis of scenarios.
  - **Project Workspace** - acts as a temporary storage container for all project data (i.e. scenarios and related data) for any given user session. At the start of a session, the project under consideration will be retrieved by the Data Manager and placed in the Project Workspace until the end of the session, when it be stored by the Data Manager.
- **Agent proxies** - are Java objects which are created by compiling the IDL interface specification. They are used by Kernel components to communicate with the Agent Controller to send or receive data.

- **Agent callback implementation** - handles service requests generated by the Agent Controller and represent objects which implement the IDL interface. This component is necessary to allow the Agent Controller get the attention of the Kernel when, for example, an agent has produced some advice and it wishes it to be given to the kernel's Advice Manager.

6.3.4 Data Manager

The Data Manager handles all aspects of the mapping from the logical view of data to its physical storage and maintenance. It is under the direct control of the System Kernel, thus all requests for data must be channelled through the Kernel. Figure 6.7 illustrates the main components of the Data Manager:

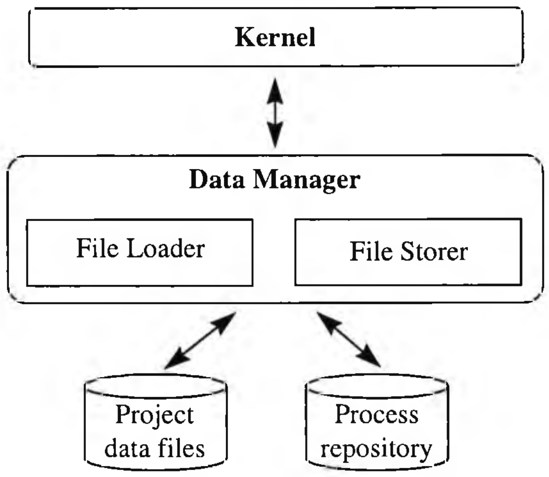


Figure 6.7 - Data Manager Component Architecture

- **File Loader** - reads in data from a physical file (project data file and/or process repository) and passes that data to the Kernel.
- **File Storer** - is used to write out to a physical data file (project data file) all data passed to it from the Kernel.
- **Project Data Files** - are the physical files containing project data.
- **Process Repository** - are the physical data files which contain all the process templates.

6.3.5 Agent Controller

The Agent Controller is a supervisor unit over the agent community, which manages the scheduling and execution of agents. In addition, it governs read / write access to the Blackboard. It is responsible for all communications between the Kernel and the Agents / Agent Library. At an architectural level it is represented by 3 main components and 4 further sub-components, as illustrated in figure 6.8:

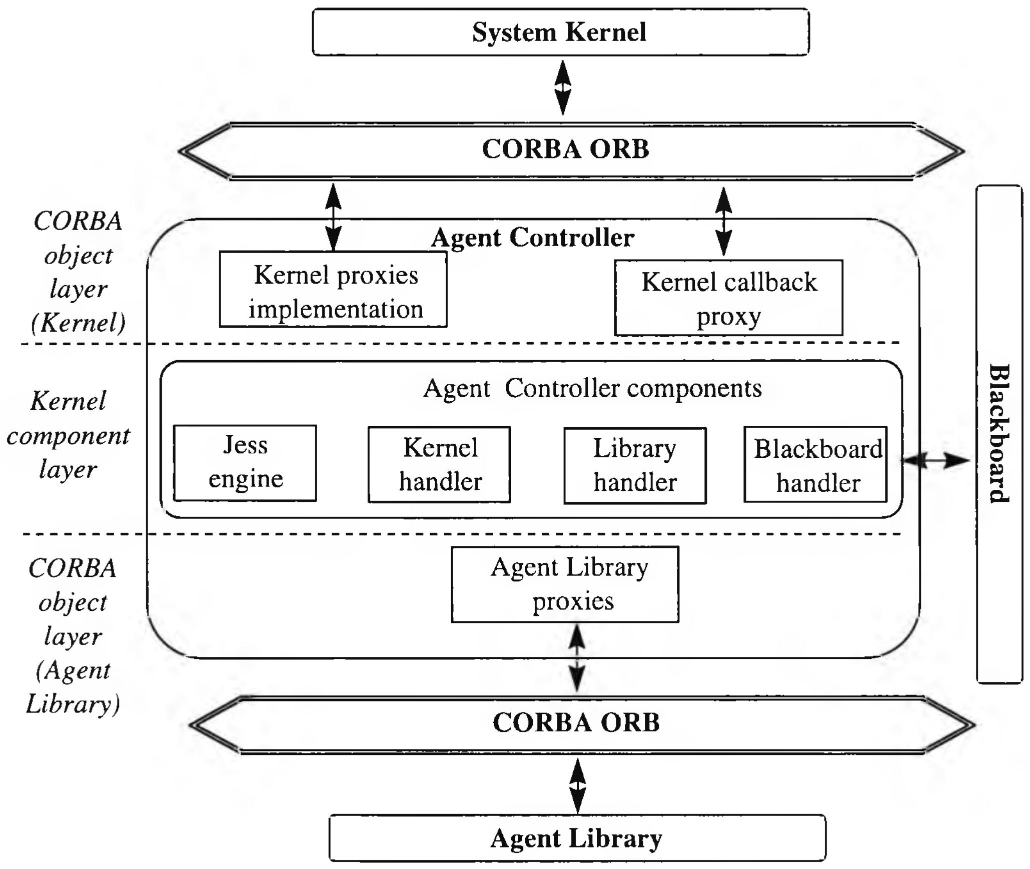


Figure 6.8 - Agent Controller Component Architecture

- **Kernel proxies implementation** - are the implementation of the Agent Proxies as seen by the Kernel. These objects implement the interface as specified by IDL specification and are used by the Kernel to communicate with the Agent Controller to send or receive data.
- **Kernel callback proxy** - are Java objects which are created by compiling the IDL interface specification. They handle service requests which are

generated by the Agent Controller and destined for the Kernel and are represented on the Kernel side by the Agent Callback Implementation.

- **Agent Controller components** - The main agent controller components are as follows:
  - **Kernel Handler** - is concerned with managing all communication between the Kernel and the other Agent Controller components. For example, when the JESS Engine requires actual (live) data values (from Data Manager) about a project, it is the Kernel Handler component that deals with processing the request. All such communication takes place via the CORBA ORB using the Kernel Proxies implementation
  - **Library Handler** - has a similar management role to the Kernel Handler. It manages all aspects of communication with the Agent Library, via the Agent Library Proxies.
  - **Blackboard Handler** - like the other handlers is responsible for the management of communications with the Blackboard. As the Blackboard interface is non-IDL (i.e. not using the CORBA ORB), communication is via a direct set of APIs. For example, when an individual agent wants to write information to the Blackboard, it is the Blackboard handler which manages the communication of this data to the Blackboard component.
  - **JESS Engine** - This component contains the inference engine and related modules for the JESS system. When an agent executes (under the control of the Agent Controller), its rules will be processed by the JESS engine and subsequent output processed by the system. As previously described, this architecture allows for a number of distinct inference mechanisms to be used, in which case they would reside as separate sub-components of the Agent Controller components.
- **Agent Library Proxies** - are Java objects which are created by compiling the IDL interface specification. They are used by Agent Controller components to communicate with the Agent Library to send or receive data.

6.3.6 Blackboard

The Blackboard represents the current problem solving state of the agents with respect to the project under consideration. For each area of expertise, the Blackboard holds state information on Tokens (data items in a process template / project model) which are manipulated by agents, state information on the agents and any advice agents have provided to date. The Blackboard is under the direct control of the Agent Controller. Any agents wishing to access the Blackboard must have read requests channelled through the Blackboard Handler component of the Agent Controller. Any write requests to the Blackboard are similarly processed, allowing co-ordination of read / write access, thus avoiding any possible access conflict.

To impose order on the information stored in the Blackboard it is necessary for the Blackboard to have a strictly defined structure. In keeping with traditional Blackboard framework design [Englemore and Morgan, 88], this Blackboard is organised as a tree structure where each node in the tree represents an area of project planning expertise. Figure 6.9 illustrates this tree structure with nodes from 1...N, where a node represents data stored about one specific area such as activity planning, risk analysis, etc.

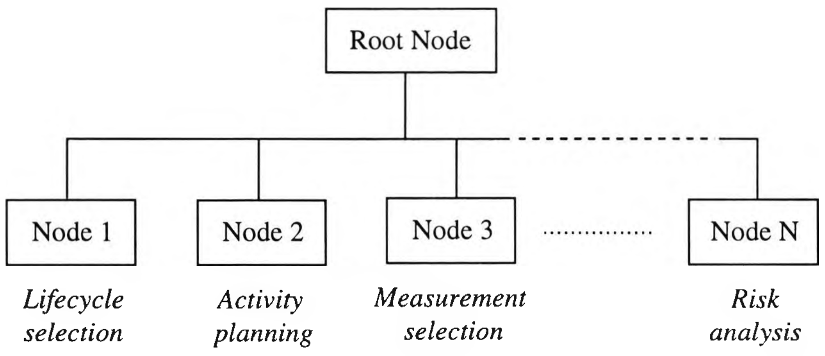


Figure 6.9 - Blackboard Node Hierarchy

As there may be many agents operating in any one area (ie. node in the tree), each agent is assigned a segment in a node within which there is a container for multiple scenarios. Figure 6.10 illustrates a detailed view of Node 1. Within this node there are segments 1...N for each of the agents operating in this area. Likewise within each segment there are a number of containers 1...J for each of the possible project scenarios.

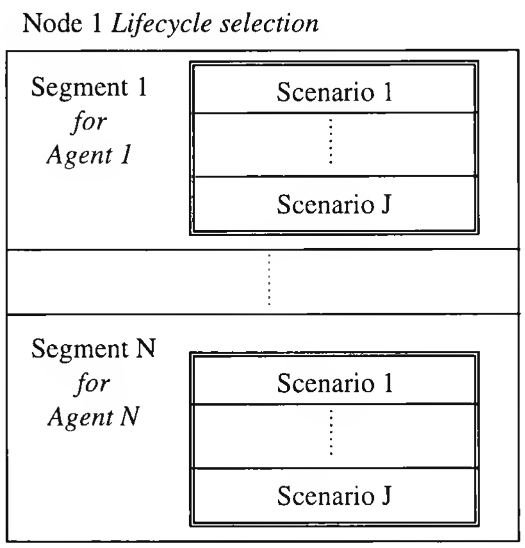


Figure 6.10 - Blackboard Segment Structure

In each scenario container there are a number of slots which hold state information on a specific agent for a specific scenario. These include a slot for each token an agent manipulates and a slot for any advice information an agent may have previously generated. Figure 6.11 illustrates a detailed view of segment 1 of node 1 and shows slots 1...K for each scenario.

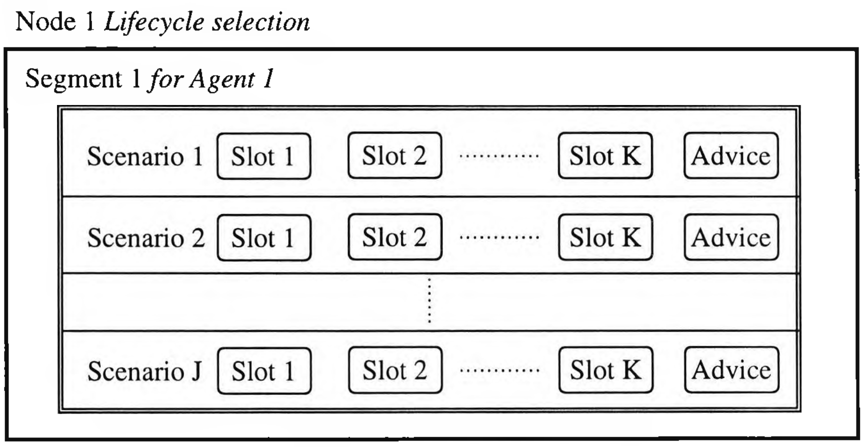


Figure 6.11 - Blackboard Scenario Structure

6.3.7 Agent Library

Agents are located in the Agent Library, but remain under the control of the Agent Controller. The purpose of the Agent Library is to manage the physical agents themselves and to service requests for agent interactions from the Agent Controller, where the physical agents themselves are represented as JESS rule scripts which are held on the file system. An example usage would be when the Agent Library interrogates the JESS scripts at the request of the Agent Controller and creates an instance of an agent for execution. The architecture of the Agent Library is illustrated in figure 6.12.

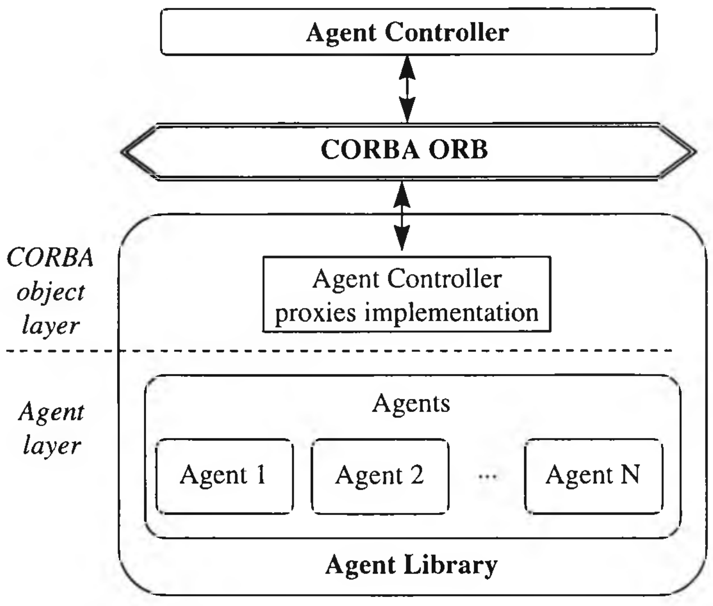


Figure 6.12 - Agent Library Component Architecture

- **Kernel proxies implementation** - are the implementation of the Agent Library Proxies as seen by the Agent Controller. These objects implement the interface as specified by IDL specification and are used by Agent Controller to communicate with the Agent Library to send or receive data.
- **Agents** - are the JESS rule scripts which implement the knowledge base. In addition to the actual rules, the script also contains identification information about the agent.

Individual Agents are represented as separate files within the system, which contain JESS rule scripts and identification / configuration information. The structure of an agent is shown in figure 6.13.

- **Agent Header** - contains all the basic information which an agent needs to identify itself to the system, including an identification and version number, and the area of proficiency of the agent.
- **Agent Tokens** - are a list of the data items in a process template / project model (that represent the actual data items held about a project) which the agent uses in its rule set.
- **JESS Rule** - is the actual JESS rule script.

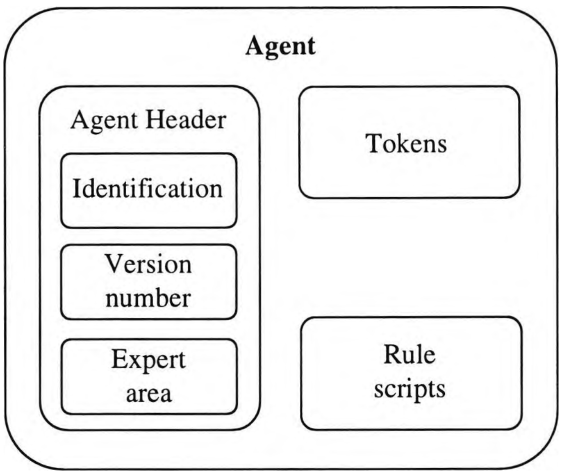


Figure 6.13 - Agent Structure

As can be seen from the above, agents are not directly embedded within the Agent Library or Agent Controller. Instead there is a link from the Agent Library to the set of available agents, i.e. agents installed on the file system. This architectural property allows for a dynamic agent population as the Agent Library will interrogate the file system at the initiation of a session and dynamically build a view of the set of available agents for the duration of that session. Furthermore, this data is passed to the Blackboard which creates the appropriate number of segments in its nodes, therefore any individual agent may be added or removed from the Agent Library by the addition or deletion of their corresponding agent file on the file system.

The architectural property of a resizable agent population is a key factor in the agent-orientated framework as it allows for dynamic updating of the knowledge base, where knowledge (agents) may be added, revised or removed without impact on the rest of the system. In particular, any such alteration in the knowledge base is completely independent of the other architectural components and does not require any explicit changes in system configuration. Thus the knowledge base may grow over time to take account of new expertise or techniques in software project planning or to add organisational specific knowledge. For example, if an organisation was planning a project to CMM level 2, they may wish to add a series of agents dedicated to that standard, or a company / project specific standard.

6.4 Review of Architectural Characteristics

Chapter 4 proposed a number of desirable architectural characteristics which should be taken into consideration when developing an architecture for an intelligent assistant system for less well understood domains, and in particular for the domain of software project management. In the light of this, and having discussed the proposed architecture above, it is necessary to pause and consider the how well this architecture satisfies these desirable characteristics.

1. **Specialism** - In this architecture, each agent is a specialist in a particular area of software project management, thus providing for a community of cooperating agents and the associated flexibility of the system.
2. **Information exchange** - This multi-agent approach provides for enhanced information sharing, as agents within the community can exchange information. Although each agent is a specialist in its own area, they have the ability to communicate with agents of other specialisms via the Blackboard.
3. **Collaboration** - The concern here is the overhead of both complexity and system performance due to collaboration. This has been streamlined by the use of a Blackboard attached to the Agent Controller, thus reducing both

system overhead and complexity of implementation, whilst providing for collaboration.

4. **Blackboard** - The blackboard approach was put forward in chapter 3 as being an aid to reducing both the level of complexity and volume of communications traffic. As stated in point 3 above, this has been incorporated into the proposed architecture.
5. **Supervision** - In an agent community, there does not have to be a single controlling agent. However, the indications are that the existence of such a supervisory agent led to more organised communications in the community. The provision of the Agent Controller will lead to more organised activity within the agent community.
6. **Hierarchy** - The concern here was that a hierarchy could diminish the flexibility of a system, thus agents should be allowed to operate regardless of rank. In the proposed system architecture, all agents are equal, but the Agent Controller may choose to impose a level of importance at run-time in accordance with local system performance, conditions, etc..
7. **Knowledge evolution** - The multi-agent approach to knowledge base implementation allows for dynamic updating of the knowledge base by adding, updating and deleting agents. In addition, CORBA provides for remote discovery of objects and services, and thus provides an ideal transportation bus for new agents.
8. **Data separation** - Is concerned with the separation of data and agents leading to a more open and flexible system, which is provided by the data communication between the Data Manager and the Agent Controller. This allows alterations in the data storage mechanisms and the agents themselves to be carried out independently of each other.
9. **Platform independence** - As the proposed architecture takes full advantage of both CORBA and Java, true platform independence is achieved.
10. **Multiple inference strategies** - As previously described, the proposed architecture allows for a number of distinct inference mechanisms to be employed. Each separate inference engine would reside as a separate sub-component of the Agent Controller.

11. **Component separation** - The separation of the agents components (which represent the 'intelligence' of the system) from the Kernel and GUI components (which represents the user-orientated framework or 'application') results in a flexible architecture, where either application or intelligent components may be altered independently.
12. **Multiple paradigms** - An agent approach allows the development of a number of different supporting frameworks. A traditional expert system approach may be employed by some agents, while others can adopt an expert critiquing system approach of critiquing user entered plans or data.

6.5 Summary

This chapter presented the architecture of the proposed intelligent assistant system. Each of the main architectural components have been further described, with the interfaces between these components outlined and the module interactions outlined.

Chapter 7 presents the issues surrounding the design and implementation of a prototype of the proposed system. It will outline both the design approach and the actual system design. In addition, the development of the prototype implementation of the system is presented.

CHAPTER 7 PROTOTYPE IMPLEMENTATION

7.1 Introduction

This chapter describes the design and the development of a prototype implementation of the architecture for an intelligent assistant system. In addition it describes a typical tool session from both a user and system level perspective. Finally, some observations and lessons learned from building the prototype system are presented.

7.2 System Design

One of the first questions that presents itself is the choice of design methodology. These choices are restricted to object orientated methods as this system will utilise both Java and CORBA which are both fully object orientated.

