City University
London

Dynamic Application Integration Using Peer to Peer Technology

by

Stelios Christofi

Research Supervisor

Dr. Bill Karakostas

A thesis submitted to

THE CITY UNIVERSITY OF LONDON

for the degree of

DOCTOR OF PHILOSOPHY

School of Informatics
Centre for Human Computer Interaction Design

October 2003

TABLE OF CONTENTS

TABLE OF FIGURES

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my project supervisor Dr. Bill Karakostas for his valuable help, suggestions, time, technical guidance, continuing motivation and support throughout the project period.

The support of the Overseas Research Students Awards Scheme, which partially pay the university fees, are gratefully acknowledged. I would like to thank all colleagues in INLECOM Systems Ltd for their help and support. In particular, I would like to thank the director of INLECOM Systems, Dr. Takis Katsoulakos for his help, advice and financial support, as well as Mrs. Anna Katsoulakos for her valuable suggestions.

I owe my parents, Titos and Chrystalla Christofi, the greatest thanks for their continuous encouragement, generous financial support and love they have given me throughout my studies at City University. Thanks are also expressed to my brothers, Fanos and Louis for their help and suggestions for this Thesis report.

Last but not least I would like to express my greatest thanks to my girlfriend Agathi Lambrou for her continuous support throughout the duration of my Thesis.

DECLARATION

No portion of the work referred in this Thesis has been submitted in support of an application for another degree or qualification at this university or other institution of learning.

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 acknowledgement.

ABSTRACT

Today's business imperatives are clearer than ever. Businesses are trying to beat competitors by introducing new products to the market, deliver personalised services, increase customer loyalty and evolve at electronic speeds. These imperatives demand a technology infrastructure that is more flexible, more dynamic and more intelligent than ever. Java Web services based on *Universal Description, Discovery and Integration* (UDDI) are an evolution in e-business applications that will help businesses reach these goals and take *Business-to-Business* (B2B) to the next level.

In the last couple of years, the concept of a *Web Service* (WS) has emerged as an important paradigm for general application integration in the internet environment. More particularly, WS is viewed as an important vehicle for the creation of dynamic e-business applications and as a means for the *Java 2 Enterprise Edition* (J2EE) and Microsoft .NET platforms to come together. This will be achieved through WS standards and several companies have been collaborating in such standardisation activities.

This Thesis describes research aiming to allow the dynamic integration of different software applications and information sources, including legacy systems, using the latest state of the art Internet technology called "*Peer-to-Peer* (P2P)". The term "dynamic" is used in order to indicate that different software applications, in different geographical locations, that need to be integrated are able to establish a communication link and interchange data without manual intervention or without any intermediate integration server. To achieve this goal, several technologies have been combined, including Java Web Services and the Extensible Stylesheet Language (XML), in order to design, develop and implement an innovative architecture that will satisfy such requirements. Amongst others, the main programming language used is the widely accepted Java language, which acts catalytically in the creation of the system architecture described in this Thesis.

PRINCIPAL ABBREVIATIONS

A2A	Application to Application
AOL	America Online
ART	Adaptive Runtime Technology
API	Application Programmers Interface
ASI	Application Service Interface
B2B	Business to Business
BPML	Business Process Modeling Language
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
CRM	Customer Relationship Management
CSV	Comma Separated Values
DCOM	Distributed Component Object Model
DNS	Domain Name Service
DTD	Document Type Definition
E2A	End 2Anywhere
EAI	Enterprise Application Integration
EBNF	Extended Backus Naur Form
EDI	Electronic Data Interchange
EJB	Enterprise Java Beans
ERP	Enterprise Resource Planning
FSM	File Storage Manager
FTP	File Transfer Protocol
GISP	Global Information Sharing Protocol
GSM	Global System for Mobile Communication
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
IP	Internet Protocol
IT	Information Technology

J2EE	Java 2 Enterprise Edition
J2ME	Java 2 Micro Edition
JAXM	Java API for XML Messaging
JAXR	Java API for XML Registries
JAX-RPC	Java API for XML-based Remote Procedure Call
JDBC	Java Database Connectivity
JMS	Java Message Service
JVM	Java Virtual Machine
JWS	Java Web Service
JWSDP	Java Web Services Developer Pack
JWSI	Java Web Service Integrator
JWSIMT	Java Web Service Integrator XML Mapper Tool
LDAP	Lightweight Directory Access Protocol
MQ	Message Queues
MQSeries	Message Queues Series
MSMQ	Microsoft Message Queues
NAICS	North American Industry Classification System
NASSL	Network Accessible Service Specification Language
OASIS	Organisation for the Advancement of Structured Information Standards
ODBC	Open Database Connectivity
P2P	Peer to Peer
RDBMS	Relational Database Management System
RMI	Remote Method Invocation
ROI	Return On Investment
RPC	Remote Procedure Call
SDK	Software Development Kit
SGML	Standard Generalized Mark-up Language
SME	Small and Medium Enterprise
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
UDDI	Universal Description, Discovery and Integration

UML	Unified Modelling Language
UN/CEFACT	United Nations Centre for the Facilitation of Procedures and Practices in Administration, Commerce and Transport
URL	Uniform Resource Locator
UUID	Universal Unique Identifier
VPN	Virtual Private Network
W3C	World Wide Web Consortium
WS	Web Service
WSDL	Web Services Description Language
WYSIWYG	What You See Is What You Get
XML	Extensible Mark-up Language
XSD	XML Schema Definition Language
XSL	Extensible Stylesheet Language
XSLT	Extensible Stylesheet Language Transformations

Chapter 1

INTRODUCTION

1.1 INTRODUCTION

Recent technology advances have enabled *Business-to-Business* (B2B) integration: linking a business tightly with those valued networks partners, to provide a quantum leap in competitiveness [15, 16, 66]. B2B integration presents an enormous opportunity for companies to create a coordinated community of partners that collaborate in a variety of areas, from planning to distribution to customer service.

Coordination of business processes, both internally as well as between the organisation and its trading partners can be problematic if the processes are not supported by adequate, relevant and timely information.

As the business processes in general rely for support on information generated by software applications, the adequacy of such applications comes under scrutiny. Applications are expected to be able to interoperate in order to exchange mutually understood data in a timely manner. Interoperability is a broad term with scope that ranges from application interfacing to full application integration. Naturally, the costs of achieving application interoperation escalate as one moves from simple interfacing to complete integration.

The problems of application integration (or the lack of it) due to applications that do not inter-operate are faced by all industries. Application integration has grown to a multi-billion dollar industry but has yet to arrive at truly universal solutions. Application integration is still a highly customised, ad-hoc and costly process.

Optimising and automating business processes within and across enterprises will continue to drive enterprise Information Technology (IT) initiatives for the foreseeable future. Application integration is essential to the automation of business processes. Enterprises need to improve process integration with their business partners, customers, and suppliers. Existing enterprise application integration (EAI) solutions are highly proprietary and centralised, and do not fully meet the needs of B2B and Web Services integration. Customers demand lower costs, standards-based and lightweight integration approaches that are capable of complete process integration between businesses.

The topic of this thesis is concerned with defining an innovative approach and providing the technology required to support Dynamic Application Integration using the latest state of the art peer-to-peer (P2P) technology. For the thesis it has been decided to use as a case study, the shipping sector, because the author has experience in designing and developing applications related to the shipping industry. This does not mean that the architecture that will be discussed in this thesis will not be applicable for other sectors but on the contrary, the architecture has been designed in such a way so that it can be universally applied to any industrial sector.

1.2 PROBLEM DEFINITION AND OBJECTIVES

The goal of application integration, whether it is within or beyond the enterprise, is to allow disparate enterprise systems to interact automatically with each other in order to more effectively optimise business processes. These systems are often provided by different vendors and have no pre-established way of communicating or exchanging data.

It is up to application integrators to provide mechanisms to facilitate the coordination of applications and a number of solutions are available in the market for this purpose.

The vast majority of the solutions for application integration used today are based on a "*hub-and-spoke*" architecture, as illustrated in figure 1.1. A central hub manages all message exchange between the applications. As such, the hub is responsible for:

- **Message Routing**: Receiving and forwarding all messages
- **Message Transformation**: Translating message formats
- **System Management**: Establishing secure connections, authentication, transaction handling, monitoring, and logging.
- **Coordination of the Business Process**: Directing which applications should do what tasks and when.

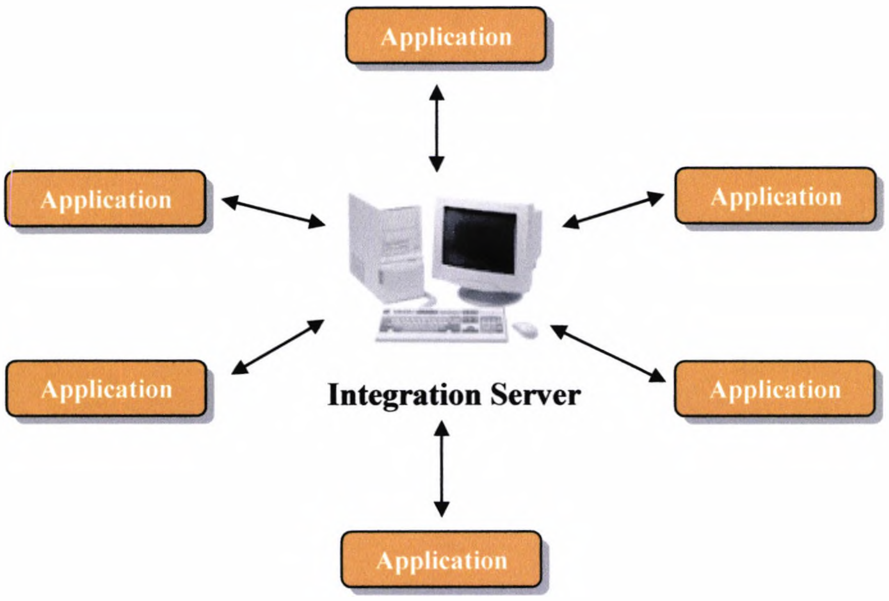


Figure 1.1 *The traditional hub-and-spoke architecture for EAI*

However, a number of problems are associated with the hub-and-spoke architecture, some of which are as follows:

- **Expensive to deploy**: A large monolithic server is expensive to develop and to deploy. Interfaces must be agreed upon beforehand and changes are difficult to incorporate.
- **Overkill for simple processes**: Most business-to-business interactions require only simple business processes, for example notification of purchase order status, dealing with electronic change orders, etc. Such simple processes call for lightweight and flexible solutions.
- **Expensive to scale and build reliability**: The hub must accomplish many processing and communication tasks. Typical message brokers can only handle 10 to 100 messages a second. The hub presents a major bottleneck; scalability and reliability issues quickly arise. Current systems attempt to avoid these issues by using increasingly sophisticated software and hardware, but this only alleviates the symptoms as opposed to curing the problem. Merely linking hubs together also does not help, as ultimately message traffic must then flow through all the hubs, slowing them down even further.
- **Hard to manage changes**: With the centralised business logic engine ultimately being tailored for those applications that are integrated, it is hard to manage changes in the system, for example, adding new applications or detaching and reattaching existing applications. Updating the business logic to new circumstances is a difficult task. As such, hub-and-spoke based solutions quickly become legacy systems in their own right. An enterprise is quickly plagued with many of these “islands” of integration. Once the interfaces are established, it is hard to change them to accommodate new data formats. It is hard to incorporate new business processes on the fly.
- **Lack of autonomy**: Individual participants do not preserve control over their participation in the business process and cannot incorporate their own business processes. This is especially important in B2B scenarios.
- **Resistance to proprietary deployments**: The deployed solutions require the use of proprietary networks, interfaces, and implementations. Therefore all participants are required to deploy the same software. Many

participants do not wish to be dictated to, as to which software to deploy. This is especially important in B2B scenarios, where participants may be participating in different networks.

To avoid the problems and limitations of the "*hub-and-spoke*" architecture, this thesis has concentrated in proving a dynamic application integration architecture using the latest and currently most advanced technology of EAI, the P2P architecture [3,38,39,49,75,76,88,91,132]. This is similar to the message broker architecture, described above, but with one main difference. There is no central integration server. Instead, the message translation, routing, splitting, and combining occur at the source and target systems within little message brokers known as agents or peers, as shown in figure 1.2.

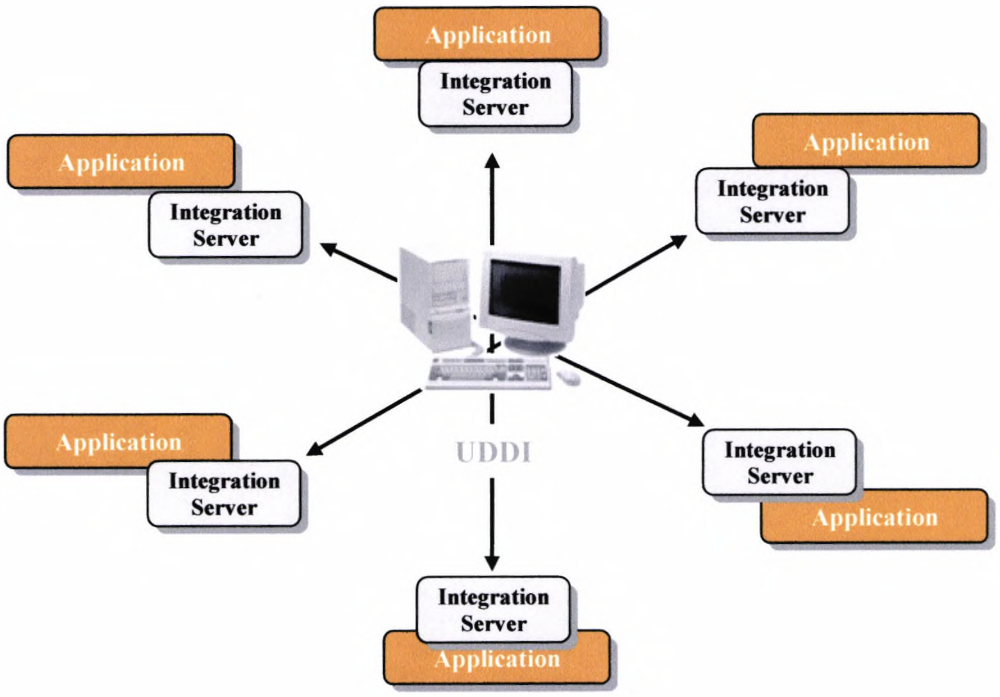


Figure 1.2 *A general Peer-to-Peer Architecture*

The agents consume information from the single system they are connected to, process that information and send it directly to any target system(s) interested in receiving that information or message. The result is a virtual "*hub-and-spoke*" type architecture, where the description of each service is located on a central location called UDDI.

The main advantages of a Peer-to-Peer architecture are [97, 98, 111, 113]:

- A P2P application s capable of locating other peers in the network.
- Once an application is able to locate other peers, it is capable of communicating with them using messages.
- Once the communication is established with other peers, the application is able to receive and provide information, such as content.

The objective of this thesis is to design, develop and implement an innovative Peer-to - Peer architecture that will be used for dynamic application integration in order to solve the problems stated above. This architecture will be based on the widely accepted Java language and the Web Services technology. Using this architecture, enterprises will be able to integrate their applications effortlessly thus avoiding manual intervention during the integration process.

The following section describes a typical problem that shipping companies are facing today during the processes of designing and constructing a ship.

A hypothetical ship designer wishes to test his company's passenger evacuation software using different ship designs, generated by various ship design tools or different ship hull forms from different companies in order to generate some evacuation statistics to measure the performance of the software. Nowadays, to achieve this requirement, the company has to either purchase a license for each and every ship software needed, test it locally or physically transfer the software to another company to be tested with existing hull forms tools. Both the above approaches require a considerable amount of effort, time and money since the transfer of software, especially for the shipping sector, cannot be accomplished using the latest electronic means for many reasons. The available electronic means may be:

- Electronic mail (e-mail) using the Simple Mail Transfer Protocol (SMTP)
- File Transfer Protocol (FTP)
- Hyper Text Transfer Protocol (HTTP)

The above electronic means are not so efficient. This is mainly because the software used for the shipping sector is very specialised and can cost a great amount of money even for a single license. Buying the license for the purpose of evaluating an evacuation program is not worth while and even transferring the software from one vendor to another using the above electronic means is not a practical solution. In this thesis the attempt to solve this problem is presented by proposing a different and innovative approach. A dynamic application integration methodology can be used in order to avoid the problems mentioned above. The dynamic application integration will be achieved using the Peer-to-Peer technology, which is the latest technology in the modern Internet environment.

During the lifecycle of the design of the ship, a ship designer (the client), wishes to test his company's passenger evacuation software using ship designs from other companies, in order to generate some evacuation statistics to measure the performance of the software. The above software requires a hull generator tool. This tool creates basic hull designs for ships in order for other software to get these hulls and tests them prior to designing the final hull form of a ship. In order to accomplish this task, he/she configures the client (Java Web Service Integrator) JWSI based on the input and output information of the above passenger evacuation software. The JWSI is software that will be developed for this thesis. This software will be installed on the client and server sites in order to perform the P2P integration. Then the JWSI contacts the appropriate UDDI to discover existing Java Web Services that offer this kind of service. When a Web Service is found that closely matches the criteria of the client software, the client JWSI and the server JWSI establish between each other a Peer-to-Peer connection for the purpose of integrating the two software applications. The whole integration is done dynamically without the intervention of the client or the server user. Once the hull design is downloaded to the client site, the passenger evacuation software runs in order to generate the appropriate evacuation statistics. By completing the above, the two software applications are integrated without the physical transfer of the whole software and the installation on the local client, but rather the transfer of only the integrated data.

The system is developed in such a way, so that any two applications requesting to be integrated together do not need to know about each other. This ensures that systems on any location can be deployed independently and still interoperate, as long as they comply to certain interchange standards that are specified in Chapter 3.

1.3 TECHNICAL MERITS

The difficulty of the thesis lies with the design and implementation of an advanced peer-to-peer architecture for the purpose of achieving dynamic application integration. A Peer-to-Peer enabled application should be capable of locating other peers in the network. Once an application is able to locate other peers, it should be able to communicate with them using messages or data.

Also, the creation and discovering of Java Web Services, which comply with the *Simple Object Access Protocol* (SOAP) and the *Web Services Description Language* (WSDL) specification, on universally accepted registries like the *Universal Description, Discovery and Integration* (UDDI) was another difficulty during the design and development of this platform. This is extremely important because in order for a P2P system to function properly it requires a global registry where all the Java Web Services will be registered and hence to be discovered dynamically by Java peers.

The actual integration between the various applications, especially to legacy applications that do not provide any open *Application Programmers Interface* (API) creates another challenging area for establishing a technical solution. In addition, the use of some verification algorithms was required to ensure the integrity, confidentiality and accuracy of the data sent by the two integrated applications.

Last but not least, the difficulty of this thesis extends to the use of a mechanism for negotiating the communication and security protocol during the actual integration process. This is extremely important since the bandwidth all over the world is expensive and selecting the appropriate communication protocol is of great importance in order to minimize integration costs for a business.

1.4 STRUCTURE OF THE THESIS

The rest of the thesis is structured in six chapters.

Chapter 2: The second chapter of the thesis discusses the need for a Peer to Peer architecture in application integration and what are the advantages over other technologies. It continues by providing a literature survey of some existing integration systems in the market followed by a comparison between them and in relation to the proposed integration server.

Chapter 3: This chapter demonstrates the detailed requirements specification of the system. It begins by providing a description of the system requirements followed by a detailed survey of methodologies, highlighting the basic features of each methodology and giving reasons why they have been selected to satisfy the requirements set above. It continues by presenting a detailed description of the latest techniques used in this thesis. Finally, a more detailed description of the overall architecture of the integration platform is presented explaining how the selected methodologies can be integrated to achieve the goals of the proposed architecture.

Chapter 4: This chapter covers the design activities for the prototype system. It explains the procedure that has been followed for the design of this system and the output diagrams created. The models that were produced are then shown along with a detailed description. The *Unified Modelling Language* (UML) and Rational Rose software are also presented.

Chapter 5: This chapter presents the implementation of the prototype application. This is achieved by proving a prototype execution demonstration followed by a detailed description of how the implementation part has actually been achieved.

Chapter 6: This chapter covers the evaluation of the final implementation. This is organised as a general discussion of the evaluation techniques used for this system, followed by some testing of the results using advanced testing tools.

Chapter 7: The final chapter summarises the work that has been done and presented in this thesis. The conclusions drawn from the implementation are also discussed. Finally, suggestions for further work are also indicated.

1.5 CONCLUDING REMARKS

The introductory chapter provides essential information about the rest of the thesis. A definition of the problem that the thesis addresses is introduced first, along with some initial objectives derived from the identification of the problem. Following the description of what the thesis deals with, this chapter presented the technologies used to confront this problem. It ends with a layout of the thesis structure.

Chapter 2

LITERATURE SURVEY

2.1 INTRODUCTION – DYNAMIC APPLICATION INTEGRATION

Application integration is the process of bringing data or a function from one application program together with that of another application program [6, 19, 36, 130, 142].

Dynamic application integration is the integration of applications in a "*dynamic*" business environment [106,127]. A dynamic business environment is one that changes at any time without informing the participants about the changes. An application participating in that environment must in a way inform the other applications, which are part of that environment, about the changes. Dynamic application integration in the context of this Thesis means that any application may be integrated and hence interchange data with another application on the Network or Internet dynamically, without prior knowledge of the underlying infrastructure or data structure of the other application. This is accomplished with the help of a *Java Web Service Integrator* (JWSI), which generates java web services on the fly (dynamically), while the potential applications to be integrated are running. These Java web services are published on a registry, called *Universal Description, Discovery and Integration* (UDDI) in order to be dynamically discovered by other applications on the Internet for the purpose of interchanging information [55, 56, 77, 134, 135]. A UDDI is an XML registry infrastructure that enables the building, deployment, and discovery of Web services. It is a neutral third party that facilitates dynamic and loosely coupled business-to-business (B2B) interactions. A registry is available to organisations as a shared resource, often in the form of a Web-based service.

The more common usage of "*dynamic discovery*" is to describe systems, in which clients search through registries to first discover and then invoke services supporting the capabilities they require. Such systems are described as dynamic because the clients supposedly have no prior knowledge of the services they are searching for. The client finds a service based on some criteria, and then interacts with the service based on communication instructions that it finds as part of the service characteristics stored in the registry.

Since integration is probably one of the most important factors that determines the success or failure of a business, many software companies have tried and are still trying to implement efficient B2B solutions that take into consideration dynamic integration issues, as mentioned above. Few of them have actually succeeded in doing so [22, 23, 24, 48, 64, 71, 82, 85, 86, 87]. What is more important, is the fact that none of the above have ever succeeded in developing such a dynamic integration solution addressing essential issues like the dynamic re-configuration of java web services on the fly. This Thesis addresses these issues concentrating on the dynamic modification of published web services without the manual intervention of the user.

2.2 THE NEED FOR PEER-TO-PEER ARCHITECURE IN APPLICATION INTEGRATION.

A number of integration efforts in most companies today are being driven by the need to automate business processes, both within and across enterprises [7, 16, 120]. This is very important since business partners can gain greater operational efficiencies through supply chain integration.

Some enterprises are even building their own custom integration servers to solve sourcing problems and are looking for applications to provide real-time views of work in progress and inventories. In addition, the demand for businesses to integrate disparate components of automation through cross-enterprise application integration has never been as high or as crucial to keep on competitive advantage.

Because this demand for integration is rising and system integrators are being pressured increasingly to deploy solutions faster and more cost-efficiently, current integration tools have not evolved to meet their needs.

Unfortunately, centralised "*hub-and-spoke*" architectures that grew out of intra-enterprise application integration are ill suited to meet the demands of what is fundamentally a distributed problem [16]. A "hub" architecture is a messaging server that exchanges important information between different applications so that the applications can interact in a meaningful way. Centralised architectures applied to decentralised business relationships result in a loss of efficiency and increased costs for companies since the initial design of these centralised architectures do take into consideration the different aspects of decentralised architectures. Web services promise a potential solution, but the standards and solutions being developed do not provide the necessary process integration capabilities [5, 18, 33, 42, 43, 63, 90].
The answer to this problem is a viable, fully distributed, network-based integration architecture, known as peer-to-peer or P2P which can help meet these new cross-enterprise challenges [68, 81, 118].

The P2P technology, like many other emerging technologies, needs to interoperate with other platforms in order to become well-established in today's business environment. There may not yet be a specific law of physics about this, but it seems that for every $1 spent on an application, at least $3 more (and maybe as much as $10 more) needs to be spent on integrating that application with other applications and systems [32]. That is why it is extremely important to implement a robust P2P architecture for application integration, being at the same time widely accepted.

The essential element of any distributed system is to enable the participants to maintain autonomy and control. This greatly simplifies development and ongoing change management, resulting in dramatically lower implementation costs, deployment time and the ongoing cost of maintaining the network. A major disadvantage in the

centralised "*hub-and-spoke*" model is that it requires a top-down development methodology for what has always been a bottom-up problem [16,120].
The cross-enterprise integration problem can be defined as "bottom-up" because applications that are to be integrated across a distributed network are already pre-existing software in their own right. To operate at maximum profitability, application owners need to keep specific control over their applications and the way in which they are accessed [120]. That is to be able to change or replace an application without impacting the entire integration solution is important to the stability and effectiveness of the network.

Finally, application owners must be able to be involved in a number of different systems across the Internet, simultaneously. To be precise, they need the ability to offer their services to separate businesses without having to purchase additional software systems for communicating with them [120].

Current integration solutions can be defined as "top-down" because they use software engineering techniques that look at an entire system as a single program distributed across several nodes, as opposed to individual pre-existing applications with individual needs for autonomy, security, etc. This requires specifying and implementing all components of the system and their interactions in a predefined or "*hard-coded*" method.

In general, message brokering, transformation and the business processes are controlled through a single hub that acts as a client to all involved applications. Changes to the API and other interfaces of the applications, as well as the addition of new applications, require modifying and updating the hub or centralised business process engine [120]. Even if a standard client interface is used on the hub side, the owner of the hub, not the application, has complete control over the business process.

Additionally, the current messaging constructs are based on the client-server methodology of request/reply. This requires both sided communication, whether synchronous or asynchronous, between any two nodes on the network. As communication cannot flow directly between network participants but, rather, must pass through the original request initiator, information that needs to be sent to a third party, the system becomes extremely inefficient. Publish/subscribe messaging may be great for information or data dissemination, but it is highly impractical for obtaining responses to specific queries [120].

Finally, the resulting system itself becomes a legacy system because the code written for a specific business process can quickly discarded, as the dynamic nature of business and Internet requires immediate transitions in process.

On the other hand, the need for a viable, fully distributed, network-based integration architecture that can meet new cross-enterprise challenges, supports the notion that a new peer-to-peer approach will better serve the technology requirements of competitive businesses. A peer-to-peer architecture, developed with a software engineering methodology that emphasises the variables of security, control, dynamic accessibility and flexibility, will deliver significant time and cost savings. The use of the bottom-up approach in software development will deliver power to the individual nodes on the network rather than leaving it locked with one central node, just like the centralised client/server methodology [32, 68, 81, 100, 118, 121, 144].

Napster and *Gnutella* are two of the most popular and early to market P2P applications were, however, built from scratch. They have followed different approaches to sharing information [32, 81] (see figure 2.1):

- Napster uses a central server directory to communicate the location of music servers in a P2P network. Napster has a centralised index where the scalability can be limited by the machine power and the network bandwidth of the central point. There is a way to have multiple index servers which are all identical, but it would only scale if the machine power were big enough and there were much less frequent insert messages than query messages [93].

- Gnutella, which began as an underground product in an AOL (America Online) acquired company, avoids a central server but is relying instead on the point to point communication amongst peers in a network. Gnutella is a fully decentralised system and have no single point of failure. However, its messaging mechanisms are based on application-level broadcasting, which is likely to limit scalability [44].

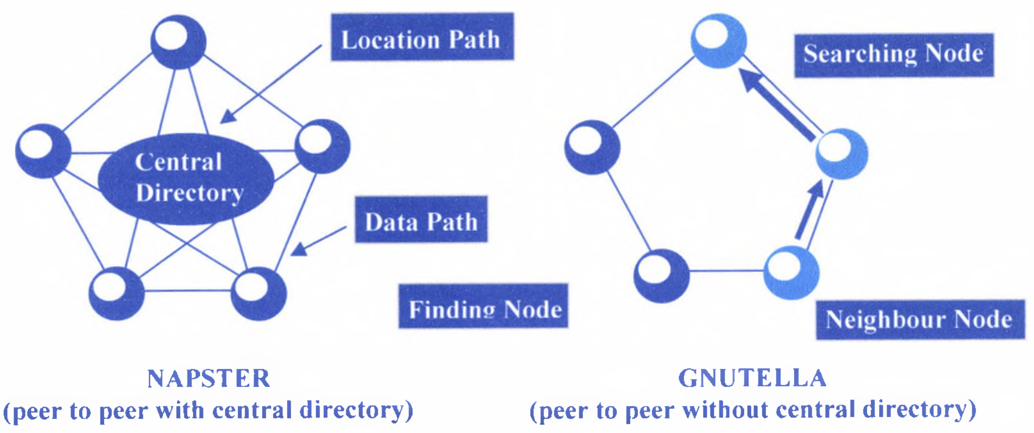


Figure 2.1 *Napster and Gnutella approaches*

To manage change and to provide for essential network functions that may be independent from the actual applications, such as security, organisations should move to a peer-to-peer architecture to enable automated provisioning, location, preparation and execution of services across a network [15, 32, 49, 132]. Rather than trying to solve problems, such as scalability and reliability, that are often associated with standard messaging technologies by creating more sophisticated and elaborate *Enterprise Application Integration (*EAI) projects, businesses should move to a fully distributed framework [143].