Object Orientated Analysis (OOA) is a semiformal specification technique for the object-orientated paradigm. There are currently over 40 different techniques for performing OOA [Schach, 97], and new techniques are put forward on a regular basis. The most popular techniques are: Booch's technique [Booch, 94], OMT [Rumbaugh et al., 91] and UML [Fowler, 97] - although it would currently appear that UML is becoming the more popular [Pooley and Stevens, 99]. However, most techniques are largely equivalent and consist of three basic steps:

- **Class modelling** - Determine the classes and their attributes. Then determine the interrelationships between the classes.
- **Dynamic modelling** - Determine the actions performed by or to each class or subclass.
- **Functional modelling** - Determine how the various results are computed by the various products.

The choice of which particular method of OOA to employ for a given project is usually arbitrary and linked to the experience or preference of the system designer, or dictated by outside influences. With this in mind, OMT was chosen as the OOA method for the design of the prototype system.

For the sake of clarity, all OMT class diagrams in this thesis are represented as 'class only', and as such, no attributes and operations appear in the class diagrams.

7.2.1 Design Approach

The proposed architecture can be viewed as containing three major components: User Interface, Kernel and Agents. This architectural division allowed for an allocation of design and implementation tasks among the three developer partners of the P3 project, with the Agent components being the sole responsibility of this researcher.

The use of CORBA imposes a sequence on the design activity. Firstly, each component in the system should be defined as either a client or a server (in the CORBA sense) and subsequently the appropriate IDL interfaces between them defined. These IDL interfaces define the services (methods) provided by each component (client or server) and the data structures that are transferred between them.

7.2.2 Client-Server Components

Although CORBA uses the terms *Client* and *Server*, this does not mean that CORBA systems have a traditional client-server architecture where a set of clients use a single server. Instead, the objects in one server may use the objects in other servers. For example, a client may invoke an object in a server, and that object may invoke others in order to fulfill the client's request. This can be achieved using the callback mechanism which reverses the roles, allowing servers to call a remote object, thus acting as a client for the duration of that call. This mechanism is often used by a server to get the attention of a client or another server.

From a CORBA perspective, the proposed architecture has four major components as illustrated in figure 7.1, which may be classified as either clients or servers:

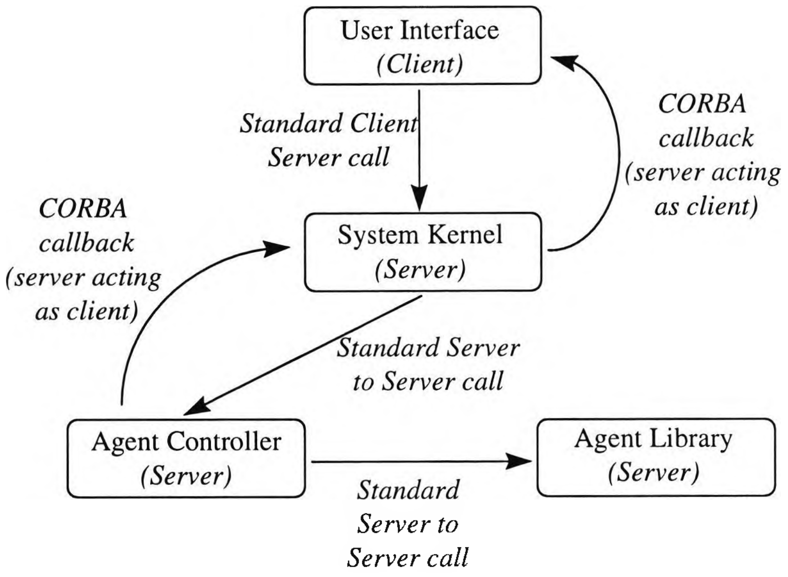

User Interface
(Client)
Standard Client Server call
CORBA callback (server acting as client)
System Kernel
(Server)
CORBA callback (server acting as client)
Standard Server to Server call
Agent Controller
(Server)
Agent Library
(Server)
Standard Server to Server call

Figure 7.1 - Client-Server Structure

- **User Interface** - Is the only true client in the classical sense. It appears as the 'front end' of the system and will always reside on the client machine, i.e. users desktop. It only invokes methods on System Kernel objects. However, the System Kernel may invoke methods on the User Interface (callbacks), if it requires its attention.
- **System Kernel** - As Acts as a server to the User Interface as it provides a number of services for it, for example, management activities associated with storing data. In addition, the System Kernel views the Agent Controller as a server, as it will request certain services from it such as "give me some advice for this situation".
- **Agent Controller** - Provides services to the System Kernel and as such acts as a server to it. However, the Agent Controller requires certain services from the System Kernel (such as data retrieval) for which it will perform a callback, thus using the System Kernel as a client for the duration of the call.
- **Agent Library** - Performs services on behalf of the Agent Controller and as such views it as a client. For example, the Agent Controller would request the Agent Library to extract a particular agent for execution.

These four CORBA components have several possible implementation configurations in terms of which component(s) may be installed on which host machine. There are three typical configurations envisaged, as illustrated in figures 7.2 to 7.4.

1. **Desktop configuration** - Where all components are installed on a single machine, say a typical end user desktop PC.
2. **Network configuration** - Where the User Interface is installed on the desktop and all other components are on the network server machine.
3. **Intranet configuration** - A variation of a standard network configuration, with the Agent Library being hosted by the corporate Intranet server and therefore a central knowledge base available to a wider corporate user group.

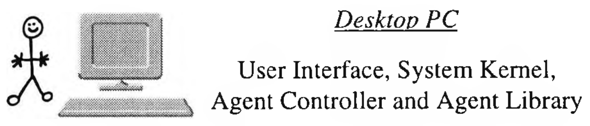

Desktop PC
User Interface, System Kernel,
Agent Controller and Agent Library

Figure 7.2 - Desktop Configuration

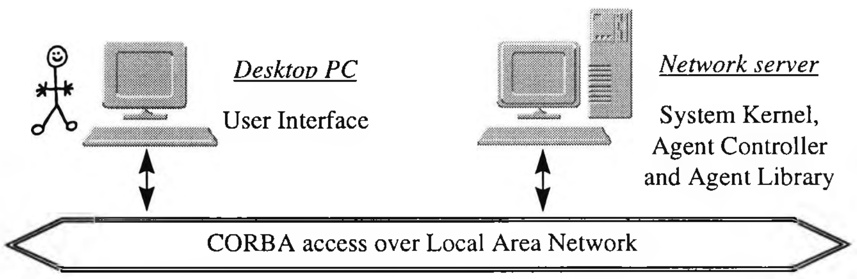


Figure 7.3 - Network Configuration

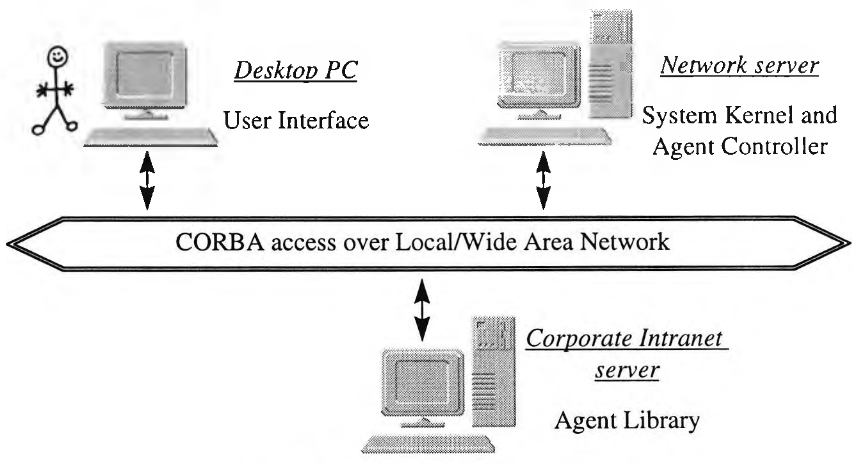


Figure 7.4 - Intranet Configuration

7.2.3 Component Interfaces

As can seen from figure 7.1 there are three IDL component interfaces which have to be specified, as follows: The User Interface - System Kernel interface (figure 7.5) shows the User Interface sending either token values or advice requests to the System Kernel, and the System Kernel sending advice objects or details on the project.

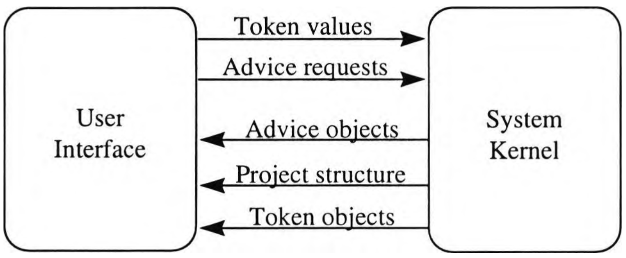


Figure 7.5 - User Interface - System Kernel Interface

The System Kernel - Agent Controller interface (figure 7.6) shows the three main types of information communicated between the two components - information or values of tokens, advice requests advice and the advice itself.

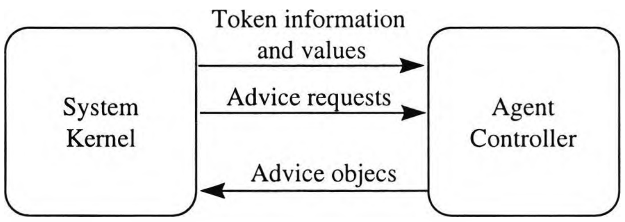


Figure 7.6 - System Kernel - Agent Controller Interface

The Agent Controller - Agent Library interface (figure 7.7) shows the Agent Library sending information on agents dependent tokens to the Agent Controller and servicing requests for agents to be extracted by returning a handle to an agent.

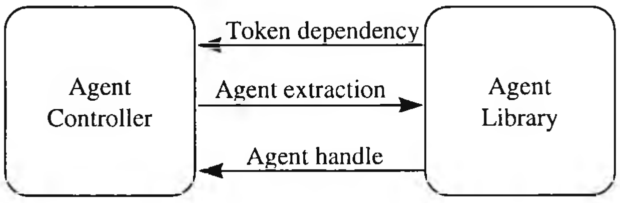


Figure 7.7 - Agent Controller - Agent Library Interface

7.2.4 User Interface

The User Interface components are organised into three main areas:

- The main window of the application, which deals with the display issues and CORBA communication initialization.
- The scenario window, which displays a project and allows the user to create scenarios based on a project situation.
- Dialog boxes which display information and allow the user to enter data.

The main classes in the User Interface are illustrated in figure 7.8:

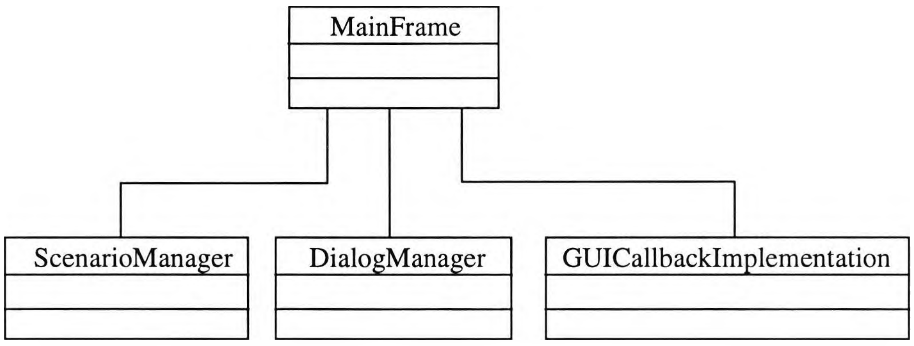


Figure 7.8 - Overview of User Interface Classes

- The MainFrame manages all the GUI components (images, windows and other resources) and initialises CORBA communications between the User Interface and the System Kernel.
- The ScenarioManager is used to display the scenario windows and control all actions associated with it.
- The DialogManager acts as a generic manager for all dialog boxes and manages all actions associated with them.
- The GUICallbackImplementation provides the implementation to the proxies for the System Kernel callbacks. It runs in a thread and listens to events coming from the System Kernel. A typical usage would be the System Kernel indicating the arrival of some advice from an agent.

7.2.5 System Kernel

The main classes of the System Kernel are illustrated in figure 7.9:

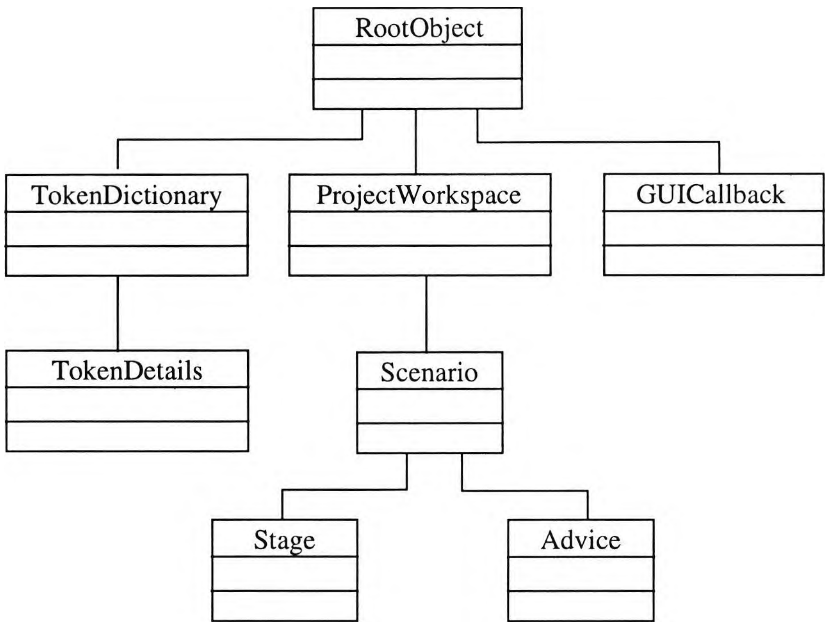


Figure 7.9 - Overview of System Kernel Classes

- The RootObject is the controlling class of the System Kernel and provides a central storage and access point for different ProjectWorkspace objects. Its main functionality is concerned with the creation and maintenance of the list of ProjectWorkspace objects. It also provides access to some of the data files and is also used to pass CORBA information from the User Interface to the System Kernel concerning hostnames and callback functions.
- The ProjectWorkspace object is the central storage object for data in the package. It allows the user to access the scenarios (work breakdown structures) and therefore the stages and tokens associated with a project. The principal functionality of this object is the loading and storing of data for this Java package.
- The Scenario objects contain pointers to the stages of a project and the associated advice generated by the agents for those stages. Any given ProjectWorkspace may contain multiple scenarios which represent different

work breakdown structures or views of a project. This allows the user to examine a project from different perspectives.

- The Stage object is used to store the elements of the lifecycle (i.e. the different activities such as requirements, specification, etc.) and characteristics connected to a particular part of the lifecycle.
- The Advice object is used to convey the agent generated advice information to the User Interface component.
- The GUICallback object implements the callback features of CORBA which allow the server (ie. System Kernel) to use part of the functionality of a client (user Interface).
- The TokenDictionary groups different TokenDetails objects for display, where these groupings correspond to different classifications of tokens.
- The TokenDetails object stores the static information for each token. Examples of this type of information would be a question that would be asked (to get a token value) or an explanation (of a tokens meaning). Other information such as bounds, data types and enumerated lists are stored.

7.2.6 Agent Controller

The main classes of the Agent Controller are illustrated in figure 7.10:

- The Supervisor object is the main controlling object in the Agent Controller component. It ensures that all requests are serviced and all advice is forwarded to the System Kernel. Its main tasks are:
    - It is responsible for initialising CORBA communications with the System Kernel and the Agent Library.
    - It is responsible for creating new threads of control for new agents.
    - It must maintain these agent threads and time them out after a period.
    - It is responsible for acquiring both the status and values of tokens from the System Kernel.
    - It is responsible for the creation and maintenance of the AdviceTable.

- It must ensure all advice generated by agents is placed in the AdviceTable and the System Kernel is notified.
- It manages the creation of the Blackboard.

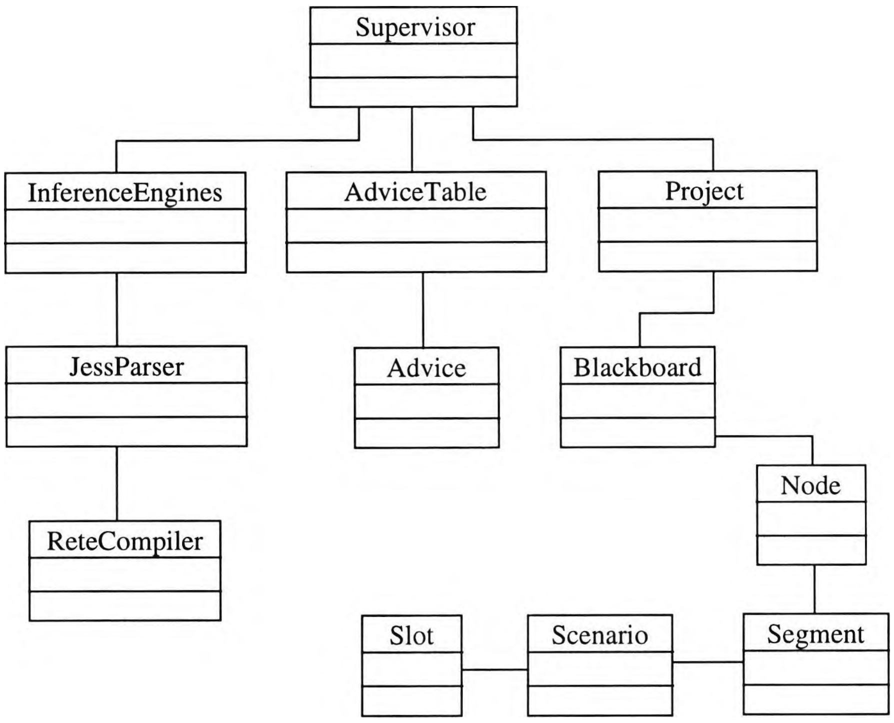


Figure 7.10 - Overview of Agent Controller Classes

- The Project object implements the core functionality of the Agent Controller and is linked to a single project. It does not have to concern itself with Kernel communications as this is managed by the Agent Supervisor object.
- The AdviceTable is responsible for storing all items of advice (i.e. Advice objects) which are generated by agents. Essentially this object contains a vector of Advice objects and the methods necessary to maintain the vector.
- The Advice objects are responsible for storing the actual advice itself.
- The Blackboard object is responsible for the control of the Blackboard structure. It must ensure that all relevant data is kept up to date. The object structure of the Blackboard is the same as described in section 7.3.6.
- The InferenceEngine is a generic object used to control the execution of an agent. It is responsible for the interrogation the agent to be executed and passing it to the appropriate inference engine, along with the actual token

and related data values. In addition, it communicates generated advice to the AdviceTable via the Project object.

- The JessParser object is responsible for the integration of JESS into the system. It prepares and interprets data sent to/from the ReteCompiler.
- The ReteCompiler is responsible executing an agents rules. This object is supplied as part of the JESS system.

7.2.7 Agent Library

The main classes of the Agent Library are illustrated in figure 7.11:

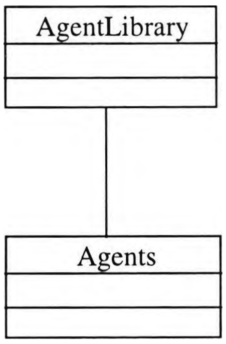


Figure 7.11 - Overview of Agent Library Classes

- The AgentLibrary object is responsible for all the management aspects of the Agent Library component of the system and for the direct interaction between the agents and the rest of the system, via its CORBA communications link with the Agent Controller. Upon system startup it is responsible for passing information to the Agent Controller about the agents, which is used to construct the Blackboard.
- The Agents object acts as a storage container for information (tokens, rule scripts and head information) about each of the agents in the system.

7.3 Systen Implementation

The following sections will present a discussion on the implementation of the prototype system. Firstly the development tools used will be presented, followed by an example of the IDL interfaces and CORBA client-server setup procedures. This is followed by a discussion on the implementation of the main system component and the knowledge base.

7.3.1 Development Tools

The development platform used was a standard Windows NT PC connected via a LAN to several Netware and UNIX servers. For the development of the system, three main tools were required: a Java compiler, a Java CORBA ORB and a suitable development environment.

There are currently several vendors of Java CORBA ORBs in the marketplace, but at the time of the design phase there were only two main Java CORBA ORBs available: Iona's OrbixWeb and Visigenic's VisiBroker. For the purposes of this research, both of the ORBs provided the necessary features for the development of the prototype. OrbixWeb was selected as the implementation ORB primarily because Iona technology are the recognised world leader in CORBA technology and also because of this researchers previous research connections with the organisation.

The compiler chosen was Sun's JDK (initially version 1.1.2) because of the problems associated with developing CORBA components which needed 100% pure Java, thus (at that point in time) ruling out compilers supplied by vendors such as Microsoft [Hunt, 98]. This was used in conjunction with the JBuilder environment which provided a graphical editing and debugging tool only, not a compiler.

7.3.2 IDL Interfaces

As discussed in section 7.2.3, there are three IDL interfaces, each of which is represented in a single IDL file which is compiled by the OrbixWeb IDL compiler. This produces several Java files that implement the CORBA communications facilities for the interface. The main elements that constitute a CORBA IDL file are:

- **Interfaces** - which define a set of methods (or 'operations' in OMG terminology) that a client can invoke on an object. Essentially this is a class definition without an implementation section.
- **Operations** - denote a service (method) that clients can invoke.
- **Data types** - denote the accepted values of CORBA parameters, attributes, exceptions and return values. These are named CORBA objects which are used across multiple languages, operating systems and ORBs.

To illustrate the contents of an interface, the following is an extract from the IDL file for the System Kernel - Agent Controller interface.

```
interface AdviceObject {
    readonly attribute long projectID;
    readonly attribute long scenarioID;
    readonly attribute long agentID;
    readonly attribute long adviceID;
    readonly attribute string advice;
};
typedef sequence<AdviceObject>AdviceTable_vector;
interface AdviceTable {
        readonly attribute AdviceTable_vector AdviceTable;
};
```

```
interface AgentController {
        oneway void startSession();
        oneway void createScenario(in long projectID, in long scenarioID);
        AdviceTable getAdvice();
};
```

The components of the IDL specification are as follows:

- **Advice object interface** - describes five data members (attributes) of the AdviceObject class, four of which are long and one string[4]. The keyword ‘readonly’ indicates the attributes cannot be directly modified by a client.
- **Advice table datatype** - This typedef statement defines an array (sequence) of AdviceObjects called ‘AdviceTable_vector’.
- **Advice table interface** - describes the one data member (attribute) of the AdviceTable object, which is an array (sequence) as above.
- **Agent controller interface** - defines three of the methods (operations) which are performed in the AgentController interface.
  - startSession() is used to instruct the Agent Controller that a tool session has started. This is a ‘oneway’ call, which means the client which makes the call (System Kernel) does not block while the remote object processes the call.
  - createScenario() instructs the Agent Controller to create a new scenario for a given projectID and scenarioID. The ‘in’ keyword indicates that the parameters are being passed from the client (System Kernel) to the server (Agent Controller). This is also a oneway call.
  - getAdvice() is used by the System Kernel to retrieve advice from the Agent Controller. The return value is an instance of the object AdviceTable, which contains an array of AdviceTable_vector as defined by the typedef statement above. As this is not a oneway call, the client (System Kernel) will block while this operation is being serviced.

---

[4] In IDL, a Java int maps to an IDL long and a Java string maps to an IDL string.

To illustrate how a method (interface) is implemented in Java on the server side, the following code segment shows the getAdvice() method call as described above.

```
class ACImpl implements _AgentControllerOperations {
        public AdviceTable getAdvice() {
                AdviceTable adviceTableRef = null;
                adviceTableRef = new _tie_AdviceTable(adviceTable);
                return adviceTableRef;
        }
}
```

This method call is defined in the ACImp class which implements the _AgentControllerOperations - a class generated by the IDL compiler, which implements the AgentController interface as described in the IDL file. This 'implements' approach to interface implementation is standard when developing Java based CORBA systems.

The getAdvice() method itself contains three steps:

1. Create a new reference variable adviceTableRef of type AdviceTable (as defined in the AdviceTable interface in the IDL file) and assign it to null.
2. Assign the reference variable to the previously created adviceTable variable by way of the _tie_AdviceTable() method (which was generated by the IDL compiler).
3. Finally return the instance of the AdviceTable reference.

To illustrate how the above method (interface) is called on the client side, the following code segment shows how the System Kernel uses the getAdvice() method:.

```
AdviceTable adviceTableRef = null;
adviceTableRef = ACproxy.getAdvice();
```

This call involves two steps:

1. Create a new reference variable adviceTableRef of type AdviceTable (as defined in the AdviceTable interface in the IDL file) and assign it to null.
2. Assign this reference to the results of the getAdvice() call. This call is made by prefixing the call with a reference name (proxy name) for the server (Agent Controller) on which the method call resides. This reference name (‘*ACproxy*’) is assigned when the CORBA server is first launched and a bind takes place. This notion of a bind is explained in the following section.

7.3.3 Client-Server Components

In order to service client requests on a particular interface, the server which provides the service must inform the ORB that it is available. This is done by initializing the ORB (using the ORB.init() method), creating an instance of the interfaces implementation class and informing the ORB that the server is available (using the _CORBA.Orbix.impl_is_ready() method). The Java code segment below shows how the Agent Controller server would perform this.

```
public class AgentControllerServer {
        public static void main(String args[]) {
                ORB.init();
                AgentController ACImpl = null;
                ACImpl = new _tie_AC(new ACImpl ());
                _CORBA.Orbix.impl_is_ready("AC_Server"),
        }
}
```

In order for a client to use the services of a server, it must first establish a link to the server. The following Java code segment shows how the System Kernel would establish a CORBA communications link to the Agent Controller.

```
public class SystemKernel {
        public static void main(String args[]) {
                ORB.init();
                String hostname;
                AgentController ACproxy = null;
                hostname = new String(_CORBA.Orbix.myHost());
                ACproxy = ACHelper.bind("AC_Server", hostname);
        }
}
```

The above procedure involves the following steps:

1. Initialize the ORB using the ORB.init() method.
2. Create a reference variable (proxy) to act as a pointer to the server.
3. Identify the host on which the server program resides. In this case it is assumed to be the same as the client program.
4. Bind to the server using the ACHelper.bind() method and specifying the name of the server and the host on which it resides.

Finally, all servers must register their presence to the CORBA ORB via an implementation repository, which acts as a database of mappings from server names to Java bytecodes. This allows the CORBA ORB to find the actual Java bytecode for a given implementation when a client binds to it. In OrbixWeb this is done by using the 'putit' utility with the parameters of server name and location.

7.3.4 Implementing System Components

The following tables summarise the development of a series of five prototype implementations of the system:

Heading	Description
Prototype	First prototype - codename 'Cavan'.
Purpose	At the time of implementation there was almost no commercial Java-CORBA development being conducted, therefore an architectural proof was developed to highlight any technical issues associated with using Java and CORBA.
Functionality	A skeleton of the four main CORBA components which operated by sending a series of messages to a DOS window indicating the calls. Contained no other functionality or graphical user interface.
Tools used	OrbixWeb 2.0, JDK 1.1.2 and JBuilder 1.0 as editing environment.

Table 7.1 - Prototype One

Heading	Description
Prototype	Second prototype - codename 'Noumea'.
Purpose	To implement the basic functionality of the system and put in place mechanisms for the entering and management of data.
Functionality	A small number of basic JESS agents developed. This version also included a crude graphical user interface.
Tools used	OrbixWeb 2.0, JDK 1.1.4 and JBuilder 1.0 as editing environment.

Table 7.2 - Prototype Two

Heading	Description
Prototype	Third prototype - codename 'Salonika'.
Purpose	To create the first functioning version of the system to be demonstrated to users as part of the validation exercise.
Functionality	An enhanced System Kernel, including file storage and scenarios. A complete GUI and the implementation of a small set of 'realistic' JESS agents which were capable of providing advice on scenarios developed by the user.
Tools used	OrbixWeb 3.0 (which contained the official OMG mapping), JDK 1.1.5 and JBuilder 2.0 as editing environment.

Table 7.3 - Prototype Three

Heading	Description
Prototype	Fourth prototype - codename 'Burgundy'.
Purpose	To create a more fully functioning version of the system. This version was the main subject of the user validation process.
Functionality	Additional number of fully functioning agents and the optimisation of the System Kernel, including the removal of a number of bugs.
Tools used	OrbixWeb 3.0, JDK 1.1.7 and JBuilder 2.0 as editing environment.

Table 7.4 - Prototype Four

Heading	Description
Prototype	Fifth and final prototype - codename 'Tipperary'.
Purpose	The creation of a pre-commercial prototype.
Functionality	Two main enhancements: the addition of a larger set of fully functioning agents and the removal of identified bugs in the system.
Tools used	OrbixWeb 3.0, JDK 1.1.7 and JBuilder 2.0 as editing environment.

Table 7.5 - Prototype Five

7.3.5 Knowledge Base Implementation

The knowledge base (agents) are structured according to areas of expertise which are represented by nodes in the Blackboard hierarchy as described in chapter 6. This hierarchy of advice areas is illustrated in figure 7.12 and shows seven main areas of expertise and a number of sub-areas, each of which may contain a number of agents.

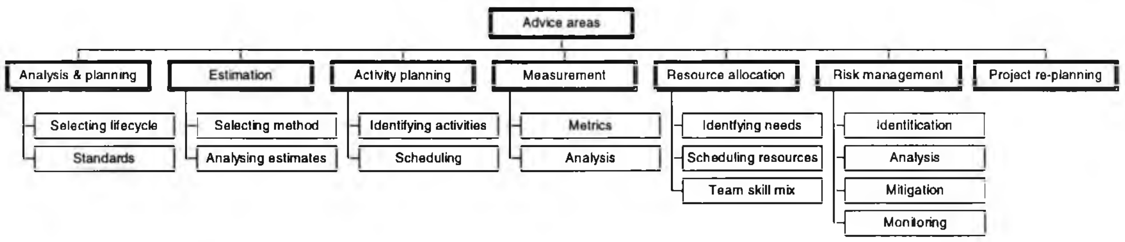


Figure 7.12 - Hierarchy of Knowledge Areas

For the series of prototypes previously described, four main areas of expertise were selected and a total of twenty five agents developed for those areas [O'Connor and Jenkins, 99a]. These areas are:

- **Lifecycle Selection** - covered advice on how to choose the most appropriate lifecycle for a project. This knowledge was elicited from several printed sources, including empirical studies such as [Alexander and Davis, 91].
- **Activity Planning** - assists with identifying activities that a project should involve, developing a schedule for the activities and the resources it will consume. This knowledge was elicited from several sources, in particular the SPIRE handbook [Sanders, 98] from ESPRIT / ESSI project 23873.
- **Risk Management** - A series of risk agents were developed to cover the four main areas of risk - cost, schedule, technical and operation risk. The primary source for these agents was the US Air Force risk management taxonomy [USAF, 88], as used in the RISKMAN2 project.
- **Measurement and Metrics** - covers advice on the selection of appropriate measures for a project to be used as indicators of product and process quality. The advice for this area was mainly provided by one of the partners in the AMI (Application of Metrics in Industry) project [Pulford et al., 96].

The structure of an agent was described in chapter 6 and contains three main components: Agent Header (identification / configuration data), Agent Tokens (data about a project) and Rules (JESS rule script). Table 7.6 shows the three sections for an agent which specialises in requirements characteristics of the activity planning area. Here four tokens (table 7.7) may be assigned one of three values which correspond to advice text (table 7.8) and associated SPICE (ISO 15504) recommended best practices [Sanders, 98]. These tables are taken from volume 2 of the Handbook and Training Guide [P3, 99] of the P3 project, which contains a complete set of decision tables.

Heading	Description
Agent Header	*; 1,1,2* *; Agent zero for Characteristics Requirements* *; Version 1.1a* *; 12,13,14,16*

Agent Tokens	*(deftemplate Agent0*
	    (slot _12) (slot _13) (slot _14) (slot _16))
	(Agent0
	    (_12 ?tk1) (_13 ?tk2) (_14 ?tk3) (_16 ?tk4))

Partial section of JESS rule script for tokens 14 and 16	*; token 14* *(if (= ?tk3 3)* *then (bind ?c "C.R.5 C.C.1 C.C.3 " crlf))* *(if (= ?tk3 2)* *then (bind ?c "C.R.5 C.C.3 C.C.1 " crlf))* *(if (= ?tk3 1)* *then (bind ?c "C.P.3 " crlf))* *(if (= ?tk3 0)* *then (bind ?c "" crlf))* *; token 16* *(if (or (= ?tk4 3)(= ?tk4 2))* *then (bind ?d "C.R.1 C.R.5 C.R.6 " crlf)* *else (bind ?d "" crlf))*

Table 7.6 - Example Agent

Token	High	Medium	Low
C.R.Complexity	C.R.1	C.R.1	None
C.R.Volatility	C.R.2, C.R.3, C.R.4	C.R.2, C.R.3, C.R.4	None
C.R.Inflexibility	C.R.5	C.R.5	C.R.5
C.R.Application	None	C.R.1, C.R.5, C.R.6	C.R.1, C.R.5, C.R.6

Table 7.7 - Token Values

No.	Advice	Recommended Activities
C.R.1	Allocate extra time for requirements analysis.	IA1, IIA2, IIA4, IIA6, IIA7, IIB1, IIB2, IIB3, IIB7, IIB8, IIC5
C.R.2	You need an extremely good configuration management system, especially for change control.	IIIB1-IIIB9
C.R.3	Prioritise your development so the parts with most volatility have a longer analysis period and the latest development slot.	IA1, IIA2, IIA4, IIA6, IIA7, IIB1, IIB2, IIB3, IIB7, IIB8, IIC5
C.R.4	Establish a good verification process.	IIID1-IIID4
C.R.5	Make sure you get the requirements right the first time with a strong requirements gathering process.	IF1-IF6
C.R.6	Try to prototype as much as possible.	

Table 7.8 - Advice Text

7.3.6 Knowledge Base Evolution

As described in section 6.3.7, the agent-orientated framework allows for a dynamic agent population, as the Agent Library builds a view of the set of available agents at the start of a session. This allows agents to be added or removed from the system by the addition or deletion of their corresponding agent file, thus providing for a dynamically updateable knowledge base, where knowledge (agents) may be added, revised or removed without impact on the rest of the system. Thus the knowledge base may grow over time to take account of new expertise or techniques in software project planning.

During the implementation phase of the prototypes a total of twenty five agents were developed, as follows:

- **Prototype 1** - contained five agents for test purposes only. They did not contain any real knowledge, rather they were used for testing component communication and data structures.
- **Prototype 2** - contained nine basic JESS agents, four of which were in the area of lifecycle selection and five in risk management.
- **Prototype 3** - contained fifteen agents, which included an addition of six activity planning agents.
- **Prototype 4** - contained twenty agents, which included an addition of three metrics and measurement agents and two further risk management agents.
- **Prototype 5** - contained twenty five agents, which included an addition of two more metrics and measurement agents and two additional activity planning agents.

As described in section 7.3.5, each agent contains three main components: Agent Header (identification / configuration data), Agent Tokens (data about a project) and Rules (JESS rule script). The process of constructing the prototype systems agents (as above) consisted of the translation of the decision tables from volume 2 of the P3 Handbook and Training Guide into the set of token value tables (figure 7.7) and the corresponding advice text tables (figure 7.8). These tables were subject to verification by the original author of the decision tables prior to the implementation of a set of JESS rules to implement the decision table. These rule scripts were then tested using the JESS interpreter and corrections made as appropriate. Finally the agent file was added to the Agent Library directory on the file system, where it would be automatically detected upon tool invocation. The process of creating a new agent (i.e. translating a decision model to JESS rule script) while not trivial, is a reasonably straightforward process, which can be completed within a short period of time.

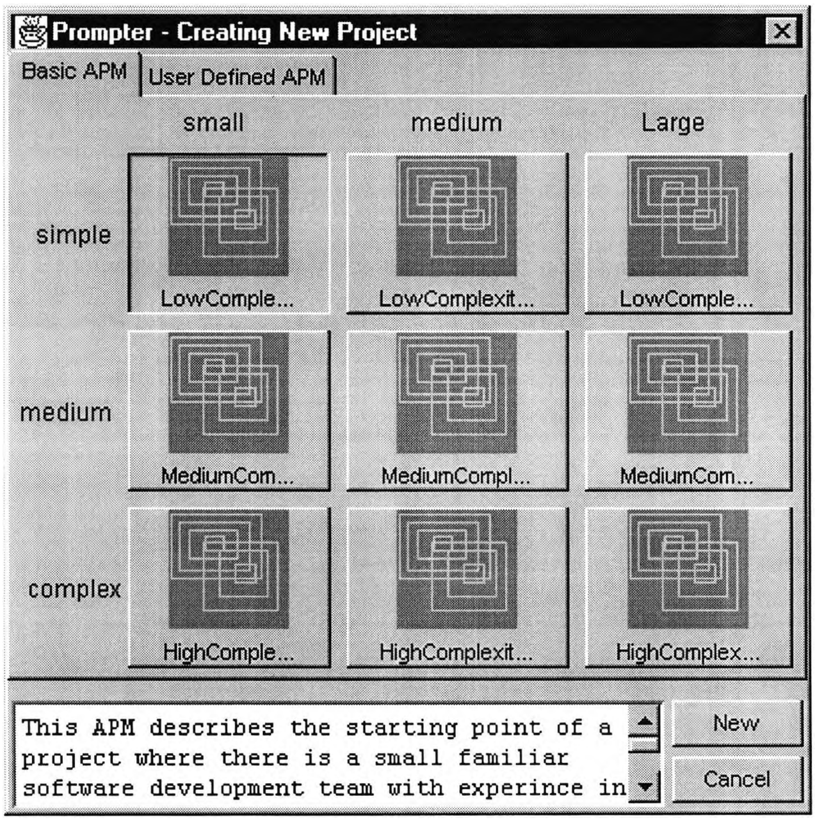
7.4 System Usage

The following two sections illustrate an example of a typical user session with the prototype system and provide an explanation of both the user level and system (component) level interaction.

7.4.1 Example User Session

This section illustrates an example of a typical user session with the prototype system and provides a narrative detailing the user level interaction with the system components.

A typical session is started by the user selecting a previous project or starting a new project. In the later case, they select New Project Workspace from the menu which displays the 'Project Model Selection' screen (figure 7.13). This screen allows the user to select the type of project model they wish to use, where the choice is represented by a grid showing project size versus complexity. For each model, a description is displayed in a text area at the bottom of the screen. When the user selects a model and clicks on the New button, a project workspace is created, along with its first scenario.



```
This APM describes the starting point of a
project where there is a small familiar
software development team with experince in
```

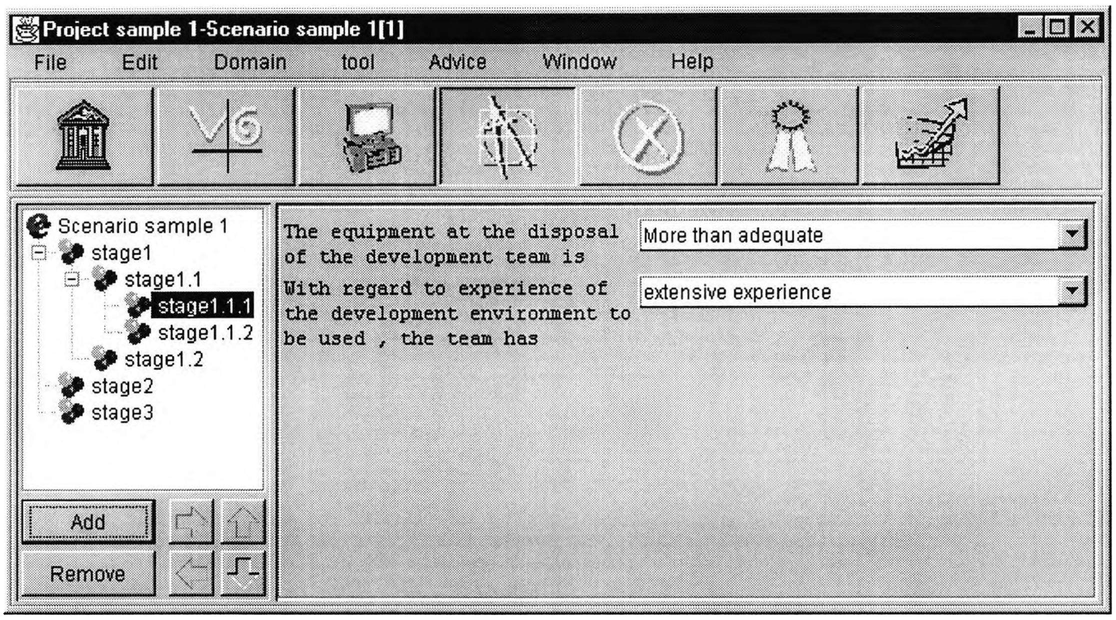
Figure 7.13 - Project Model Selection Screen

After selecting the appropriate APM, the system sets up the relevant default values for the project and then opens the main scenario window. The user now enters the second typical stage of a session, that of defining the project.