Such a framework would allow each application to automatically discover other applications via Web Services and to coordinate the business process from application to application (A2A) or peer to peer. By replacing the centralised "*hub-and-spoke*" architecture with software components associated with the individual applications that need to be integrated, businesses can construct systems that will deliver a high level of fault tolerance and scalability, require little maintenance, adapt to rapid change and be cost-efficient to deploy [16, 120].

The current goal of most application integration projects is to allow disparate enterprise systems to interact automatically with one another in order to optimise business processes. However, the centralised "*hub-and-spoke*" architectures that grew out of intra-enterprise application integration will not answer the existing demand for businesses to integrate disparate systems through cross-enterprise application integration [16]. A more effective and less costly approach to integration is the use of distributed, network-based peer-to-peer integration architectures.

While it will be a challenge to migrate from today's traditional approach to enterprise integration and development, companies that wish to achieve full integration with partners and customers, or hope to refine their internal business processes, must accept and meet this challenge.

In this Thesis, a novel peer-to-peer integration architecture is proposed, from which any application may be integrated with another application in order to share data, information and knowledge, as well as to solve typical problems that shipping companies may be facing today during the processes of designing and constructing a ship. This integration platform will be designed in such a way so that it can be universally applied to other sectors, not just to the shipping industry. The benefits of designing and implementing such a platform are enormous, since dynamic application integration is easily achieved without having to spend time and money to develop specific software code.

This will be achieved by providing to the application users some easy-to-use guidelines on how to integrate their applications to the integration platform. These guidelines, which will be in a form of a wizard, will create a Java web service that will wrap their applications in order to communicate with the platform. This Java web service will then be dynamically published on the UDDI. A second application user that has already created a wrapper for his/her application may need to share some information with other applications in this dynamic platform. The wrapper will be in a position to dynamically discover the already published Java web service and download it in order to initiate a direct peer-to-peer communication between the two applications that need to share some information. After the communication is established, the two applications will exchange all necessary information. By this way, the two applications are integrated seamlessly, without manually intervention from both users.

The sections that follow describe in detail some existing systems in the market and educational research with respect to application integration and how they differ from the approach proposed in this Thesis.

2.3 EXISTING COMMERCIAL INTEGRATION SYSTEMS

A large quantity of B2B protocols and proposals are available in today's market [15,16,105]. Some of the existing protocols are Electronic Data Interchange (EDI) [4], SWIFT [128], RosettaNet [109], ebXML [35] and OAGIS [94].

Each implementation of the above protocols is different and was designed for specific reasons. Specifically, the "*RosettaNet*" and "*ebXML*" standards focus on peer-to-peer based B2B protocols and have developed formalisms for the description of B2B protocols. However, no formal semantics are in place so that the interpretation of the formalisms might cause incompatibility in a compliant software.

Several standards like ebXML [35], XLANG [129] and Business Process Modeling Language (BPML) [12] address the definition of "processes" in the context of web services and B2B protocols. All of those are initial versions of the standards and are not widely adopted yet.

Furthermore, currently there are thousands of integration solutions in the market trying to address the integration problem in general [22, 23, 24, 64] as well, such as Microsoft BizTalk Server [48, 71, 82, 85, 86, 87], IBM Server Integration Solutions [46, 52, 53, 54], Compaq *NonStop* Solutions Integrator [28], Oracle Integration Server [95, 96], TIBCO [131], Versata [137] etc. Each of them is providing its own solution, based on some specific requirements but only a few have tried to address the "***dynamic***" issues of integration using the peer-to-peer technology. This section presents some of the existing systems that are widely accepted by the international market.

2.3.1 Metis Collaboration Platform V 5.0 (MCP)

MCP is a Peer-to-Peer application integration platform that powers large-scale commerce and e-Business solutions for global enterprises. MCP uses modules as the basis for complex technology integrations and composite application development, enabling seamless interoperability through protocol-neutral architecture. The pre-built modules leverage the existing network and applications within the enterprise, without having to be modified or reconfigured to collaborate freely [83].

MCP has three major applications within an enterprise:

a) **Peer to Peer Application Integration**

MCP integrates legacy assets, protocols, middleware, hardware and networks as well as a wide range of applications. All technologies, irrespective of protocols, operating system or language, are integrated together through MCP enabling complete interoperability across all technologies and the enterprise.

b) **Composite Application Development**

MCP can build composite applications, either web-based (in the vein of web services) or for essential enterprise-critical applications. Enabling fast, easy application development reducing time frames and rapid generation of Return On Investment (ROI).

c) **MCP's Virtual Stack**

MCP can achieve a robust, flexible, open and distributed vertical stack by acting as the business operating system. This may result in leveraging the existing systems and saving time and money in the process.

The MCP consists of three logical environments: a) a Development Environment, b) an Application Assembly Environment and c) an Execution Environment, which contain specific tools and facilities used to build, assemble, deploy and manage collaborative applications. APIs and core modules provide the basis of the environment. A Java-based graphical facility enables the configuration of modules, the definition of components and applications, the deployment and later management of executing applications [83].

a) **The Metis Development Environment:** Consisting of lightweight APIs and a graphical tool for configuring modules, the Metis Development Environment provides developers with tools to build Metis-enabled application components from new and existing C, C++ and Java application code.

b) **The Metis Application Assembly Environment:** Consisting of a graphical Application Builder tool and pre-built modules, the Metis Application Assembly Environment enables developers to build deployable applications from Metis-enabled modules and components. Metis is building a comprehensive list of pre-built components that can be used within Metis applications. These components include a data archiver and custom database modules for commercial databases and will include message queue modules for IBM Message Queues (MQ) and Microsoft Message Queues (MSMQ) products, transaction system modules for Netscape Transaction Server and Net Dynamics and a CORBA module, among others. All pre-built modules are completely configurable and extensible.

c) **The Metis Execution Environment:** Applications developed within the Application Assembly Environment are ready to be deployed and run on the network. To do this, the Network Deployment Tool is used to install the Metis Runtime Environment and distribute application components onto the network devices.

By utilising pre-built components, the Metis Collaboration Platform enables companies to extend their technology environment by leveraging existing technology investments. These components handle specific tasks and interface with specific third-party technology products. Using these components, the Metis Collaboration Platform enables integration with common industry functionality and products.

- **Data Archiving:** The Data Archiving component can be used as a generic distributed data store. It provides a general storage and retrieval facility for archived data. It enables joining across archives, grouping of data sets and secure storage and retrieval of data.
- **Database Integration:** MCP includes database access components for Sybase, Oracle, MySql, Microsoft SQL, and ODBC data sources. These components extend each vendor's native library calls directly into Metis messages, enabling Metis-enabled modules to transparently access data from traditional databases and bring the complete functionality of each vendor's DBMS into Metis applications.
- **MQ Series:** The IBM MQ Series message queuing product is used in many systems that require communication between mainframes and UNIX machines. MCP will fully support communications to and from MQ Series with a

component that enables rapid integration into systems that utilise MQ Series as a communication vehicle.

- **MSMQ:** MCP will enable access to Microsoft's MSMQ message queuing product with a transformation module that encapsulates the native MSMQ API to allow access to Microsoft-supported queues.
- **Transaction Systems:** The Netscape Transaction Server and Net Dynamics Transaction server are widely used in the industry today. MCP will support these servers bi-directionally with modules that facilitate access to these servers from any Metis-enabled module. In addition, Metis will supply a plug-in module for these transaction servers, which will allow requests to be made from the servers to any Metis-enabled module.
- **CORBA:** This component will enable Metis-enabled modules to communicate with Common Object Request Broker Architecture (CORBA) and other Remote Method Invocation (RMI) based systems. The messages that invoke CORBA methods will be loosely bound and applied to any CORBA object. The Metis CORBA component will maintain state among requests to CORBA services from other modules.
- **Palm Devices:** The Metis Runtime Environment will run in a very small footprint on Palm devices, enabling developers to write Metis-enabled modules for Palm devices.

After reviewing the Metis overview architecture, this platform may appear to be an appropriate solution for dynamic application integration. However, this is not the case, since the Metis Platform addresses the dynamic application integration problem by having all possible components integrated into one platform, so when a business requires integration, the appropriate modules are used to design the integration solution for that business. This approach inherits a lot of problems since some integration solutions may require specific integration requirements that cannot be handled by the existing modules already integrated into the Metis Platform.

In this Thesis, a more dynamic and robust solution is addressed that will satisfy the needs of any integration solution without depending on existing modules already developed for specific tasks.

2.3.2 WebV2 Peer Beans

WebV2 provides a next generation approach to application integration and business process automation using Web Services [140, 141]. WebV2 eliminates the central server and provides for totally decentralised business process execution across applications, services, EAI hubs, and application servers with lightweight software components.

WebV2's approach is to replace the monolithic hub with software components associated individually with the applications to be integrated, as illustrated in figure 2.2 [141]. These components are based upon WebV2's PeerBeans software development and execution environment. The PeerBeans service-oriented architecture allows for the automated provisioning, location, preparation, and execution of services across a network. As such, the PeerBeans components provide the functionality provided by the hub, but in a fully distributed and decentralised web services manner.

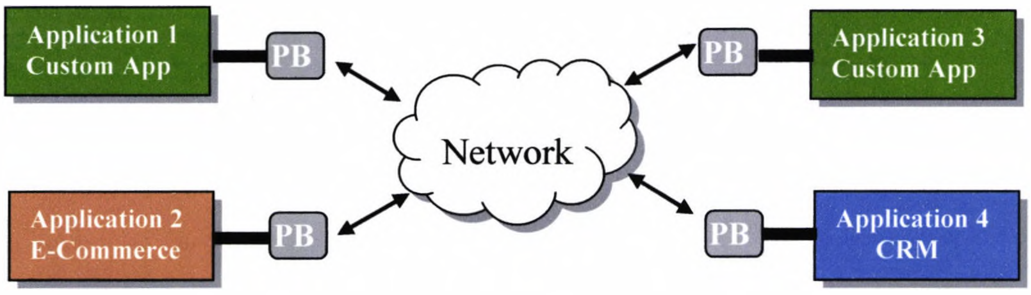


Figure 2.2 *The WebV2 Peer-to-Peer Architecture*

Applications are linked to other applications by means of the PeerBeans components, which handle the business process associated with that application, as well as message routing and transformation. The PeerBeans components interact through Intranet, Extranet or Internet and support dynamic change in the business logic and participating applications. The PeerBeans solution is totally distributed, that is, no central server is required. There is no single point of failure and massive scalability is inherent in the architecture. Furthermore, applications are provided with complete autonomy and control of the business logic is distributed to where it belongs.

The "*PeerBeans*" product is a Java-based core infrastructure development and deployment software for peer-to-peer networks. The PeerBeans product provides a comprehensive suite of peer-to-peer and application integration functions, including functions for network interconnection (opening, closing and managing network connections), message management (transformation and routing), security (authentication, encryption), and collaboration (resource sharing, transactions). The WebV2 implementation of the core PeerBeans container is one hundred percent Java and extremely compact, allowing it to run on virtually any computer system, including mobile devices. The PeerBeans infrastructure is highly scalable, fast, fault tolerant, and requires little maintenance. Each application is tied into its own PeerBeans software, automatically providing its services and resources to other applications in the network. The PeerBean framework automatically takes care of discovery of other applications, and coordinating the business processes as defined by the applications.

A peer is simply a computing process implementing a PeerBean container. Any number of peers may run on any number of machine hardware nodes in a network. The system is made up entirely of the PeerBeans software distributed across the network and no extra server is needed for management of the PeerBeans software.

In order to support the network of peers, WebV2 provides a number of so-called system peers, which themselves are implemented using the PeerBeans solution. These peers provide special domain-independent services, called interoperability services. These services can be provided by any number of peers on the network in a decentralised, distributed, and redundant fashion. These services are described in more detail below.

- **Application Discovery Service:** An intelligent registry of resources and services provided by application peers, allowing for the automatic determination of which applications participate in any given business process. When new applications are linked to their corresponding PeerBeans software, they automatically register their services with the Application Discovery Service and are immediately available for business. This extends the web services UDDI standard with the following features:
    - **Subscription**: Updates in information about services is automatically distributed to subscribing clients.

- **Service Request Forwarding**: Service requests are automatically forwarded by the Application Discovery Service to the appropriate service, which responds in turn directly to the client. This saves communication overhead.
- **Networking**: Much like the Domain Name Service (DNS) system in the Internet, arbitrarily many and partially redundant Application Discovery Services can be deployed on the network so as to ensure massive scalability and robustness.

❑ **System Monitor Service:** This provides the administrator with an overview of the system, the currently running application peers, system services, their communication, and network load.

❑ **Network Transformation Rule Library Service:** When an application peer receives a message from another application peer in a format unrecognised by the application, it must employ transformation rules to convert the message into its own format. The transformation rule library provides a repository of transformation rules; the appropriate ones can be downloaded dynamically, cached, and applied by the application peer.

❑ **Network Business Logic Library Service:** A repository of generic and application-specific business logic rules available on the network for access by application peers. The rules can be dynamically downloaded and followed by an application peer.

❑ **Security and Authentication Services:** These are used for supporting establishment of secure transactions and authentication of nodes in the network. WebV2 provides several mechanisms of establishing security and authentication i.e. username and password, kerberos.

❑ **Transaction Logging Service:** This handles and preserves a complete history of all transactions carried out among the applications registered with this service.

WebV2 eliminates the problems of flexibility, robustness, and scalability associated with the point-to-point and hub-and-spoke architectures by providing a peer-to-peer architecture based on a loosely-coupled messaging-based coordination middleware called PeerBeans software. The PeerBeans product allows for distributing message brokering and business logic control functionality across the network, thereby avoiding a single point of failure and allowing total autonomy of the applications. The peer-to-peer network provides supporting system services.

After a thorough reviewing of this platform, it was deduced that the architecture approach applied is similar to the approach taken by the Metis Collaboration Platform. This is supported by the fact that each pre-complied PeerBean can perform a specific task assigned by the Platform, since each PeerBean correspond to one component. Then these PeerBeans can work together to simulate the business process of one application. Moreover, this approach inherits many problems as well, just like the Metis Platform, for the reasons already discussed in section 2.3.1.

In this Thesis, this kind of integration is avoided, because a number of pre-compiled modules can be put together to produce a dynamic integration platform where applications can be integrated dynamically. Instead, the main thrust of this Thesis, is focused on dynamic issues that will satisfy the needs of any integration solution without depending on existing modules already developed for specific tasks.

2.3.3 Orbix E2A Web Services Integration Platform

IONA's Orbix *End 2 Anywhere* (E2A) Web Services Integration Platform is the first Web services platform built to handle critical business integration processes both inside and outside the firewall. Orbix E2A leverages Web services standards and service-oriented architectures to provide an open, flexible and low-cost integration solution. The Orbix approach eliminates the requirement to deploy one vendor's technology on both ends of an integration connection. Orbix provides all the tools and management services required for reliable, scalable and secure business integration [59, 136].

The Orbix E2A Web Services Integration Platform is available in three editions:

- The *Collaborate Edition* is a single platform for total business integration that provides a comprehensive set of integration solutions for process collaboration both inside and outside the enterprise.
- The *Partner Edition* provides a low-cost, easy-to-deploy Web services connector that enables customers to seamlessly collaborate with trading partners that have deployed the Collaborate Edition or other integration technologies.
- The *XML Bus Edition* is a low-cost Web services integration platform built for developers who want to take a 'pure' Web services approach to integration.

Orbix E2A Collaborate Edition is a comprehensive, next-generation integration broker that enables multi-vendor, multi-technology process collaboration both inside and outside the enterprise. This Web services-based platform leverages new and existing integration standards to reduce the cost of application integration and ensure complete interoperability with disparate systems. Orbix E2A Collaborate Edition offers a service-oriented architecture, based on IONA's patented Adaptive Runtime Technology (ART) - a flexible, scalable and extensible infrastructure that speeds time-to-value in any integration initiative.

Orbix E2A Partner Edition enables efficient communication across multiple tiers of the supply chain by providing a low cost and easy-to-use connector application for seamless collaboration between trading partners. Built for small and medium enterprises (SMEs), the Partner Edition provides easy access to Collaborate-based information hubs, other individual Partner Edition deployments, or any other integration or web services technology. Based on IONA's Web Services Integration Platform, the Partner Edition allows trading partners to securely send, receive and respond to business communications with minimal effort.

IONA's XML Bus Edition (part of the Orbix E2A Web Services Integration Platform family) is an integration platform for the integration of heterogeneous applications built with .NET, Java, Java 2 Enterprise Edition (J2EE), Java 2 Micro Edition (J2ME), and CORBA. This platform provides the qualities of service that mission-critical Web services demand, including rapid development tools, scalable and reliable deployment infrastructure, and enterprise-class management for deployed Web services.

The XML Bus Edition is the most interoperable Web services implementation available, with broadly-vendor-tested XML, Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Universal Description, Discovery and Integration (UDDI) compatibility, and a wide range of standard API support, including Java API for XML Registries (JAXR), Java API for XML Messaging (JAXM) and more.

E2A is driven by two simple ideas [57, 58, 60]. Firstly, any application, whether firmly established or newly developed, should be able to interact with any other relevant application, no matter where it resides or how it was developed. Secondly, application development and application integration should not be treated as separate processes. In a fully connected world, they must be part of the same process.
Orbix E2A offers a service-oriented architecture, based on IONA's patented Adaptive Runtime Technology (ART) - a flexible, scalable and extensible infrastructure that speeds time-to-value in any integration initiative. Some features of the Collaborate Edition are the following:

- ❑ High-productivity graphical tools allow rapid creation of business process models.
- ❑ Flexible architecture designed to address both EAI and B2B integration problems.
- ❑ Wide array of Customer Relationship Management (CRM) and Enterprise Resource Planning (ERP) adapters extend the value of internal systems, by seamlessly incorporating them into business process flows.
- ❑ Technology adapters support major industry technologies and protocols (i.e. Relational Database Management System (RDBMS), Common Object Request Broker Architecture (CORBA), Java, Java Message Service (JMS), Message Queues Series (MQSeries), XML) to bridge to important middleware, database, mainframe, and legacy systems.
- ❑ Graphical data transformation tools for integrating Enterprise Applications with no coding.
- ❑ Support for standard B2B collaborations enables rapid implementation of business processes.

- One hundred percent J2EE and XML-based, providing extensibility and a wide range of compatible developer tools.
- Graphical tools for the creation of Web services from existing systems, including Java, EJB, and CORBA-based applications.
- Graphical tools for monitoring and optimising business processes.
- Structured adapter framework for easy creation of custom adapters to integrate with legacy systems and mainframe applications.

The purpose of this platform is to support enterprise applications, which implement the enterprise's core, mission-critical business processes. IONA's primary focus is on applications in telecommunications, financial services, and high-technology manufacturing, and government/public sector. The Orbix platform minimises the time to integrate these applications together.

On the other hand, the purpose of this Thesis is to provide a quick and robust dynamic integration solution for legacy applications by creating java web services dynamically. This solution is important since it is not required to take the legacy applications off-line in order to accomplish the dynamic integration. In addition it not required to have a prior knowledge of the underlying data structure of the application. The time it takes to integrate two applications via the web services technology will only take few minutes and hence the cost of integration is minimised.

2.3.4 Sun ONE Integration Server

The Sun ONE Integration Server, Enterprise Application Integration (EAI) Edition. is a software product designed for enterprises that need to integrate packaged, custom, legacy, and new Java applications [125].

This platform makes it possible for companies to integrate core business processes with multiple applications running on multiple operating systems across multiple communication protocols within the enterprise. Companies benefit from the automation of business processes across distributed heterogeneous information systems with increased productivity and efficiency.

Moreover, this platform provides support for Web services through SOAP messaging, which can export a service definition via Web Services Description Language (WSDL) to UDDI or other SOAP clients. This is achieved by a remote procedure call, of type XML that invokes a call on another application using SOAP and the Web. It allows different parts of the enterprise to exchange structured information over the Web, independent of the types of systems or application platforms. SOAP functions as a wire protocol to connect Web services to multiple Web portals [125].

The Key Features of the Sun ONE integration server are:

- **Flexible Business Process Management**: Fully featured integration broker for the enterprise including business process management, message transport, data transformation, intelligent routing, and adapters. Integrating systems, data and applications requires business process management to create, deploy and manage new business solutions. This platform provides support for both fully automated process flow as well as human interaction workflow. It enables an entirely new class of flexible enterprise applications that can incorporate existing components into powerful, graphically defined process definitions.

- **Data Transformation**: Sun ONE Integration Server contains an XSL processor that utilizes the Extensible Stylesheet Language Transformations (XSLT) to transform XML documents. This process translates data elements among multiple structures of XML. These Modules for XML and XSLT, help enable faster development time to market.

- **Message Transport:** The message transport for the Sun ONE Integration Server is a two-way channel for XML/SOAP-based data messages sent over HTTP/HTTPS or the Java Message Service (JMS) Application Programmers Interface (API), a publish-and-subscribe solution with persistent message queuing. This enables integration for Web services through SOAP Messaging, which can export a service definition via WSDL to UDDI or other SOAP clients.

- **Intelligent Routing**: The flexibility that exists on this Integration Server is that the intelligent routing function enables customers to pick out arbitrary elements from the exchanged document and make routing decisions based upon the content. Unlike other approaches, a customer does not need to specify header fields. The primary benefit is that the proxy or message transport defines how the message is going to be routed.

- **Adapters**: This integration server offers technology adapters for Common component technologies, Custom, Mainframe and Packaged Systems. The vast majority of integration connections are to legacy and custom applications. The Sun ONE Integration Server facilitates the rapid development of robust, XML-based custom adapters with the XML Adapter Designer. This powerful adapter Software Development Kit (SDK) provides a fully leveraged XML and framework-based approach, ease of use, state management, application session context and performance agents.

- **Authentication**: Lightweight Directory Access Protocol (LDAP) integration provides a single source of authentication, user data, and roles definition.

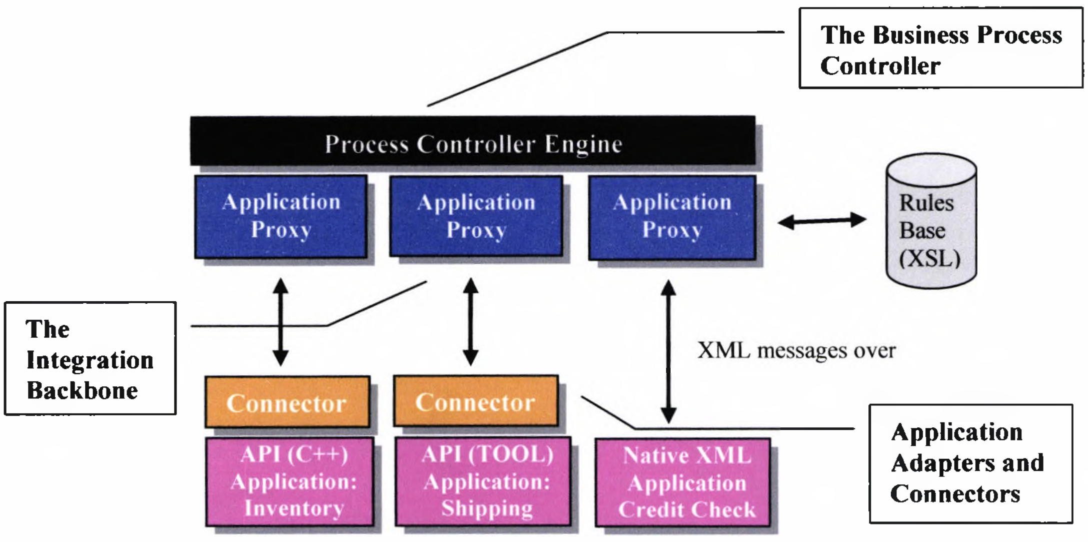

The Business Process Controller
Process Controller Engine
Application Proxy
Application Proxy
Application Proxy
Rules Base (XSL)
The Integration Backbone
XML messages over
Connector
Connector
API (C++) Application: Inventory
API (TOOL) Application: Shipping
Native XML Application Credit Check
Application Adapters and Connectors

Figure 2.3 *The Sun ONE Integration Server Architecture*

The Sun ONE Integration Server application system consists of the following components, as illustrated in figure 2.3 [125]:

- **The Business Process Controller engine:** This Integration Server offers an integrated process management solution that merges open component development with business process automation. The process controller enables a whole new class of flexible enterprise applications, by allowing business process managers to incorporate existing components, including packaged application components, into powerful, graphically defined process definitions.
  This combination of process automation controls applied to components is known as Open Component Sequencing.

- **An Integration Backbone**: The Sun ONE Integration Server Backbone facilitates communication between the business process engine and participating applications. The backbone consists of a manager and its associated proxies. A proxy acts as a two-way channel for messages sent over HTTP or JMS. It is a client of the process engine and can act as a client or server with respect to its application.
  The proxy interprets and transforms XML messages from its application to the process engine, as well as messages from the process engine to its application.
  The Integration Backbone consists of:
  - A set of application proxies, each representing the business service provided by its application.
  - A manager to handle start-up, shutdown and general administration.
  - An XSL rule base, consisting of rule documents. Specific rule documents are assigned to proxies to interpret and transform messages

- **Application Adapters and Connectors:** The Integration Server Adapters allow package or custom applications to participate in a business process by translating to and from application APIs and XML. Sun ONE Integration Server Adapters are:
  - **Lightweight**: These small in size Java or compiled C++ routines use a simple API to XML schema, and contain no application logic.

- **Standards-based**: XML/HTTP is an open, ubiquitous protocol usable by HTML or Java clients.
- **Scalable and reliable**: Adapters use XML document handling and interpretation, connector replication, standard exception handling, management agents, with logging and performance instrumentation.

To sum up, the Sun ONE Integration server provides either a set of ready-made and not application-specific tools that will enable developers or companies to create their own integration packages to be incorporated to their existing applications or a set of integration packages that require additional coding to create adapters in order to incorporate existing legacy applications with their integration servers. Either of these features is not required to create a dynamic application integration server.

On the other hand, the proposed integration solution includes the peer-to-peer technology, which is not part of the Sun's Integration architecture as illustrated in figure 2.2. This is the most important factor in order to implement robust and dynamic application integration solutions.
The systems that have been evaluated in the previous pages address a number of issues regarding application integration using the peer-to-peer technology and the web services infrastructure. Most of them use pre-configured components from other vendors and they use them in a co-operative environment so that by combining two or more of these components together they achieve the integration solution they need.

The main difference between these systems and the system proposed in this Thesis is the fact that the author suggests a more dynamic and easier application integration approach, where the Java Web Services, created by the Java Web Service Integrator (JWSI), are dynamically re-configured based on the state of the application. This is critically important since the status of the applications change rapidly according to external factors and the dynamic nature environment of the Internet. Many people wrongly assume that all users and applications are connected to the Internet and that the bandwidth is always available. For example, with the existing systems, a newly created web service could not be in a position to adapt to changes made to an application.

On the other hand, the benefits of the proposed architecture are enormous in comparison to the existing solutions discussed in section 2.3. The java web service module will be in a position to modify dynamically the characteristics of the published web service so that other applications on the Internet can make advantage of these newly added characteristics.

In addition, any application linked to the JWSI can dynamically and without the knowledge of the application user, be integrated with other applications based on some criteria, for the purpose of interchanging information using a direct peer-to-peer connection. This approach, which appears to be novel, leads to tremendous advantages, as there is no need to pre-design and thus implement specific integration solutions between the participated applications. This is of great importance since any new applications can be integrated seamlessly in a P2P environment.

As an example, a new application has been designed and developed to measure the stability of the ship in a simulated environment. This application requires information that is generated by other ship-related applications in order to function properly and to produce the correct results. In order to achieve this integration requirement in the above existing system, it would take a lot of time and money to produce a viable solution.

Using the solution proposed in this Thesis, the new application would be easily integrated into the P2P integration platform and will also be able to use the data that has been generated by another application in this platform for the purpose of producing the expected data from that application.
Moreover, in order for any application clients to be integrated with the proposed integration server, they just need to install the JWSI. This Java application is relatively small in size (1 Mbyte) and can be installed in a number of operating systems. The installation process is simple and the pre-configuration, before dynamic integration can take place, is minimal.

To sum up, it can be seen that the proposed solution contains a number of benefits over other or similar P2P architectures that will enable the easy and dynamic application integration not only in the shipping sector but essentially in any sector in the business market.

2.4 EXISTING ACADEMIC INTEGRATION SYSTEMS

As already mentioned in section 2.3, there are a number of protocols in the market which try to address the P2P architecture. One educational protocol that addresses this area is the GISP (Global Information Sharing Protocol) project, which aims at a world-wide distributed index [65]. A distributed index consists of a set of pair data (key, value) shared by many peers. Each peer is responsible for a part of the index based on a hash function. Every peer is basically flat and there is no single point of failure. A distributed index is an essential building block for peer-to-peer systems. Each peer promotes its strengths so that stronger peers contribute more than weaker peers. Redundancy is important for defending against undesirable peers. Peers replicate pair data so that each pair data of the index is covered by several peers.

In terms of Peer-to-Peer security and reliability there are a number of educational researches that try to address this subject as well. One of them is the E-Speak project [68]. E-Speak is an e-service infrastructure where services advertise, discover, and interoperate between each other in a dynamic and secure way. The E-Speak security adopts a multi-layered approach and builds a range of protection mechanisms on top of the Public Key Infrastructure. The E-Speak advertising services have a dynamic pluggable architecture and implement a scalable wide-area discovery based on distributed queries. It is argued that E-Speak may be used as the common secure, scalable infrastructure for different multiple P2P applications. In addition, substantially research has been performed by the Stanford University which concentrates on issues like locating resources in P2P systems, resource aggregation, availability and authenticity using the Peer to Peer technology [78, 79, 114]. Although these are very important factors for a reliable P2P, it is not fully applicable for the purpose of this Thesis, since this Thesis is concentrating on the dynamic Application Integration using the P2P technology and not how to design and develop a P2P infrastructure.