There are five main areas or domains in which information can be specified and choices made. Each domain is represented by a separate panel (screen), which can be selected either from the main menu or by using the toolbar buttons provided. Depending on the APM selected, many of these choices will be already selected with initial default values (any of which can be changed if required). The five domain panels are:

- **Characteristics panel** - the basic characteristics of the project such as requirements, customer, business drivers and project environment.
- **Project panel** - covers matters connected with the nature of the project itself, such as resources, estimation, schedule and cost.
- **Quality panel** - asks questions concerned with quality systems in the organisation, quality characteristics relevant to a planned product (or specified by the customer) and includes sub-domains such as organisation, product and customer.
- **Plan panel** - presents a list of standard stages making up the life cycle, and asks the user to specify some plan details about each. Each of these stages can then be further divided, if required, into sub-stages to allow for a finer degree of management of the project.
- **Metrics panel** - gives some recommendations for a minimum set of metrics for a project.

Figure 7.14 shows the 'Project Plan Panel' screen, with the project's tasks organised in a tree structure on the left hand side. Here the user has the ability to increase, decrease, re-order, add or remove tasks for a given project.

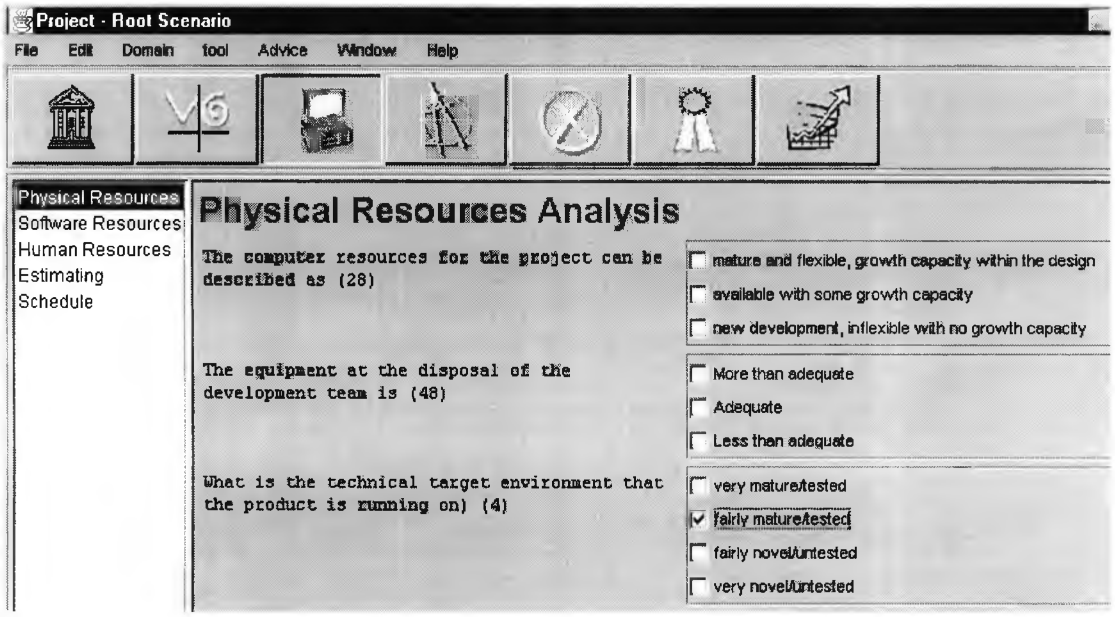


The equipment at the disposal of the development team is: More than adequate

With regard to experience of the development environment to be used , the team has: extensive experience

Figure 7.14 - Project Plan Panel Screen

Figure 7.15 shows a ‘question and answer session’ for a given scenario. The user chooses the most appropriate answer for their situation and thus communicates specific information about the project to the tool (and also to the agents).



Physical Resources Analysis

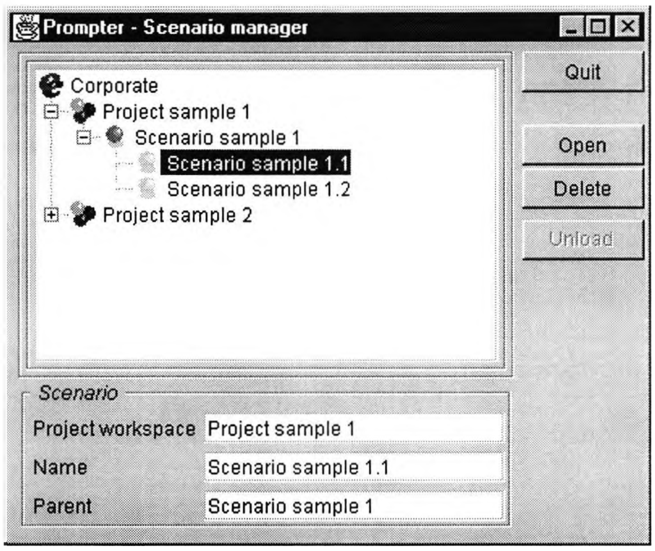
The computer resources for the project can be described as (28)
- [ ] mature and flexible, growth capacity within the design
- [ ] available with some growth capacity
- [ ] new development, inflexible with no growth capacity

The equipment at the disposal of the development team is (48)
- [ ] More than adequate
- [ ] Adequate
- [ ] Less than adequate

What is the technical target environment that the product is running on) (4)
- [ ] very mature/tested
- [✓ fairly mature/tested]
- [ ] fairly novel/untested
- [ ] very novel/untested

Figure 7.15 - Scenario Window Screen

Figure 7.16 shows the Scenario Manager window, which allows users select a scenario for further examination or delete an existing scenario. In addition, some information on the selected scenario is displayed in the bottom section of the screen.



Project workspace: Project sample 1
Name: Scenario sample 1.1
Parent: Scenario sample 1

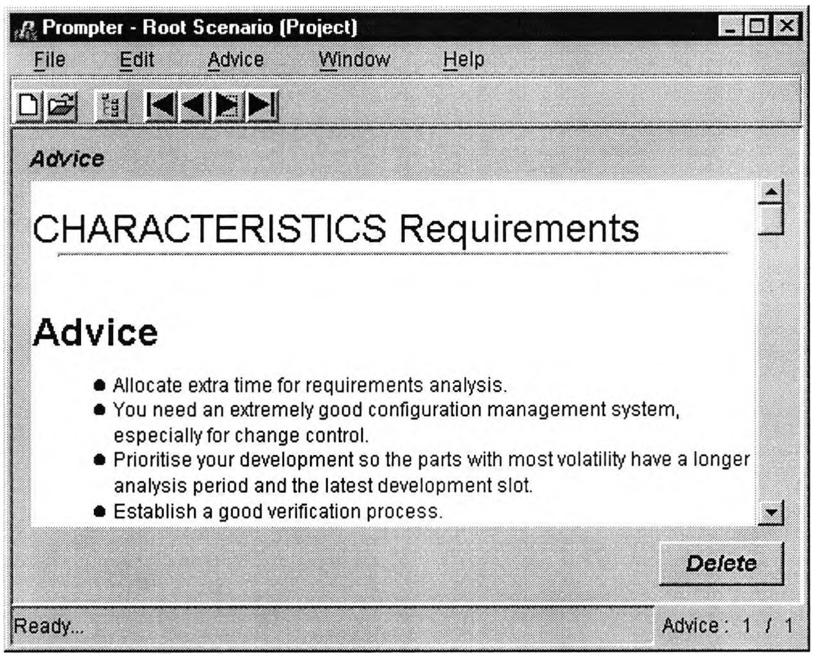
Figure 7.16 - Scenario Manager Screen

As the user continues to refine the project plan, the project parameters (tokens) are being continuously analysed by the agents. When advice is available, it is indicated on the bottom right of the main screen (see figure 7.17).

Figure 7.17 - Advice Counter

In order to view advice, the user selects the Advice Manager window (see figure 7.18) which in this case is displaying some of the advice text for the requirements characteristics agent above. This text is formatted using HTML and displayed using the Sun Swing GUI components. The user may choose to store, print or delete the advice.

The user may then choose to amend some aspect(s) of the project plan based on the advice received, which in turn may cause more advice to be generated. Thus the user enters a loop of refining project plans and exploring different project plan scenarios.


Prompter - Root Scenario (Project)
File Edit Advice Window Help
Advice

CHARACTERISTICS Requirements

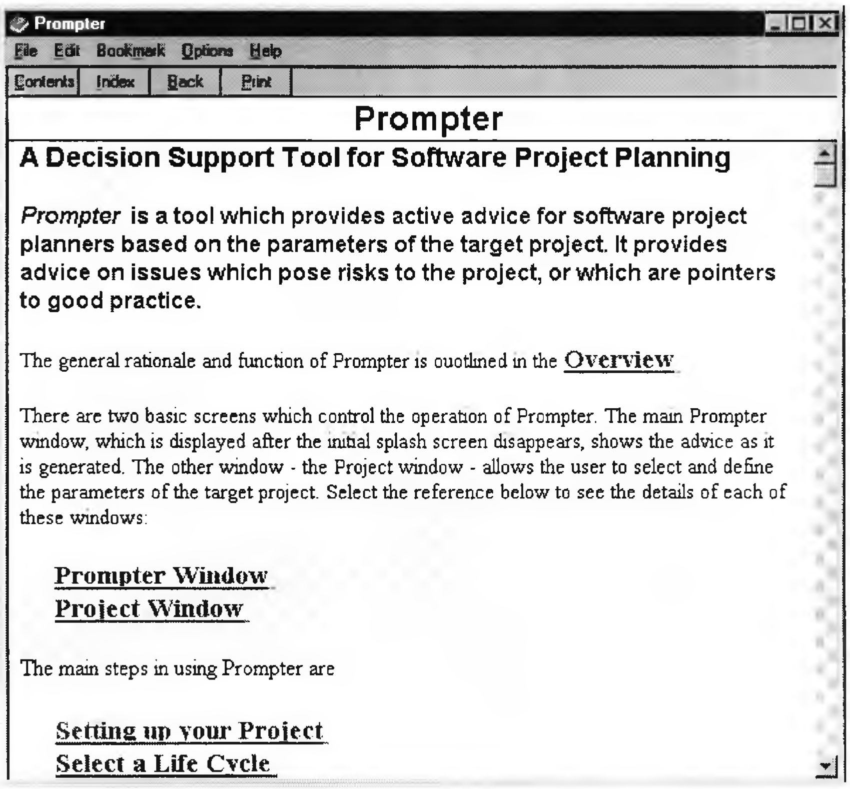
Advice

- Allocate extra time for requirements analysis.
- You need an extremely good configuration management system, especially for change control.
- Prioritise your development so the parts with most volatility have a longer analysis period and the latest development slot.
- Establish a good verification process.

Delete
Ready...
Advice : 1 / 1

Figure 7.18 - Advice Manager window

At any point, the user has access to the systems on-line help facility. Figure 7.19 shows the Main Help Screen Window. The final prototype included a limited number of help screens and a basic user manual.



Prompter

A Decision Support Tool for Software Project Planning

Prompter is a tool which provides active advice for software project planners based on the parameters of the target project. It provides advice on issues which pose risks to the project, or which are pointers to good practice.

The general rationale and function of Prompter is ouotlined in the Overview

There are two basic screens which control the operation of Prompter. The main Prompter window, which is displayed after the initial splash screen disappears, shows the advice as it is generated. The other window - the Project window - allows the user to select and define the parameters of the target project. Select the reference below to see the details of each of these windows:

Prompter Window
Project Window

The main steps in using Prompter are

Setting up your Project
Select a Life Cycle

Figure 7.19 - Main Help Screen

7.4.2 Example Component Interaction

This section illustrates the system level interaction between components which take place during a typical user session, as described in the previous section. In particular it focuses on the messages passed between the Agent Controller, Blackboard, Agent Library and Agents themselves, from the starting of the tool and a project through to the subsequent closure of both a project and the tool.

Figure 7.20 is an event trace diagram which illustrates an abstract view of the message passing between system components. Due to the complex nature and large number of messages passed between the various components and their associated objects, this diagram illustrates the message passing at a high-level abstract view and not at the level of object method invocation.

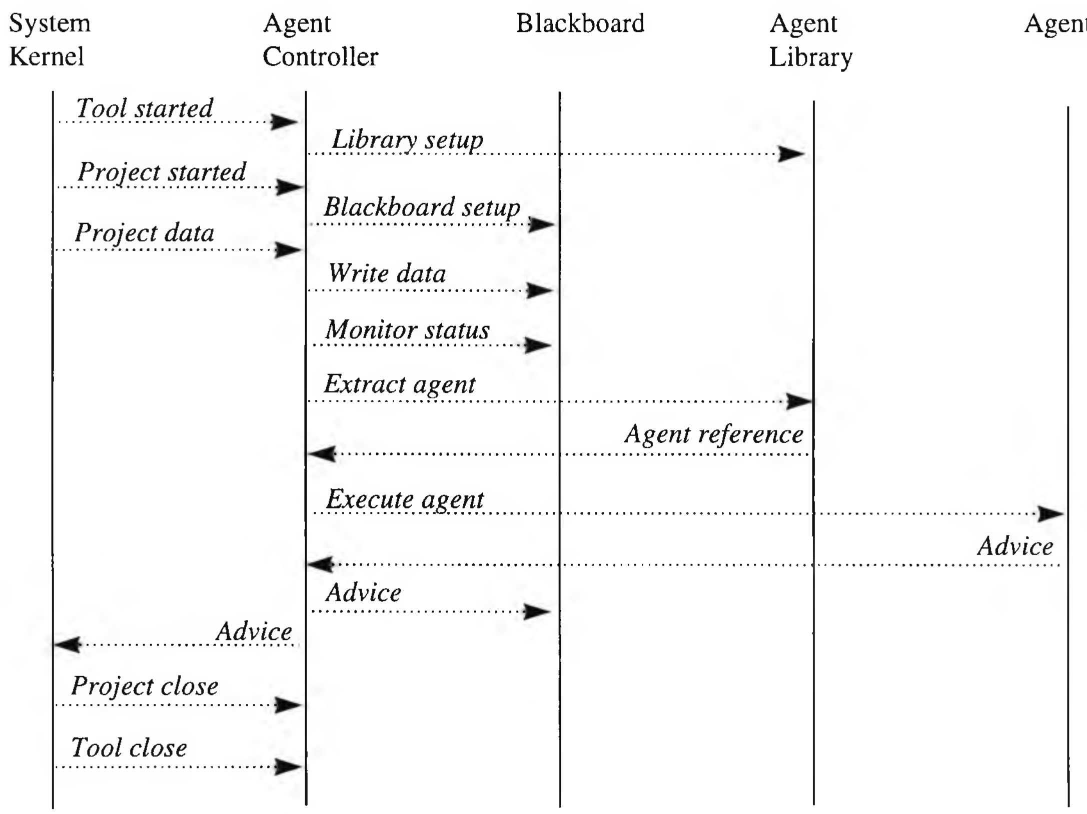


Figure 7.20 - Event Trace

- When the tool is started, the Kernel starts both the Agent Controller and Agent Library CORBA servers and initialises the Agent Controller.
- Upon being started, the Agent Controller initialises the Agent Library, which interrogates the file system in order to build a view of the set of available agents for the duration of that tool session.
- When the user creates a project via the user interface, the System Kernel informs the Agent Controller, which in turn initialises an instance of a Blackboard for that project, ie. it creates the Blackboard data structure for that project.
- At this point, the user enters a cycle of entering and refining project data (via the user interface) as described in section 7.3.6. The project data (tokens) are communicated by the Kernel to the Agent Controller, which in turn writes them to the appropriate Node, Segment and Slot in the Blackboard.
- As the user continues to enter data (which is recorded as above) the Agent Controller monitors the state of the Blackboard. When the necessary data (tokens) is available for an agent to execute, it is extracted from the Agent Library by the Agent Controller and executed using the Jess Engine.
- When an Agent has completed execution, it communicates its advice to the Agent Controller, which then forwards this advice (via the AdviceTable object) to the System Kernel for subsequent display to the user. The Agent Controller also notes the advice generation in the Blackboard.
- This sequence of component interactions will continue while the user continues to enter and amend data about a project, thus generating further advice.
- When the user closes a project, the System Kernel informs the Agent Controller. If the user has chosen to save the project under review, the appropriate contents of the Blackboard will be saved, along with other data under control of the System Kernel.
- When the user closes the tool, the System Kernel informs the Agent Controller, which shuts down the Agent Library CORBA server and the Agent Controller CORBA server.

7.5 Prototype Development Observations

This section will present some technical observations and lessons learned from the construction of the prototype system [O'Connor and Jenkins, 99b]:

- The CORBA imposed design sequence (defining IDL interfaces first) proved to be advantageous as it removed the possibility of ambiguity in the interpretation of the interfaces by the system developers, which was especially useful given the distributed nature of the development.
- Configuration management of IDL is particularly important as any change to an IDL definition must be replicated on both client and server sides.
- Not all Java structures are supported by the IDL mappings. For example; arrays of an undefined size are permissible in Java, but no mappings exist for such structures in IDL.
- Subsequent to the release of the official OMG IDL-Java mapping standard, all interfaces had to be revised to conform to the standard, which required changes to IDL and server initialisation routines.
- One of the main programming issues was the additional complexity in code due to CORBA. For example; synchronisation issues in relation to non-oneway IDL operations which complicated the introduction of Java threads.
- Debugging CORBA-Java programs is more difficult because standard debug tools are not capable of tracing a remote method call across an ORB.

At this point it is also worth noting some of the positive points in relation to the development of the prototype system:

- Java and CORBA work well together. This in the most part is due to Java's built-in multi-threading, garbage collection and error management which makes it easier to write robust network objects. Also, Java's object model compliments CORBA's as they both use the concept of interfaces to separate an object's definition from its implementation.

- For the reasons stated above, Java has proven to be a faster platform in which to develop CORBA based applications by comparison to C++.
- The incorporation of JESS proved to be a technical success in that JESS components were seamlessly integrated into the prototype system.
- The addition of new agents into the system proved to be a reasonably straightforward process, which was achieved without any re-configuration or alteration to other system components.

7.6 Summary

This chapter contains a description of the issues surrounding the implementation of the prototype system, including both the design and development of the prototype. In addition, some observations made and lessons learned from building the prototype system were presented.

Chapter 8 presents the field of research methodology and provides a review of a the approaches that are used within the field of computing and information systems. Further, it has described the approach which shaped the design user trials process.

Chapter 8 Research Methodology and Design

8.1 Introduction

This chapter gives a brief overview of the field of research methodology. It contains a review of a variety of research perspectives and approaches which are used within the field of computing and information systems and describes the methodology used in this thesis.

8.2 Research Methodologies

Research methods can be classified in various ways, however, one of the most common distinctions is between quantitative and qualitative research methods.

- **Quantitative research methods** - were originally developed in the natural sciences to study natural phenomena. Examples of quantitative methods now well accepted in the social sciences include surveys, laboratory-based experiments and simulations.
- **Qualitative research methods** - were developed in the social sciences to enable researchers to study social and cultural phenomena. Examples of qualitative methods are action research, case study research and ethnography. Qualitative data sources include observation and participant observation, interviews and questionnaires, documents and texts, and the researcher's impressions and reactions.

The motivation for pursuing qualitative research, as opposed to quantitative research, comes from the observation that the main characteristic which distinguishes humans from the natural world is the human ability to communicate. Qualitative research methods are designed to help researchers understand people and the social and cultural contexts within which they live. [Kaplan and Maxwell, 94] argues that the goal of understanding a phenomenon from the point of view of the participants and its particular social and institutional context is largely lost when textual data are quantified.

Cornford [Cornford, 96] offers the following warning in relation to the pursuit of qualitative research methods:

> *"Qualitative researchers are less certain as the possibility of the pursuit of a value-free, time and place independent, fact...the qualitative researcher, in seeking out the individuals experience and awarding it its own value, must accept a more subjective view of reality."*

[Archer, 88] suggests than within the broad heading of qualitative research there are three distinct rationales. First, there is a position that sees qualitative research as complementary to a quantitative approach and providing access to research questions that otherwise might not be accessible. The second position sees qualitative research as a precursor and poor relation, providing an entry point into new fields of study that may be subsequently treated by 'hard' approaches. In this way qualitative research provides reconnaissance and orientation before the main research effort. The third position is one which sees qualitative research as the only true approach, and a significant improvement on the 'pseudo-science' practiced by those who adhere to quantitative approaches. This view is based upon a notion of social science being in an immature or pre-science stage in which "empirical research cannot go beyond a sort of (natural) history, conducted in a disciplined fashion" [Archer, 88].

8.2.1 Research Methodologies in Computing

The question of how to undertake research within the discipline of computing and information systems is a topic that increasingly exercises the minds of those who work within this domain [Mumford et al., 84] [Galliers, 92]. There have been attempts to map out a broad research agenda for the discipline of computing [Boland and Hirschheim, 87] [Keen, 91]. The overall research endeavour in computing, as in any other discipline, involves many different styles and types of work. Loosely, these may be considered as two streams, one of 'theoretical' research and one of 'empirical' research. Theoretical research is concerned with developing and refining a body of abstract understanding of phenomena and issues. It may be undertaken through a purely mental set of procedures, though sometimes these will need to be fed with stimuli from outside sources. Empirical research, on the other hand, is work that concerns itself more centrally with observing events in the world (sometimes in a laboratory setting) and then seeking to 'make sense' of what is observed.

There are three distinct approaches to research within the domain of computing and information systems [Cornford, 96]:

- **Constructive research** - is concerned with developing frameworks, refining concepts and pursuing technical developments. The concern here is with 'models and frameworks which do not describe any existing reality, but rather help to create a new one'.
- **Nomothetic research** - is concerned with exploring empirical data in order to test hypotheses of a general character about phenomena studied. Such research emphasises systematic protocols and hypothesis testing within the scientific tradition.
- **Idiographic research** - is concerned with exploring particular cases or events and providing the richest picture of what transpires. Idiographic research emphasises the analysis of subjective accounts based on participation or close association with everyday events. In information systems there is a strong tradition of case studies which may be seen as idiographic research.

[Straub et al., 94] provide a classic taxonomy of styles of research within information systems:

- **Laboratory experiments** - These imply a research activity that is undertaken within controlled conditions. Within an experimental research design, the researcher manipulates some variables and observes the results.

Most often, data will be quantitative in nature and will relate to a limited number of phenomena.

- **Surveys** - a single survey provides a cross-sectional picture of affairs at a point in time. The basic technique may be extended to provide longitudinal data by repeating the process over time. Commonly, the researcher has to acknowledge that while a small scale survey can provide interesting information from a real population, it is not statistically representative.
- **Reviews** - one way to describe research in this category is to suggest that it looks backwards rather than forwards. That is, the researcher is concerned with charting the development of a set of ideas and with placing them within a descriptive framework. A well executed review of prior work can make its own research contribution by providing a more refined understanding of the theoretical and empirical work which has been done in a particular area.
- **Case studies** - a case study is an in-depth exploration of one situation. The strength of a case study is in the richness of data that can be obtained by multiple means when researchers restrict themselves to a single situation. This leads people to recommend the case study approach for topics and areas of study which are novel or which have little theory as yet [Cornford, 96]. The case study, in this way, might be seen as a preliminary research exercise out of which potential theories can be developed for subsequent validation through other methods.

The term 'research' itself may take on a range of meanings and thereby be legitimately applied to a variety of contexts [Eilon, 79]. Indeed, there is no single or commonly agreed approach to conducting research in the field of computing and information systems.

8.3 Choosing a Research Approach

Zuboff offers the following commentary in relation to the choice of a research methodology [Zuboff, 88]:

"Behind every method lies a belief. Researchers must have a theory of reality and of how that reality might surrender itself to their knowledge-seeking efforts. These epistemological fundamentals are subject to debate but not to ultimate proof".

The choice of method depends on many factors, in particular the nature of the research being undertaken and constraints on that research. As previously presented, research in the field of software engineering (as a specialism within the field of computing) tends towards qualitative research methods. This is due in the main to two influencing factors; the nature of the research and the time frame within which the research is conducted.

In general, software engineering research does not represent classical hypothesis testing to which empirical methods such as statistical tests can be applied [Cornford, 96]. Rather, software engineering research is more often concerned with the appropriateness of a methodology, technique or computer system for a given purpose. Such non-empirical methods usually yield large amounts of qualitative and anecdotal evidence which must be rigorously analysed to discover underlying trends. In addition, due to the dynamic and rapidly evolving nature of software systems and technologies, the time frame within which research must be evaluated is usually many times smaller than that of other domains. For example, in domains such as medicine, the evaluation of new drug therapy requires a longitudinal study over a number of years. By contrast, the domain of software engineering is so rapidly evolving that the time to market necessitates a shorter time period within which to evaluate the entity under study.

[Jeffery and Votta, 99] offer the following opinion on quantitative approaches to software engineering research:

"At this point in time, there is no widely held collective agreed model of the definition and role of empirical (quantitative) software engineering".

[Seaman, 99] advocates qualitative methods in software engineering research with the following commentary:

> *"The principal advantage of using qualitative methods in software engineering is that they force the researcher to delve into the complexity of the problem rather than abstract away from it . Thus, the results are richer and more informative...however, qualitative results are often considered 'softer' or 'fuzzier' in technical communities, but then so are the problems we study in software engineering."*

The research described in this thesis is typical of many software engineering research projects. By its nature it does not lend itself to empirical techniques of quantitative study, rather to the qualitative research methods. This presents the questions of which method, approach and style is more appropriate to this research and why. In order to satisfy the objectives of this research a number of issues must be addressed:

- An enquiry into the current state of art and practice in the software industry with regard to the usage of software project planning tools was necessary. The purpose of this enquiry was to assess the position of existing project planning tools, thus highlighting the potential benefits of incorporating intelligent assistance into project management tools.
- It was also necessary to develop some form of yardstick which may be used to assess the architecture developed in this thesis and measure (in some form) its appropriateness.
- Some appropriate form of evaluation of the prototype system was also required. This evaluation should have regard to a number of aspects, including the prototypes position with regard to existing project planning tools and users perception of the system itself and its benefits as an intelligent assistant system.

Each of the above issues has a corresponding impact in relation to the choice of research methodology:

- The most appropriate mechanism to assess the current state of art and practice in the software industry with regard to the usage of software project planning tools was to conduct a survey of software project managers who are users of such tools. It was not necessary to provide a comprehensive in-depth study of software project planning tool users, rather the purpose of the survey was to obtain an appreciation of the type of tools that are being used by project managers and to obtain a better understanding of the actual state-of-practice regarding these tools. As this survey is concerned with user opinion and perception, qualitative methods would be more appropriate than quantitative methods.
- In order to develop a yardstick against which to assess the proposed architecture, it is first necessary to conduct a review of the architecture of existing related systems. From this a set of desirable architectural characteristics can be derived, against which the proposed architecture may be compared.
- The most appropriate form of evaluation of the prototype system would be to expose the system to actual users, i.e. software project managers in commercial organisations. There are a number of options for an evaluation such as laboratory experiments, where a group of users are exposed to the prototype system in a controlled environment, or a case study situation where a 'real world' project is reconstructed and executed using the prototype system.

From the above it is clear that qualitative research methods are more appropriate for this type of software engineering research project. This leads to the question of which is the most appropriate approach to take. A mixture of constructive and idiographic approaches would be the most suitable, as the constructive approach is concerned with developing new frameworks and models for technical development and as such it fits with the task of developing a novel architecture to support the development of an intelligent assistant system. In addition, idiographic research approaches may be regarded as a supporting approach to the conducting of user trials for the prototype system.

The final decision on style of research has been addressed above with the use of surveys on existing tool users and reviews of software architectures. The user trials may be seen as a preliminary case study through which a decision may be made in respect to subsequent validation.

The overall approach to evaluation of this research may be characterised as a combination of participation and observation. The decisions in relation to the choice of research methodology for this work have been motivated by a number of factors, including practical considerations such as the time frame and available resources. It may be that the 'ideal' approach would be to conduct a more in-depth study over an extended period of time, with a larger number of people involved in the survey and user trials process.

8.4 Summary

This chapter described the field of research methodology and provides a review of a variety of research perspectives and approaches that are used within the field of computing and information systems. Further, it has described the research method, approach and style which shaped this thesis.

Chapter 9 contains an overview of the user trials process as well as details on the actual user trials themselves. A summary of the results of these trials and the feedback gained from users is then provided.

CHAPTER 9 PROTOTYPE DEPLOYMENT

9.1 Introduction

This chapter describes the trial usage of the system by a group of end users and presents an overview of the trial usage process and details on the actual trials. A summary of the results of these trials including the feedback provided by users is also presented.

There are several reasons for conducting user trials of the system. Firstly, it exposes the system to 'real world' project managers and obtains feedback from them in relation to the systems functionality and advice. In addition, it provides a mechanism to elicit opinion from users as to the added value of the system as compared to traditional project planning systems. It should be noted that it was not the purpose of these trials to provide a comprehensive in-depth test of the system, but rather to gain an appreciation of user perception of the system - its usefulness and added value.

9.2 Trial Usage Process

The user trial process was conducted in two main phases:

- **Phase 1** - involved the P3 member organisations using the prototype system as described in chapter 8. It was conducted over an eight month period and contained four sets of user trials - one for each of the major prototype releases.
- **Phase 2** - involved non-P3 organisations using the pre-commercialisation beta version of the system. It was conducted over a two month period and contained two sets of user trials.

The six trials are summarised in table 9.1 below:

Trial	Version	Users	Main Objective
1	Noumea	10 users from 1 P3 partner organisation.	Test the user interface with respect to data capture.
2	Salonika	6 users from 2 P3 partner organisations.	Test scenarios and generated advice associated with them.
3	Burgundy	11 users from 2 P3 partner organisations.	Test the total functionality of the system and quality of the advice produced by the system.
4	Tipperary	9 users from 2 P3 partner organisations.	Test the total functionality of the system and quality of the advice produced by the system.
5	Pre-commercial beta prototype	5 users from 4 non-P3 organisations.	Additional feedback from non-P3 project organisations.
6	Pre-commercial beta prototype - same version as used in trial 5	4 users from 4 non-P3 organisations.	Additional feedback from original tool users survey participants.

Table 9.1 - Overview of User Trials

Each of the trials consisted of three main steps:

1. A formal presentation given to the trial participants to introduce the scope and nature of the system.
2. A period of actual usage of the tool by the participating users from which a trial usage report was produced.
3. A review meeting involving all trial participants in which the trial usage report was discussed and analysed.

The actual usage of the tool consisted of one or more of the following steps, depending on the prototype version being used and the individual user:

- Creation of a fictitious project to allow experimentation with the system.
- Selection of a recent (previous) project and the creation of a project plan for it using the system, including a comparison of the two plans.
- Selection of a new or forthcoming project and the creation of a project plan for it using the system.
- Creation of a project plan for a supplied case study (see Appendix C).

At the end of each trial, the users were required to formally document their findings in a trial usage report and specifically to consider the tool under headings such as:

- Defining and refining a project plan.
- Creation and manipulation of project plan scenarios.
- Advice produced by the system while conducting the above.
- Suitability of the decision support framework provided by the system.
- Comparison in relation to other project planning tools used.
- Interaction issues such as GUI look and feel, etc.

The following six sections will present the main findings of the each of the trials. For each trial a high level view of its participants, duration and objectives is given. A summary of the main finding of each trial usage report which are pertinent to the objectives of this research are presented. For the sake of clarity, details on issues such as the GUI and installation program are not presented.

9.2.1 Trial 1

Trial	Trial 1 using Noumea prototype.
User details	Ten staff from various departments of one P3 organisation. These included software developers, project managers and quality managers.
Duration	Trial conducted over a three day period.
Objectives	• Assess user perception of added value of an intelligent assistant system as compared to traditional project planning tools.

	• Evaluate user understanding of the tokens used to characterise project modes in the system. • Test the user interface with respect to data capture.
Main finding of trial usage report	• Generally, users considered that the system appeared to provide a good framework for supporting decision making - although much of the detailed functionality had not yet been implemented. They considered the tool provided a novel approach to decision making and had the potential to be of use in a commercial setting. • They also noted that the tool was not a replacement for decision making by the project manager, but potentially a useful aid to support the decision making process. • Users suggested that the system could be used from two complementary perspectives depending on a persons job function: The first could be for the quality manager who defines an organisations good practices from those of state-of-the-art and locally used standards. The second could be the person in charge of a project who applies these defined practices. • The users considered that the overall success of the system was directly linked to the quality (suitability, understandability, etc.) of the advice it offered. Some participants noted that they would be disappointed if the tool only provided general or non-pertinent advice. • Users considered it important to have a balance between the requested information by the user and the automated provision of information so the user will not be swamped with information. • The perception of a number of participants was that the system could be viewed as a complementary tool to other existing project support tools, such as PSN6. It would therefore be desirable to have links between these tools and also to spreadsheet packages. • Generally the tokens (project parameters) were easily understood by users. A small number of them were not understood or were considered not to be valid in a particular project / product context.

	• The prototype was difficult for users to evaluate as it only contained a basic element of the main functionality of the system. Additionally, as the prototype only contained a small number of simplistic agents, users commented that they were unsure about the exact nature of the advice as it would appear in future prototypes.

Table 9.2 - Summary of Trial 1

This trial was very important as it was the first time the system was exposed to end users. Therefore, of principal concern was user reaction to the approach and framework provided by the system. The majority of users responded in a positive manner to the framework, however, it should be noted that the prototype system used in this trial contained only the basic functionality of the system. Because of this, some of the users comments may have been based on their perception of what the fully functioning system would provide from the perspective of the presentation made to them and the partially functional prototype used as the basis for the trial.

Many of the users specific comments were based on the fact that the prototype (as evaluated by them) only contained some of the main functional services to be provided. In retrospect, this may have been caused by inappropriate guidance during the initial presentation session. The specific comments and concerns of this group were considered valid and were taken into account in the development of the third prototype, although some requests, such as an export facility were considered to be outside the scope of the early prototypes.

The second trial session was based on a substantially more complete prototype - for example, scenario support was not included in the prototype for the first trial. This additional functionality allowed trial participants to get a more complete picture of the decision support framework provided by the system.

9.2.2 Trial 2

Trial	Trial 2 using Salonika prototype.
User details	Six staff from two P3 partner organisations.
Duration	Trial conducted over a two week period.
Objectives	• The main goal of this prototype was the development of scenario support, therefore the emphasis was on obtaining user feedback on scenario usage and associated advice generated in scenarios. • Assess users attitudes towards the nature and type of advice being offered by the agents. • Gain additional user feedback with respect to the user interface, having taken account of the changes made since the last prototype. • As with the previous trial, assessing user perception of the added value of the system was an important issue.
Main finding of trial usage report	• This review confirmed the report from the first trial that the system was generally considered to provide a good framework for supporting decision making and the system had the potential to be of use in a commercial setting. • Users considered a strength of the system was that it could provide guidance and assistance on the "good rules of planning" by providing a clearly defined way of building a project. • Users also remarked that the system acted as a "planning reminder", in that its approach provided a subtle mechanism to remind or prompt users about certain actions or issues to be considered in a plan which they may have inadvertently omitted. This facility may be of use to both new / inexperienced and experienced project planners. • In general, users considered that the use of scenarios to develop alternative views and paths through a project was useful. They considered it was useful to simulate variants of a plan and suggested that for most projects the capability to create five fundamentally different scenarios would be sufficient.

	• A number of specific comments and suggestions were made in relation to improving the mechanism by which scenarios are developed for a project. • Participants expressed a need for the system to offer guidance on the selection of the most appropriate lifecycle for a project. • Many of the users considered the advice given by the agents to be general and not sufficiently specific. Some of the users expected the advice to be presented in a more conversational format and make explicit reference to the specific values associated with a particular scenario. • Users requested that the advice displayed should be permanent (for the duration of a session) so users can see an advice history. The advice was considered by users to contribute to their overall understanding of a project and also assisted with the evaluation of risk to the project. • There were issues relating to a duplication of advice being offered during any one user session. Users considered this could be a distraction from important issues. • A small number of users commented that it would also be useful if some advice was expressed in more quantitative terms (for example, 70% probability of the project failing in a specific issue). • Some users expressed the desire for the system to provide support for the on-going tracking of a project during its execution and not just during the planning phase.

Table 9.3 - Summary of Trial 2

This trial was important as it was the first version of the tool to contain scenario support and a number of realistic agents. Further, this trial was conducted using a wider audience and over a longer time frame. The general comments of the users were very supportive of the concept of a support tool as they considered it a novel approach to planning by comparison to existing systems.

At the review meeting there was substantial discussion on the possible usage patterns of the tool. The consensus was that the system was most appropriate for medium to large projects or projects where the level of uncertainty was high. The real potential of the system was perceived to be as an aid to pre-planning, to assist the project manager "get a better mental picture of the shape of the project" prior to creating full and detailed plans, possibly using existing traditional systems. For this reason the ability to export project plan data from the tool was considered a priority issue.

Another important issue which arose during this trial was the desire for specificity and quantitative attributes of an agents advice. In particular, the issue of more quantitative advice - which could offer exact percentages or absolute numbers - in relation to a given situation was considered to be very important by a small number of users in one of the organisations. The possibility of offering more quantitative advice was discussed with the authors of the P3 Handbook and Training Guide and was considered to be very difficult and often not appropriate in many circumstances. For example, while it may be appropriate (and not unusual) for a risk management expert (agent) to comment that a particular project (or scenario based on a given project) was 70% likely to fail for a given reason, it would be unusual for an expert to comment that a particular lifecycle was 70% appropriate for a given project. However, as a direct response to this request, the scope of the search for suitable material - for use as the basis for agents - was widened to include more empirical studies which could be used to produce more quantitative advice.

The initial comments of the users in relation to performance and speed were considered to be a reasonably important issue, but given time restrictions it was considered more appropriate to use the majority of remaining time in the development of new agents, with a small time allocation for performance issues. Comments regarding the construction of scenarios were considered and some alterations made to the presentation of this information in subsequent prototypes.

9.2.3 Trial 3

Trial	Trial 3 using Burgundy prototype.
User details	Eleven staff from two P3 partner organisations which included most of the reviewers from trial 2.
Duration	Trial conducted over a three week period.
Objectives	• One of the principal goals of this prototype was to include a larger set of agents, therefore the emphasis of this trial was on gathering user feedback with respect to the advice being offered by the agents. • Get additional user feedback in respect of the construction and usage of scenarios - having taken into account the changes made since the last prototype. • As with both previous trials, assessing users perception of the added value of the system was also an important issue. • A secondary objective was the testing of a comprehensive automated installation program. This was an important issue as the development of an automated installation program for a distributed CORBA based system is a challenging process.
Main finding of trial usage report	• This review confirmed the report from the previous two trials, that users generally considered the system provided a useful framework for supporting their decision making process. • There were a large number of comments made about the type and content of the advice being produced by agents. These included issues such as the same advice being produced more than once in a session, ambiguity in the wording of advice and the relevance of advice not always being obvious. • A number of users (mostly from one organisation who employ a rigorous process and quality programme) again expressed the desire for (some) advice to be expressed in more quantitative terms. • In addition, suggestions were made that specific corrective advice

	should be given. For example, "...as your project is running 10% late, increase staff overtime". • The general user opinion was that using the scenario-based method to examine alternative paths through a project was valuable. However, there was a need expressed to have a mechanism to visualize the currently active paths / scenarios through a project. This arose as some users considered it was possible to "get lost" or disorientated when multiple paths existed.