One the other hand, Application Integration can also be achieved via Web and not using the P2P technology. An example of such an implementation is a framework for web-based Enterprise Application Integration from San Jose State University [102]. This framework provides a very flexible architecture for seamless integration of enterprise applications using available technologies like XML. This implementation does not refer to the dynamic nature of Application Integration with the help of P2P, but rather concentrates on the Application Integration via the Web.

A very good research topic that refers to the dynamic nature of application integration is the proposal from University of Twente in England. This proposal suggests a framework for dynamic B2B interaction [51]. A B2B transaction is divided into the interaction part and business implementation part to support flexible interaction. A component based system framework is proposed to support the B2B transaction execution. To support dynamic B2B services, dynamic component composition is required. Service and component notions are combined into a composable service component.

Although this implementation is very much close to the proposed solution of this Thesis, this implementation does to refer to the dynamic Application Integration using the P2P technology but uses a component-based approach to solve the problem of application integration.

However, all of these solutions require manual coding to integrate the different types of applications. Given the multi-tier nature of B2B applications, where each tier has to be developed separately, it becomes a very complex task to integrate these applications without an integrated visual and code generator environment. A framework that addresses these issues is called GAIL (the Gen-it Abstract Integration Layer) [67]. GAIL provides a B2B application framework and customised model-driven architecture approach for deploying and integrating different applications together. In addition, GAIL plugs into most of the leading Unified Modeling Language (UML) modeling tools and provides a parameterised architecture to generate most of the code needed in deploying and integrating applications.

To sum up it can be seen that there are several on-going educational projects and proposals that try to solve the Application Integration problem, which use either the Peer-to-Peer or other related technologies. None of them addresses the problem stated in Chapter 1, using the "dynamic" factor on top of the Peer to Peer technology, which is one of the most important aspects of this Thesis.

2.5 CONCLUDING REMARKS

In this chapter, an overview of the dynamic application integration is presented and the importance of a peer-to-peer architecture in application integration is discussed in detail. A comprehensive literature survey for existing commercial systems or educational projects in the market has been presented in conjunction with a comparison between them and in relation to the dynamic integration solution proposed in this Thesis.

The chapter ends with a summary of what has been discussed giving an overall picture of the differences between the proposed solution and the integration servers discussed earlier in this chapter.

Chapter 3

REQUIREMENTS SPECIFICATION

3.1 INTRODUCTION

Details relating to the organisational context and requirements for the thesis will be presented in this chapter. A detailed survey of methodologies is then conducted, highlighting the basic features of each methodology, giving reasons why they have been chosen to satisfy the thesis's requirements.

Specific focus is given to the Web Services and how they can work together to provide a P2P architecture for dynamic integration. This is important since, one of the requirements of this thesis is to investigate the dynamic issues of application integration and how each peer program, without any user intervention, can find other peers on the Internet for interchanging data.

The focus of this chapter is to provide a detailed description of the architecture of the designed system.

3.2 SYSTEM REQUIREMENTS

The following section lists the requirements considered in this thesis. This will help not only the author but also the users of this system to better evaluate and verify the integrity of the system. In addition, this section will help to prepare a comparison between the initial and accomplished requirements of this thesis. The system is designed and implemented taking into consideration the following requirements, in order to solve the problems that have been identified in Chapter 1.

i. **Provide Dynamic Application Integration**

   As explained in section 2.1, dynamic application integration in the context of this thesis means that any application can be integrated and hence interchange data with another application in the Network or Internet, dynamically without prior knowledge of the underlying infrastructure or data structure of the other application. This is accomplished with the help of the *Java Web Service Integrator* (JWSI), which generates Java Web Services on the fly, that is dynamically, while the potential applications to be integrated are running. How this is achieved, is described later in section 3.5 of this Chapter.

ii. **Provide peer to peer functionality**

   One of the main requirements of this thesis is to design and develop a system with peer-to-peer capabilities. A P2P enabled application should be capable of locating other peers in the network. Once an application is able to locate other peers, it should be able to communicate with them using messages. Once the communication is established with other peers, the application should be able to receive and provide information, such as content.

iii. **Provide dynamic discovery and publishing of Java Web Services**

   An essential requirement on top of the P2P functionality is the ability of the system to dynamically discover and publish Java Web Service using the *Universal Description, Discovery and Integration* (UDDI) registry. This is an important requirement because in order for a P2P system to function properly it requires a global registry where all Java Web Services will be registered and subsequently be discovered dynamically by other Java peers.

iv. Provide small Java peers

To establish a communication link between the two dynamically integrated applications, peers are required to act as agents on the client's and server's machine, to send and receive requests from both locations. These peers are universal enough and can be installed in different machines under different operating systems because they are based on the Java language. The purpose of these peers is to dynamically publish Java Services on a UDDI and at the same time dynamically discover new Java Services for the purpose of integrating two or more applications together.

v. Provide SOAP and WSDL specification compatibilities

Another requirement of the proposed system is the ability to generate dynamic Java Web Services, which comply with the *Simple Object Access Protocol* (SOAP) and the *Web Services Description Language* (WSDL) specification. This requirement is also significant in case external systems may want to discover and hence use the already published Java Web Services produced by the system. This makes the system extremely flexible and adaptable to future collaboration with other systems, which are already compatible with these standard specifications.

vi. Provide various transfer and security protocols

Another important aspect of this thesis is the dynamic selection and negotiation of different transfer and security protocols between the two peers. Firstly, this is accomplished by having the two peers negotiate the appropriate transfer protocol based on the bandwidth availability of the communication between them. Secondly, based on some parameters found on each Java Web Service, the security protocol is dynamically selected, in case the information sent is confidential and based on the availability of that protocol between the peers. Sometimes, the information sent is not so confidential and the two peers may decide not to use any security protocols for that session. This is important since sometimes the communication link, for example a satellite link between the ships and the shore offices, is very expensive especially for large data transfers.

vii. Provide dynamic XSLT transformations

Occasionally, the web services are not capable to handle all relevant information needed to be sent between two peers. In such a case, a mechanism is required to dynamically convert the data structure from the source peer to the data structure of the destination peer. This is archived via the use of the XSLT language. This language creates the mappings of two different data structures by describing a template that needs to be applied to the given data structure. The user of each application usually creates these mappings manually. This thesis is trying to suggest ways to dynamically generate these mappings and hence dynamically apply them to transform the given data format to the destination.

In addition, some applications may use databases to store their information and hence these data must be exported in a format that is readable by the requested peer. This can be also achieved using the XSLT generator, which will transform the data stored into the database to XML and vice-versa.

viii. Demonstration of all the above features in a simulated environment

This requirement will help the author and the users of this system to evaluate the overall functionality and performance of the system as well as to identify whether the system has been designed and developed according to the requirements specified and explained in this section.

Since the requirements have been identified and explained, it is now proper to perform a research on methodologies that exist in the market, which will be used to attain, in a way acceptable to the users, the proposed above requirements. The following section concentrates on this research.

3.3 SURVEY OF METHODOLOGIES

To achieve the above-mentioned requirements, a thorough research has been carried out to find the most suitable methodologies to be used for this thesis. The following section discusses the methodologies used, giving at the same time an explanation of why these particular methodologies have been selected for the proposed goal. Because some of these methodologies have limitations in achieving a particular set of tasks, combination and collaboration between these methodologies may be required, which will be discussed further in this report.

3.3.1 Peer-to-Peer Architecture

Message broker processing is a mixture of schema and content transformation, rules processing, message splitting, and combining, as well as message routing. Once the processing is complete, the information is sent to any target systems that need to receive that information using whatever native format the target application can understand, such as XML, Java Message Service (JMS) message, proprietary, etc, see figure 3.1.

The hub-and-spoke EAI solution is resource-constrained because all the processing takes place on a single server. Eventually, the number of connected systems and the information traffic will saturate the available resources of the integration server (memory, processor, and disk), resulting in reduced performance.

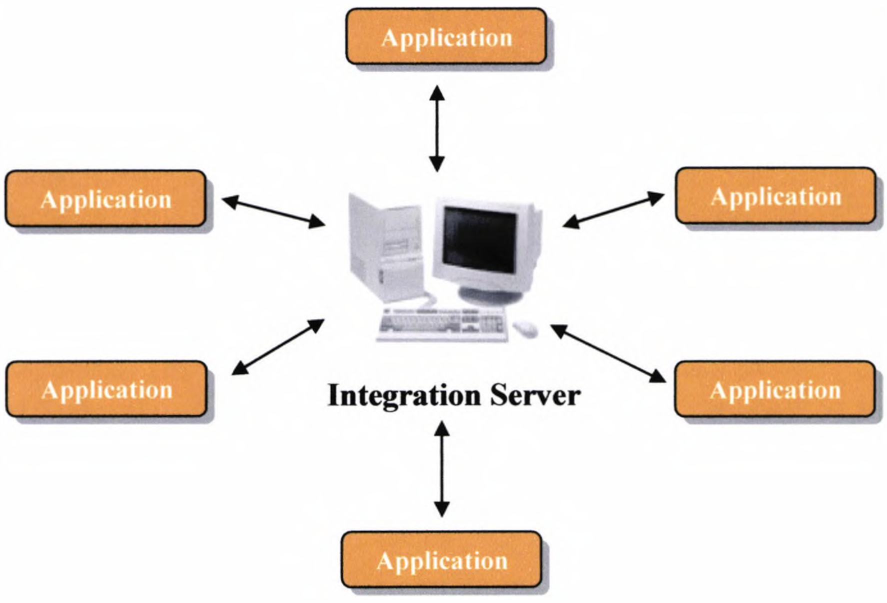


Figure 3.1 *The traditional hub-and-spoke architecture for EAI*

The latest and most advanced technology in the world of EAI is the *peer-to-peer* architecture [25, 76, 110, 121, 132]. This is similar to the message broker architecture (see figure 3.1) but with one difference. There is no central integration server. Instead, the message translation, routing, splitting, and combining occurs at the source and target systems within little message brokers known as agents or peers (see figure 3.2).

The agents consume information from the single system they are connected to, process that information, and send it directly to any target system(s) interested in receiving that information or message. The result is a virtual *"hub-and-spoke"* type architecture, where the description of each service is located on a central location called UDDI.

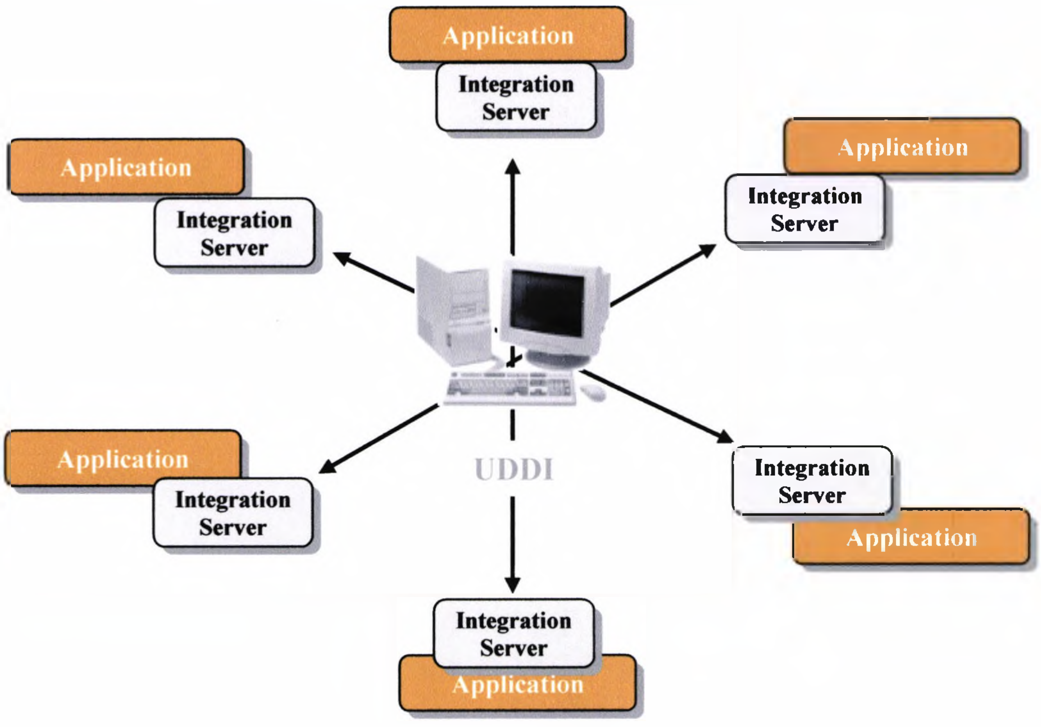


Figure 3.2 *A general Peer-to-Peer Architecture*

Because the processing occurs on many systems in parallel, this architecture has the potential for integrating hundreds if not thousands of systems, within both EAI and B2B solution domains. Peer-to-peer eliminates many problems, including single point of failure and single-server saturation.

The advantages of a Peer-to-Peer architecture are [113,120]:

- A P2P application is capable of locating other peers in the network.
- Once an application is able to locate other peers, it is capable of communicating with them using messages.
- Once the communication is established with other peers, the application is able to receive and provide information, such as content.

Finally, peer-to-peer is the future of EAI, and there are a number of aggressive middleware companies, both start-up and existing, looking to move into this architecture.

3.3.2 Web Services

A Web Service is an application that exposes a programmatic interface using standard, Internet-friendly protocols. Web Services are designed to be used by other programs or applications rather than by humans. Programs invoking a Web Service are called clients. *Simple Object Access Protocol* (SOAP) over *Hyper Text Transfer Protocol* (HTTP) is the most commonly used protocol for invoking Web Services. By exposing data and functionality using standard protocols, Web Services make it easy to build sophisticated dynamic applications that integrate many features and content. There are three main uses of Web Services [42, 50, 55, 56, 90, 99]:

- **Application Integration**. Web Services within an intranet are commonly used to integrate business applications running on disparate platforms. For example, a .NET client running on Windows 2000 can easily invoke a Java Web Service running on a mainframe or UNIX machine to retrieve data from a legacy application.
- **Business Integration**. Web Services allow trading partners to engage in e-business leveraging the existing Internet infrastructure. Organisations can send electronic purchase orders to suppliers and receive electronic invoices. Doing e-

business with Web Services means a low barrier to entry, because Web Services can be added to existing applications running on any platform without changing legacy code.

- **Commercial**. Web Services focus on selling content and business services to clients over the Internet similar to familiar Web pages. Unlike Web pages, commercial Web Services target applications not humans as their direct users. Many airlines, for example, expose flight schedules and status Web Services for travel Web sites and agencies to use in their applications. Like Web pages, commercial Web Services are valuable only if they expose a valuable service or content. It would be very difficult to get customers to pay for using a Web Service that creates business charts with the customers' data. Customers would rather buy a charting component, for example a *Component Object Model* (COM) or .NET component and install it on the same machine as their application. On the other hand, it makes sense to sell real-time weather information or stock quotes as a Web Service.

In order to understand better the advantages of Web Services, the following section lists a few of the most essential differences between traditional *Enterprise Application Integration* (EAI) solutions and Web Services [5, 18, 33, 43, 111, 112, 115, 119, 133]:

- **Simple**: Web Services are much simpler to design, develop, maintain, and use as compared to a typical EAI solution, which may involve distributed technology such as *Distributed Component Object Model* (DCOM) and *Common Object Request Broker Architecture* (CORBA). Once the framework of developing and using Web Services is ready, it will be relatively easy to automate new business processes spanning across multiple applications.

- **Open Standards**: Unlike proprietary EAI solutions, Web Services are based on open standards such as UDDI, SOAP, HTTP and this is probably the single most important factor that would lead to the wide adoption of Web Services. The fact that they are built on existing and ubiquitous protocols eliminates the need for companies to invest in supporting new network protocols.

- **Flexible**: Since EAI solutions may require point-to-point integration, changes made at one end have to be propagated to the other end, making them very difficult and time consuming in nature. Web Services based integration is quite flexible, as it is built on loose coupling between the application publishing the services and the application using those services.

- **Cost Effective**: EAI solutions, such as message brokers, are very expensive to implement. On the other hand, Web Services may accomplish many of the same goals faster and are much more cost effective than EAI solutions.

- **Scope**: EAI solutions, such as message brokers, integrate applications treating them as single entities, whereas Web Services allow companies to break down large applications into small independent logical units and build wrappers around them. For example, a company can write wrappers for different business components of an *Enterprise Resource Planning* (ERP) application, such as order management - purchase order acceptance, status of order, order confirmation, accounts receivable and accounts payable.

- **Efficient**: As mentioned in the previous point, Web Services allow applications to be broken down into smaller logical components, which make the integration of applications easier as it is done on a granular basis. This makes Web Services solutions for EAI much more efficient than traditional EAI solutions.

- **Dynamic**: Web Services provide a dynamic approach to integration by offering dynamic interfaces, whereas traditional EAI solutions are pretty much static in nature.

In conclusion, by using Web Services the requirement ***"i."*** in section 3.2 is satisfied. The following section discusses in detail the architecture of Java Web Services and how they can be applied in order to develop the system proposed in this thesis.

3.3.3 Java Web Services

As already discussed in the previous section, Web Services are essential for the design and development of a dynamic platform. For this thesis, it is proposed to use Java Web Services, which have all the advantages of traditional Web Services plus they are platform independent. The following section discusses how Java Web Services fit into the architecture proposed in this thesis.

The *Java Web Services Developer Pack* (Java WSDP) is an integrated toolset [70, 77], that in conjunction with the Java platform, allows Java developers to build, test and deploy XML applications, Web Services, and Web applications. The Java WSDP provides Java standard implementations of existing key Web Services standards including WSDL, SOAP, ebXML, and UDDI. These Java standards allow developers to send and receive SOAP messages, browse and retrieve information in UDDI and ebXML registries, and quickly build and deploy Java Web Service.

Java Web Services are platform-independent, like Java, language-agnostic, a clear advantage over *Java Remote Method Invocation* (RMI), can easily be tunnelled through firewalls, which are one of the drawbacks in modern enterprise networks, object-oriented and tend to be loosely coupled allowing more flexible application development [62].

Among other features, the *Java Web Services Developer Pack* Version 1.0 includes the following:

- Java XML Pack which includes the following:
    - Java API for XML Messaging (JAXM)
    - Java API for XML Processing (JAXP) with XML Schema support.
    - Java API for XML Registries (JAXR)
    - Java API for XML-based Remote Procedure Call (JAX-RPC)
    - SOAP with Attachments API for Java (SAAJ)

- Java WSDP Registry Server
- Web Application Deployment Tool
- Apache Tomcat container

Figure 3.3 illustrates how a Web Service is registered, found and called in a scenario based on Java technology [63].

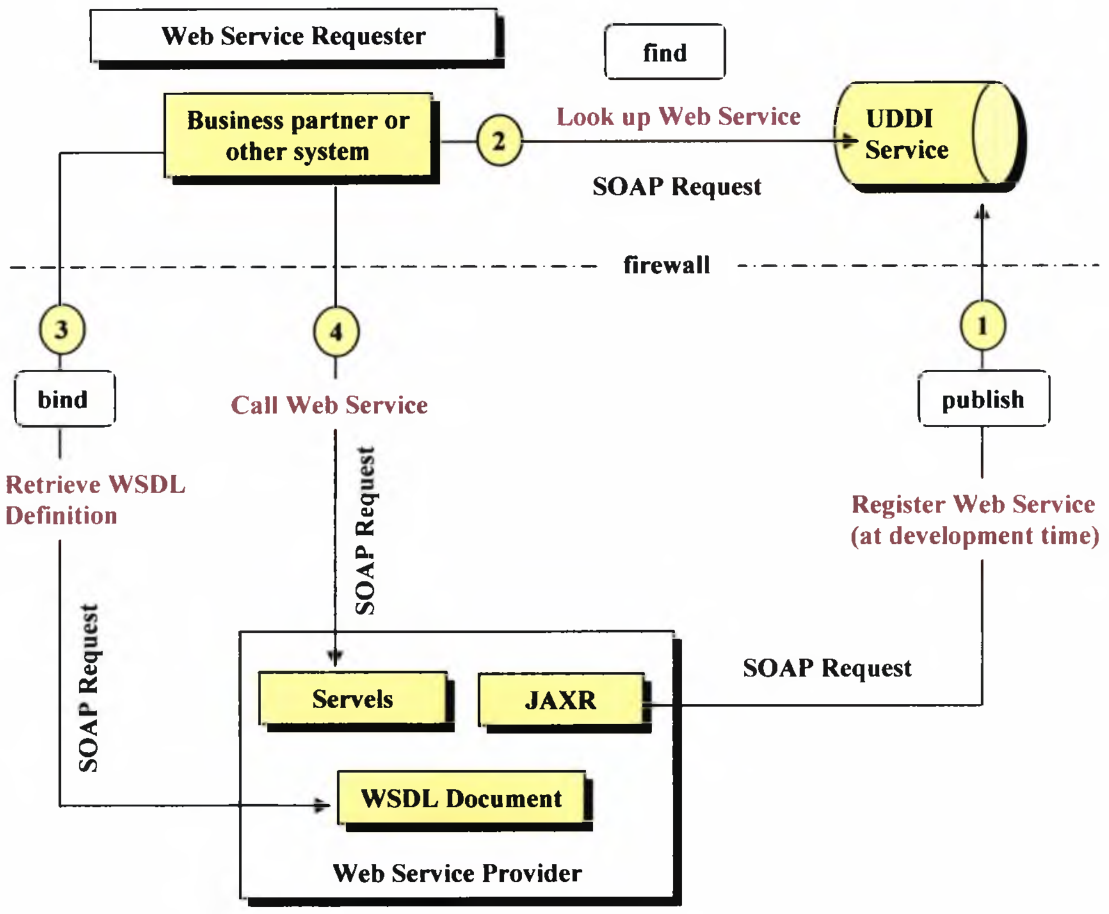


Figure 3.3 *Java Web Service Architecture*

In figure 3.3, the Web Service is registered in a UDDI repository using the Java API for XML Registries (JAXR, step 1), where a business partner or other system can find the service (step 2). The registry information from UDDI is used to locate a WSDL document that details the call semantics for the Web Service (step 3). With the WSDL document in hand, the Java programmer can then feed it to a tool that can generate a Java object proxy to the Web Service, or simply use it as a reference document along with a lower-level SOAP API (step 4).

3.3.4 Web Services Description Language

Web Services Description Language (WSDL) is an XML-based language used to define Web Services and describe how to access them [21, 117, 139]. Being XML-based, WSDL is both machine and human readable, which is an important advantage. Some modern development tools can generate a WSDL document describing any Web Service as well as consume a WSDL document and generate the necessary code to invoke the Web Service.

WSDL is the keystone of the *Universal Description, Discovery, and Integration* (UDDI) initiative spearheaded by Microsoft, IBM, and Ariba. UDDI is an XML-based registry for businesses worldwide, which enables businesses to list themselves and their services on the Internet. WSDL is the language used to do this.

WSDL is derived from Microsoft's *Simple Object Access Protocol* (SOAP) and IBM's *Network Accessible Service Specification Language* (NASSL). WSDL replaces both NASSL and SOAP as the means of expressing business services in the UDDI registry.

Figure 3.4 shows an example of a Web Service and a client invoking it in two different ways: Either by using SOAP or HTTP GET. Each invocation consists of a request and a response message.

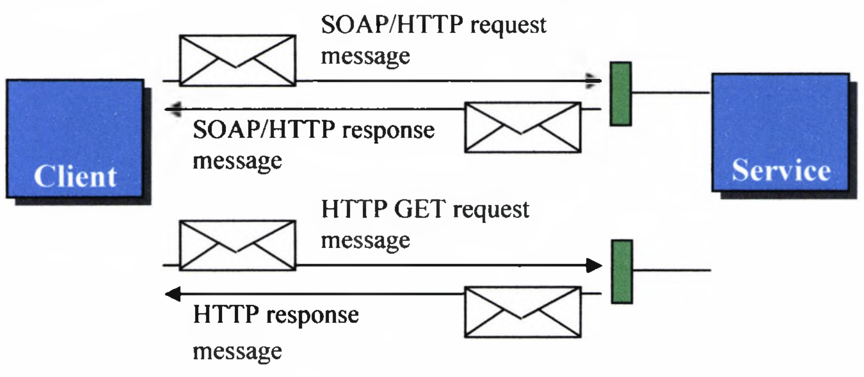


Figure 3.4 *A client invoking a Web Service*

Figure 3.5 shows the same example with WSDL terminology pointing out the various elements that WSDL describes.

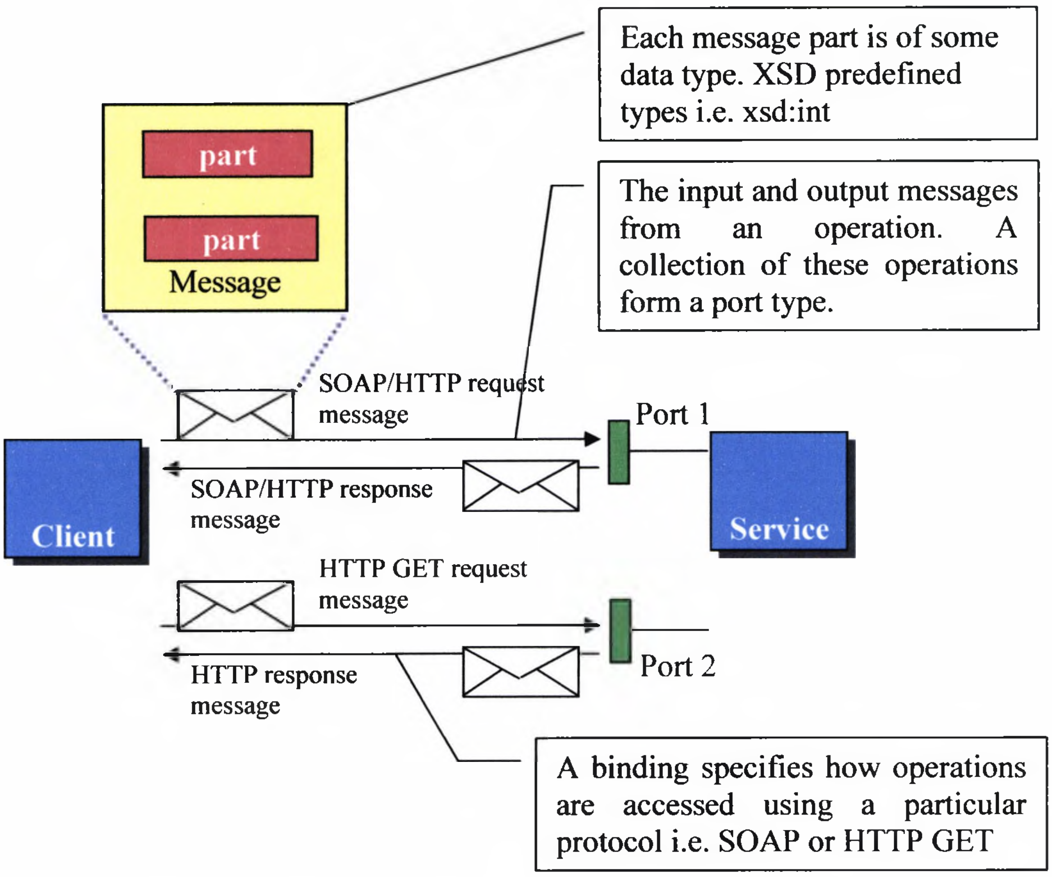
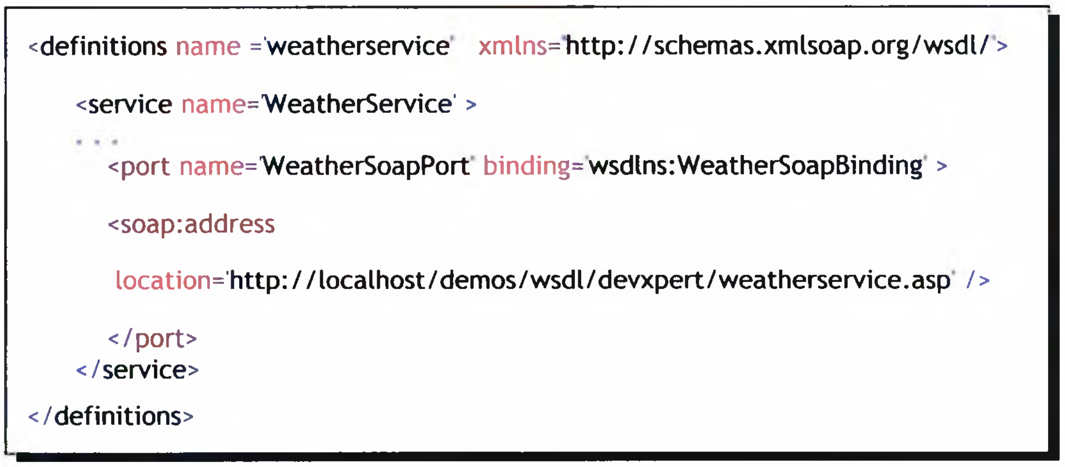

Each message part is of some data type. XSD predefined types i.e. xsd:int
part
part
Message
The input and output messages from an operation. A collection of these operations form a port type.
SOAP/HTTP request message
Port 1
SOAP/HTTP response message
Client
Service
HTTP GET request message
HTTP response message
Port 2
A binding specifies how operations are accessed using a particular protocol i.e. SOAP or HTTP GET

Figure 3.5 *WSDL terminology describing Web Services*

In order to better understand the WSDL, an example is illustrated in figure 3.6 which describes a service called "*weatherservice*".



```
<definitions name ='weatherservice'  xmlns='http://schemas.xmlsoap.org/wsdl/'>

    <service name='WeatherService' >
    ...
       <port name='WeatherSoapPort' binding='wsdlns:WeatherSoapBinding' >

       <soap:address

        location='http://localhost/demos/wsdl/devxpert/weatherservice.asp' />

       </port>
    </service>

</definitions>
```

Figure 3.6 *An example of WSDL technology*

The *<definitions>* element is the root element of the WSDL document. In this section the WSDL namespace is declared as the default namespace for the document so all elements belong to this namespace unless they have another namespace prefix. All other namespace declarations are omitted from this example to keep it clear.

Each service is defined using a service element. Inside the service element, the different ports can be specified on which this service is accessible. A port specifies the service address, for example, *http://localhost/demos/wsdl/devxpert/weatherservice.asp*.

Each port has a unique name and a binding attribute. When using SOAP, the port element contains a *"<soap:address/>"* element with the actual service address. In this example the soap namespace prefix refers to the namespace "*http://schemas.xmlsoap.org/wsdl/soap/*".

This namespace is used for SOAP-specific elements within WSDL. Such elements are also known as WSDL SOAP extension elements.

A Web Service does not have to be exposed using SOAP. For example, if the Web Service is exposed via HTTP GET, the port element would contain an *"<http:address/>"* element similar to the following script:

<http:address location="http://localhost/demos/wsdl/devxpert/weatherGET.asp"/>

A Web Service can be accessible on many ports. For example, it makes the Web Service available via SOAP and HTTP GET and possibly even via SMTP. For this Web Service, all three ports are available, each one with a different name.

3.3.5 Universal Description, Discovery and Integration

Universal Description, Discovery and Integration (UDDI) is a platform-independent framework for describing services, discovering businesses, and integrating business services by using the Internet. More specifically, UDDI is a directory for storing information about Web Services, which they communicate via the SOAP standard. This directory consists of Web Service interfaces described by the WSDL language.

UDDI is a cross-industry effort driven by all major platform and software providers like Dell, Fujitsu, HP, Hitachi, IBM, Intel, Microsoft, Oracle, SAP, and Sun, as well as a large community of marketplace operators, and e-business leaders [134,135].

A UDDI registry provides simple information about:

- **Who it is**. A registration includes the name of the Web Service provider, and can include additional identifiers such as a *North American Industry Classification System* (NAICS) code or a *Dun & Bradstreet* D-U-N-S number. D&B D-U-N-S Number is a unique nine-digit identification sequence, which provides unique identifiers of single business entities.
- **What it is**. A registration includes the name of a Web Service and, typically, a brief description.
- **Where it is**. A registration contains "binding templates" that point to an address where the service can be accessed.
- **How to request it**. A registration contains the ways that describe the interface for the Web Service.

Each registry that conforms to UDDI is operated at its own site and hence the UDDI specifications refer to this as the "operator" site. The operator site contains the master copy of its registry. However, the collection of UDDI registries is replicated. If a business searches for a Web Service in a registry at one operator site, the search is done across the information in the master copy, including the information replicated from the other UDDI registries.

The UDDI specifications define how to publish and discover information about Web Services in a UDDI-conforming registry. More specifically, the specifications define a UDDI schema and a UDDI API. The schema identifies the types of XML data structures that comprise an entry in the registry for a Web Service. The API describes the SOAP messages that are used to publish an entry in a registry, or discover an entry in a registry. The UDDI schema and the UDDI API are described in detail in the following sections.

3.3.5.1 UDDI Schema

The UDDI schema identifies the types of XML data structures that comprise an entry in the registry for a Web Service. The following figure illustrates the schema. It shows the five types of XML data structures that comprise a registration.

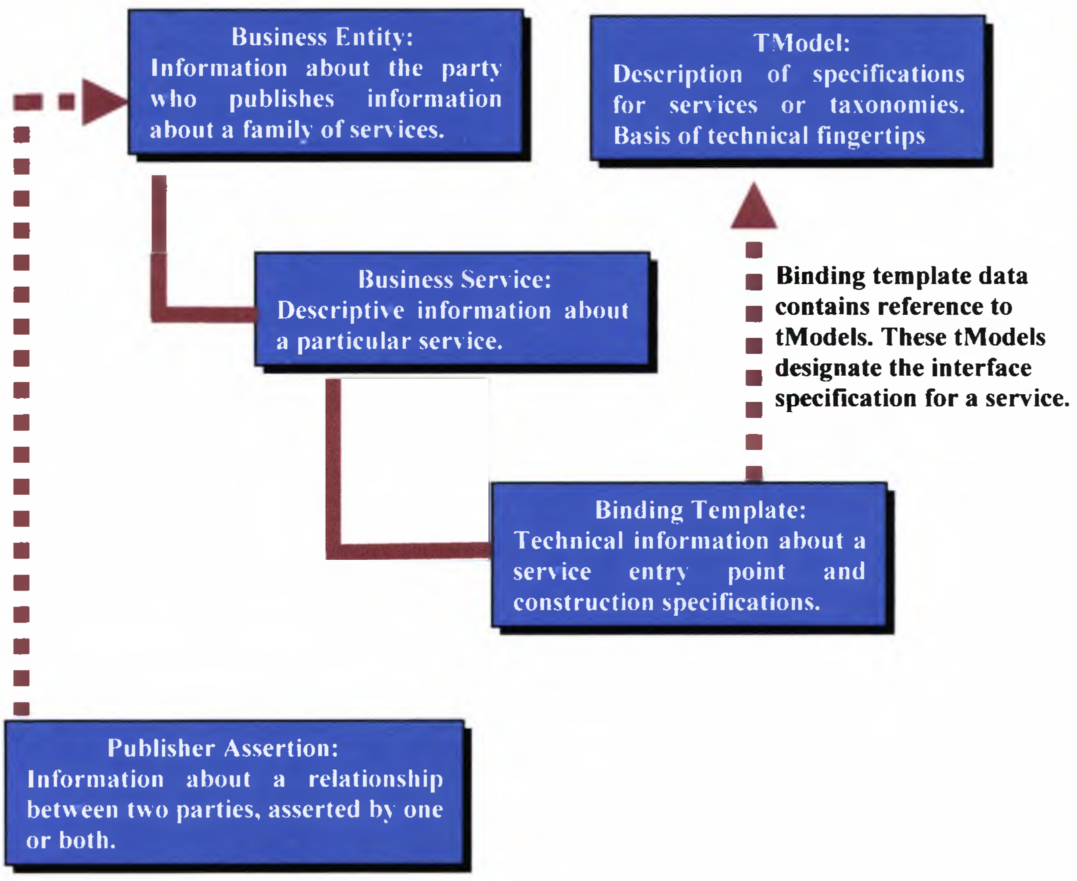

Business Entity:
Information about the party who publishes information about a family of services.
TModel:
Description of specifications for services or taxonomies. Basis of technical fingertips
Business Service:
Descriptive information about a particular service.
Binding template data contains reference to tModels. These tModels designate the interface specification for a service.
Binding Template:
Technical information about a service entry point and construction specifications.
Publisher Assertion:
Information about a relationship between two parties, asserted by one or both.

Figure 3.7 *The five types of XML data structures that comprise the UDDI schema*

The five types of XML data structures that comprise the UDDI schema are described below:

Business Entity: This structure represents all known information about a business or entity and the services that it offers. From an XML standpoint, a business Entity is the top-level data structure that holds descriptive information about a business or entity.

A business Entity element contains attributes that:

- Uniquely identify the business Entity with a *businessKey*. The key is a Universal Unique Identifier (UUID) that is generated by the registry. The mechanism that produces UUIDs guarantees uniqueness through a combination of hardware addresses, timestamps and random numbers.
- Name the individual who published the business Entity data (*authorisedName*).
- Specify the UDDI registry site that manages the master copy of the businessEntity data (*operator*).

Other elements within the structure identify the name recorded for the business or entity (*name*), and optionally specify additional information about the business or entity.

Figure 3.8 shows part of a complete Business Entity structure for a company named Inlecom Systems Ltd. The company provides various Web Services, including an online product ordering service.

```
<businessEntity businessKey="35AF7F00-1419-11D6-
A0DC-000C0E00ACDD"
authorizedName="0100002CAL"
  operator="www.inlecom.com/services/uddi">
<name>Inlecom Systems Ltd</name>
<description xml:lang="en">
   The source for all products
</description>
<contacts>
<contact>
<personName>Christofi Stelios</personName>
<phone>
  (0161)1111111
</phone>
</contact>
</contacts>
```

Figure 3.8 *Part Business Entity Structure for UDDI Schema*

Business Service: This structure identifies a service provided by the business or entity that is represented by the parent Business Entity. A business or entity can provide multiple services, so there can be multiple business Service elements within a business Entity. The set (or "family") of business Service elements are specified in a business Services structure.

Figure 3.9 shows part of a complete business Service structure for an online product ordering service offered by Inlecom Systems Ltd. Notice especially the "*categoryBag*" element, which is used to specify a classification scheme (formally this is known as a "taxonomy"). In this example, the taxonomy is NAICS.

```
<businessServices>
<businessService serviceKey="
  2AB346C0-2282-43B0-756B-0003CC35CC1D">
<name>Online Product Ordering</name>
<description xml:lang="en">
  Ordering products online
</description>
<categoryBag>
  ntis-gov:naic
<catgeoryBag>
```

Figure 3.9 *Part Business Service Structure for UDDI Schema*

Binding Template and **tModel**: A binding Template along with a tModel provides two important pieces of information about a Web Service: its technical specification and its *accessPoint*. A technical specification (also called a "technical fingerprint") typically provides details about things such as protocols and interchange formats used in communicating with the service.

The *accessPoint* is an address, such as a Uniform Resource Locator (URL) or email address, at which the service can be called. Contained within a binding Template structure is a *tModelInstanceInfo* element that references a tModel by its key. The referenced tModel provides the technical specification.

There can be multiple binding Template elements for the family of services identified in a business Services structure. The multiple binding Template elements are specified in a binding Templates structure.

Figure 3.10 illustrates part of a complete binding Template structure for the online product ordering service offered by Inlecom Systems Ltd. Notice the reference to the tModelKey in the "*tModelInstanceInfo*" element.

```
<bindingTemplates>
<bindingTemplate bindingKey="
  4BC7C340-2398-12E6-887C-0005AC33CC2D"
<description>
  JAXRPC (SOAP/HTTP ) based binding
</description>
<accessPoint>
  http://www.inlecom.com:8080/products/
</accessPoint>
<tModelInstanceDetails>
<tModelInstanceInfo tModelKey="UUID:
  36E13526-4553-3265-B5F7-C4B522E75A05" />
</tModelInstanceDetails>
```

Figure 3.10 *Part Binding Template Structure for UDDI Schema*

Publisher Assertion: This structure is used as a way of asserting a relationship between one business Entity and another. For example, the structure can be used to show that two businesses are subsidiaries of the same company. To assert the relationship, each of the two businesses Entity structures specify its own publish Assertion structure. However the information in each structure needs to be exactly the same.

3.3.5.2 UDDI API

The UDDI API describes the Simple Object Access Protocol (SOAP) messages that are used to publish an entry in a registry (the "Publish" API), and those that are used to discover an entry in a registry (the "Inquiry" API).

- **Publish API**. There are a variety of messages defined in the Publish API. What they have in common is that they perform some action related to one of the types of structures that comprise the UDDI schema. For example, the Publish API message "*save_business*" is used to save or update one or more complete business Entity structures in a registry.

Figure 3.11 shows an example of a message that saves the business Service structure (named Online Product Ordering) that was presented in section 3.3.3.1. The "*authinfo*" element is used to specify an authentication token. This is a value that is a standard part of UDDI's authentication mechanism and needs to be specified in all Publisher API calls. The authentication token is obtained by issuing a call to "*get_authToken*".

```
<save_service generic="2.0" xmlns=um:uddi-
org:api-v2">
<authinfo>authentication token goes here
...</authinfo>
<businessServices>
<businessService serviceKey="
  2AB346C0-2282-43B0-756B-0003CC35CC1D">
<name>Online Product Ordering</name>
<description xml:lang="en">
  Use this service to purchase products
over the Web
</description>
```

Figure 3.11 *Part Binding Template Structure for UDDI Schema*

- **Inquiry API**. The Inquiry API contains two types of messages: "**find**" messages that search UDDI registries for entries that meet specified search criteria, and "**get**" messages that retrieve detailed information about a specified registration. For example, the Inquiry API message "*find_business*" is used to search for all registered business entities that meet search criteria specified in the call. Figure 3.12 shows the "*find_business*" message that searches for all business entities whose registered name begins with the characters "Products".

```
<find_business generic="2.0" xmlns=um:uddi-
org:api-v2">
<name>Products%</name>
</find_business>
```

Figure 3.12 *Part structure of an Inquiry API for UDDI API*

The call returns a "*businessList*" structure that contains information about each matching business, and summaries of the "*businessService*" elements exposed by those businesses. The "get" messages include those that return detailed information about one or more business entities i.e. "*get_businessDetail*" or "*get_businessDetailExt*".

3.3.6 Simple Object Access Protocol

SOAP stands for Simple Object Access Protocol and provides the standard *Remote Procedure Call* (RPC) mechanism used for invoking Web Services. It implies that the underlying Web Service representation is an object when in fact it does not have to be. The Web Service can be written as a series of functions in JAVA and still invoke it using SOAP. The SOAP specification provides standards for the format of a SOAP message and how SOAP should be used over HTTP. SOAP also builds on XML and XML Schema Definition (XSD) Language to provide standard rules for encoding data as XML [11,20].

Like UDDI, SOAP is XML based. Moreover, SOAP is a generally accepted standard for Web Services and the reason for SOAP's widespread adoption is its simplicity. SOAP is "lightweight," that is, it involves a relatively small amount of code, and it is fairly easy to understand. The basic item of transmission in SOAP is the SOAP message, which consists of a mandatory SOAP envelope, an optional SOAP header, and a mandatory SOAP body.

Figure 3.13 illustrates a SOAP message, designed to retrieve the current price of a stock. Specifically, the SOAP message makes a request to "*GetLastTradePrice*", passing it a symbol called DEF. "*GetLastTradePrice*" is an operation performed by a Web Service, and DEF is the symbol for a specific stock. Clearly, for the request to be satisfied, "*GetLastTradePrice*" needs to be described, and its description needs to specify that the operation takes a stock symbol as input and returns a stock price as output. This is not done in SOAP. Instead it's done through Web Services Description Language (WSDL), an XML-based language for describing a Web Service. A WSDL description for a Web Service is contained in a WSDL document for the service.

For the purposes of this example, it is assumed that a WSDL document exists for a Web Service and in that document, the "*GetLastTradePrice*" operation is appropriately defined.

```
<SOAP-ENV: Envelope
  xmlns:SOAP-ENV=
    "http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:
    encodingStyle=
      "http://schemas.xmlsoap.org/soap/encoding/">
        <SOAP-ENV:Header>
           <t:Transaction xmlns:t="some-URI">
              SOAP-ENV:mustUnderstand="1">5
           </t:Transaction>
        </SOAP-ENV:Header>
        <SOAP-ENV:Body>
            <m:GetLastTradePrice xmlns:m="some-URI">
              <symbol>DEF</Symbol>
           </m: GetLastTradePrice>
        </SOAP-ENV:Body>
</SOAP-Envelope>
```

Figure 3.13 *An example of a SOAP Message*

The SOAP Message consists of the following items:

- **Envelope**. The Envelope element is the top element of the envelope. In figure 3.13, the Envelope element specifies two parameters: an XML namespace and an encoding style. An XML namespace is a collection of names that can be used in XML element types and attribute names, in other words, it is an XML schema. The example points to the *Uniform Resource Identifiers* (URI) "*http://schemas.xmlsoap.org/soap/envelope*". This URI defines the XML schema for SOAP messages.
  The "*encodingStyle*" attribute identifies the encoding style. An encoding style identifies the data types recognised by SOAP messages and specifies rules for how these data types are serialised, that is transformed into a stream of bytes, for transport across the Web. The example points to the URI *http://schemas.xmlsoap.org/soap/encoding/*. This URI specifies the encoding style for "Section 5" encodings, the ones described in Section 5 of the SOAP specification.

- **Header**. As mentioned earlier, the header is optional. However, if it is included in a SOAP message it must be the first child of the Envelope element. The Header element, through attributes, extends the SOAP message in a modular way. A SOAP message travels from an originator (a client application) to a final destination, potentially passing through a set of intermediate nodes along the message path. Each node is an application that can receive and forward SOAP messages. The SOAP header can be used to indicate some additional processing at a node independent of the processing done at the final destination. In the example in figure 3.11, the Header element indicates that this is a transaction (a URI specifies the namespace for the transaction). The header could just as easily have specified attributes for another type of process, such as authorisation checking. The attribute value "*mustUnderstand=1*" means that the initial node in the SOAP message path must process the header. The value 5 is passed to the initial node as input.

- **Body**. The SOAP body contains the main part of the SOAP message. In particular, the Body element contains information for the final recipient of the SOAP message. In the above example, the Body element contains two items of information: *GetLastTradePrice* (with its namespace) and the symbol DEF. This information is passed to the final destination. The application at that destination needs to understand the request and take the appropriate action. As mentioned earlier, for the purposes of this example, it is assumed that a WSDL document exists for a Web Service, and in that document, the "*GetLastTradePrice*" operation is appropriately defined. Based on the information in the WSDL document, it is assumed that the application calls the "*GetLastTradePrice*" operation, passing it the DEF ticker symbol as input. The WSDL document also indicates that the output returned by the "*GetLastTradePrice*" operation is a price. The returned price information is also passed in a SOAP message. The body of the response is illustrated in figure 3.14.

```
<SOAP-ENV:Body>
   <m:GetLastTradePriceOutput xmlns:m="some-URI">
           <price>42.50</price>
   </m: GetLastTradePriceOutput>
</SOAP-ENV:Body>
```

Figure 3.14 *An example of the body of a SOAP Response*

3.3.7 The XML language

"*Extensible Markup Language (XML) is a simple, very flexible text format derived from SGML. Originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere*" [29].

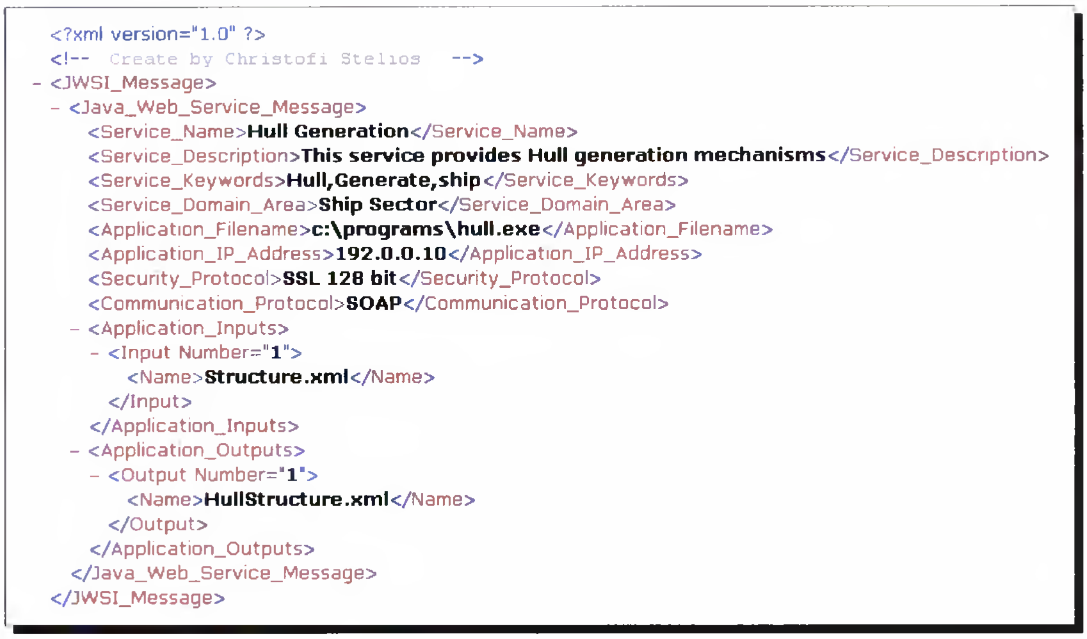
XML, describes a class of data objects called "XML documents" and partially describes the behavior of computer programs which process them. XML is an application profile or restricted form of SGML, the Standard Generalized Markup Language. By construction, XML documents are conforming SGML documents.

XML documents are made up of storage units called "entities", which contain either parsed or unparsed data. Parsed data is made up of "characters", some of which form "character data", and some of which form "markup". Markup encodes a description of the document's storage layout and logical structure. XML provides a mechanism to impose constraints on the storage layout and logical structure [29, 31, 92, 138].

The main differences between XML and *Hyper Text Markup Language* (HTML) are:

- XML is **not a replacement** for HTML
- XML and HTML were designed with **different goals:**
- XML was designed to **describe data** and to focus on **what data is**
- HTML was designed to **display data** and to focus on **how data looks**
- HTML is about **displaying** information, XML is about **describing** information
- XML allows new elements to be defined by the programmer.
- XML separates presentation from data
- XML is case-sensitive.
- HTML is for web browsers, XML is for any sort of data transfer.

An example of an XML document is illustrated in figure 3.15.



```
  <?xml version="1.0" ?>
  <!-- Create by Christofi Stelios  -->
- <JWSI_Message>
  - <Java_Web_Service_Message>
      <Service_Name>Hull Generation</Service_Name>
      <Service_Description>This service provides Hull generation mechanisms</Service_Description>
      <Service_Keywords>Hull,Generate,ship</Service_Keywords>
      <Service_Domain_Area>Ship Sector</Service_Domain_Area>
      <Application_Filename>c:\programs\hull.exe</Application_Filename>
      <Application_IP_Address>192.0.0.10</Application_IP_Address>
      <Security_Protocol>SSL 128 bit</Security_Protocol>
      <Communication_Protocol>SOAP</Communication_Protocol>
    - <Application_Inputs>
      - <Input Number="1">
          <Name>Structure.xml</Name>
        </Input>
      </Application_Inputs>
    - <Application_Outputs>
      - <Output Number="1">
          <Name>HullStructure.xml</Name>
        </Output>
      </Application_Outputs>
    </Java_Web_Service_Message>
  </JWSI_Message>
```

Figure 3.15 *An example of an XML Document*

To better understand the structure of the above XML document, it is converted into its equivalent logical structure using a tree-like hierarchical structure. This is illustrated in figure 3.16.

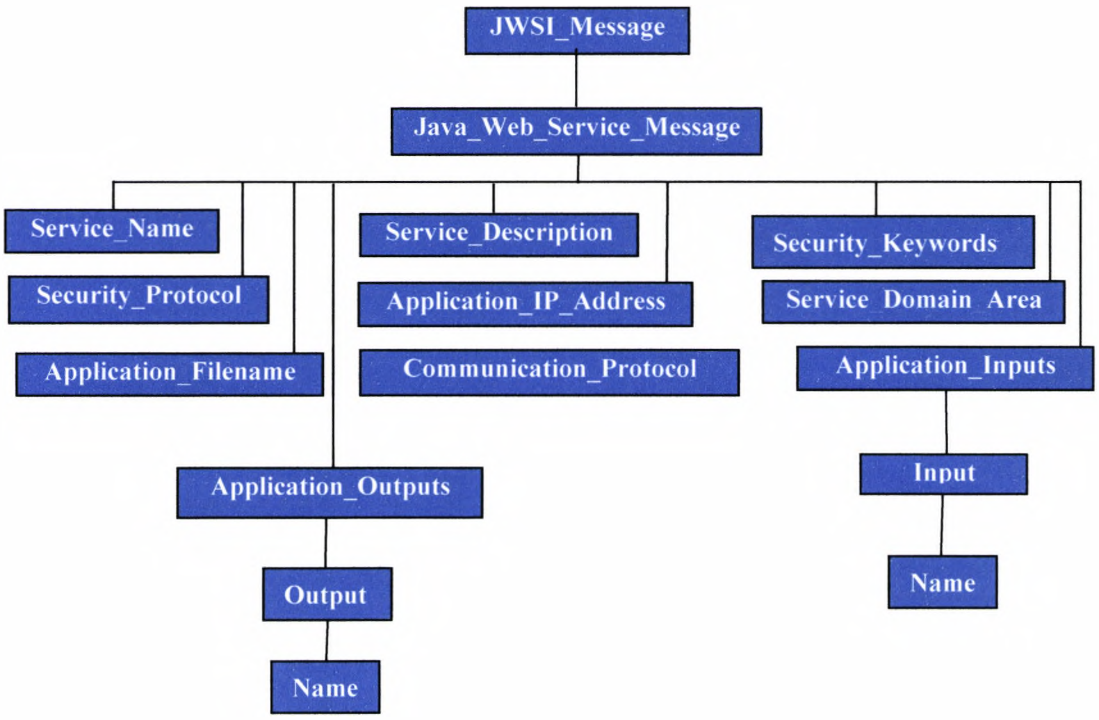


Figure 3.16 *An example of XML Document Logical Structure*

As it can be seen from figure 3.16, an XML document has a tree-like structure, with the *root* element (***<JWSI_Message>***) at the top of the tree. All the elements that are inside the root element are also contained within each other. The document must contain one and only one root element. An element is the parent of the elements it contains. The elements that are inside an element are called *children*. Similarly, the elements that have the same parent element are called *siblings*.

In this example, ***<JWSI_Message>*** is the parent of all other elements, ***<Service_Name>*** is a child of ***<Java_Web_Service_Message>***, and ***<Service_Name>*** and ***<Service_Description>*** are siblings. Going down the element tree, each child element must be fully contained with its parent element. Sibling elements may not overlap.

The XML language has been chosen as the infrastructure for data manipulation in contradiction to the *Comma Separated Values* (CSV) files also known as "***text***" files, for the following reasons [37, 80, 101, 103]:

- **Standardisation**

  Standardisation in information representation and transfer is crucial to both B2B and *Business-To-Client* (B2C) E-Commerce. XML is platform, application independent, and vendor-neutral mechanism. XML relies on other technologies, in particular, SGML for syntax, URIs for name identifiers, *Extended Backus Naur Form* (EBNF) for grammar and Unicode for character encoding, which are all standards.

- **Longevity**

  Electronic document formats can and do become "legacy formats" just like punched-cards or micro-fiche. XML on the other hand, will never become a legacy data format as everything about XML is open. In 20 years time XML 1.0 will certainly be "old" compared to, XML 6.0 but it will always be possible to programmatically access the structure and content of XML 1.0 documents. XML will never be legacy data. XML data will never need to be re-keyed. Even if a system becomes obsolete, the XML data will live on and will remain accessible in the long term.

- **Manageability**

XML is very programmable. This is one of its major attractions when processing high volume document collections. Other document formats-notably What You See Is What You Get (WYSIWYG) formats-are notoriously difficult to process in an automated fashion. The benefits of the high level of automation that can be achieved with XML really become apparent as the volume of information increases. For this thesis, it could have manually performed the conversion from one XML document to another. However, the amount of time involved would have been excessive. Moreover, mistakes due to human editing would have been an inevitable consequence of manual intervention. The amount of human effort involved in a manual XML transformation system is enormous.

- **Neutral**

XML is the basic format for representing data on the Web Services platform. In addition to being simple to create and parse, XML was chosen because it is neither platform nor vendor specific. Being neutral is more important than being technically superior. Software vendors are much more likely to adopt a neutral technology rather than one that was invented by a competitor.

- **Business-to-Business Communication**

Conducting Business-to-Business requires communicating with other companies and often poses a challenge. XML simplifies B2B communication, particularly in vertical industries for the following reasons:

- The only thing that is to be mutually agreed upon is the XML vocabulary that will be used to represent data.
- Neither company has to know how the other's back-end systems are organised, which does not put any extra technical burden whilst

maintaining privacy requirements. All that is required is that each company develops the *mappings* to transform XML documents into the internal format used by the back-end systems.

- XML-based solution is scalable. If there is an addition of another partner, there is no need by the host company to interact with the systems of the new company. What is needed is that the new company follow the protocol established by the hosting company. (The XML vocabulary).

3.3.8 XML Schema Languages

XML Schemas express shared vocabularies and allow applications to carry out rules made by people. They provide a means for defining the structure, content and semantics of XML documents [14, 30]. There are currently a number of different schema language representations in the market. In this section, a description of the two most known schema languages will be presented.

3.3.8.1 Document Type Definition

The *Document Type Definition* (DTD) [73] specifies the structure of an XML document, thereby allowing XML parsers to understand and interpret the document's contents. The DTD contains the list of tags, which are allowed within the XML document along with their types and attributes.

More specifically, the DTD defines how elements relate to one another within the document's tree structure and specifies which attributes may be used with which elements. Therefore, the DTD constrains the element types that can be incorporated in the document and determines its conformance. An XML document, which conforms to its DTD, is said to be "***valid***".

Figure 3.17 illustrates the document type definition of the XML described in figure 3.15.

```
<?xml version="1.0"?>
<!DOCTYPE JWSI_Message [

<!ELEMENT JWSI_Message        (Java_Web_Service_Message+)>
<!ELEMENT Java_Web_Service_Message  (Service_Name,Security_Protocol,
Application_Filename, Service_Description, Application_IP_Address, Communication_Protocol,
Security_Keywords, Service_Domain_Area, Application_Inputs+, Application_Outputs+)>
<!ELEMENT Service_Name               (#PCDATA)>
<!ELEMENT Security_Protocol          (#PCDATA)>
<!ELEMENT Application_Filename       (#PCDATA)>
<!ELEMENT Service_Description (#PCDATA)>
<!ELEMENT Application_IP_Address     (#PCDATA)>
<!ELEMENT Communication_Protocol     (#PCDATA)>
<!ELEMENT Security_Keywords  (#PCDATA)>
<!ELEMENT Service_Domain_Area        (#PCDATA)>
<!ELEMENT Application_Inputs         (Input+)>
<!ELEMENT Input              (Name)>
<!ELEMENT Name                       (#PCDATA)>
<!ELEMENT Application_Outputs        (Output+)>
<!ELEMENT Output                      (Name)>
]>
```

Figure 3.17 *An example of a Document Type Definition*

An XML language is defined in a Document Type Definition (DTD). The DTD is either contained in a "*<!DOCTYPE>*" tag, contained in an external file and referenced from a "*<!DOCTYPE>*" tag, or both.