Table 9.4 - Summary of Trial 3

The general comments in regard to advice (ambiguity, relevance, etc.) were dealt with by making minor changes in either the advice structure and format or trapping the execution mechanism to ensure that duplicate advice is either not produced or not passed to the User Interface. The revised advice presentation format (which including the ability to browse the advice history) was considered to by the previous trial participants to be useful and meet their expressed need. They pointed out that this allowed them to look back at the reasons/motivation behind decisions made and also to refer back to suggested best practices and advice when necessary.

Once again, the point of major concern of these user comments was the desire to have advice expressed in more quantitative terms. This request was again discussed at length with the authors of the P3 project Handbook and Training Guide in addition to a widened search for more empirical published studies. However, there was little by was of quantitative advice incorporated into the agents in this prototype as little (suitable) knowledge was elicited prior to the trial of this prototype.

There were a number of performance issues in respect of this version which were subsequently rectified. In particular, a change to use the "InProcess" method of server threads increased the general speed of the tool. However, the initial tool startup was still extremely slow, due in the most part to the delay in starting the ORB and launching the CORBA servers associated with the tool.

9.2.4 Trial 4

Trial	Trial 4 using Tipperary prototype.
User details	Nine staff from two P3 partner organisations and included most of the reviewers from trials 2 and 3.
Duration	Trial conducted over a two week period.
Objectives	• The main addition to this prototype was a larger and more comprehensive set of agents with a wider scope of expertise. Therefore, particular emphasis was placed on user reaction to the nature and type of advice being offered by the expanded agent library. • This prototype also included a limited number of help screens and a basic user manual, therefore user assessment of this material was also required. • As with previous trials, assessing user perception of the added value of the system as a decision support framework was also an important issue. • A secondary objective was user testing of the efficiency and reliability of the system as a whole. This information was particularly important from a commercialisation perspective. Additionally, the testing of an updated CD-ROM based install and uninstall utility was being monitored.
Main finding of trial usage report	• Again as with previous trials, users generally considered the system provided a good framework for supporting the decision making process and had commercial potential. • The users who had been involved in the previous three trials considered that the development prototypes to date had (in the most part) brought the tool in line with the expected functionality of a standard pre-commercial (beta or release candidate) system. • There were numerous suggestions made as to the possible usage patterns of the system, ranging from a project pre-planning tool to a project manager training tool.

	• There were a number of favourable comments made regarding users understanding of a project and its parameters and the quality of decisions subsequently made. Users commented that they "had a better feel for" and "understood potential danger areas" of a project. • The comments received in respect of advice being produced by the agents were generally favourable and were due to the expanded amount of advice being produced as a direct result of an increase in the number of agents in the agent library. • A number of specific comments were made about the type and content of the advice being produced by agents. Users appreciated the expanded amount of advice being produced by agents in this prototype version and also the reduction in duplicate advice. • As with two previous trials, there were more suggestions made with regard to the provision of quantitative advice. • Some participants expressed the desire to have a high level of customisability in terms of a process model, tokens, etc., as they found the predefined models a little restrictive. • The improvements in the appearance of certain aspects of the GUI which were made in this prototype were well received by users, in particular the alteration of a number of icons on the main screen which were now considered more intuitive.

Table 9.5 - Summary of Trial 4

During the review meeting there was substantial discussion on the possible usage patterns of the tool. As had emerged from previous trials, users considered it most appropriate for medium to large projects and not for small-scale projects. It was considered that the potential customer would be interested in using a tool which would help them "get an understanding of what the project required", or use the tool to "help with thinking aloud about different approaches to a project". This pre-planning usage was considered to be a reasonably unique aspect of the tool which could be used to complement existing traditional planning and management tools.

9.2.5 Trial 5

Trial	Trial 5 using pre-commercial beta prototype.
User details	5 users from 4 non-P3 organisations.
Duration	Trial conducted over a one week period.
Objectives	• The main objective of this trial was to expose the system to users who were not connected to the P3 project or P3 project organisations, thus getting feedback from a different perspective and from users who had not seen the tool evolve over time. • As with previous (P3 project based) trials, assessing user perception of the added value of the system as a decision support framework was an important issue. • Also of particular interest was user reaction to the nature and type of advice being offered by the expanded agent library. • A secondary objective was testing user reaction to the final system in terms of its commercial readiness, particularly in relation to issues such as install program, GUI look and feel, ease of use, help files, user manual and related product attributes.
Main finding of trial usage report	• The users considered that the system provided a novel approach to the creation of project plans and had potential to assist project planners with decision making. • They considered that the system could help project planners in "respecting good planning rules" and "remembering all the small things" about a plan. • It was suggested that the system may be of use to senior project managers to get "a view of what a project might look like" and therefore be better placed to understand the resources it may need. • Further, it was considered that the system had potential to "close the distance" between senior and junior project managers, as it could be used by the two managers to "sketch a basic high-level plan", thus leading to common agreement on a basic project plan. • Users felt it would be useful if the system provided a facility in

	which they could attach there own (free text) comment to certain stages of a project as a memory aid on a particular project aspect or decision made. • On a related point, some users expressed the desire to be able to annotate advice (with their own additional comments) created by the system and have that stored along with the system generated advice. • In terms of the advice presentation (advice window) some users commented that it would be useful to have the facility to save advice to disk or export it to a word processor. Also they thought it would be useful if the advice contained web like hyperlinks to the appropriate sections of the handbook and training guide or relevant web sites to assist with getting more information on certain topics. • The users placed great emphasis on the systems data export and reporting facilities. In particular they expressed a wish to have a data export connection to more tools that just Microsoft Project.

Table 9.6 - Summary of Trial 5

This trial was of particular importance as it was the first time the system was tested by non-P3 project users. The five trial participants were from four medium to large-scale software organisations. Two of the users elected to use the supplied case study, two used a recent project and the last developed a fictitious project.

The users responded positively to the system and considered it provided a novel approach to project planning and associated decision making, by comparison to existing systems. Much of the general feedback was in relation to the potential marketing strengths of the system - it was suggested that it would complement existing traditional planning and management tools and/or that it could be used to create high-level project plans by senior project managers in a multi-project environment as an aid to assigning personnel to projects.

9.2.6 Trial 6

Trial	Trial 6 using pre-commercial beta prototype.
User details	4 users from 4 non-P3 organisations who had participated in the earlier tool user survey. Only four of the original six surveyed were able to participate.
Duration	Trial conducted over a three week period.
Objectives	• The main objective of this trial was to ‘close the loop’ between the survey of tool users conducted prior to system development and the actual system developed. This provided feedback from a different perspective and from users who had not seen the tool evolve over time but who were familiar with the initial objectives of the research. • Another objective was assessing if the prototype system fulfilled the expectations of the survey participants, based on their understanding of what the proposal system was going to provide. • In line with previous (P3 project based) trials, assessing user perception of the added value of the system as a decision support framework was an important issue. • A secondary objective was testing user reaction to the final system in terms of its commercial readiness, particularly in relation to issues such as install program, GUI look and feel, ease of use, help files, user manual and related product attributes.
Main finding of trial usage report	• In general the users considered the system met their expectations in terms of provision of advice on project planning and the construction of project plans. However, some had expected the system to be more orientated towards the subsequent management and tracking of a live project rather than just planning. • They considered that one of the potential benefits / uses of the system was in allowing a project planner “walk around” a project and get a “feel for” or understanding of “what the final plan could look like”.

	• A criticism was that the system did not have any explicit / dedicated services directed at the planning of the software maintenance stage of a product or legacy system. It was suggested that the lifecycle of a maintenance project differs from that of systems development. • The users considered that the system could be useful in a wide range of projects and should have the potential to be tailored to different types of project. For example, they suggested that the agent library be expanded to include features for typical data processing applications, internet applications, embedded systems, etc. and provide a feature for the user to select the general classification of project they are working on and provide specific advice for that classification. • Likewise the users suggested that the agent library should contain a series of agents which were specialists in different standards, such as CMM, ISO 15507, etc.. The user could then select the standard they wished to work with and receive advice on creating project plans using that standard. • The users further suggested that the system should provide the ability to 'turn off' agents that a user 'didn't like' or 'didn't agree with'. • One of the potential strengths of the system which was identified was the ability to expand the agent library. It was suggested that of great potential benefit to an organisation would be the facility to create a set of company-specific agents (for internal standards, practices, etc.) and have these inserted into the agent library. This suggestion fitted closely with the intranet / network system deployment approach. • It was suggested that the system could be used in a 'training mode' or as a training tool to assist with training new project planning / management staff, or as an aid to transfer of know-how within a company or department.

	• Furthermore, they suggested that the system could be of potential use in an academic environment as a training tool which could be used to complement a project planning / management course.

Table 9.7 - Summary of Trial 6

This trial was important, because, as with trial five, the participants were also non-P3 project users. Furthermore, these users had participated in a survey of project planning / management tools and were aware of the initial objectives of this research. Of the original six survey participants, two had worked in P3 partner organisations at the time of the survey, but no longer worked for those organisations at the time of this trial. Furthermore, only four of the original six survey participants were able to be part of the trial process. Three of the users elected to used a recent project and re-create a project plan for it, while the final participant developed a fictitious project.

Overall the users response to the system was favorable and they made a large number of suggestions and recommendations. Of particular interest were their suggestions of several possible enhancements to the agent library and the functioning of the agents. Many of these are suggestions which should be incorporated into further releases of the tool.

One of the objectives of this trial was assessing if the prototype system fulfilled the expectations of the users based on their participation in the survey. Notwithstanding the suggestions noted above, the general impression of these users was that the system did provide the type of features they had expected. Further, they considered the system was commercially viable but probably not to small software organisations. Of particular importance from a commercial perspective was tool interoperability and data export facilities, which were considered to be of high level importance to potential customers. They also noted that the system should be marketed as a companion or complementary system to existing (traditional) project planning and management tools. In this respect they shared the views of the users in trial five that the system could be aimed towards the pre-planning or feasibility stage of a project.

9.3 Trial Usage Findings

The motivation for conducting the user trials was twofold. Firstly to test the tool in operation by project managers and secondly to assess user opinion of the added value of an intelligent assistant system. The trials were conducted in six distinct stages, one for each of the four major prototype releases plus two additional trials based on the pre-commercial prototype release. The initial four trials were conducted at an early enough stage during this research to influence the evolution of the subsequent prototypes.

For the first phase of trials a total of fifty person weeks effort over a six month period involving twenty two different participants was logged. These users represented project management staff from two of the P3 project partner organisations and represented a broad range of experience from novice to highly experienced.

The second phase of trials involved fifteen person weeks effort over a two month period involving nine different participants. These users represented project management and quality assurance staff from eight different organisations and represented a broad range of experience from relatively novice to highly experienced.

The main output of these trials was a set of review documents which detailed the comments and opinions of the users involved in each trial. To summarise, the combined findings of these reports were:

- **Decision support** - The general feeling of users was that the prototype system demonstrated that the notion of intelligent assistance for software project planning was feasible. In addition, they considered that the prototype implementation provided a suitable framework for supporting decision making and had the potential to be of use in a commercial setting.
- **Plan descriptions** - The general opinion of users was that the mechanisms of describing projects plans (via models and scenarios based on a model) was an appropriate and useful device to capture information about a project.

- **Scenarios** - Users considered the ability to create multiple scenarios to examine multiple views or a projects plan (with corresponding advice) to be very useful.
- **Advice** - Of paramount interest in these trials was user feedback in relation to advice produced by agents. The overall trend was that novice users considered the advice appropriate and useful as either a reminder of a particular aspect of planning or as an indicator of which direction to consider. However, more experienced users expressed the desire for more specific and quantitative advice.
- **Plan understanding** - There were many comments made such as “get a better mental picture of the shape of the project”, “get a better feel for a project”, “understand potential danger areas” and “remember all the small things about a plan”. These comments related to the project planner / managers understanding of a project and the parameters which influence the decisions made about it. A common theme in many user comments was that this aspect of understanding a project (either a specific project or general project planning) was increased.
- **Training tool** - A suggestion put forward by a number of users was the possibility of a repositioning of the system for use as a training tool in which users could develop a model of a fictitious project and thus practice project planning skills on a ‘virtual project’.
- **Deployment** - The prototype system was successfully deployed on a number of user machines via an automated setup program. Notwithstanding a number of small problems with ORB / JVM identification and access to the Windows NT registry, the prototype was successfully deployed by end users on a variety of machines.
- **Operation** - The prototype system was successfully operated by a number of users on a variety of machines. This was an indication that the prototype system was capable of being executed in a commercial environment, although the slow speed of execution in early prototypes was an important issue. However, users acknowledged that the speed issue was not of great importance for a research prototype, but would be for a commercial version.

- **User interface** - In general, users were satisfied with the GUI and its ability to handle data input. A large number of comments were made in the early stages and these were incorporated in subsequent prototypes.

One of the most difficult issue to tackle which arose during the user trials was the request for advice which was more quantitative in nature. This has proven difficult for two reasons; Firstly, little suitable source material was available which contained quantitative data / results that could be used as the basis for agents. Secondly, it is difficult for humans to discern the differences between quantitative values at a fine grain level with domains such as software project planning. For example, there is no appreciable difference between the values of 70% and 75% if they were expressed as a measure of suitability for a given lifecycle model. However, it is worth noting that this quantitative issue - while important in its own right - is not a central issue to the proposed architecture of this thesis. It is however an indicator of the nature of advice users perceive to be useful in addition to advice already produced.

Following from the series of users trials described in this chapter, the pre-commercial beta prototype system is currently being further enhanced - based on many of the comments made above - with a view to launch on the commercial tool market.

9.4 Summary

This chapter has presented a discussion on a series of user trials. For each trial the objectives were presented and the users comments discussed. In addition, this chapter also presented a brief discussion on the outcome of the trials.

Chapter 10 will present the conclusions of this research based on the proposed architecture, the construction of a prototype implementation and the series of user trials. This chapter will also propose directions for future work in all areas addressed in this research.

CHAPTER 10 CONCLUSIONS

10.1 Research Goals

It was the proposition of this research that there are a number of weaknesses in the current approaches being taken in the provision of software project planning tools and that there is significant scope to improve on existing systems in areas such as:

- **Creation of project plans** - few tools offer automatic guidance on the creation of technical and management plans. Of use to the project planner would be the automatic creation of an outline plan from specified (pre-defined) types of projects, which could be further refined to the particular project under consideration. In addition, users have to input large amounts of project plan data with little or no support for the accuracy, completeness and quality of the plan.
- **Decision support** - Most systems fall short of supporting the project planner in the decision making process. They do not offer assistance in representing knowledge about plans, or provide mechanisms for reasoning about plans in flexible ways.
- **Flexible Knowledge Base** - In the continuously evolving domain of software technologies, of great importance in the provision of automated decision support is the ability to dynamically update the knowledge base to cope with new and evolving standards and best practices.
- **Scenarios** - Although some tools offer 'what if' analysis in response to changing parameters, they do not offer direct 'recommendations' for action given a certain situation. Of use to the project planner would be the ability to create several different scenarios (or work breakdown structures) of possible future plan, based on the variation of a number of key project parameters, with associated decision support and plan verification. This would allow the project planner to examine alternate project plans from different perspectives.

- **Distributed cross-platform systems** - A further aspect to supporting the software project planner which is not addressed by today's support systems is the distributed and cross-platform nature of systems development. Users of existing software project planning systems could benefit greatly from support for the distributed cross-platform nature of modern client-server development.

This research set out to explore the role of artificial intelligence techniques in the provision of an intelligent assistant based software project planning tool which would address the issues described above. In addressing these aims, this research devised a framework and architecture which was used as the basis for the design of an intelligent assistant system. From this design a prototype system was implemented for use by software project managers in the planning of a distributed multi-platform software development project.

The main outcome of this was an architecture for an intelligent assistant system for use in software project planning. A prototype system was constructed to test the proposed architecture, and feedback from a series of end user trials by commercial tool users was evaluated to assess the usefulness and suitability of the system.

10.2 Research Outcomes

This thesis reviewed the main approaches to providing intelligent assistance and proposed that in the complex domain of software project planning, a useful tool to support the project manager would be a hybrid of a number of techniques - Decision Support System, Expert System, Blackboard and Intelligent Agents - encapsulated within an agent-orientated framework. This approach enables the inter-working of a variety of well understood techniques within a single underlying framework.

This research also reviewed the typical services currently provided by software project planning tools and conducted a survey of tool users to highlight user need for an intelligent assistant within a software project planning tool and to further identify

possible shortcomings in the services provided by existing tools. The results of this review and survey were an increased understanding of the needs of software project planners and the perceived deficiencies in existing tools. This survey also provided supporting evidence for the usefulness of an intelligent assistant based system.
This research conducted a review of a number of diverse architectures based around the agent-orientated paradigm and proposed a set of desirable architectural characteristics - which should be taken into consideration when developing an architecture for an intelligent assistant system - and compared the proposed architecture against these characteristics, demonstrating that the system satisfied these criteria.

Following this investigation, Java and CORBA were adopted as suitable solution technologies to implement the architectural properties of distributed platform-independent systems. Furthermore, JESS was put forward as a suitable agent language for the purpose of implementing a prototype system within a Java-CORBA architecture.

In order to validate this research, a prototype application based on the proposed architecture was developed. The successful construction of this application demonstrated that the architecture was implementable and could be deployed in a commercial setting. Furthermore, this facilitated the end user testing of the application which concluded that the system provided a novel approach to the creation of project plans and had potential to assist and support project planners with making project plans and the associated decision making.

10.3 Further Research

This section will briefly outline some of the research directions which could be further pursued as a result of the research reported in this thesis. This list is not intended to be exhaustive.

10.3.1 System Architecture

The agent-based architecture described in this research provides a novel, open, flexible and adaptable approach to the implementation of an intelligent assistant system for in software project planning. However, there are a number of areas which could be further investigated.

The User Interface is a light-weight system component which handles the management of all the screen elements (menus, dialog boxes, etc.), validates data entered by the user and passes on clear functional messages to the rest of the system. The User Interface component is an ideal candidate for incorporation into a web browser, as the existing functionality could be re-implemented using Java applets. This would provide for a thin network client executing in a web browser with the remainder of the system located elsewhere on the network, thus extending the system to any client platform for which there exists a web browser.

The Agent Controller is a supervisory unit over the agent community which manages the scheduling and execution of agents. It contains the inference engine and related modules for the JESS system. This approach of allowing a separate inference engine for each agent language allows for a number of distinct inference mechanisms to be used, in which case they would reside as separate sub-components of the Agent Controller. A natural extension to this architecture would be to extend the number of inference engines, possibly to consider some of the other languages and knowledge representation schemes.

Agents are located in the Agent Library, but remain under the control of the Agent Controller, whose purpose is to manage the physical agents themselves and to service requests for agent interactions. This architecture allows for the updating of the knowledge base by adding, updating and deleting agents in the Agent Library. There are a number of strategies which may be employed to update the Agent Library, with the most promising being via the Naming and Trader service of CORBA. The CORBA Trader Service provides an 'advertising directory' for CORBA objects, which allows objects to publicise their services and bid for jobs, whereas the Naming service provides a ‘name directory’ for objects, which allows applications to look up objects by name. Using these two CORBA services it would be possible to maintain an Internet site (single or multiple) which contains new agents and has a situation where the Agent Controller could interrogate the CORBA ORB to identify new (remote) agents. These agents could then be acquired and inserted into the local Agent Library and possibly be subject to an electronic commerce style of payment for such new agent.

10.3.2 Knowledge Base

The knowledge base for this intelligent assistant system is encapsulated in a series of intelligent agents which are located in the Agent Library. In the previous section, a number of proposals for further work have been put forward. In addition there are also other knowledge-orientated aspects which warrant further investigation.

For the purpose of this research it was considered that the particular method of knowledge acquisition employed was not of primary concern. Rather, the method should elicit a sufficient quantity of useful data which could be used to construct agents. This research has used a combination of printed sources and informal interviews as the primary method of knowledge acquisition. At this point it is worth considering the appropriateness of these methods of knowledge acquisition within the context of this research. It would be useful to explore other methods of knowledge acquisition as well as the broader consideration of the knowledge lifecycle. For example, given a large set of agents (and thus elicited knowledge), some form of knowledge management procedure would be necessary. Another aspect of the knowledge lifecycle which would benefit from further consideration would be the validation of elicited knowledge prior to its incorporation into agents.