XML provides an application independent way of sharing data. With a DTD, independent groups of people can agree to use a common DTD for interchanging data. An application can use a standard DTD to verify that the data sent or received from the outside world is valid.

3.3.8.2 XML Schema Definition Language

An *XML Schema Definition* (XSD) consists of components such as type definitions and element declarations. These can be used to assess the validity of well-formed element and attribute information items, and furthermore may specify augmentations to those items and their descendants.

This augmentation makes explicit information which may have been implicit in the original document, such as normalised and/or default values for attributes and elements and the types of element and attribute information items [17].

Schema-validity assessment has two aspects:

- Determining local schema-validity, that is whether an element or attribute information item satisfies the constraints embodied in the relevant components of an XML Schema.
- Synthesising an overall validation outcome for the item, combining local schema-validity with the results of schema-validity assessments of its descendants, if any.

Figure 3.18 illustrates the XML Schema Definition (XSD) of the XML described in figure 3.15.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
        <xs:element name="Application_Filename" type="xs:string"/>
        <xs:element name="Application_IP_Address" type="xs:string"/>
        <xs:element name="Application_Inputs">
                <xs:complexType>    <xs:sequence>
                                <xs:element ref="Input" maxOccurs="unbounded"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
        <xs:element name="Application_Outputs">
                <xs:complexType>    <xs:sequence>
                                <xs:element ref="Output" maxOccurs="unbounded"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
        <xs:element name="Communication_Protocol" type="xs:string"/>
        <xs:element name="Input">
                <xs:complexType>    <xs:sequence>
                                <xs:element ref="Name"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
        <xs:element name="JWSI_Message">
                <xs:complexType>    <xs:sequence>
                        <xs:element ref="Java_Web_Service_Message" maxOccurs="unbounded"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
        <xs:element name="Java_Web_Service_Message">
                <xs:complexType>    <xs:sequence>
                                <xs:element ref="Service_Name"/>
                                <xs:element ref="Security_Protocol"/>
                                <xs:element ref="Application_Filename"/>
                                <xs:element ref="Service_Description"/>
                                <xs:element ref="Application_IP_Address"/>
                                <xs:element ref="Communication_Protocol"/>
                                <xs:element ref="Security_Keywords"/>
                                <xs:element ref="Service_Domain_Area"/>
                                <xs:element ref="Application_Inputs" maxOccurs="unbounded"/>
                                <xs:element ref="Application_Outputs" maxOccurs="unbounded"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
        <xs:element name="Name" type="xs:string"/>
        <xs:element name="Output">    <xs:complexType>    <xs:sequence>
                                <xs:element ref="Name"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
        <xs:element name="Security_Keywords" type="xs:string"/>
        <xs:element name="Security_Protocol" type="xs:string"/>
        <xs:element name="Service_Description" type="xs:string"/>
        <xs:element name="Service_Domain_Area" type="xs:string"/>
        <xs:element name="Service_Name" type="xs:string"/>
</xs:schema>
```

Figure 3.18 *An example of an XSD*

3.3.8.3 Selecting the XML schema language

XML Schema is a more advanced version of DTD. DTD has lots of disadvantages over schema, such as it does not support strong data typing, has syntax other than XML, and it is not expandable. Schema was introduced to overcome those drawbacks. The most common features of XML Schema are [17]:

- Syntax is very similar to XML. This means that the schema can be edited by using any XML editor.
- Data types are not limited to string, integer, long, float but also it supports custom data types.
- XSDs are more robust and flexible than DTDs. Schemas are XML documents, unlike DTDs, which contain non-XML syntax. Schemas also support namespaces, which are required to avoid naming conflicts, and offer more extensive data type and inheritance support.
- XML Schema provides Content-Based Validation (the order in which the child elements are nested) and also provides Data Type validations. Its functionality and validation checks are supported for simple and complex data types.
- XML Schema is easily extendible to incorporate more features in the future.

XML provides a simple way of representing data but it says nothing about the standard set of data types available and how to extend that set, for example, whether an integer is a 16, 32 or 64 bit. Such details are important to enable interoperability. The W3C XML Schema (XSD) is a standard that specifies some built-in types and language to define additional types. The Web Services platform uses XSD as its type system. When a Web Service is build, the data types that are used must be translated to XSD types to conform to the Web Services standards.

Taking into consideration the above limitations of DTDs, and the wide use of XSD, it has been decided to use the XML schema definition language as the infrastructure for describing the XML documents that will be used in this thesis.

3.3.9 Extensible Stylesheet Language

XSL stands for ***eXtensible Stylesheet Language***. XSL [1, 13] is a language for expressing stylesheets and consists of two parts [116]:

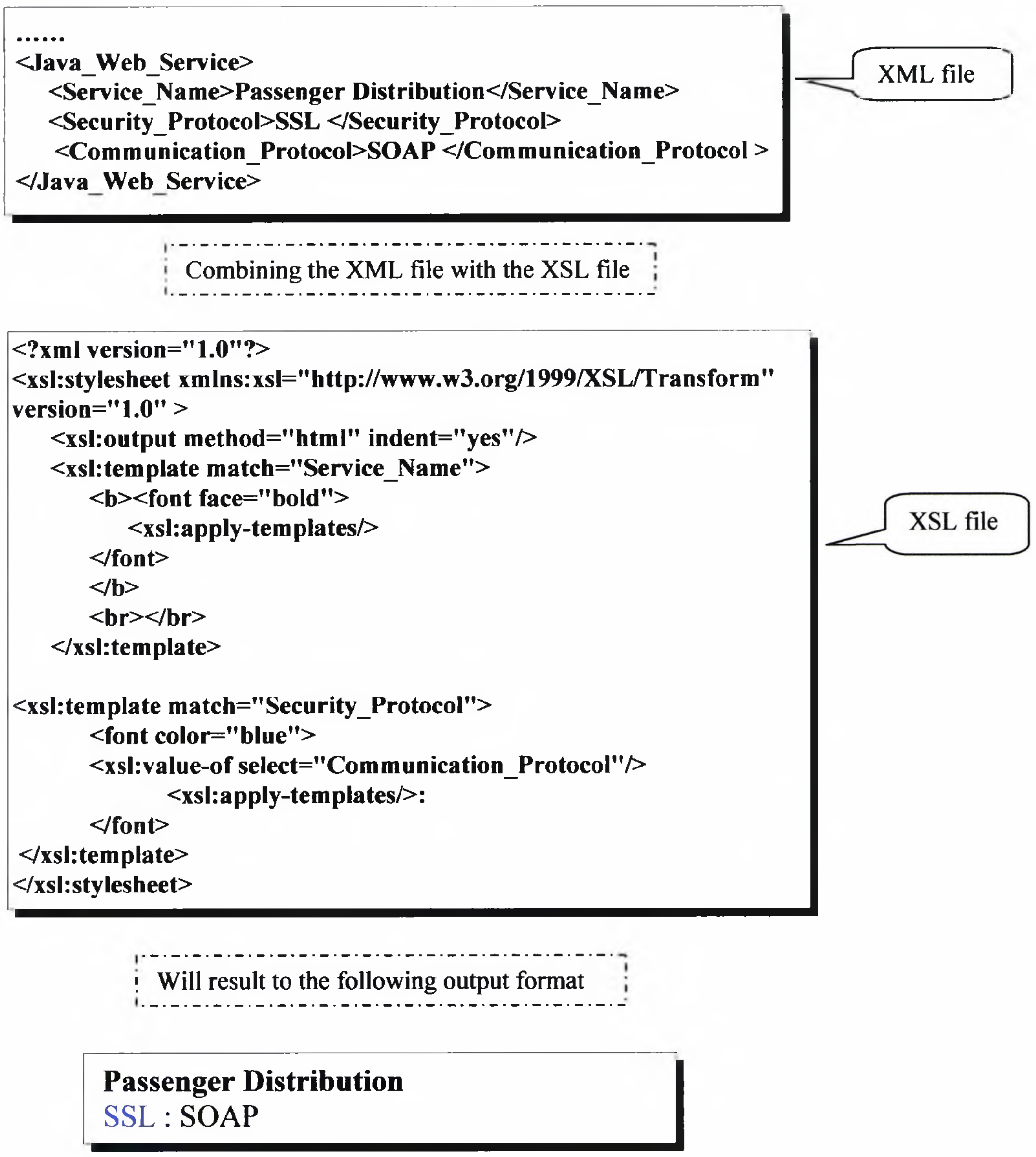
- ❑ XSL Transformations (XSLT), a language for transforming XML documents, and
- ❑ XSL Formatting Objects, an XML vocabulary for specifying formatting semantics.

An XSL stylesheet processor accepts a document or data in XML and an XSL stylesheet and produces the presentation of that XML source content as intended by the designer of that stylesheet.

There are two aspects of this presentation process: first, constructing a result tree from the XML source tree and second, interpreting the result tree to produce formatted results suitable for presentation on a display, on paper, or onto other media. The first aspect is called "*tree transformation*" and the second is called "*formatting*". The process of formatting is performed by the "*formatter*". This formatter may be a rendering engine inside a browser.

Tree transformation allows the structure of the result tree to be significantly different from the structure of the source tree. For example, a table-of-contents can be added as a filtered selection of an original source document, or source data can be rearranged into a sorted tabular presentation. In constructing the result tree, the tree transformation process also adds the information necessary to format that result tree.

Figure 3.19 shows a sample XML file and how it can be transformed and rendered.


XML file
Combining the XML file with the XSL file
XSL file
Will result to the following output format

```
......
<Java_Web_Service>
   <Service_Name>Passenger Distribution</Service_Name>
   <Security_Protocol>SSL </Security_Protocol>
    <Communication_Protocol>SOAP </Communication_Protocol >
</Java_Web_Service>
```

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0" >
    <xsl:output method="html" indent="yes"/>
    <xsl:template match="Service_Name">
        <b><font face="bold">
            <xsl:apply-templates/>
        </font>
        </b>
        <br></br>
    </xsl:template>

<xsl:template match="Security_Protocol">
        <font color="blue">
        <xsl:value-of select="Communication_Protocol"/>
                <xsl:apply-templates/>:
        </font>
 </xsl:template>
</xsl:stylesheet>
```

Passenger Distribution
SSL : SOAP

Figure 3.19 *Applying XSL on an XML file*

The stylesheet can be used to transform any instance of the XML it was designed for. The first rule says that a "*Service_Name*" element will be transformed into an html block with a bold font. "*<xsl:apply-templates/>*" is a recursive call to the template rules for the contents of the current element. The second template applies to all "*Service_Protocol*" elements and formats them as text colour blue and the third template is applied to all "*Communication_Protocol*" elements as shown in figure 3.19.

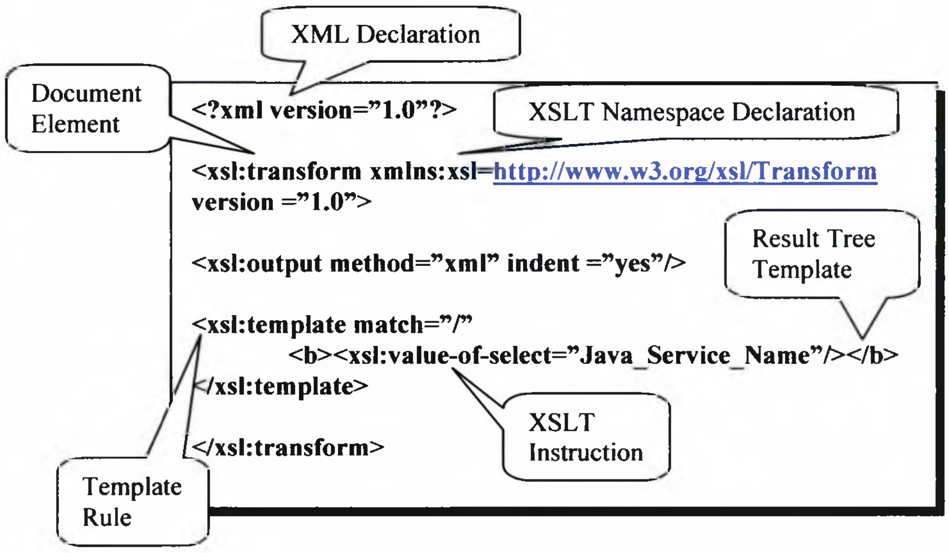
3.3.10 XSL Transformations

XSLT [26, 34, 80] stands for ***eXtensible Stylesheet Language Transformations***. XSLT is a language for transforming XML documents into other XML documents.

XSLT is designed for use as part of XSL, which is a stylesheet language for XML. In addition to XSLT, XSL includes an XML vocabulary for specifying formatting. XSL specifies the styling of an XML document by using XSLT to describe how the document is transformed into another XML document that uses the formatting vocabulary.

XSLT is also designed to be used independently of XSL. However, XSLT is not intended as a completely general-purpose XML transformation language. Rather it is designed primarily for the kinds of transformations that are needed when XSLT is used as part of XSL.

Figure 3.20 illustrates and gives a brief description of an example of an XSLT document.



```
<?xml version="1.0"?>
<xsl:transform xmlns:xsl=http://www.w3.org/xsl/Transform
version ="1.0">
<xsl:output method="xml" indent ="yes"/>
<xsl:template match="/"
              <b><xsl:value-of-select="Java_Service_Name"/></b>
</xsl:template>
</xsl:transform>
```

Figure 3.20 *An example of an XSLT*

After discussing the different technologies of the XML language, the next section is set to discuss the advantages and drawbacks of using the Java language to implement the proposed system.

3.3.11 Java Language

Based on the 4th requirement of this thesis [see section 3.2], it is necessary to design and develop small peers to participate into the integration process. These peers will be developed using the Java language for the reasons that will be discussed further in this section. The purpose of these Java peers is to establish a communication line between the two dynamic integrated applications, which will act as agents on the client's and server's machine, to send and receive requests from both locations. Furthermore, they will have the ability to dynamically publish Java Services on a UDDI and at the same time dynamically discover new Java Services for the purpose of integrating two or more applications together.

More specifically, the main requirements of these Java peers are:

- The portability to many operating systems.
- Usability, since most of the users may not be computer literate.
- Ease of navigation and control, for the purpose of minimising the time required to initiate communication between the client and the server.
- Easy setup and configuration.

After evaluating a number of programming languages like Visual Basic, Visual C++, Visual Fortran, .NET language, it has been decided to use the Java language, which satisfies all the above requirements. A more detailed discussion about the Java language follows.

"*Java is a simple, object-oriented, network-savvy, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, dynamic language.*"[126]

The above statement taken from the Sun Microsystems's Web site gives an overall picture of the capabilities of the Java language.

The Java programming language offers several distinct advantages [27, 40, 74, 80, 122]:

- **Simplicity**

  The Java programming language is easy to learn and use. The language also brings the productivity enhancements from object-oriented methodologies. There is a large and rapidly growing supply of developer talent in the industry, making it easier to staff new projects.

- **Security**

  For distributed networked applications, the Java programming language features robust security. The run-time environment creates a "*sandbox*" where applications can execute safely, with fine-grained access restrictions on system resources. The lack of pointers also eliminates direct memory access, resulting in fewer bugs.

- **Portability**

  By using an underlying Java run-time environment, applications can be easily developed on a desktop system using standard software development tools. By taking into account the underlying target hardware characteristics, developers can then move applications with minimal effort to the target device.

As a result, applications written in the Java programming language are easily migrated across product lines, greatly simplifying porting and development efforts. In this way, the Java programming language can provide a much higher degree of standardisation in the real-time world.

As it can be seen, the advantages of using the Java language are great, as already discussed in this section, and that is why it has been selected as the main programming language of developing the Java peers.

To satisfy the requirements set out in section 3.2, it is now appropriate to describe the overall peer-to-peer architecture and how the methodologies stated in that section fit into the overall architecture. The following section provides a detailed description of the *Java Web Service Integrator* (JWSI), as well as a detailed discussion of the overall technical peer-to-peer platform proposed in this thesis.

3.4 Java Web Service Integrator

This section describes in detail the functionality of the *Java Web Service Integrator* (JWSI). It continues with a detailed description of each JWSI component. The JWSI is a small java agent that is installed on the client machine together with the application that needs to be integrated. It acts as a peer for the client and provides the necessary functionality to the client to integrate the client application with other applications on the Internet. This system has been developed in order to prove the concepts already discussed in this thesis and to demonstrate the proposed P2P architecture in a fully simulated environment. Figure 3.21 illustrates the architecture of the Java Web Service Integrator.

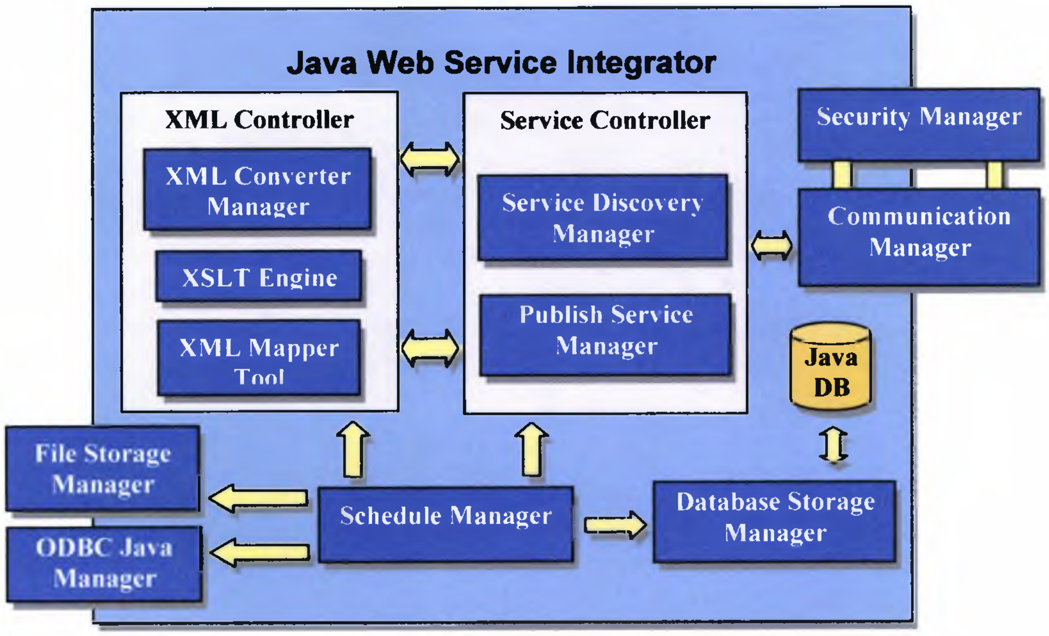


Figure 3.21 *The Java Web Service Integrator*

The Java Web Service Integrator comprises of the following components:

- **File Storage Manager**

  The purpose of the File Storage Manager is to enable the interaction between the JWSI and the File System of the Operating System. The main role is the manipulation of physical files (i.e. Read, Write, Check file timestamp) on the Physical drive of the machine that is running the JWSI. These functions are executed in predefined interval of times as scheduled by the Schedule Manager. The Communication Manager will transfer these physical files to other peer agents after a Java Web Service request.

- **ODBC Java Manager**

  The purpose of the Open Data Base Connectivity (ODBC) Java Manager is to enable the interaction between the JWSI and the client Database. This is achieved via the ODBC driver that is found in all operating systems and the Java Data Base Connectivity (JDBC) driver that exists as part of the JWSI library. The two drivers work together so as the JWSI can access information that is stored in any ODBC enabled database on the client site. An ODBC enabled database is a database that can communicate with an ODBC driver. The advantage of this component is to enable the JWSI to extract any information that is stored in a custom made database, unknown to the JWSI and publish this information on the UDDI. Other legacy systems running the JWSI can download this information via a Java Web Service and integrate it with their existing custom made databases to be used by their legacy systems.

- **Database Storage Manager**

  The purpose of the Database Storage Manager is the storage and retrieval of information, including the transmitted XML files, in the local Java Database of the JWSI. The advantages of using a light Java Database Engine as part of the JWSI architecture are endless. Firstly, the JWSI does not have to retrieve any information which is part of the local application profile from a remote server, and hence reduce the time of accessing the information and minimise the remote communication costs. Secondly, the JWSI is independent of any other application or server and hence can be installed on any machine with minimal requirements. Last and not least, the JWSI keeps a history of all integrated applications in the Database so that next time an application needs to be integrated with another application, the mappings between these two does not have to be created again. By doing that, the JWSI keeps a history of all information that is stored in the Database and hence the JWSI uses this information intelligently to be used for future integrations.

- **Schedule Manager**

  The Schedule Manager has an important role on the JWSI overall architecture. The Schedule Manager is responsible to schedule almost all components that are found in the JWSI and subsequently, provide a better way to increase the efficiency of the JWSI. It schedules not only the File Storage Manager, the ODBC Java Manager and the Database Storage Managers but also it schedules the Service Controller to publish and subscribe the different services that the JWSI has to offer. Moreover, it schedules the XML controller to start at specific times in order to convert any file coming from the local legacy application to an XML file where it can be converted to a Java Web Service.

- **XML Controller**

  The XML Controller comprises of three main parts.

  - The XML Converter Manager
  - The XSLT Engine
  - The XML Mapper Tool

  Each part has specific responsibilities. The XML Converter Manager is responsible for converting Comma Separated Values (CSV) files to XML files and vice versa. Furthermore, it is responsible to convert from any XML file to another XML file with different XML schemas. This is accomplished using the XSLT Engine as a catalyst during the XML conversion process. The XML conversion starts and finishes according to the instructions sent by the Schedule Manager during the lifecycle of the integration process.

  The role of the XSLT Engine is twofold. Firstly, it is responsible to act as a catalyst to the XML conversion process by applying the XSLT templates during the conversion and secondly, it is responsible to create the dynamic XSLT templates during the creation of the XML mappings. This is done via the XML Mapper Tool. This Tool provides an interface where the source and destination schemas of two different CSV or XML files are loaded and hence provides to the client an easy to use tool for the purpose of creating dynamically XSLT templates for the two corresponding files. It is important to note that the XSLT of any pair files can be created only once and the corresponding XSLT mappings are being stored into the local Java Database for future access. In cases when the Java Web Service is clearly understood (i.e. each attribute of the published Java Web Service corresponds to the attributes needed to be consumed by the client) by a newly created application wanted to be integrated into the architecture, then the client does not have to create the XSLT mappings. These mappings will be created by the JWSI automatically and therefore, this helps the user to simplify the dynamic integration process with other applications.

- **Service Controller**

The Service Controller comprises of two main parts.

- The Publish Service Manager
- The Service Discovery Manager

The purpose of the Publish Service Manager is to publish on the UDDI any new Java Web Services that will be created by the JWSI. The JWSI provides to the client an easy-to-use wizard where the service can be created according to the inputs and outputs of the legacy application. Besides inputs and outputs, this service will contain information, such as the purpose of the service, any special integration requirements, security restrictions etc. The JWSI will be in a position to recognise these service "tags" and will take appropriate actions to initiate the integration process. A number of services can be published based on the same application but with different "tags".

The purpose of the Service Discovery Manager is to search periodically the UDDI registry for new Java Web Services. These newly found services are downloaded and stored in the local Java database to be accessed by the local legacy applications. Only the Java services that are compatible with the local legacy application are downloaded to the client so as to avoid the download of unnecessary potential services that are not applicable to the client. These newly found services are evaluated based on some criteria that the client has to supply prior to initiating the Service Discovery Manager. Moreover, the Discovery Manager is responsible to synchronise the existing services between the above registry and the services that are found in the local Java database in case the client has modified them. This is done to ensure consistency and to guarantee that no unexpected errors will occur during the integration process of the two applications. As already mentioned above, the Schedule Manager initiates the Service Controller.

- **Communication Manager**

  The purpose of the Communication Manager is to negotiate the available transport protocols that exist between the two peers and establish a secure communication between them. During the negotiation, a lot of parameters have to be taken into consideration before both peers agree to communicate. One of these parameters is the security, which is handled by the Security Manager. The Java Web Service specifies the level of security that needs to be used during the communication. If any of the security parameters does not exist in a service, then this means that the information needed to be interchanged between the peers is not confidential and consequently, no security measures need to be taken. After the handshake has been achieved, the Communication Manager sends the necessary information to the requested application. The *Internet Protocol* (IP) address of the '*server*' application that published the service can be found on the envelope of the Java Web Service. This process enables the dynamic discovery of Java Web Services on the UDDI and the dynamic integration of legacy applications without prior knowledge of the two legacy applications.

- **Security Manager**

  Due to the architecture of the Internet and Intranet, there will always be ways for unscrupulous people to intercept and replace data in transit. Without security precautions, users might encounter security problems when sending information over the Internet or an Intranet. The Security Manager is responsible to ensure the secure communication between the two peers. This is accomplished by using a number of security protocols that will guarantee the delivery and the integrity of the data sent between the two peers. Each Java Service has to list all security parameters that are applicable for the published service so that the requested peer knows what security parameters needs to be inserted to that service before the two peers come to a handshake. The following security parameters may be used to ensure a secure communication. For each service a unique identifier is generated bound between the published service and the legacy application that is involved. When the requested peer tries to check the credentials of that service then this unique identifier will be cross-checked against the unique identifier that is found on the JWSI of the published service. By this way, it is guaranteed that the requested peer using the service is communicating with the correct peer holding the same unique identifier as the published service.

Another major security problem is impersonation, where information passes to or from a person who poses as the intended recipient or as the sender of the message. The chances of impersonation can be reduced if people are forced to authenticate (or verify their identities) before communicating information. On the Internet and Intranet someone can use digital certificates to ensure that users or computers are who they say they are. In this way, a certificate acts as a digital ID. This is one of the security technologies that the Security Manager is using to solve the problem of impersonation.

On top of these security techniques, an encryption algorithm of 128bit key is being used in order to make sure that nobody can manipulate the communication lines and hence read the transmitted data.

By using the above security techniques, it is guaranteed that the communication path between the two peers is secured and ready for transmission. These security techniques can be used either individually or by a combination of them to increase the security according to the confidentially of the data needed to be integrated.

To sum up, the above section discussed in detail the functionally of each component found in a JWSI peer and hence satisfied the requirement "***vi***" mentioned in section 3.2. The following section is set to describe the overall technical P2P platform and how the JWSI peers fit into this architecture proposed in this thesis. This will be achieved by the use of a "*Use Case*" scenario.

3.5 OVERALL TECHNICAL P2P PLATFORM

Summarising all the requirements that have been identified and explained in section 3.2 and after a detailed survey of methodologies was conducted; the overall P2P platform architecture is presented in figure 3.22.

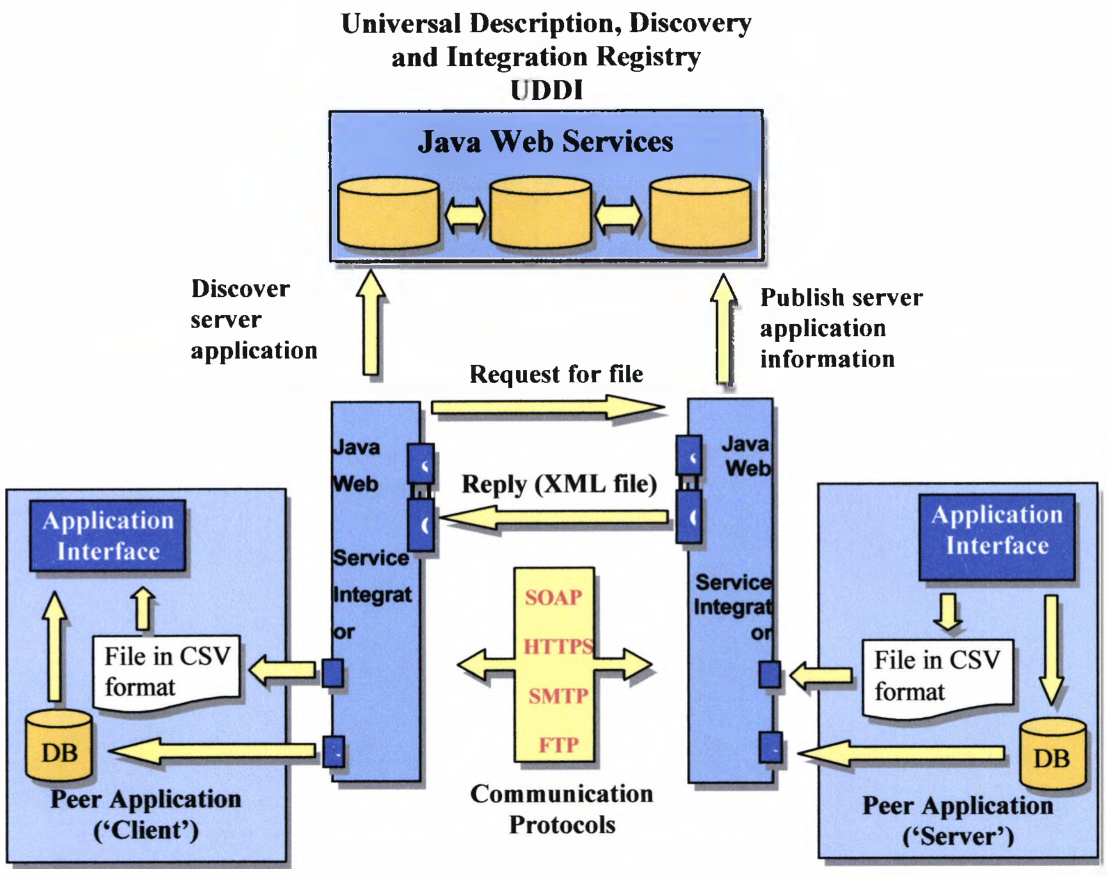

Universal Description, Discovery and Integration Registry UDDI
Java Web Services
Discover server application
Publish server application information
Request for file
Reply (XML file)
Java Web Service Integrator
Java Web Service Integrator
Application Interface
Application Interface
File in CSV format
File in CSV format
DB
DB
SOAP
HTTPS
SMTP
FTP
Communication Protocols
Peer Application ('Client')
Peer Application ('Server')

Figure 3.22 *An Overall Architecture of the Platform*

The above architecture comprises of three main units:

- The Client Application and Java Web Service Integrator
- The Server Application and Web Service Integrator
- The Universal Description, Discovery and Integration (UDDI) Registry

Although in this architecture the terms '*Client*' and '*Server*' are used, it must be noted that this is only done for convenience purposes, i.e. so that the two Applications participating in the scenario discussed below can be distinguished. In fact, all applications acting in a P2P fashion are both client and servers.