In relation to the knowledge base, the main issue which arose during the user trials was the request for advice which was more quantitative in nature. During the development of agents as part of this research, the request proved difficult primarily because little suitable source material was available which contained quantitative data / results which could be used as the basis for agents. In addition, a barrier to developing such material would be the difficulty which humans have in discerning the differences between quantitative values at a fine grain level for less specific domains such as software development. For example, there is no appreciable difference between the values of 70% and 75% if they were expressed as a measure of suitability for a given lifecycle model. However, this request is an indicator of the nature of advice users perceive to be useful in addition to the advice already produced by the system. It is therefore worth investigating the issues surrounding the provision of such quantitative advice in conjunction with the previous issue of knowledge acquisition and validation.

The prototype system developed as part of this research contained a relatively small number of agents which operated in a small number of potential advice areas. To provide a more complete knowledge base a larger set of agents, covering a larger number of areas, would be necessary. For example, a series of agents dedicated to standards such as ISO 15504, CMM, etc., would be useful for users operating within those standards or for those considering adopting such standards.

The task of constructing agents (as described in this thesis) is a manual procedure and consists of first writing the agent header and subsequently constructing the agent rules in a suitable language - in this case JESS. Of potential interest would be automated tool support to assist agent developers (possibly user-organisations) in constructing their own agents. For example, the RISKMAN2 project (cf. section 4.2.3) developed an associated product, "Daemon Writers Pack" [Power, 94], to assist experts in risk management write Daemon's (RISKMAN agents). It consisted of a series of forms with a strict set of heading / formatting guidelines which acted as a generic rule template from which a developer would code a C++ Daemon. A major limitation of this approach was the complex implementation issues surrounding the translation of the text based forms into C++ Daemons.

To date all agents developed for the prototype system have been manually coded following the process described in section 7.3.6. The translation from a decision table to JESS rule scripts is a reasonably straightforward process, as JESS IF-statements follow a clear and simple format. It should therefore be possible to create a 'user friendly' GUI based tool which allows a user to enter conditions (dependent token values) and associated actions (advice to be generated), and produce a JESS rule script in the appropriate agent format. As part of the P3, project an experimental prototype of a 'Agent Developer Kit' was developed using Visual Basic, which was capable of taking simplistic conditions / actions and producing basic JESS scripts. However, this was not pursued further for commercial reasons.

It should be possible to develop a series of add-on tools for end users with features for automatically generating agents via a GUI based system and the editing of existing agents. Such a system could also implement Agent Library housekeeping and other procedures. Such a tool would be of benefit to users interested in creating a set of agents dedicated to internal company standards or practices.

10.3.3 Prototype System

For the purpose of this research, issues such as the execution speed of the system were not of primary concern, although a number of measures were taken to address such issues. However, a number of issues remain, mostly in relation to CORBA ORB initialisation, the launching of server objects and the presence of the OrbixWeb debug window. Successfully addressing these issues, would result in a subsequent prototype system operating at more acceptable levels (from a commercial perspective) and thus be in line with users comments received during the trials.

The addition of extended functionality within the prototype system, in particular enhanced scenario analysis and the provision of a larger set of agents would lead to a pre-commercial prototype system which could be used as the subject of further user trials to assess the commercial viability of an intelligent assistant system for software project planning.

During the users trials there was the suggestion of repositioning the system for use as a training tool, in which users could develop a model of a fictitious project and thus practice project planning skills on a 'virtual project'. There are a number of possibilities for such a repositioning which are worthy of further consideration, which include the possibility for an interactive (possibly web based) version of the P3 Handbook and Training Guide with a slimmed down version of the system acting in a project simulator / advisor role.

This research set out to examine the provision of an intelligent assistant system within the domain of software project planning. However, it should also be possible to extend this approach to the parent domain of software project management and the associated tasks of managing and tracking a software development project.

10.4 Concluding Remarks

Due to the growing complexity of software systems and the current economic context, software projects are facing more and more constraining production objectives in terms of time, cost, quality and risk. This evolution in the nature of projects being undertaken by software organisations results in increased difficulties associated with planning, managing and executing software development projects. Indeed, software projects often fail because the project managers lack knowledge of good practices and effective processes which can reduce risk and increase the likelihood of success.

Existing project planning tools do give support to the project manager and have several basic strengths; planning calculation and re-calculation; recording progress and feedback data; comparison of planned against actual achievement and re-calculation of the plan in relation to progress update. These reflect the strengths of data processing by computers applied to project planning. However, such tools do not provide support for the project manager in the decision making process and do not offer assistance in representing domain knowledge about plans and designs or provide mechanisms for reasoning about plans and designs in flexible ways.

This research has proposed an agent-orientated framework as the basis for an intelligent assistant system for use by software project planners and designed a novel architecture upon which this system can be constructed. This architecture is based on a fusion of a number of techniques within a multi-agent framework which aims to improve the quality of the decision making process of software project planners. This framework incorporates the information gathering and analysis techniques of a Decision Support System with the ability of an Expert System to propose possible solutions using expert knowledge and best practices and the power of Blackboard to exchange information between components. This novel approach enables the inter-working of a variety of well understood techniques within a single underlying framework - that of the agent-orientated paradigm.

To assist with validating the proposed architecture, a prototype application was developed and a series of trials conducted. The conclusion of these trials was that the prototype system demonstrated that the notion of an intelligent assistant system for software project planning was a viable concept, worthy of commercial investigation. Further, it demonstrated that the proposed multi-agent framework provided a viable architecture for supporting decision making which has the potential to be of use in a commercial setting.

This research is a significant step forward in the development of a new generation of software project planning tools. The approach described in this thesis is a fusion of a number of well understood techniques within a single unifying framework. An important characteristic of this approach is the combination of these techniques in an open distributed environment with the potential for continuous evolution.

APPENDIX A SUMMARY OF P3 PROJECT

Project Acronym: P3

Title: Project and Process Prompter

Objective: The P3 project intends to develop a prototype tool, Prompter, which can be used to assist project planners in deciding: What resources will be needed for their project? What parameters need to be measured during the course of the project? What trade-offs in project variables could lead to reduced risk and greater chance of success?

General Information: Prompter has the potential to satisfy a need in the software industry for greater understanding of the options available during planning and why one choice should be made over another. For example, certain criteria have been established for selection of one life cycle rather than another; these criteria are not always well-understood, nor are the ramifications of that selection. Prompter will give the project planner the opportunity to input project goals and certain project-specific variables, match them against a generic model to create a specific project model, than analyse a set of options which may be used to organise the project so that it will meet its goals. Part of the project model will be definition of measurements that should be taken during the project and submitted as input to Prompter so that the original choice of options is either validated as correct, or possibly needs changing.

The final deliverable of this project will be a pre-commercial tool which will be further "productised", then marketed initially to the software industry. Prompter will be a stand-alone Windows-based software tool; it will run on any version of Microsoft Windows later than Windows 3.1 since this is where the major market opportunity exists. The user interface will be standard in that pull-down menus and other well-understood features will be used. In addition, this basic user interface will be enhanced as far as is feasible with advanced visualisation and dialogue techniques to: improve understanding of the underlying tool complexities; allow a user perspective rather than a purely technical perspective; and support teaching as part of project and process planning.

Start Date: 01-Sep-1996

End Date: 28-Feb-1999

Duration: 30 months

Programme Acronym: ESPRIT 4

Subprogramme Area: Emerging Software Technologies

Programme Type: 4th FWP (Fourth Framework Programme)

Prime Contractor

Organisation: Catalyst Software

Organisation Type: Industry

Org. Country: Ireland

Other Contractors

Organisation Name: Dublin City University

Organisation Type: Education

Org. Country: Ireland

Organisation Name: Objectif Technologie

Organisation Type: Industry

Org. Country: France

Organisation Name: Intracom Sa

Organisation Type: Industry

Org. Country: Greece

Organisation Name: Schneider Electric Sa

Organisation Type: Industry

Org. Country: France

APPENDIX B SURVEY OF TOOL USERS

Each project manager was asked a series of questions, ranging from the highly specific to the more generic. The questions were divided into the following categories:

- General background questions regarding the person and employer.
- Questions to ascertain the type of projects normally undertaken.
- Questions about methodologies, standards and development tools used.
- Questions about the use of project management tools.
- Questions regarding the potential benefits of an intelligent assistant system

Each project manager was asked a series of questions aimed at finding out what (if any) tools were being used, the manner in which they were being used and the usefulness of certain types of features. Each manager was also asked to consider the proposal of an intelligent assistant in the context of a software project planning tool and offer an opinion on the proposed features. The following list shows a sample of the type of tool-orientated questions that was asked of each interviewee:

- What project management tools do you use?
- What exactly do you use the tool for?
- What tool feature(s) do you find most useful?
- What tool feature(s) do you find least useful?
- What feature(s) do you think they lack?
- What was the most successful project you have worked on, and could tool support (intelligent or otherwise) have made it better?
- What was the least successful project you have worked on, and could tool support (intelligent or otherwise) have made it better?
- Do you think intelligent assistance could help you in managing your projects?
- What features do you think an intelligent assistance tool should provide?
- What do you consider such a tool could offer the first-time project manager?

APPENDIX C CASE STUDY

Introduction

Accurate Data Ltd is a company which develops and maintains inventory software (***MEDIN***) for hospitals and medical supply companies. The original system which has to be maintained is based on an Oracle database and DEC equipment and has a decidedly old-fashioned mainframe look to it. AD is planning a project to:

1. move to a client server environment using DEC servers and workstations
2. update the user interface to include graphics
3. provide running inventory control with scanners

AD has not had too many competitors until a few years ago, but now they must make these changes or lose their market share and probably go out of business. Newer systems are already Y2K compatible and one competitor is ISO 9001 certified. In the medical business this is starting to be important. AD has managed to keep their customers in spite of this because of frequent and regular presence at customer sites to understand any problems and keep au courant of the customer business needs. Software emergencies are dealt with quickly; regular maintenance releases deal with non-emergency bug fixes and minor upgrades. AD has the best reputation in the business for customer service, but that is not enough any more.

The AD customer base is two large hospitals, 32 medium sized hospitals, and 30 large clinics. The older product is sold in three 'sizes' to suite the differences in these markets. AD would like to take advantage of the new changes to move to only one product. Old customers can upgrade for a graduated fee, depending on their size. One of the medium sized hospitals has a specialist (Grace Ibeza) who is interested in being part of the development team for the new product, but most of the other users just meet in a User Group meeting once a year to formally submit suggestions for improvement and any complaints. AD has promised this User Group that a major upgrade will be released every 6 months with the entire project to be finished (and

Y2K compliant) by the year 2000. AD realises this is a very tight schedule, but they feel there is no choice. Somehow they have to make the schedule. AD has no long-term debt and could borrow money if they need to, but of course the less borrowed, the happier the Managing Director (Corbin Corvette) will be.

AD is not interested in being on the bleeding edge of technology. They want to use tested equipment that is upgradable and likely to be around for the next five years or that can easily be replaced with other brands if necessary. In the past, they satisfied these needs by using transportable COBOL. Now they are interested in a language like Java or something else which can cross platforms.

AD has 5 software employees:

- Jean Rolls, senior software engineer who has been with the company since its beginning,
- Mercedes Carrumba, system designer who is only a few years out of University but has a very good knowledge of software engineering in general and the languages and platforms being used at Accurate Data
- Haley Ford and Sue Subaru, junior software engineers who mostly code/test/document
- Tonya Trap, part-time student who is just starting out

In addition, there is a part-time software manager (Kelly Volvo) who is also supporting sales and a full-time sales manager (Volks Polo) who gets the customer requirements and does the acceptance testing of the products before they go to the customers.

This is a massive job compared to anything they have ever tried and the AD personnel are not sure how to begin the work. What do they do now? Can *Prompter* help?

Part I A bird's eye view of this project

I-1 When to start

The schedule is defined (18 months), the budget is flexible, although Corbin would like to see a solid estimate before they begin. The decision has already been made to do the following:

- No more fixes to the old system except for an emergency. The software is very stable and this should be possible so that the entire group is working on the new system.
- Document the processes the software people follow now, the location of hard copy/soft copy data that might be useful for employees, and generally 'clear the decks for action'.
- Hire from the outside a project manager who's sole responsibility is to make this happen, co-ordinating the new software releases through Kelly Volvo and Volks Polo. Corbin Corvette is talking to Grace Ibeza from Mercy Hospital about becoming the project manager and she looks like she will be hired.

I-2 Objectives of the project manager

Since the project manager hasn't been hired yet, Kelly and Volks worked together to develop a preliminary schedule. The initial tasks to document the current processes, inventory what exists and where it was located were scheduled to take place first and these did not require the project manager at first. Based on what had been done in the past, the activities were defined but only loosely scheduled. When Grace arrives in 4 weeks, she can put more detail to the schedule.

I-3 Where are the risks?

- Technical risks do not seem too high here; although nobody is an expert in Java, Jean and Mercedes both have a good background in other languages. They are not concerned about the client/server environment since Mercedes worked in such an environment while at University.
- Cost isn't the highest risk, although such an ambitious set of goals may have unknown and unexpected costs.
- Schedule is the highest risk, particularly for Y2K issues.
- Operational risk is also high since two of the three upgrades (graphics and scanners) require an new way for the user to view the system; they may not like what AD produces.

Grace will have to include risk management in the project plan and monitor the risks carefully during the project.

I-4 Making decisions

Grace has already started working on the project management plan, including:

- risk management
- control of processes and corrective actions when problems occur
- communications links to the customers, Corbin Covette and the rest of the staff
- analysis of options for make or buy decisions
- quality management
- design and development planning

Grace has experience with ISO 9001 and knows that these cover some of the key areas in that.

I-5 Managing expectations

Grace knows it is important to communicate to everyone the information they need to know. So she starts by proposing a public display of the schedule, metrics which show planned vs. actual work completed, and high-level bullet points of decisions made or project status, to be updated at the end of every day or two. This will provide the raw data to Volks and Corbin so they can communicate with the customers, the media, measure status against the promises made to the User Group. In addition, Grace plans a daily 'stand-up' meeting with the development staff to discuss any issues and prevent problems from growing too large. Other meetings will be held formally and informally, and technical and management reviews included on the schedule.

I-6 Managing change

Grace is not worried too much about organisational change because AD is such a small company, but she is very conscious of how dependent she is on each person in the staff. Everyone has their special skills that would be difficult to replace. Enough time will be scheduled for reviews and audits so that at least two people are aware of all parts of the new system.

Grace will depend very much on Kelly and Volks to keep up with customer requirements and needs. Grace comes from a hospital herself so she knows the business, but only at one place. She plans to go with Kelly and Volks on some of the customer visits so she can see for herself the differences and commonalties in the customer base.

I-7 Managing quality in your project

This is somewhat difficult to manage. Grace knows the processes which can be included to help ensure quality but AD is a small company and has always relied on the personal commitment to quality of its people. Is that good enough for this project? Nobody is quite sure how to set 'quality targets' or even define when is good, good enough?

REFERENCES

[Agha et al., 93]
G.Agha, P.Wegner and A.Yonezawa (Eds.), "Research Directions in Concurrent Object-Orientated Programming", MIT Press, 1993.

[Alexander and Davis, 91]
L.Alexander and A.Davis, "Criteria for Selecting Software Process Models", In Proceedings of the 15th Annual International Computer Software and Applications Conference (COMPSAC), IEEE Computer Society, 1991.

[Alty et al., 94]
J.Alty, D.Griffiths, N.Jennings and E.Mamdani, "ADEPT - Advanced Decision Support Environment for Process Tasks: Overview and Architecture", In Proceedings of the 1994 BCS Expert Systems conference, British Computer Society, 1994.

[Alty, 97]
J.Alty, (Lutchi Research Centre, Loughborough University), Personal Communication, March 1997.

[Archer, 88]
S.Archer, "Qualitative Research and Epistemological Problems of the Management Disciplines", in A.Petigrew (Ed.), "Competitiveness and the Management Process", Blackwell, 1988.

[Bander et al., 88]
J.Bander, J.Edwards, C.Jones and D.Hannaford, "Practical Engineering of Knowledge Based Systems", Information and Software Technology, Vol. 30, No. 5, 1988.

[Bass et al., 98]
L.Bass, P.Clements and R.Kazman, "Software Architecture in Practice", Addison Wesley, 1998.

[Bates et al., 92]

J.Bates, A.Bryan-Loyall and W.Scott-Reilly, "Integrating reactivity, goals, and emotion in a broad agent", Technical Report CMU-CS-92-142, Carnegie-Mellon University, 1992.

[Benech and Desprats, 97]

D.Benech and T.Desprats, "A KQML-CORBA based Architecture for Intelligent Agents Communication in Cooperative Service and Network Management", in Proceedings of IFIP / IEEE International Conference on Management of Multimedia Networks and Services, 1997.

[Berson, 92]

A.Berson, "Client/Server Architecture", McGraw-Hill, 1992.

[Bochsler, 88]

D.Bochsler, "A Project Management Approach to Expert System Applications", In Proceedings of ISA'88, 1988.

[Boland and Hirschheim, 87]

R.Boland and R.Hirschheim (Eds.), "Critical Issues in Information Systems Research" Wiley, 1987.

[Bonczek et al., 81]

R.H.Bonczek, C.W.Holsapple, and A.Whinston, "Foundations of Decision Support Systems", Academic Press, 1981.

[Booch, 94]

G.Booch, "Object Orientated Analysis and Design With Applications" (2nd Edition), Addison Wesley, 1994.

[Boy, 91]

G.Boy, "Intelligent Assistant Systems", Academic Press, 1991.

[Bradshaw, 97]

J.Bradshaw (Ed.), “Software Agents”, MIT Press, 1997.

[Brooks, 87]

F.P.Brooks, “No Silver Bullet: Essence and Accidents of Software Engineering”, IEEE Computer, 20 (4), 1987.

[Brooks, 89]

R.Brooks, “A Robot that Walks: Emergent Behaviors from a Carefully Evolved Network”, Journal of Neural Computation, Vol.1, No.2, 1989.

[Brown et al., 82]

J.Brown, R.Burton, and J.deKleer, “Pedagogical, Natural Language and Knowledge Engineering Techniques in SOPHIE”, In D.Sleeman and J.Brown (Eds.), “Intelligent Tutoring Systems”, Academic Press, 1982.

[Brownston et al., 85]

L.Brownston, R.Farrell and E.Kant, “Programming Expert Systems in OPS5- An Introduction to Rule-Based Programming”, Addison-Wesley, 1985.

[Buchanan and Feigenbaum, 78]

B.Buchanan and E.Feigenbaum, “DENDRAL and Meta-DENDRAL: Their Applications Dimension”, Artificial Intelligence, 11, 1978.

[Buchanan et al., 83]

B.Buchanan, D.Barstow, R.Bechtel, J.Bennet, W.Clancey, C.Kulikowski, T.Mitchell and D.Waterman, “Constructing Expert System”, in F.Hayes-Roth, D.Waterman and D.Lenat (Eds.), “Building Expert Systems”, Addison-Wesley, 1983.

[Budd, 98]

M.Budd (Ed), “Ovum Evaluates: CASE Products”, Ovum, 1998.

[Caglayan and Harrison, 97]

A.Caglayan and C.Harrison, "Agent Sourcebook", Wiley, 1997.

[Catalyst, 99]

"Plan Your Projects Better", Catalyst Software, 1999.

[Cawsey, 98]

A.Cawsey, "The Essenance of Artificial Intelligence", Prentice Hall, 1998.

[Clocksin and Mellish, 81]

W.F.Clocksin and C.S. Mellish, "Programming in Prolog", Springer-Verlag, 1981.

[Cohen and Levesque, 97]

P.Cohen and H.Levesque, "Communicative Actions for Artificial Agents", in J.Bradshaw (Ed.), "Software Agents", MIT Press, 1997.

[Cooke and McDonald, 86]

N.Cooke and J.McDonald, "A Formal Methodology for Acquiring and Representing Expert Knowledge", In Proceedings of the IEEE, pp. 1422-30, IEEE, 1986.

[Coutaz, 87]

J.Coutaz, "PAC - An Object Orientated Model for Dialog Design", in H.Bullinger and B.Shackel (Eds.), "Human Computer Interaction - INTERACT'87", North-Holland, 1987.

[Corkhill, 97]

D.Corkhill, "Countdown to Success: Dynamic Objects, GBB and RADARSAT-1", Communications of the ACM, Vol. 40, No.5, 1997.