The UDDI is an Internet based registry containing searchable descriptions of published *Application Services Interfaces* (ASIs) and of other related information (such as the IP address of the server, supported communication protocols etc). In this Integration architecture, these ASIs referred to 'Java Web Services' because they are services, which are created by the Java peers. A Java Web Service is an XML Schema that describes a type of business document that can be provided by the application. Once a client identifies a suitable application, it uses information contained in the UDDI registry to contact the server application directly. The UDDI, therefore, acts only as an intermediary (broker) without actually intervening in the integration process. The UDDI specification is the building block that will enable businesses to quickly, easily and dynamically find and transact business with one another using their preferred applications [134, 135]. Today's business imperatives are clearer than ever. Business are trying to beat competitors by introducing new products to the market, deliver personalised services, increase customer loyalty, and evolve at electronic speeds. These imperatives demand a technology infrastructure that is more flexible, dynamic and business-intelligent than ever. Java Web services based on UDDI are an evolution in e-business applications that will help businesses reach these goals and take Business-to-Business (B2B) to the next level.

In order to understand the overall architecture presented in figure 3.22 a "Use Case" scenario will be discussed in the following section.

Before the integration process takes place, the user of the application server needs to create the Java Web Service. This is achieved by using the "Java Web Service Wizard". In this wizard, the user needs to follow some steps before the service is created and published on the UDDI registry. Some main parameters that have to be specified for this service are: Service name, service description, inputs values for the application, outputs of the application, the IP address of the client running the application, available communication protocols, confidentiality of information and the security protocols applicable. After these parameters have been specified, the Published Service Manager publishes the service on the UDDI registry. The integration process can now be initiated.

Before the user of the client application initiates the Service Discovery Manager, the JWSI needs to be configured. The configuration process is essential since the JWSI must know the inputs and outputs of the local application needed to be integrated. This is important in order for the JWSI to know which dynamic services are needed to be downloaded from the UDDI. This is accomplished by comparing the description of the service on the UDDI and the description of the local application. Not all Java Web Services found on the UDDI are downloaded since this is unnecessary and inefficient.

After the two above steps have been completed, the integration process may commence. During interval amounts of time the Discovery Service Manager searches the UDDI for any new services that closely match the specification of the pre-configured JWSI, to which the local application requested to be integrated. If a new service is dynamically discovered for the first time then the schema of this service is downloaded and listed under the newly found services window where the user has to create an XSLT between the Java Web Service and the local application. This is accomplished via the XML Mapper Tool, where the user maps the fields of the XML schema of the Java Web Service with the fields of the schema of the local application. The outcome of this process is an XSLT, which corresponds to the mappings of the two schemas. If a new service is dynamically discovered and the XSLT for that service already exists, then control is given to the Communication Manager to resolve the destination IP address of the server application that offers this service.

If for any reason the destination server is unavailable because of a temporary computer failure, the Communication Manager is re-scheduled to initiate a second handshake after a predefined interval of time. If after 10 re-tries the server is still unavailable, then the Communication Manager marks the recent service as “Not Available” and returns control back to the Discovery Service Manger to search for any new services. If the server is “Available” and hence the service is ready to be used, the two JWSI peers try to negotiate the security as well as the communication protocols that are available between each other, in order to establish a secure and reliable connection. The first step is to negotiate the available communication protocols and the second step is to decide on the security. For this scenario, the available communication protocols that are listed on the envelope of the service are: HTTP, HTTPS, SMTP, FTP, and SOAP. Since all communication protocols are available, the two peers need to decide which of them is more appropriate to be used in this situation. For example, the following decision rules are applied in order to select the appropriate protocol. One of the rules is the measurement of the communication bandwidth. This is accomplished by having the two peers send some "*test data*" of different sizes to each other. In this way, the bandwidth and speed of the communication line is measured and hence the selection of the available protocol is based on these results. Below, is a list of protocols and explanations on where these protocols may be used for each specific scenario:

- **Simple Mail Transfer Protocol**

  The *Simple Mail Transfer Protocol* (SMTP) is likely to be used, in the shipping context environment, when ships are in the middle of the ocean and are using the low bandwidth but expensive satellite link as means of communication. This asynchronous communication is also applicable when the transmission has to obey certain timing and delivery constrains. In a general context environment, SMTP is likely to be used when the server supporting the Java Web Service is not available. This is critical since the requested application can send the information via SMTP and then when the server is available can extract this information from the SMTP server and process the transaction at a later stage. This kind of integration is applicable when the integrated data is not so confidential and does not require an immediate response from the server.

- **File Transfer Protocol**

  The *File Transfer Protocol* (FTP) is used when the transfer of data is very large. This is essential when large volumes of data are interchanged between the two applications in a timely manner. In the future, this protocol can be extended to support *Virtual Private Networks* (VPNs), as well as where the transfer of data is even faster.

- **Hyper Text Transfer Protocol or Simple Object Access protocol**

  Hyper Text Transfer Protocol (HTTP) over *Transmission Control Protocol/Internet Protocol* (TCP/IP), in the shipping context environment, is likely to be used when ships are near shore and can use dial-up connections using *Global System for Mobile Communication* (GSM) or similar carrier. This protocol works over TCP/IP networks that have firewalls installed and only the

HTTP protocol is employed throughout. Firewalls are software or hardware systems, which are installed to block any malicious incoming events from unauthorised persons. They usually permit communication only on port 80, which is the default port for this protocol. *Hyper Text Transfer Protocol Secure* (HTTPS) will be used if confidential information needs to be exchanged between the two legacy applications. This is because it adds a layer of complexity, which may reduce the transfer rate of the data. In a general context environment, the SOAP protocol over the HTTP or the HTTP alone is likely to be used for almost all transactions taking place between the two peers.

After the most appropriate communication protocol is selected, based on the above criteria, the Security Manager takes place to negotiate the level of security that needs to be applied on the communication line. It is important to note that different security techniques may be applied on different communication protocols. In this scenario, the available security protocols are as follows:

- **Encryption over SMTP** with different encryption algorithms i.e.3DES. 3DES is a cryptosystem, which can encrypt and decrypt data using a single secret key.

- **Encryption over FTP and HTTP** with 40-bit or 128-bit key. The *Secure Socket Layer* (SSL) security protocol provides data encryption, server authentication, message integrity, and optional client authentication for a TCP/IP connection.

- **Digital Signatures**. Asymmetric (or public key) cryptography involves two related keys, one of which only the owner knows (the 'private key') and the other which anyone can know (the 'public key'). The advantages this technology has provided are that only one party needs to know the private key; and that knowledge of the public key by a third party does not compromise security.

  A digital signature is a 'message digest' (created by processing the message contents using a special algorithm) encrypted using the sender's private key. The recipient can, by re-creating the message digest from the message that they receive, using the sender's public key to decrypt the digital signature. Then by comparing the two results, satisfy themselves not only that the contents of the message received must be the same as that which was sent (data integrity), but also that the message can only have been sent by the supposed sender (sender authentication), and that the sender cannot credibly deny that they sent it (non-repudiation) [108].

- **No Security**. This is the default value of all Java Web Services and it is used when security is not an issue and the information that needs to be interchanged is not confidential.

After the security protocol has been selected and before it is ready to be used by the communication protocol, a "*validation check*" is performed to identify if the selected security protocol is applicable for the selected communication protocol. For example, if the selected communication protocol was HTTP and the selected security protocol was the "Encryption over SMTP" then the security protocol is discarded since it cannot be applied over the HTTP protocol. In such a case, the Security Manager continues to the next available security protocol and the "*validation check*" is performed once again. For this scenario, the next available protocol will be the "Encryption with 128-bit key", which is compatible with the HTTP protocol.

Once the communication protocol and the security protocols have been selected, the two peers are now ready to interchange the XML data for the purpose of integrating the two applications.

The JWSI of the server, via the *File Storage Manager* (FSM), reads the exported CSV file that has been created by the local application. The FSM converts the exported data into the corresponding XML file, which in turn is passed to the Communication Manager via the Schedule Manager in order to be transmitted to the requested JWSI of the client.

When the transmitted XML file is reached to the client site then the Communication Manager passes the XML file to the XML converter Manager. This is in order to convert the incoming XML file to an XML file readable by the client application. This is accomplished by the use of the XSLT Engine and the XSLT file that was created at the beginning of this scenario. The output XML file is then converted to a CSV file and stored into a specific folder on the Computer in order to be accessed at a later stage by the client application. By doing that, this completes the integration process of the two peers and the Communication Managers of both peers close the communication line.

The JWSI then enters into the same stage, as before, that is to monitor for new Java Web Services on the UDDI registry.

As can be seen, the above scenario is composed of three phases.

A. ***Create/publish Server Java Web Service***
B. ***Configuration of client legacy application***
C. ***Dynamic integration***

Below is a list of steps that summarises the process followed in these three phases.

Phase A: Create/publish Server Java Service

- ❑ Start "*Create Service Wizard*"
- ❑ Specify Service attributes.
- ❑ Specify service application's inputs
- ❑ Specify service application's outputs.
- ❑ Prepare Service Schema.
- ❑ Store Service Schema into local database.
- ❑ Publish Service in UDDI.

Phase B: Configuration of client legacy application

- ❑ Start "Configuration" Wizard.
- ❑ Specify client application description
- ❑ Specify client application domain area
- ❑ Store configuration into local database

Phase C: Dynamic integration

- ❑ New data found from legacy application.
- ❑ Initiate the Service Discovery Manager.
- ❑ Suitable Service found and downloaded from UDDI.
- ❑ Contact Server peers and prepare for handshake.
- ❑ Negotiate Communication Protocol.
- ❑ Negotiate Security Protocol.
- ❑ Establish connection between peers.
- ❑ Transfer of data in a SOAP envelope.
- ❑ Convert received data into local schema based on the XSLT.
- ❑ Convert transformed XML data into CSV format.

The key issue of a shipping integration approach is the fact that most shipping applications can be classified as legacy ones, i.e. they do not provide an open *Application Programming Interface* (API). Application integration without the use of API is not so efficient. But on the contrary, this research suggests a unique dynamic integration methodology using the latest technologies. This is possible without the dependence on the application provider, as the legacy application does not have to be modified internally in any way.

In conclusion, it can be seen that the above detailed architecture of the platform satisfies all the requirements listed in section 3.2 in order to achieve the dynamic application integration using the peer to peer technology.

3.6 CONCLUDING REMARKS

This chapter provided the requirements analysis for this thesis and a survey of relevant methodologies. Java Web Services provide the potential solution for integration. These ingredients are required to exchange information in a peer-to-peer manner for dynamic application integration.

An approach to the dynamic application integration has been proposed, which is followed by a detailed description of the technical architecture of the overall system. Finally, the different communication protocols have been described, giving a summary of the steps involved in achieving the dynamic application integration.

In the next chapter, the design of the proposed system will be presented taking into account the results of the requirements analysis and the conclusions drawn from the methodological and architectural issues.

Chapter 4

SYSTEM DESIGN AND ANALYSIS

4.1 INTRODUCTION

Following the requirements specification phase, the design process aims to provide an efficient solution to the problem stated, that is to achieve P2P dynamic application integration. Efficiency is assessed in terms of delivering a system that meets the requirements in a way acceptable to the users, subject to constraints by the limitations of tools and methodologies available.

The design phase uses models that can be understood by technically and none technically agnostic people. This chapter presents such models, which are needed for the design aspects of the Dynamic Integration platform. During the description of the design, the *Unified Modeling Language* (UML) is applied, which is a standard language for visual modeling. In this respect, Rational *Rose 2002* will be used, since this tool supports and incorporates the UML notation. The basic elements of UML and Rational Rose together with their corresponding notations are also presented in this chapter.

4.2 UML AND RATIONAL ROSE

Modeling is an essential part of large software projects, and helpful to medium and even small projects as well. A model plays the analogous role in software development that blueprints and other plans, such as site maps, physical models, etc, play in the construction of a building. Using a model, those responsible for a software development project's success can assure themselves that business functionality is complete and correct, end-user needs are met, and program design supports requirements for scalability, robustness, security, extendibility, and other characteristics, before implementation in code renders changes difficult and expensive to make. There are many additional factors for a project's success, but having a rigorous modeling language standard is one essential factor [107]. For these reasons a modeling language for modeling the system during the design phase is essential. It has been decided to use the UML as the modeling language for the reasons outlined in the following section.

4.2.1 The Unified Modeling Language

The *Unified Modeling Language* (UML) is the industry-standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems. It simplifies the complex process of software design, making a "blueprint" for construction [2,8].

The UML specifies a modeling language that incorporates the object-oriented community’s consensus on core modeling concepts. It allows deviations to be expressed in terms of its extension mechanisms. The objectives of the UML modelling language are to:

- provide sufficient semantics and notation to address a wide variety of contemporary modeling issues in a direct and economical fashion.
- provide sufficient semantics to address certain expected future modeling issues, specifically related to component technology, distributed computing, frameworks, and executability.

- provide extensibility mechanisms so individual projects can extend the metamodel for their application at low cost.
- provide extensibility mechanisms so that future modeling approaches could be grown on top of the UML.
- provide sufficient semantics to facilitate model interchange among a variety of tools.
- provide sufficient semantics to specify the interface to repositories for the sharing and storage of model artifacts.

There are nine types of *Diagrams* in the UML, which are outlined as follows.

- **Class diagram:** Shows a set of classes, interfaces, and collaborations and their relationships. This is the most common type of diagram used when modeling Object Oriented systems.
- **Object diagram**: Shows a set of objects and their relationships. Can be thought of as an instance of a Class diagram.
- **Use case diagram**: Shows a set of use cases and actors and their relationships. These types of diagrams drive the whole development process since they describe the requirements of the system.
- **Sequence diagram**: Shows an interaction, consisting of a set of objects and their relationships, including the messages that may be dispatched among them. Emphasizes the time-ordering of messages.
- **Collaboration diagram**: Shows an interaction, consisting of a set of objects and their relationships, including the messages that may be dispatched among them. Emphasises the structural organization of the objects that send and receive messages.
- **Statechart diagram**: Shows a state machine, consisting of states, transitions, events, and activities.

- **Activity diagram**: Special kind of a Statechart diagram that shows the flow from activity to activity within a system. They are very similar to flowchart diagrams except that concurrency may be modelled in Activity diagrams.

- **Component diagram**: Shows the organizations and dependencies among a set of components.

- **Deployment diagram**: Shows the configuration of run - time processing nodes and the components that live on them.

There are also five types of Views in the UML, which are the following:

- **Use case view**: Encompass the use cases that describe the behaviour of the system as seen by its end users, analysts, and testers.

- **Design view** : Encompass the classes, interfaces, and collaborations that form the vocabulary of the problem and its solution.

- **Process view** : Encompass the threads and processes that form the system's concurrency and synchronization mechanisms.

- **Implementation view** : Encompass the components and files that are used to assemble and release the physical system.

- **Deployment view** : Encompass the nodes that form the system's hardware topology on which the system executes.

Each of the five views is a projection into the organisation and structure of the system, focused on a particular aspect of that system. Each of these five views can stand alone so that different stakeholders can focus on the issues of the system's architecture that most concern them. The five views also interact with each other. For example, nodes in the deployment view hold components in the implementation view that, in turn, represent the physical realisation of classes, interfaces, collaborations, and active classes from the design and process views.

4.2.2 Rational Rose

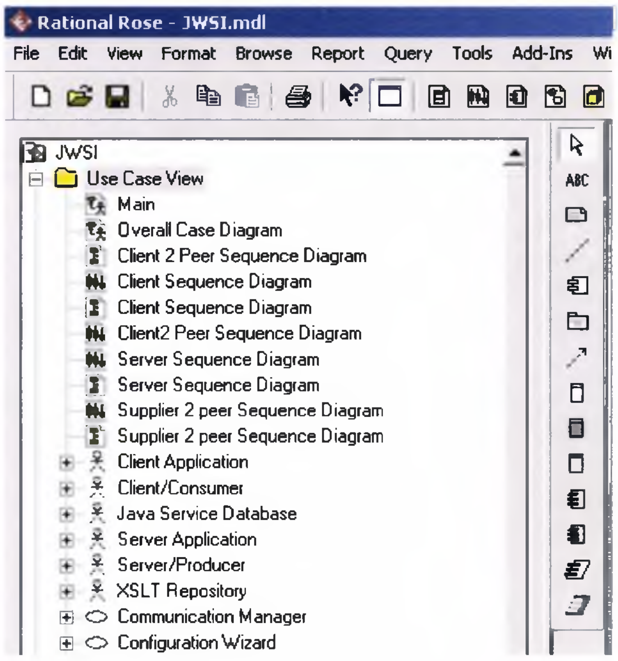
The analysis and design of the system has been supported by the use of a modeling tool called *Rational Rose*. Rational Rose supports the UML (see section 4.2.1), an industry standard, that allows analysts and designers to express object-orientated concepts.

Rational Rose uses some district views during the stages of analysis and design. These views are: Use Case view, Logical view, Component view and Deployment view. These views allow the users to model the components and interfaces of a system, specify its behaviour and define the collaborations among the system elements.

A brief description of each one of these views follows.

- **Use Case View**

  The *Use Case View* includes all of the actors, Use Cases, and Use Case Diagrams in the system. The Use Case view is an implementation-independent presentation of the system. It focuses on a high level picture of **what** the system will do, without worrying about the details of how the system will do it. The Use Case View is illustrated in figure 4.1.

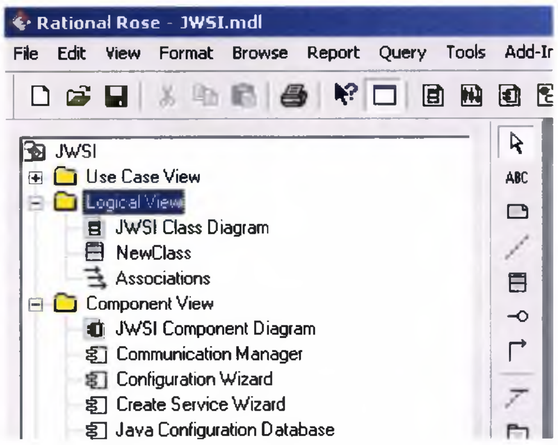


JWSI
Use Case View
Main
Overall Case Diagram
Client 2 Peer Sequence Diagram
Client Sequence Diagram
Client Sequence Diagram
Client2 Peer Sequence Diagram
Server Sequence Diagram
Server Sequence Diagram
Supplier 2 peer Sequence Diagram
Supplier 2 peer Sequence Diagram
Client Application
Client/Consumer
Java Service Database
Server Application
Server/Producer
XSLT Repository
Communication Manager
Configuration Wizard

Figure 4.1 *Use Case View in the Rational Rose Browser*

- **Logical View**

The *Logical View* focuses on **how** the system will implement the behaviour in the Use Cases. It provides a detailed picture of the components of the system and describes how the components are interconnected. The Logical View includes among other things, the Analysis Model and the Design Model. The Logical View is illustrated in figure 4.2.


Rational Rose - JWSI.mdl

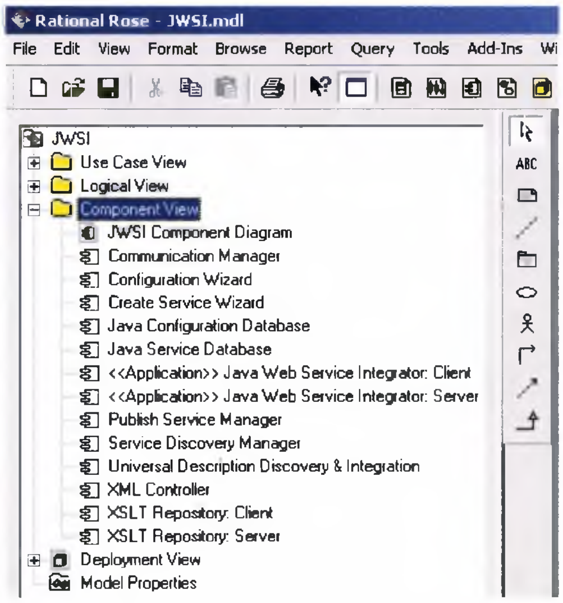
File Edit View Format Browse Report Query Tools Add-Ir

JWSI
Use Case View
Logical View
JWSI Class Diagram
NewClass
Associations
Component View
JWSI Component Diagram
Communication Manager
Configuration Wizard
Create Service Wizard
Java Configuration Database

Figure 4.2 *Logical View in the Rational Rose Browser*

- **Component View**

The *Component View* contains information about the executable files, runtime libraries and other components of the system model. A *Component* is a physical module of code. In Rational Rose, Component Diagrams are displayed in the Components View. The Component View allows you to see the relationships between the modules. The Component View can be seen in figure 4.3.


Rational Rose - JWSI.mdl

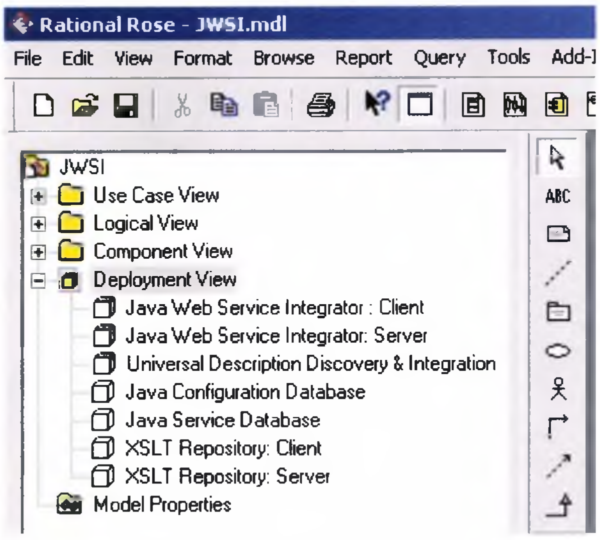
File Edit View Format Browse Report Query Tools Add-Ins Wi

JWSI
Use Case View
Logical View
Component View
JWSI Component Diagram
Communication Manager
Configuration Wizard
Create Service Wizard
Java Configuration Database
Java Service Database
<<Application>> Java Web Service Integrator: Client
<<Application>> Java Web Service Integrator: Server
Publish Service Manager
Service Discovery Manager
Universal Description Discovery & Integration
XML Controller
XSLT Repository: Client
XSLT Repository: Server
Deployment View
Model Properties

Figure 4.3 *Component View in the Rational Rose Browser*

- **Deployment View**

  The final view in Rational Rose is the *Deployment View*. The Deployment View is concerned with the physical deployment of the system. The Deployment View is illustrated in figure 4.4.



JWSI
- Use Case View
- Logical View
- Component View
- Deployment View
  - Java Web Service Integrator : Client
  - Java Web Service Integrator: Server
  - Universal Description Discovery & Integration
  - Java Configuration Database
  - Java Service Database
  - XSLT Repository: Client
  - XSLT Repository: Server
- Model Properties

Figure 4.4 *Deployment View in the Rational Rose Browser*

4.2.3 Elements and Notations

This section presents some elements and notation of the Unified Modeling Language. These elements are used during the analysis and design stages of the system. The graphical notation and textual syntax are essentially the most basic part of the UML, utilised by tools and end-users in order to model systems.

The following are some of the important elements and graphical notations of UML [2, 27, 41, 104].

- **Use Case**

  A Use Case (see figure 4.5) can be described as a specific way of using the system from a user's (actor's) perspective. A more detailed description might characterise a Use Case as:

- A pattern of behaviour the system exhibits
- A sequence of related transactions performed by an actor and the system
- Delivering something of value to the actor

Use cases provide a means to:

- Capture system requirements
- Communicate with the end users and domain experts
- Test the system

Use cases are best discovered by examining the actors and defining what the actor will be able to do with the system.

Since all the needs of a system typically cannot be covered in one use case, it is usual to have a collection of use cases. Together, this Use Case collection specifies all the ways of using the system.

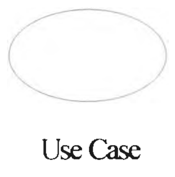


Figure 4.5 *UML notation for a Use Case*

- **Actor**

  Actors represent system users (see figure 4.6). They help delimit the system and give a clearer picture of what the system should do. It is important to note that an actor interacts with, but has no control over the use cases. An actor is someone or something that:

  - Interacts with or uses the system
  - Provides input to and receives information from the system
  - Is external to the system and has no control over the use cases

Actors are discovered by examining:

- Who directly uses the system
- Who is responsible for maintaining the system
- External hardware used by the system
- Other systems that need to interact with the system

The needs of the actor are used to develop Use Cases. This insures that the system will be what the user expected.

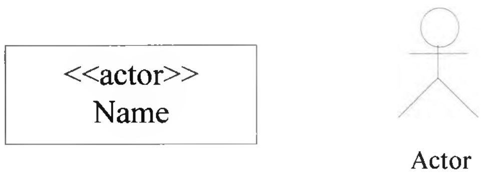


Figure 4.6 *UML notation for an Actor*

- **Use Case Relationships**

  An association provides a pathway for communication. The communication can be between Use Cases, Actors, Classes or Interfaces. Associations are the most general of all relationships and consequentially the most semantically weak. If two objects are usually considered independently, the relationship is an association.

  An *Extend relationship* (see figure 4.7) is a stereotyped relationship that specifies how the functionality of one Use Case can be inserted into the functionality of another Use Case. Extend relationships between Use Cases are modeled as dependencies by using the Extend stereotype. Moreover extend relationships are important because they show optional functionality or system behavior.

Uses relationship (see Figure 4.7) is a stereotype set on the Generalised specification that can be attached to a model element to give it a specialised meaning. The Uses Generalisation indicates that one Use Case uses the functionality of another Use Case. The Use Case being used typically contains functionality that a number of other Use Cases may need or want.

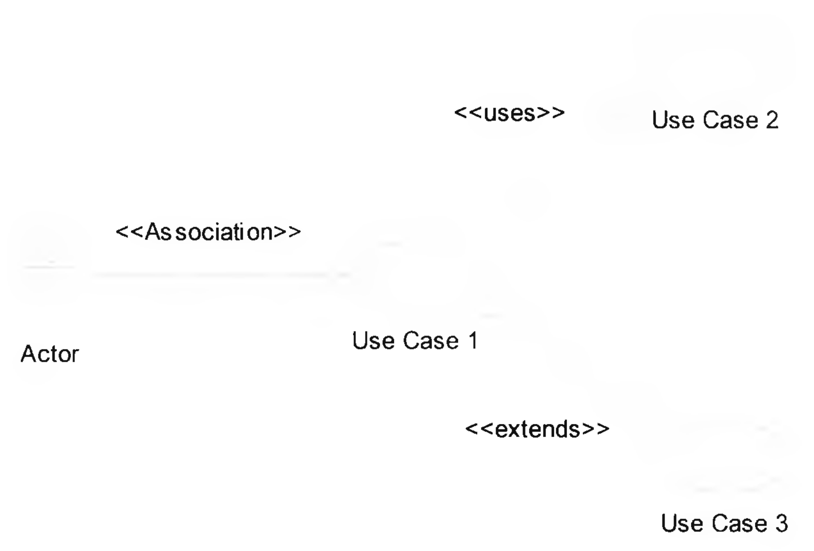


Figure 4.7 *UML notation for Use Case Relationships*

- **Sequence Diagrams**

A *Sequence diagram* is a graphical view of a scenario that shows object interaction in a time-based sequence; what happens first, what happens next. Sequence diagrams establish the roles of objects and help provide essential information to determine class responsibilities and interfaces. This type of diagram is best used during early analysis phases in design because they are simple and easy to comprehend. Sequence diagrams are normally associated with use cases.

A Sequence diagram has two dimensions: typically, vertical placement represents time and horizontal placement represents different objects. Figure 4.8 illustrates the UML notation for sequence diagrams.

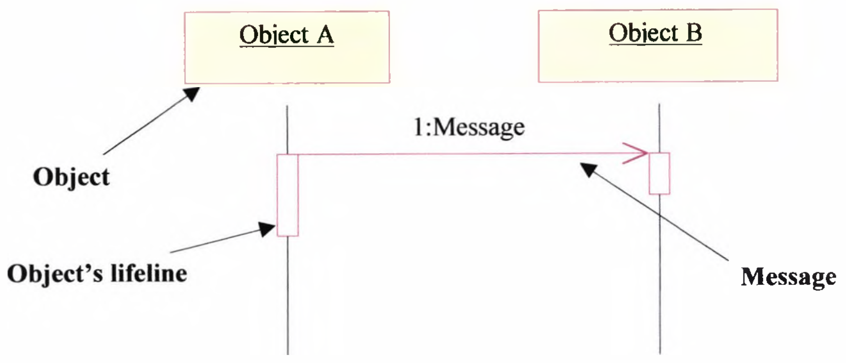


Figure 4.8 *UML notation for Sequence Diagrams*

- **Collaboration Diagrams**

Collaboration diagrams (see figure 4.9) show objects, their links and their messages. They can also contain simple class instances and class utility instances. Each Collaboration diagram provides a view of the interactions or structural relationships that occur between objects and object-like entities in the current model.