[Cornford, 96]

T.Cornford, "Project Research in Information Systems: A Students Guide", Macmillan, 1996.

[Cox, 94]

B.Cox, "Explaining and Understanding Engineering Problems - An Intelligent Tutoring Approach", International Journal of Engineering Education, Vol.10, No.3, 1994.

[Dennett, 91]

D.Dennett, "Consciousness Explained", Penguin Press, 1991.

[DTI, 92]

Department of Trade and Industry, "KBS: Survey of UK Applications", Touche Ross and Co., 1992.

[Eilon, 79]

S.Eilon, "Seven Faces of Research", In S.Eilon, "Aspects of Management" (2nd Edition), Pergamon Press, 1979.

[Engelmore, 93]

R.Engelmore (Ed.), "JTEC Panel on Knowledge-based Systems in Japan", ARPA Technical Report, Advanced Projects Research Agency, 1993.

[Englemore and Morgan, 88]

R. Englemore, T.Morgan, "Blackboard Systems", Addison Wesley, 1988.

[Fairclough, 96]

J.Fairclough (Ed.), "Software Project Management", in "Software Engineering Guides", Prentice Hall, 1996

[Finin et al., 97]

T.Finin, Y.Labrou, and J.Mayfield, "KQML as an agent communication language", in J.Bradshaw (Ed.), "Software Agents", MIT Press, 1997.

[Finkelstein et al., 94]
A.Finkelstein, J.Kramer and B.Nuseibeh (Eds.), "Software Process Modelling and Technology", Research Studies Press, 1994.

[Fischer, 87]
G.Fischer, "A Critic for LISP", In Proceedings of the 10th International Joint Conference on Artificial Intelligence, Los Altos, USA, Morgan Kaufman, 1987.

[Fischer et al., 91]
G.Fischer, A.C.Lemke, T.Mastaglio, A.Morch, "The Role of Critiquing in Cooperative Problem Solving", ACM Transactions on Information Systems, April 1991.

[Fodor, 83]
J.Fodor, "The Modularity of Mind", MIT Press, 1983.

[Forgy, 82]
C.Forgy, "Rete: A Fast Algorithm for the Many Pattern / Many Object Pattern Match Problem", Artificial Intelligence, No. 19, 1982.

[Fowler , 97]
M.Fowler, "UML Distilled - Applying the Standard Object Modelling Language", Addison Wesley, 1997.

[Frasson and Gauthier, 86]
C.Frasson and G.Gauthier (Eds.), "Intelligent Turoring Systems - Proceedings of the Third International Conference (ITS'96)", Lecture Notes in Computer Science 1086, Springer, 1996.

[Friedman-Hill, 99]
E.J.Friedman-Hill, "Jess - The Java Expert System Shell", User Manual version 5.01b1, Sandia National Laboratories, USA, 1999.

[Galliers, 92]

R.Galliers, "Choosing Appropriate Information Systems Research Methods", in H.Nissen (Ed), "Information Systems Research: Contemporary Approaches and Emergent Traditions, North-Holland, 1992.

[Genesereth and Fikes, 92]

M.Genesereth and R.Fikes, "Knowledge Interchange Format Version 3 Reference Manual", Technical Report 92-1, Logic Group, Stanford University, 1992.

[Genesereth and Ketchpel, 94]

M.Genesereth and S.Ketchpel, "Software Agents", Communications of the ACM, Vol. 37, No. 7, 1994.

[Giarratano and Riley, 94]

J.Giarratano and G.Riley, "Expert Systems - Principles and Programming", PWS Publishing Company, 1994.

[Goal, 95]

"The GOAL Project", ESPRIT 6283 - Generic Object-Oriented Multi-Application Project Management Tool for Large Inter-Organisational Projects, Public Report, 1995.

[Gosling and McGilton, 96]

J.Gosling and H.McGilton, "The Java Language Environment", White Paper, Sun Microsystems, May 1996.

[Gupta et al., 96]

S.Gupta, W.Regli and D.Nau, "Integrating DFM with CAD through Design Critiquing", Concurrent Engineering, Vol. 2, No. 2, 1996.

[Grosz and Davis, 94]

B.Grosz and R.Davis (Eds.), "A Report to APRA on Twenty-First Century Intelligent Systems", American Association for Artificial Intelligence, 1994.

[Hall, 97a]

C.Hall (Ed.), "Intelligent Software Strategies", Cutter Information Corp., Summer 1997.

[Hall, 97b]

C.Hall (Ed.), "Intelligent Software Strategies", Cutter Information Corp., Fall 1997.

[Hampton, 97]

"A Buyer Guide to Selecting Project Management Software", The Hampton Group, 1997.

[Hayes-Roth, 83]

B.Hayes-Roth, "The Blackboard Architecture - A Generic Framework for Problem Solving?", Technical Report, HPP-83-30, Stanford University, 1983.

[Hebb, 49]

D.Hebb, "The Organisation of Behavior", Wiley, 1949.

[Heck and Mitchell, 96]

M.Heck and K.Mitchell, "Project Management Solutions", InfoWorld, Vol. 18, Issue 23, 1996.

[Henry, 94]

W.Henry, "Software Project Risk Management: A Support Tool", M.Sc. Thesis, Dublin City University, 1994.

[Hodges and Rogers, 97]

J.Hodges and S.Rogers, "Cross-Industry Application: 1997 Worldwide Markets and Trends", Report 13705, International Data Corporation, 1997.

[Hughes and Cotterall, 99]

B.Hughes and M.Cotterall, "Software Project Management" (2nd Edition), McGraw-Hill, 1999.

[Humphrey, 89]

W.Humphrey, "Managing the Software Process", Addison-Wesley, 1989.

[Humphrey, 95]

W.Humphrey, "A Discipline for Software Engineering", Addison-Wesley, 1995.

[Hunt, 98]

J.Hunt, "Tooled up for Java", SIGS Application Development Advisor, Vol. 2, No. 2, 1988.

[Huhns and Sing, 97]

M.Huhns and M.Sing, "Conversational Agents", IEEE Internet Computing, Vol. 1, No. 2, 1997.

[Jackson, 90]

P.Jackson, "Introduction to Expert Systems" (2nd edition), Addison-Wesley, 1990.

[Jeffery and Votta, 99]

R.Jeffery and L.Votta, "Guest Editors Special Section Introduction", IEEE Transactions on Software Engineering, Vol. 25, No. 4, 1999.

[Jenkins et al., 87]

J.O.Jenkins, R.Verbruggen and M.Bosco, "Integrated Management Process Workbench (IMPW): Intelligent assistance for the Software Project Manager", In Proceedings of CASE'87 - 1st International Workshop on Computer-Aided Software Engineering, Cambridge, USA, 1987.

[Jennings et al., 96a]

N.Jennings, J.Corera, I.Laresgoiti, E.Mamdani, F.Perriollat, P.Skarek and L.Varga, "Using ARCHON to develop real-world DAI applications for electricity transportation management and particle accelerator control", IEEE Expert, Vol. 11, No. 6, 1996.

[Jennings et al., 96b]

N.Jennings, P.Faratin, T.Norman, P.O'Brien, M.Wiegand, C.Voudouris, J.Alty, T.Miah, E.Mamdani, "ADEPT: Managing Business Processes using Intelligent Agents", In Proceedings of the 1996 BCS Expert Systems conference, BCS, 1996.

[Jones, 94]

M.Jones, "Project Management Report", Cambridge Market Intelligence, 1994.

[Kamsteeg and Blerman, 89]

P.Kamsteeg and D.Blerman, "Differences Between Expert Systems and Domain Components of Intelligent Tutoring Systems", in J.Cambell and J.Cuena (Eds.), "Perspectives in Artificial Intelligence", Wiley, 1989.

[Kaplan and Maxwell, 94]

B.Kaplan, and J.Maxwell, "Qualitative Research Methods for Evaluating Computer Information Systems", in J.Anderson, C.Aydin and S.Jay (Eds.), "Evaluating Health Care Information Systems: Methods and Applications", Sage, 1994.

[Kawaguchi et al., 91]

A.Kawaguchi, H.Motoda and R.Mizoguchi, "Interview Based Knowledge Acquisition Using Dynamic Analysis", IEE Expert, Vol. 6, No. 5, 1991

[Keen, 91]

P.Keen, "Relevance and Rigour in Information Systems Research: Improving Quality, Confidence, Cohesion and Impact", in H.Nissen (Ed), "Information Systems Research: Contemporary Approaches and Emergent Traditions, North-Holland, 1992.

[Klein and Methlie, 95]

M.Klein and L.B.Methlie, "Knowledge Based Decision Support Systems with Applications in Business" (2nd edition), John Wiley & Sons, 1995.

[Lange and Oshima, 97]
D.Lange and M.Oshima, "Programming Mobile Agents in Java with the Java Aglet API", White Paper, IBM Japan, 1997.

[Lederberg, 87]
J.Lederberg, "How DENDRAL was Conceived and Born", ACM Symposium on the History of Medical Informatics, National Library of Medicine, 1987.

[Leonard, 98]
A.Leonard, "Bots - The Origin of New Species", Penguin, 1998.

[Lesnick and Moore, 97]
L.Lesnick and R.Moore, "Creating Cool Intelligent Agents for the Net", IDG Books, 1997.

[Luger and Stubblefield, 89]
G.F.Luger and W.A.Stubblefield, "Artificial Intelligence and the Design of Expert Systems", Benjamin-Cummings Publishing, 1989.

[Luger and Stubblefield, 98]
G.Luger and W.Stubblefield, "Artificial Intelligence - Structures and Strategies for Complex Problem Solving", Addison Wesley, 1998.

[Maes, 97]
P.Maes, "Software Agents", In Proceedings of the Unicom conference on Agents and Intelligent User Interfaces, Unicom, 1997.

[Mair, 92]
P.Mair, "CASE: A State of the Market Report", Unicom, 1992.

[Mallach, 94]
E.Mallach, "Understanding Decision Support Systems and Expert Systems", Irwin, 1994.

[Mayfield et al., 95]
J.Mayfield, Y.Labrou, T.Finin, "Evaluation of KQML as an Agent Communication Language", in M.Wooldridge, J.Muller, M.Tambe (Eds.), "Intelligent Agents II: Agents Theories, Architectures and Languages", Lecture Notes in Computer Science 1037, Springer Verlag, 1995.

[Microsoft, 97]
"Introduction to Microsoft Agent", White Paper, Microsoft Corporation, August 1997.

[Miller, 97]
J.Miller, "I Dream of Genie: Microsoft's new active agent control can grant your wish", Lecture presented at Microsoft SiteBuilder Conference, 1997.

[Minsky, 85]
M.Minsky, "The Society of Mind", Simon and Schuster, 1985.

[Montgomery, 88]
A.Montgomery, "Gemini - Government Expert Systems", In Proceedings of the 1988 BCS Expert Systems conference, British Computer Society, 1988.

[Moynihan et al., 94]
T.Moynihan, J.Power, W.Henry, "A Critiquing System Architecture for Software Risk Management", In Proceedings of 5th European Software Control and Metrics conference (ESCOM), Italy, May 1994.

[Muller et al., 99]
J.P.Muller, M.P.Singh and A.S.Rao (Eds.), "Intelligent Agents V: Agents Theories, Architectures and Languages", LNCS 1555, Springer Verlag, 1999.

[Mumford et al., 84]
E.Mumford, R.Hirschheim, G.Fitzgerald and A.Wood-Harper (Eds.) "Research Methods in Information Systems", Proceedings of the IFIP WG 8.2 Colloquium, North-Holland, 1984.

[OMG, 96]
"CORBA: Architecture and Specification", Object Management Group, 1996.

[O'Connell, 96]
F.O'Connell, "How To Run Successful Projects II", Prentice Hall, 1996.

[O'Connor et al., 97a]
R.O'Connor, T.Moynihan, T.Renault and A.Combelles, "PROMPTER - A Decision Support Tool for Software Project Management", Technical Report CA-2997, Dublin City University, 1997.

[O'Connor et al., 97b]
R.O'Connor, T.Renault, C.Floch, T.Moynihan and A.Combelles, "Prompter - A Decision Support Tool using Distributed Intelligent Agents", In Proceedings of 9th International Conference on Artificial Intelligence Applications, England, 1997.

[O'Connor, 98]
R.O'Connor, "A Multi-Agent Approach to Knowledge Base Implementation", In Proceedings of the International Postgraduate Research Student Conference, Dublin, Ireland, 1998.

[O'Connor and Jenkins, 98]
R.O'Connor and J.O.Jenkins, "Supporting Effective Software Project Management and Control by the use of Intelligent Knowledge-based Guidance", In Proceedings of 9th European Software Control and Metrics conference (ESCOM), pp. 133 - 141, Rome, Italy, 1998.

[O'Connor and Moynihan, 98]
R.O'Connor and T.Moynihan, "A Multi-Agent Approach to Project Planning", In Proceedings of 10th International Conference on Artificial Intelligence Applications (EXPERSYS), Virgina, USA, 1998.

[O'Connor and Renault, 98]
R.O'Connor and T.Renault, "Designing an Internet Enabled Decision Support Tool in the Domain of Software Project Management", In Proceedings of the International Symposium on Engineering of Intelligent Systems, Spain, 1998.

[O'Connor and Jenkins, 99a]
R.O'Connor and J.O.Jenkins, "Intelligent Project Guidance", In Proceedings of 10th European Software Control and Metrics conference (ESCOM), pp. 27 - 36, Herstmonceux, England, 1999.

[O'Connor and Jenkins, 99b]
R.O'Connor and J.O.Jenkins, "Using Agents for Distributed Software Project Management", In Proceedings of 8th International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, IEEE Computer Society Press, USA, 1999.

[Ould, 90]
M.Ould, "Strategies for Software Engineering: The Management of Risk and Quality", Wiley, 1990.

[Orfali et al., 99]
R.Orfali, D.Harkey and J.Edwards, "Client/Server Survival Guide", Wiley, 1999.

[Orfali and Harkey, 98]
R.Orfali and D.Harkey, "Client/Server Programming with Java and CORBA", Wiley, 1998.

[Orlikowski and Baroudi, 91]
W.Orlikowski and J.Baroudi, "Studying Information Technology in Organizations: Research Approaches and Assumptions", Journal of Information Systems Research Vol. 2, 1991.

[P3, 99]
"The P3 Project - Handbook and Training Guide", ESPRIT 22241, 1999.

[Polson and Richardson, 88]
M.Polson and J.Richardson (Eds.), "Foundations of Intelligent Tutoring Systems" Lawrence Erlbaum Associates Inc., 1988.

[Pooley and Stevens, 99]
R.Pooley and P.Stevens, "Using UML - Software Eningeering with Objects and Components", Addison Wesley, 1999.

[Power, 94]
J.Power, "A Critiquing System Architecture in the Risk Management Domain", M.Sc. Thesis, Dublin City University, 1994.

[Pulford et al., 96]
K.Pulford, A.Kuntzmann-Combelles and S.Shirlaw, "A Quantative Approach to Software Management - The AMI Handbook", Addison Wesley, 1996.

[Pressman, 97]
R.Pressman, "Software Engineering - A Practitioners Approach", McGraw Hill, 1997.

[Procter and Bouchier, 94]
I.Procter and S.Bouchier, "Software Tools for Project Management", Information Technology Report, Unicom, 1994.

[PMI, 96]
"A Guide to the Project Management Body of Knowledge", Project Management Institute, 1996.

[Quillian, 68]
M.R.Quillian, "Semantic Memory", In Semantic Information Processing, M.Minsky (Ed.), MIT Press, 1968.

[USAF, 88]
"Software Risk Abatement", AFSC/AFLC Pamphlet 800-45, US Department of the Air Force, 1988.

[Ringland and Duce, 88]
G.A.Ringland and D.A.Duce, "Approaches to Knowledge Representation", Research Studies Press, 1988.

[Rumbaugh et al., 91]
J.Rumbaugh, M.Blaha, W.Premerlani, F.Eddy and W.Lorensen, "Object Orientated Modeling and Design", Prentice Hall, 1991.

[Russell et al., 89]
D.Russell, T.Burton, D.Jordan, M.Jensen, R.Rogers and J.Cohen, "Creating Instructions with IDE: Tools for Instructional Designers", Technical Report PS-00076, Xerox Palo Alto Research Center, 1989.

[Sanders, 98]
M.Sanders (Ed.), "The SPIRE Handbook - Better, Faster, Cheaper Software Development in Small Organisations", ESPRIT/ESSI project 23873, European Commission, 1998.

[Seaman, 99]
C.Seaman, "Qualitative Methods in Empirical Studies of Software Engineering", IEEE Transactions on Software Engineering, Vol. 25, No. 4, 1999.

[Selfridge, 59]
O.Selfridge, "Pandemonium - A Paradigm for Learning", Symposium on the Mechanization of Thought, HMSO, London, 1959.

[Schach, 97]
S.Schach, "Software Engineering with Java", Irwin, 1997.

[Schreiber et al., 93]
G.Schreiber, B.Wielinga and B.Brenker (Eds.), "KADS: A Principled Approach to Knowledge Based Systems Development", Knowledge Based Systems, Vol. 11, Academic Press, 1993.

[Shaw and Garlan, 96]
M.Shaw and D.Garlan, "Software Architecture - Perspectives on an Emerging Discipline", Prentice Hall, 1996.

[Shen, et al., 97]
D.Shen, C.Wu and S.Lee, "EXPIDER: A Channel Routing Expert for VLSI Design", Journal of Expert Systems with Applications, Vol. 12, No. 2, 1997.

[Shoham, 93]
Y.Shoham, "Agent-oriented programming", Artificial Intelligence, Vol. 60, No. 1, 1993.

[Silverman, 92]
B.Silverman, "Survey of Expert Critiquing Systems: Practical and Theoretical Frontiers", Communications of the ACM, Vol. 35, No. 4, April 1992.

[Smith et al., 94]
P.Smith, P.Ross and E.Awad, "A Survey of the Skills and Personality Attributes of the Knowledge Engineer in the UK", KnowledgeBase, Vol. 8, No. 2, 1994.

[Smith, 96]
P.Smith, "An Introduction to Knowledge Engineering", International Thomson Computer Press, 1996.

[Sommerville, 95]
I.Sommerville, "Software Engineering", Addison Wesley, 1995.

[Srinivas et al., 97]
K.Srinivas, V.Jugannathan and R.Karinthi, "Java and Beyond Executable Content", IEEE Computer, June 1997.

[Stone, 97]
W.Stone, "Project Management for the Internet", Mainspring Communications, June 1997.

[Straub et al., 94]
D.Straub, S.Ang and R.Evaristo, "Normative Standards for IS Research", Database, Vol. 25, No.1, 1994.

[Tierney and Davison, 95]
M.Tierney and R.Davison, "Decision Support for the Provision of Services", Communicate, Vol.2, No.1, June 1995.

[Turban, 95]
E.Turban, "Decision Support and Expert Systems: Management Support Systems", Prentice Hall, 1995.

[Verbruggen et al., 89]
R.Verbruggen, T.Moynihan and G.McCloskey, "RISKMAN1: A Prototype Risk Analysis Tool for Software Development Project Managers", In Proceedings of CASE'89 - 3rd International Workshop on Computer-Aided Software Engineering, 1989.

[Watson, 97]
M.Watson, "Intelligent Java Applications", Morgan Kaufmann, 1997.

[Weld, 95]
D.Weld (Ed.), "The Role of Intelligent Systems in the National Information Infrastructure", AI Magazine, Fall 1995.

[Wittig et al., 94]

T.Wittig, N.Jennings and E.Mamdani, "ARCHON - A Framework for Intelligent Co-operation", Journal of Intelligent Systems Engineering, Vol. 3, No. 3, 1994.

[White, 94]

J.White, "Telescript Technology: The foundation for the electronic marketplace", White paper, General Magic Inc., USA, 1994.

[White et al., 97]

J.White, C.Helgeson and D.Steedman, "Introduction to the Odyssey API", White Paper, General Magic Inc., USA, 1997.

[Wooldridge and Jennings, 95]

M.Wooldridge and N.Jennings, "Intelligent Agents: Theory and Practice", Knowledge Engineering Review, Vol. 10, No. 2, 1995.

[Zuboff, 88]

S.Zuboff, "In the Age of Smart Machines - The Future of Work and Power", Heinemann Professional Publishing, 1988.