Sequence diagrams are closely related to Collaboration diagrams and both are alternate representations of an interaction. There are two main differences between Sequence and Collaboration diagrams: Sequence diagrams show time-based object interaction while Collaboration diagrams show how objects associate with each other.

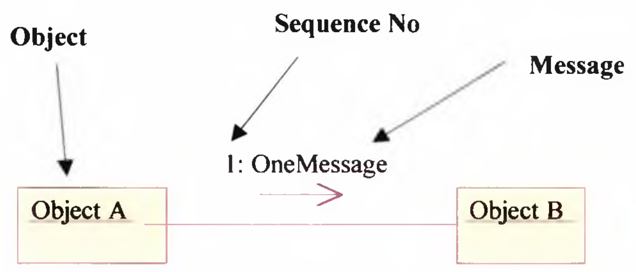


Figure 4.9 *UML notation for Collaboration Diagrams*

❑ **Component Diagrams**

Component diagrams (see figure 4.10) provide a physical view of the current model. A component diagram shows the organisations and dependencies among software components, including source code components, binary code components, and executable components.

These diagrams also show the externally-visible behaviour of the components by displaying the interfaces of the components. Calling dependencies among components are shown as dependency relationships between components and interfaces on other components. Note that the interfaces actually belong to the logical view, but they can occur both in class diagrams and in component diagrams.

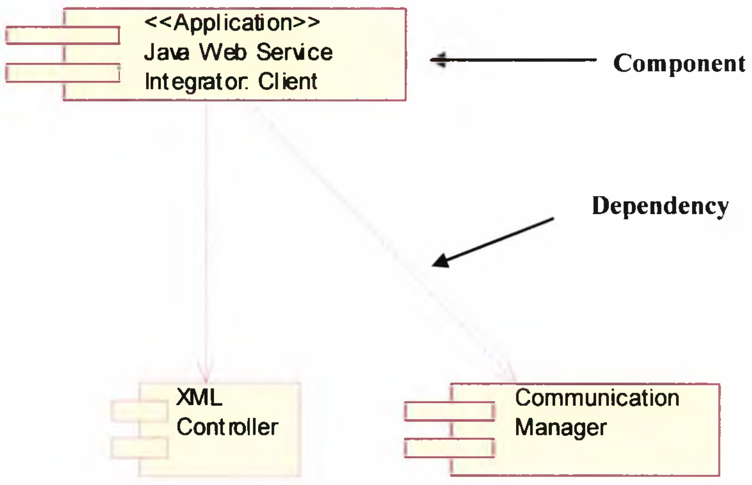


Figure 4.10 *UML notation for Component Diagrams*

❑ **Deployment Diagrams**

A *Deployment diagram* (see figure 4.11) shows processors, devices, and connections. Each model contains a single deployment diagram, which shows the connections between its processors and devices, and the allocation of its processes to processors.

Processor Specifications, Device Specifications, and Connection Specifications enables modification of the respective properties. The information in a specification is presented textually; some of this information can also be displayed inside the icons.

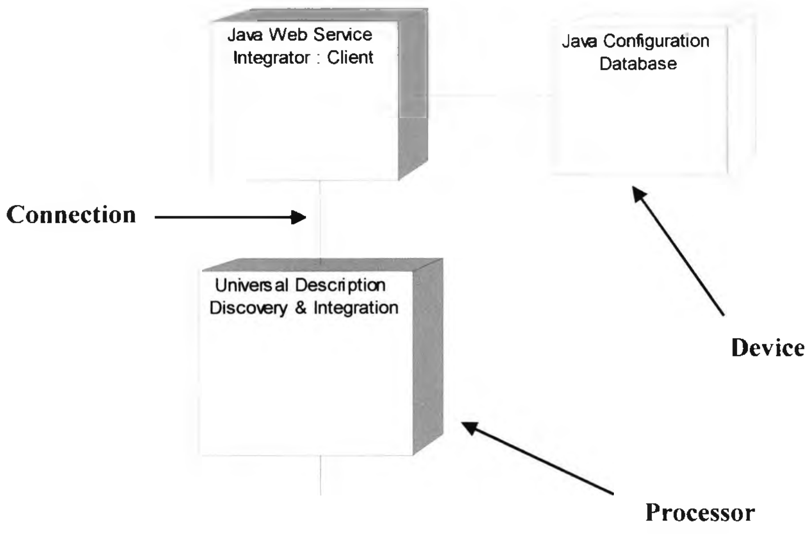

Java Web Service Integrator : Client
Java Configuration Database
Connection
Universal Description Discovery & Integration
Device
Processor

Figure 4.11 *UML notation for Deployment Diagrams*

- **Class & Object Diagrams**

A *Class diagram* (see figure 4.12) is a picture for describing generic descriptions of possible systems. Class diagrams and collaboration diagrams are alternate representations of object models. Class diagrams contain classes and *Object diagrams* contain objects, but it is possible to mix classes and objects when dealing with various kinds of metadata, so the separation is not rigid. Class diagrams may be constructed in conjunction with Object diagrams but are more common than object diagrams. Normally, the class diagrams are built first, plus occasional object diagrams, illustrating complicated data structures or message-passing structures.

Class diagrams contain icons representing classes, interfaces, and their relationships. One or more class diagrams can be created to depict classes contained by each package in a model; such class diagrams are themselves contained by the package enclosing the classes they depict; the icons representing logical packages and classes in class diagrams.

An object diagram shows the existence of objects and their relationships in the logical design of a system. An object diagram may represent all or part of the object structure of a system, and primarily illustrates the semantics of mechanisms in the logical design. A single object diagram represents a snapshot in time of an otherwise transitory event or configuration of objects.

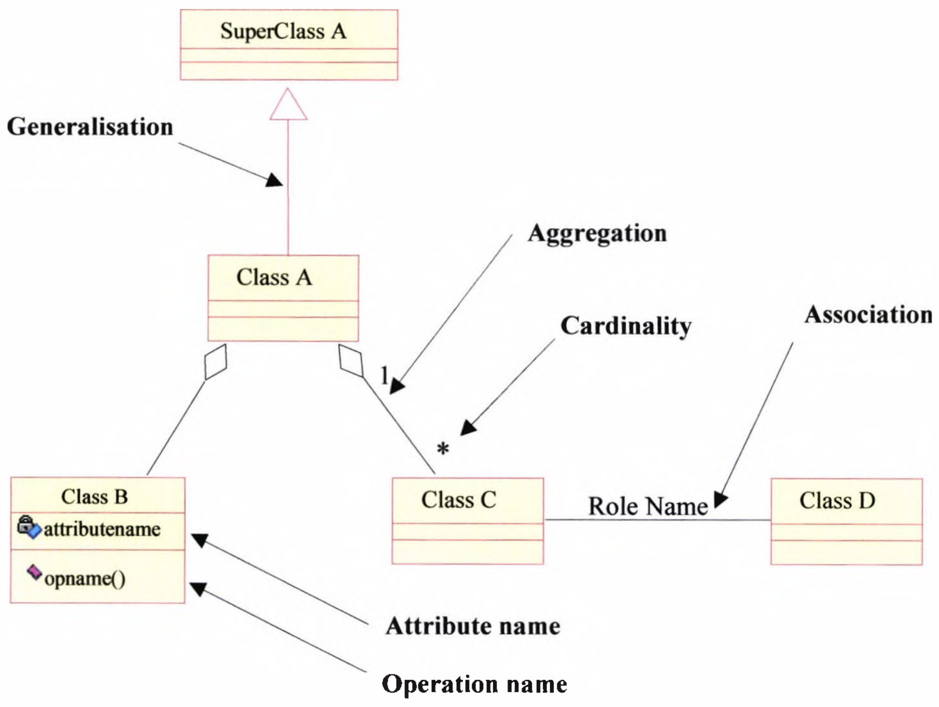


Figure 4.12 *UML notation for Class Diagrams*

4.3 USE CASE VIEW

The first view to be presented is the *Use Case View* (see section 4.2.2*)*. The Use Case View typically consists of *Use Cases* and the associated Sequence and Collaboration diagrams. The following sections present these diagrams for the proposed dynamic peer-to-peer application integration platform.

4.3.1 Use Case Diagrams

Use-case diagrams graphically depict system behaviour (use cases). These diagrams present a high level view of how the system is used as viewed from an actor's perspective. A use-case diagram may depict all or some of the use cases of a system.

A use-case diagram can contain:

- ❑ Actors, which are entities outside the system.
- ❑ Use Cases, which are system boundaries identifying what the system should do. *Use Cases* were first introduced by Ivar Jacobson [61] in the early 1990s. Use Cases document the behaviour of the system from the user's point of view.
- ❑ Interactions or relationships between actors and Use Cases in the system including the associations, dependencies, and generalisations.

Use-case diagrams can be used during analysis to capture the system requirements and to understand how the system should work. During the design phase, use-case diagrams can be used to specify the behaviour of the system as implemented.

They are used primarily for visualising the Use Cases. The following two sections describe the Actors and the Use Cases of the integration system.

4.3.1.1 Actors

- **Server/Producer**

  The *Server,* or the *Producer,* is the actor that uses the *Java Web Service Integrator* (JWSI) for creating and publishing a Java Web Service. This is achieved by having the JWSI initiating the Create Service Wizard. The purpose of this wizard is to retrieve all relevant information from the supplier application and any other information that is considered important by the Supplier. Examples of this information may include the inputs and outputs of the server application, the purpose of the application, a description etc. Then, the wizard stores this information into a Java Service Database for future reference. Subsequently, the JWSI initiates the Publish Service Manager. The purpose of this manager is to retrieve the service information from the Java Service Database and to create a Java Web Service based on this information to be published on the Universal Description, Discovery and Integration (UDDI) directory. Before the actual publishing of the service takes place, the Publish Service Manager creates automatically some mappings between the data fields of the published service and the data fields of the supplier application. These mappings are stored into an XSLT Repository. This is very important since the mappings will be used when the client application request some data from the server via the Java web service. The Server/Producer may be a Client/Consumer actor as well.

- **Server Application**

  The Server Application is an important actor of the system. Its main purpose is twofold. Firstly, the Server application is being used by the Create Service Wizard component in order to collect any information regarding that application. This is important since this information will be used to create the published service. Moreover, this information is stored into the Java Service Database for further processing. Secondly, the server application is being used by the XML controller on the server site, to execute the server application and get the returned data. Then, the XML controller can convert these results into the compatible format of the Java Web Service in order to be returned back to the requested client application for further processing.

- **Java Service Database**

  The *Java Service Database* is another actor of the system, since it is also something external to the system, which interacts and exchanges data with it. The responsibility of this Database is to store information related to Java Web Services. This information is taken from either the Server application or the server actor. The information stored in this database is extremely important since this is the information that will be used when creating and publishing the web service on the UDDI.

- **XSLT Repository**

  The *XSLT Repository* is also an actor of the system. It is another database of the system that holds information related to the mappings between the client or server and the Java Web Service. These mappings are created before the Java Web Service is ready to be published on the UDDI and are used when the JWSI on the server site request a specific web service. In such a case, the relevant XSLT mappings are loaded to be used when converting the server data into the web service compatible format. The same applies on the client site when the data has returned from the server and need to be passed to the client application.

- **Java Configuration Database**

  The *Java Configuration Database* is another actor of the system since it is also something external to the system, which interacts and exchanges data with it. The responsibility of this configuration Database is to store information about potential web services that need to be discovered by the JWSI. This information is taken from either the Client application or the client actor. The information stored in this database is extremely important since this is the information that will be used when the Service Discovery Manager searches for potential web services on the UDDI matching the specification of the configuration file and subsequently, the specification of the client application needed to be dynamically integrated.

- **Client Application**

  The Client Application is an important actor of the system. Its main purpose is also twofold, just like the Server Application. Firstly, the Client application is used by the Configuration Wizard component so as to collect any information regarding that application. This is important since this information will be used to search for web services matching the specifications of the client application. Furthermore, this information is stored into the Java Configuration Database for further processing. Secondly, the client application is being used by the XML controller on the client site, to return the data received from the Java Web Service.

- **Client/Consumer**

  The *Client* or the *Consumer* is the actor that actually instantiates the dynamic integration process. The client uses the JWSI, in order to initiate the Configuration Wizard. The purpose of this wizard is to create a configuration file that will describe the purpose and operations of the client application that need to be integrated dynamically. This is achieved by loading all relevant information from the client application and any other inputs that are considered important by the client. Examples of this configuration information may include the inputs and outputs of the application, the purpose of the application, a description, etc. Then, the wizard stores this information into a Java Configuration Database for future access. Subsequently, the JWSI initiates the Service Discovery Manager, which will search for Java Web Services on the UDDI directory based on the configuration data stored previously by the client. The service discovery manager will download only the web service that matches exactly the specification of the configuration data in order to prevent any redundancies and avoid inconsistencies during the dynamic integration. The Client/Consumer may be a Supplier/Producer actor as well.

4.3.1.2 Use Cases

The diagram in figure 4.13 illustrates the overall Use Case Model of the system.

As can be seen from the Use Case Diagram in figure 4.13, the *Server/Producer* initiates the *Java Web Service Integrator (JWSI),* which runs as a service application on the server machine. The JWSI uses the Create Service Wizard to create a web service specification according to the input data taken either from the server application or the Producer itself. All relevant information regarding the server application is important so as to create a web service as accurate as possible. If incorrect data are passed through this wizard, then the web service will not be completely compatible with the server application and the potential client application needed to be integrated will not be in a position to find that web service on the UDDI.

After the service specification is created, the JWSI initiates the Publish Service Manager, which will publish the service on the UDDI. The Publish Service Manager also creates an XSLT file, which corresponds to the mappings between the fields of the web service and the fields of the server application. The JWSI on the server site can now get into an idle mode waiting for incoming requests from potential clients.

On the other site, the *Client* needs to configure the client application before the integration takes place. This is accomplished by initiating the JWSI and subsequently using the Configuration Wizard, the configuration file holding the specification of the client application, is created. This is important since the configuration data describes in detail the purpose of the client application in order to find potential web services based on that configuration.

After the configuration data has been saved into the Java Configuration Database, the JWSI initiates the Service Discovery Manager for the purpose of searching the UDDI and to find the Java Web Services that best fit into the specification of the client application. After the appropriate web services have been discovered, they are downloaded on the client site in order to initiate the integration process.

Subsequently, the JWSI initiates the Communication Manager, which will execute the web service in order to retrieve the data directly from the server site and hence achieve the peer-to-peer communication without any intermediate servers or applications.

Currently, the communication Manger supports a number of communication protocols. After the server peer is contacted, the communication manager of the server site requests the data from the XML Controller. The XML Controller will use the already created XSLT file and will apply it to the data taken from the server application in order to be compatible with the requested web service.

Once the conversion is completed, the converted results are returned back to the Communication Manager of the client through the Communication Manager of the server to be fed to the client application. Before the data are passed to the client application, they are converted by the XML Controller on the client site using the automatically created XSLT mappings.

The XSLT file is created based on the information taken from the configuration database and the information taken from the service found on the UDDI. Only in very special circumstances where the JWSI is not able to automatically create compatible mappings, the client will use the configuration wizard again to manually create these mappings.

By completing all the above steps, the dynamic peer-to-peer application integration is achieved.

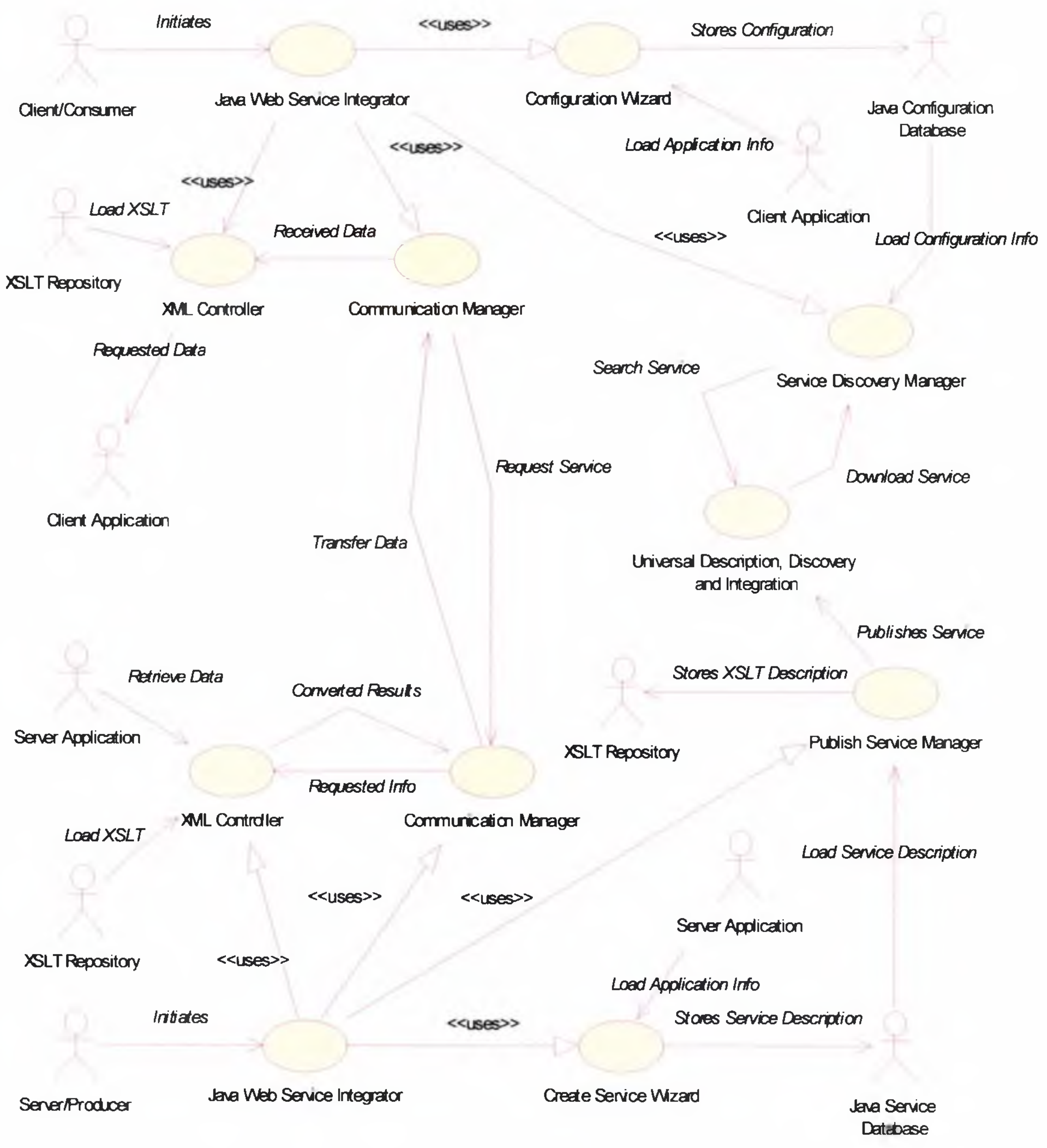


Figure 4.13 *Overall Use Case Diagram of the system*

4.3.2 Sequence Diagrams

The purpose of the Use Case knowledge that has been presented in the previous section was to provide a guideline in the design model that would organise the artifacts related to the Use Cases. These artifacts consist typically of the Collaboration and Sequence diagrams. A description of these diagrams is provided in section 4.2.3. The Sequence diagrams identified for the proposed Peer-to-Peer platform are presented in the next sections. Because of the high complexity of the system, only the high-level sequence diagrams will be presented, and also they have been broken into four smaller manageable sequence diagrams for better understanding.

4.3.2.1 Server/Producer Sequence Diagram

The *Server/Producer Sequence Diagram* shows object interaction in a time-based sequence that occurs on the server's site. This is illustrated in figure 4.14. As already mentioned, a server may act as a client as well. It is important to also note here that in order to have a complete and dynamic Peer-to-Peer integration scenario, the actions performed on this figure must be performed before the actions performed that are illustrated in figures 4.15 and 4.16.

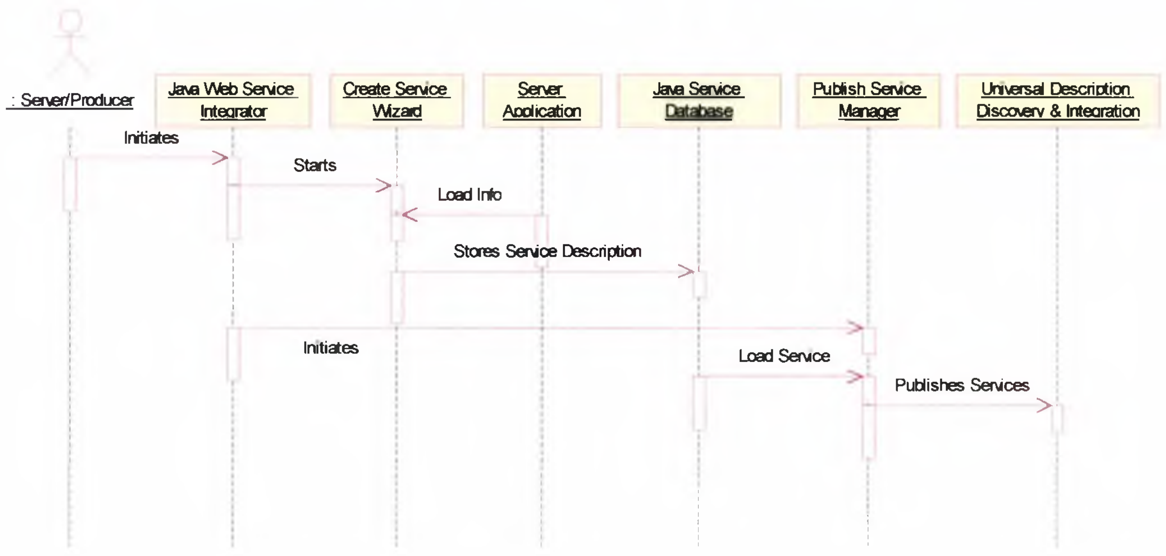


Figure 4.14 *The Server/Producer Sequence Diagram*

The first action shown in the diagram in figure 4.14, is performed by a user on the server site, who initiates the Java Web Service Integrator. The JWSI then starts the Create Service Wizard component. This component is responsible for creating the Java Web service that needs to be published; hence, it requests data from the server's application running locally. After the specification of the service is created, it is stored into the Java Service Database for further access. Subsequently, the JWSI initiates the Publish Service Manager component, which after it loads the service specification from the Database, it creates and publishes the service on the UDDI. The service is stored there until it is accessed from a client requesting that service.

4.3.2.2 Client/Consumer Sequence Diagram

The *Client/Consumer Sequence Diagram* shows object interaction in a time-based sequence that occurs on the client's site. This is illustrated in figure 4.15. It is important to note here that the actions performed on figure 4.14 precede the actions performed on this diagram.

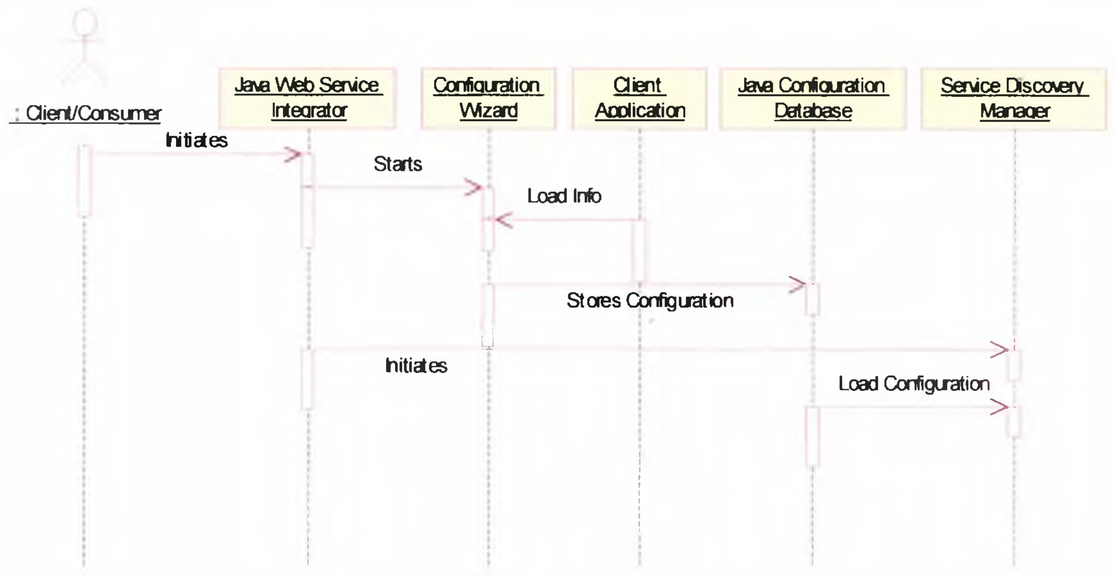


Figure 4.15 *The Client/Consumer Sequence Diagram*

The Client/Consumer, who initiates the Java Web Service Integrator, performs the first action shown in the diagram, in figure 4.15. The JWSI then starts the Configuration Wizard component. This component is responsible for creating a configuration file describing in detail the client application and how it can be accessed. In order to achieve this, the component loads information from the client application and after the configuration file is created, is stored into the Java Configuration Database.

Next, the JWSI initiates the Service Discovery Manager component. After the component is loaded, the configuration data are fetched from the database. This component is responsible to discover any new Java Web services located on the UDDI. If any new services are found, they are then downloaded in order to begin the dynamic peer-to-peer application integration.

4.3.2.3 Client/Consumer to Peer Sequence Diagram

Because of the large number of steps invoved between the two peers after the dynamic integration has been initiated, it has been decided to separate the entire peer-to-peer process into two smaller sequence diagrams. The first one is called Client/Consumer-to-Peer Sequence Diagram and the second one is called Server/Producer-to-Peer Sequence Diagram. The latter will be discussed in section 4.3.2.4.

The *Client/Consumer to Peer Sequence Diagram* shows object interaction in a time-based sequence that occurs between the client and the peer, after the dynamic integration has been initiated. This is illustrated in figure 4.16. It is important to note here that the actions performed on figure 4.15 precede the actions performed on this diagram.

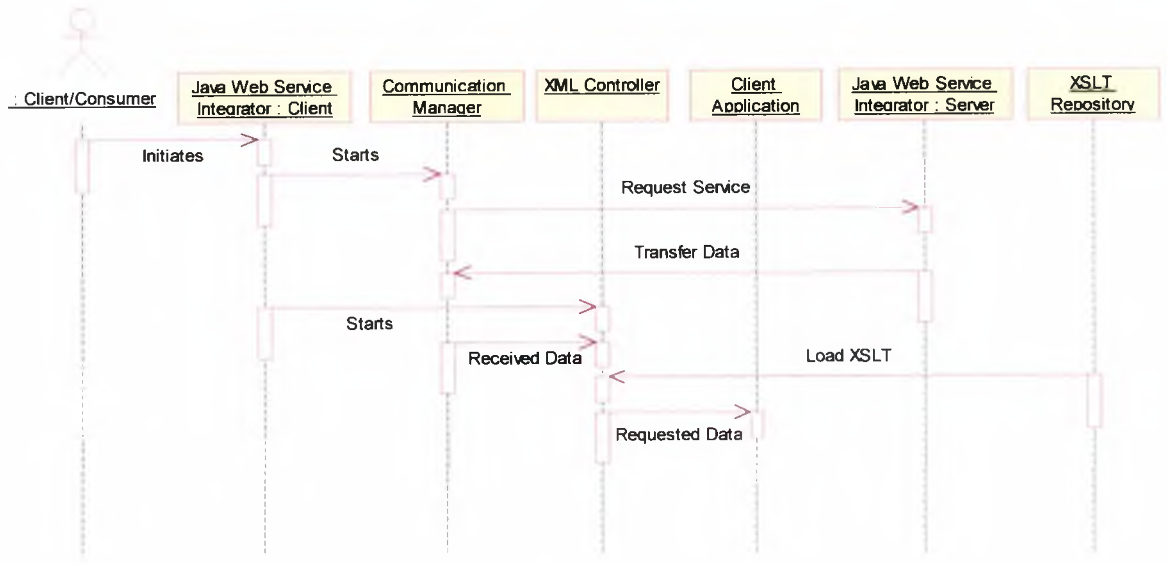


Figure 4.16 *The Client/Consumer to Peer Sequence Diagram*

The first action of the diagram in figure 4.16 is performed by the Client/Consumer, who initiates the JWSI. This action has already been shown in figure 4.15. Following the final step of figure 4.15 and after a new Java Web Service is found and downloaded from the UDDI, the JWSI starts the Communication Manager component. This component is responsible for negotiating with the server, based on a suitable list of communication protocols and prepares a robust and secure communication line between the two peers. After the communication has been performed, the Communication component invokes the service.

On the server site, the Java Web Service Integrator receives this request and after processing it, it returns the results back to the Communication component of the client. The steps involved on the server site will be discussed in section 4.3.2.4. Once the client receives the data, the JWSI initiates the XML Controller component in order to transform the returned results into a format compatible and readable by the client application. This is achieved by loading an XSLT file from the XSLT Repository, which describes in detail the mappings between the data of the Java Service and the data of the client application. Finally, the transformed results are passed into the client application for further processing. By completing the above steps the dynamic peer-to-peer application integration is completed.

4.3.2.4 Server/Producer to Peer Sequence Diagram

The *Server/Producer to Peer Sequence Diagram* shows object interaction in a time-based sequence that occurs between the server and the peer, after the dynamic integration has been initiated. This is illustrated in figure 4.17.

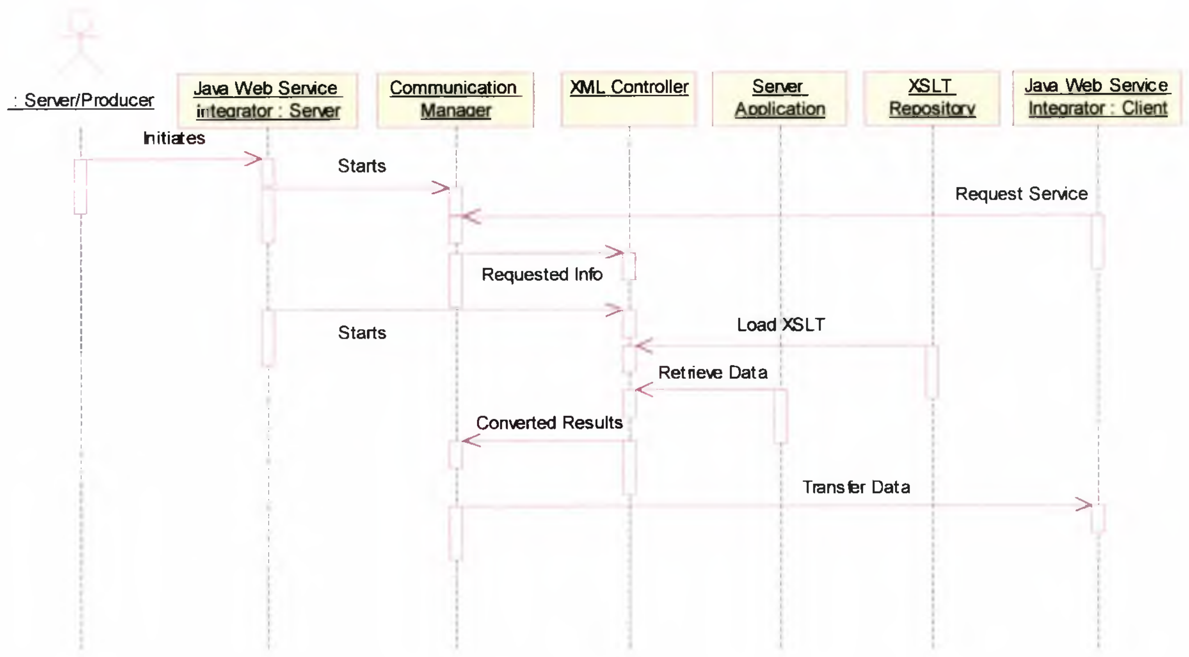


Figure 4.17 *The Server/Producer to Peer Sequence Diagram*

The first action shown in the diagram in figure 4.17 is performed by the Server/Producer, who initiates the JWSI. This action has already being shown in figure 4.14. Following the intermediate steps of figure 4.16 and after a request has been received by the client to invoke the web service, the JWSI starts the Communication Manager component. This component is responsible to negotiate a suitable and secure communication protocol before it receives the actual request from the client. Next, the requested information is passed to the XML Controller. The controller executes the service in order to get the data from the server application and then it loads an XSLT file from the XSLT Repository database.

Subsequently, the XSLT file is applied to the data for creating the transformed data that will be compatible with the service specification. Once the transformation is completed, the data are transferred back to the client for further processing. By completing the above steps, the dynamic peer-to-peer application integration is completed.

4.3.3 Collaboration Diagrams

Sequence and Collaboration diagrams express similar information, but present it in different ways. *Collaboration diagrams* (see figure 4.9) show objects, their links, their messages and are used to demonstrate how objects interact with each other, to perform the behaviour of a particular Use Case or part of a Use Case. The Collaboration diagrams identified for this thesis are presented in the next sections, taking into consideration their equivalent Sequence diagrams. Once again, because of the high complexity of the system, only the high-level Collaboration diagrams will be presented.

4.3.3.1 Server/Producer Collaboration Diagram

The *Server/Producer Collaboration Diagram* shows how the objects interact between each other to perform a specific task on the server's site. This is illustrated in figure 4.18.

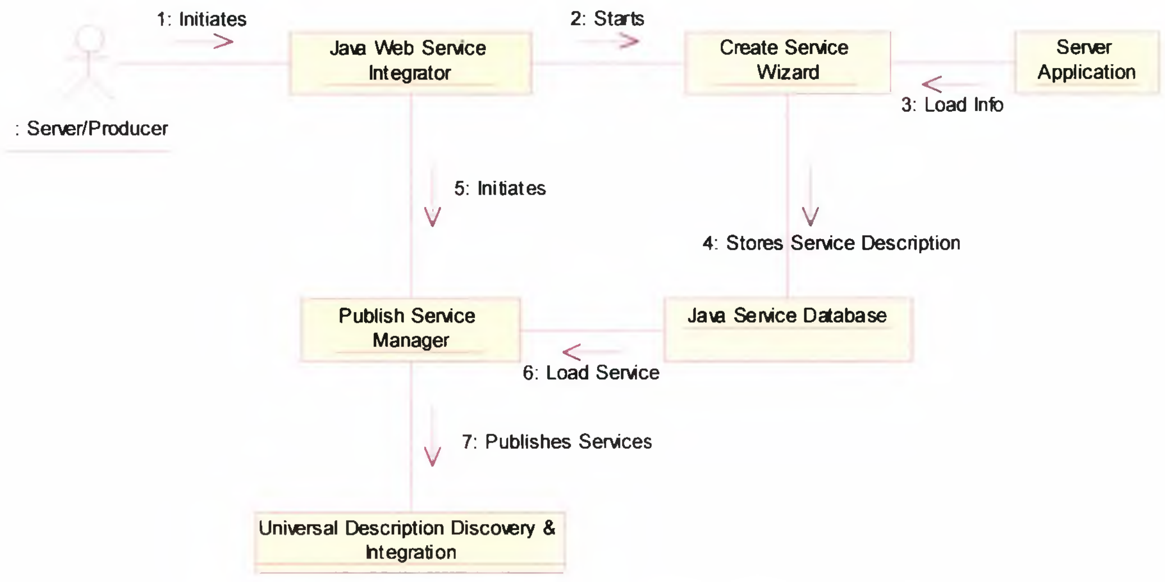


Figure 4.18 *The Server/Producer Collaboration Diagram*

As can be seen from the diagram in figure 4.18, the action "*1:Initiates*" is firstly performed by the *Server/Producer* on the JWSI. The second action "2:Starts" is performed by the JWSI on the *Create Service Wizard* object to create a specification of the service that needs to be published.

Next, action "3:Load Info" is performed by the latter object, on the Server Application, in order to collect the necessary information, before the service is created. Action "4:Stores Service Description" is then executed to store the service specification into the Java Service Database for further processing. Subsequently, action "5:Initiates" is performed by the JWSI to load the Publish Service Manager object. This object is responsible for publishing the service that has been created previously on the UDDI. Before this is done, action "6:Load Service" is executed to load the specified service information from the Java Service Database. Finally, the action "7:Publishes Service" is executed to actually publish the service on the UDDI directory.

4.3.3.2 Client/Consumer Collaboration Diagram

The *Client/Cosnumer Collaboration Diagram* shows how the objects interact between each other to perform a particular task on the client's site. This is illustrated in figure 4.19.

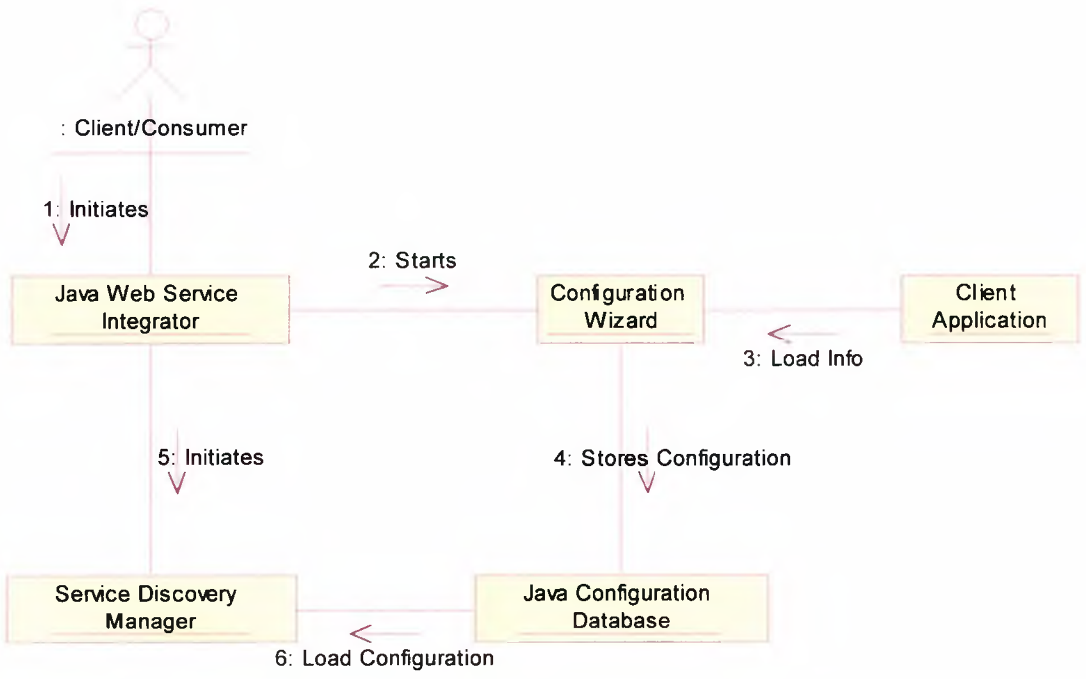


Figure 4.19 *The Client/Consumer Collaboration Diagram*

As can be seen from the diagram in figure 4.19, the Client/Consumer on the JWSI performs the action "1:Initiates" firstly. The second action "2:Starts" is performed by the JWSI on the *Configuration Wizard* object to create a configuration file for the client application. This is important because the configuration file describes the client application in detail so that only Java Services found on the UDDI directory that match the configuration file will be downloaded. This file is created by executing action “3:Load Info” on the client application.

Then action “4:Stores Configuration” is executed to save that configuration file into the Java Configuration Database. After this has been completed, the JWSI executes action “”5:Initiatess” to load the Service Discovery Manager object. This object is responsible for discovering any new services that are found on the UDDI directory. This is done after action “6:Load Configuration” is executed, to retrieve the configuration information from the Java Configuration Database.

4.3.3.3 Client/Consumer to Peer Collaboration Diagram

The *Client/Consumer to Peer Collaboration Diagram* shows how the objects interact between each other to perform a specific task between the client and the Server Peer. This is illustrated in figure 4.20.

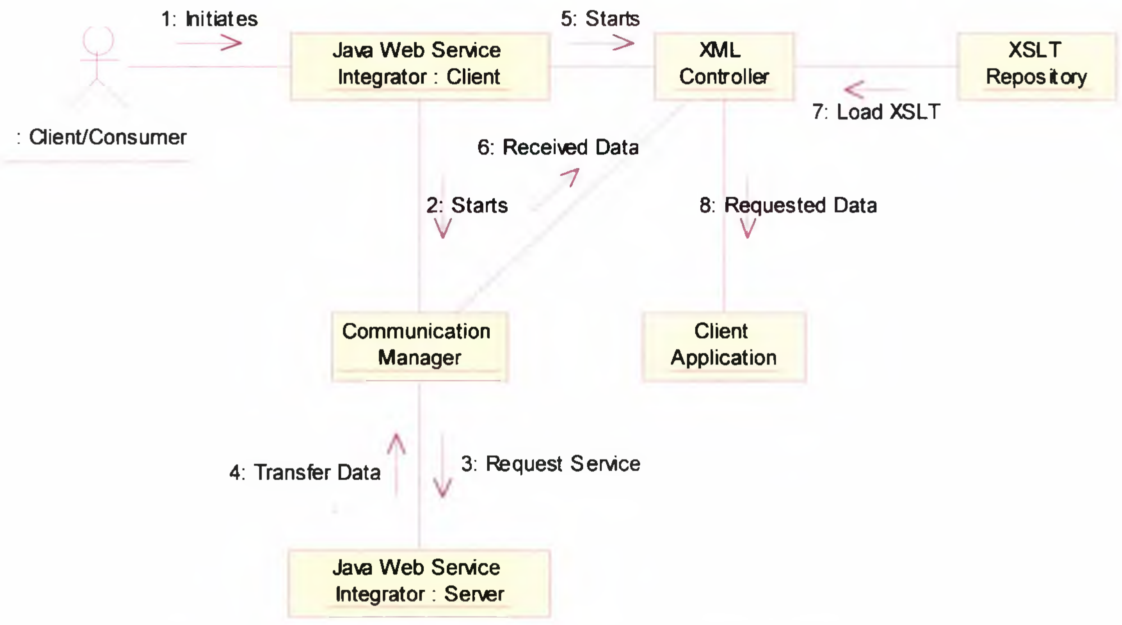


Figure 4.20 *The Client/Consumer to Peer Collaboration Diagram*

As can be seen from the diagram in figure 4.20, the Client/Consumer on the JWSI performs the action “1:Initiates” firstly. This action has already been performed by the Client/Consumer in figure 4.19. It is also demonstrated here in order to better explain the whole scenario. The second action "2:Starts" is performed by the JWSI on the *Communication Manager* object after a new Java Web Service has been dynamically discovered from the UDDI.

The Communication Manager object then executes action "3:Request Service" to actually invoke the service on the server's site. After the service is invoked and processed by the JWSI of the server's site, it returns the appropriate results back to the Communication Manager. This is accomplished by performing action "4:Transfer Data". The XML Controller is then initiated by the JWSI using action "5:Starts". After that, action "6:Received Data" is executed by the Controller to get the results received from the server. Subsequently, action "7:Load XSLT" is performed to load the XSLT file from the XSLT Repository. This is important since the returned data will be transformed based on that XSLT being compatible with the client's application.

Finally, after the transformation is performed, action "8:Requested Data" is executed to pass the transformed data to the client application for further processing. By completing the above steps, the dynamic Peer-to-Peer application integration is achieved.

The Collaboration diagrams give a better idea of how these objects collaborate between each other in a uniform way.

4.3.3.4 Server/Producer to Peer Collaboration Diagram

The *Server/Producer-to-Peer* Collaboration Diagram shows how the objects interact between each other to perform a specific task between the Server/Producer and the Client Peer. This is illustrated in figure 4.21.

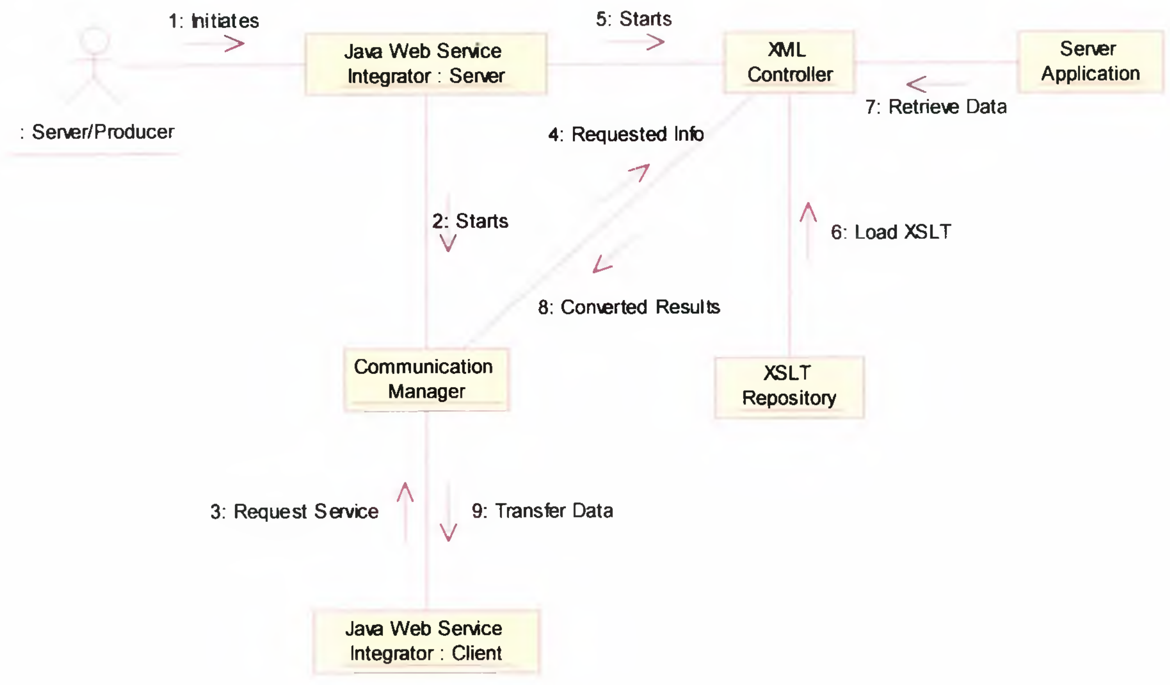


Figure 4.21 *The Server/Producer to Peer Collaboration Diagram*

As can be seen from the diagram in figure 4.21, the Server/Producer on the JWSI performs the action "1:Initiates" firstly. This action has already been performed by the Server/Producer in figure 4.18. It is also demonstrated here to better explain the whole scenario. The second action "2:Starts" is performed by the JWSI on the *Communication Manager* object after a Client Peer requested to invoke the service on the server site.

The Communication Manager object then executes action "3:Request Service" to actually retrieve the specification of the requested service. The requested service with the actually requested info is passed to the XML Controller by executing action "4:Requested Info". At the same time action "5:Starts" is performed to load the XML Controller. The XML controller executes the service and retrieves the data from the Server Application by using action "7:Reteive Data". Before that, action "6:Load XSLT" is performed to load the selected XSLT file associated with that server. This is important in order to convert the data retrieved from the server application to a format compatible to the specification of the requested service. These converted results are returned back to the Communication Manager object via action "8:Conveted Results". Finally, the Communication Manager object transfers the transformed results back to the requested Client using action "9:Transfer Data". By completing the above steps, the dynamic Peer-to-Peer application integration is achieved.

4.4 LOGICAL VIEW

The *Logical View* is mainly used to address the functional requirements of the system. The Logical View focuses on how the system will implement the behaviour in the Use Cases. Moreover, it includes the main class diagram that contains classes and their logical relationships.

4.4.1 Class Diagram

Class Diagrams describe the static structure of a system, or how it is structured rather than how it behaves. They also show the attributes and operations of a class and the constraints that apply to the way objects are connected. Figure 4.22 illustrates the overall class diagram of the system.

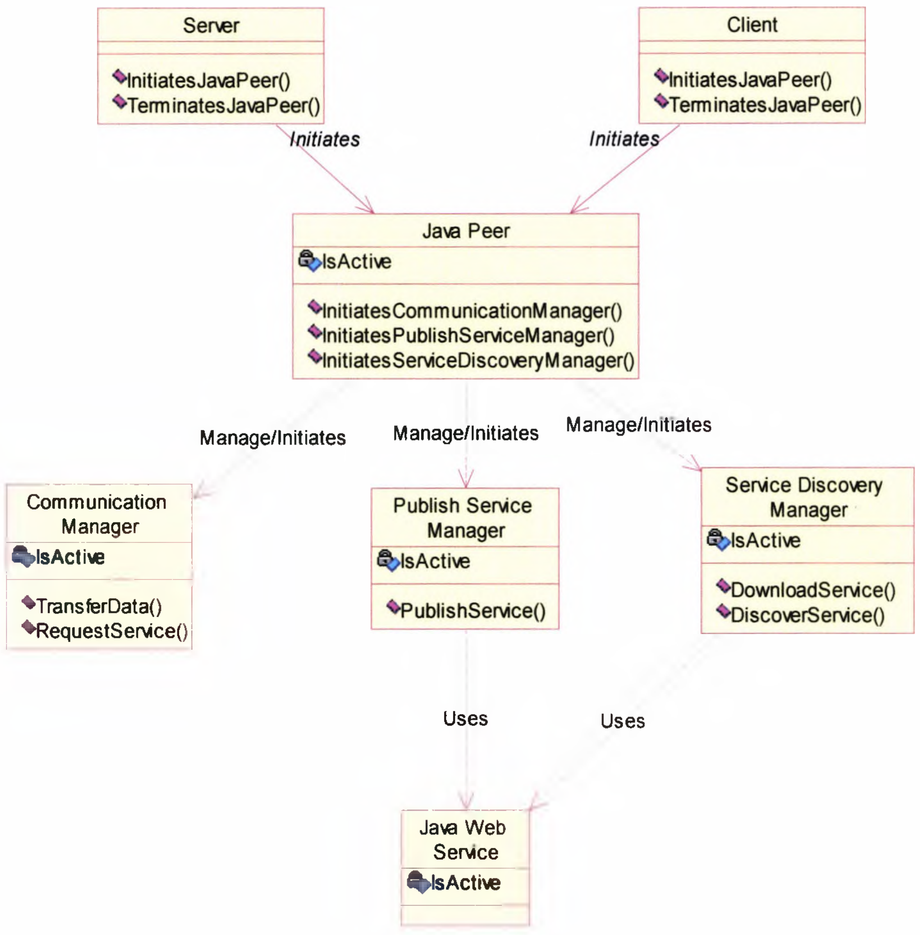


Figure 4.22 *Overall Class Diagram of the System*

As it can be seen from figure 4.22, a *user* on the server machine or a user on the client may initiate the Java Web Service Integrator for the purpose of integrating the two applications. Each user may initiate only one instance of the JWSI, since there is no essential benefit from initiating more than one instance of the JWSI application. Furthermore, once the JWSI is initiated, it can execute three main classes in order to achieve the dynamic integration. These three classes can only have one instance running at any time in order to avoid inter-class conflicts. The Publish Service Manager class may use the Java Web Service class for the purpose of publishing the service on the UDDI. Similarly, the Service Discovery Manager class may discover and download the service from the UDDI in order for the two peers to start negotiation and achieve the dynamic integration of the two applications.

The idea behind the "*IsActive*" attribute is to indicate if a particular instance of the class is enable/active or not. For example, if a Java Service is published on the UDDI and the owner of this service does not wish any more to have the Java Service listed there, then this service can be disabled in order not to be used by other user or applications. This is the case when the service can no longer serve the purpose of its creation.

4.5 CONCLUDING REMARKS

The design phase in a software development project bridges the gap between what is required and what is eventually implemented. During this process, the requirements are analysed and a course of actions is clearly specified. The output of the design process serves as a guideline for the implementation of the actual peer-to-peer dynamic platform, which follows next. The accuracy of the models derived from the design is a critical aspect of the system's success in meeting the initial requirements.

The design phase approach used for the purposes of designing the dynamic architecture and its corresponding outputs were explained in this chapter. Among these outputs, the Sequence and Collaborations diagrams were described, which will act as input to the next phase. Finally, the overall Class diagram of the system was presented, which will act as a blueprint for the implementation phase.

Chapter 5

SYSTEM IMPLEMENTATION

5.1 INTRODUCTION

Using the results derived from the design phase as blueprints, the implementation phase aims to realise the requirements specified in Chapter 2. This chapter describes the Dynamic Peer-to-Peer platform developed using this process.

The Web Service implementation protocols are initially discussed as they appear in the system architecture, alongside some examples of their corresponding source codes.

A description of the implemented system follows next. This is accomplished by a demonstration of how the software components react to specific tasks from the various users. The demonstration includes the various functionalities that both the client/consumer and server/producer peer components incorporate to meet the user's needs, as well as the flow of actions performed for the completion of a task.

The software that has been implemented is described next along with the relevant benefits. The chapter ends by presenting an overview of the implementation models of the Peer-to-Peer platform.

5.2 WEB SERVICE PROTOCOLS

The following section describes in detail the SOAP (Simple Object Access Protocol) implementation and how, in collaboration with the built-in software Classes of the Java language, the appropriate modules have been implemented for dynamically manipulating a UDDI (Universal Description, Discovery, and Integration) registry. This is supported by providing some examples of source codes actually written for this purpose.

5.2.1 SOAP IMPLEMANTATION

This section describes the SOAP Implementation, which has been implemented in this thesis in order to achieve the requirements specified in Chapter 2.

In the proposed platform, the Java API (Application Program Interface) for XML Messaging (JAXM) has been used, which makes it possible to implement Java Components based on XML messaging via the Java platform. This is achieved by making method calls using the JAXM API, in which the XML messages can be created and sent over the Internet. This section also describes how the JAXM API has been used to provide SOAP functionality to the proposed Peer-to-Peer platform.

The JAXM API conforms to the SOAP v1.1 specification and the SOAP with Attachments specification. The complete JAXM API is presented in the following package:

- ***Javax.xml.soap***: The package defined in the SOAP with Attachments API for Java specification. This is the basic package for SOAP messaging, which contains the API for creating and populating a SOAP message. This package has the entire API necessary for sending request-response messages.

All SOAP messages are sent and received over a connection. The connection can go directly to a *"particular destination"* or to a *"messaging provider"*. A messaging provider is a service that handles the transmission and routing of messages and provides features not available when a connection goes directly to its ultimate destination. Only a connection to a *"particular destination"* is applicable in this thesis and not to a *messaging provider* because of the architecture of the proposed platform.

The JAXM API supplies the following class and interface to represent the above connection:

- ***Javax.xml.soap.SOAPConnection****:* A connection from the sender directly to the receiver (a Peer-to-Peer connection)

A *SOAPConnection* object, which represents a Peer-to-Peer connection, is simple to create and use since it does not required any configuration and also because it does not need to be run in a Java Servlet container or in a J2EE (Java 2 Enterprise Edition) container. *Java Servlet* technology provides Web developers with a simple, consistent mechanism for extending the functionality of a Web server and for accessing existing business systems. A Servlet can almost be thought of as an applet that runs on the server side but without an interface [124].

Figure 5.1 illustrates a code fragment that creates a *SOAPConnection* object and after creating and populating the message, it uses the connection to send the message. The parameter *request* is the message being sent; *endpoint* represents where it is being sent.

```
SOAPConnectionFactory factory = SOAPConnectionFactory.newInstance();

SOAPConnection con = factory.createConnection();

. . .// create a request message and give it content

SOAPMessage response = con.call(request, endpoint);
```

Figure 5.1 An Example of a SOAP Connection in Java

When a *SOAPConnection* object is used, the only way to send a message is through the method *"call"*, which transmits its message and then blocks until it receives a reply.

Because the method "*call*" requires that a response be returned to it, this type of messaging is referred to as *request-response* messaging.

A Web Service implemented for *request-response* messaging must return a response to any message it receives. When the message is an update, the response is an acknowledgement that the update was received. Such an acknowledgement implies that the update was successful. Some messages may not require any response at all. The Web Service that gets such a message is still required to send back a response, because one is needed to unblock the "*call*" method. In this case, the response is not related to the content of the message; it is simply a message to unblock the call method. For example, in the context of this thesis, a client application may execute the Web Service for the purpose of retrieving some data. If the data that will be returned by the Server application takes several hours to be prepared, then a message is sent back to client indicating that the message is being received and that the data will be returned back to the client application as soon as they are ready.

Figure 5.2 illustrates, as an example, the steps that are required to create a SOAP message using the JAXP API, as well as how to send it to the specified Peer for execution.

```
import javax.xml.soap.*;
import java.util.*;
import java.net.URL;

public class Request {
 public static void main(String[] args) {
   try {
          SOAPConnectionFactory scFactory = SOAPConnectionFactory.newInstance();
          SOAPConnection con = scFactory.createConnection();
          MessageFactory factory = MessageFactory.newInstance();
          SOAPMessage message = factory.createMessage();
          SOAPPart soapPart = message.getSOAPPart();
          SOAPEnvelope envelope = soapPart.getEnvelope();
          SOAPHeader header = envelope.getHeader();
          SOAPBody body = envelope.getBody();
          header.detachNode();
          Name   bodyName   =   envelope.createName(""GetServiceFile"",""JWSI"",
          ""217.35.115.230"");
          SOAPBodyElement gltp = body.addBodyElement(bodyName);
          Name name = envelope.createName("ServiceName");
          SOAPElement symbol = gltp.addChildElement(name);
          symbol.addTextNode(""Hull Generation"");
          java.net.URL endpoint = new URL(“217.5.155.230/JWSI”);
          SOAPMessage response = con.call(message, endpoint);
          con.close();
```

Figure 5.2 An Example of a SOAP Request using the JAXM API

As a continuation of the example shown in figure 5.2, figure 5.3 illustrates the steps that are required to receive a response from the Server Peer after a SOAP request has been initiated.

```
            // Read the SOAP Response
            SOAPPart sp = response.getSOAPPart();
            SOAPEnvelope se = sp.getEnvelope();
            SOAPBody sb = se.getBody();
            Iterator it = sb.getChildElements(bodyName);
            SOAPBodyElement bodyElement = (SOAPBodyElement)it.next();
            String Result = bodyElement.getValue();
            System.out.print("The output execution of the JWSI Service is ");
            System.out.println(Result);

    } catch (Exception ex) { ex.printStackTrace(); }
  }
}
```

Figure 5.3 An Example of a SOAP Response using the JAXM API

When the SOAP request is sent over the communication line, it is converted into an XML file format, the content of which is shown in figure 5.4. It is important to note here that the IP (Internet Protocol) Address illustrated in this example is taken from the specification of the Web Service when it is dynamically discovered under the UDDI directory. In case the IP Address was mistyped during the creation of the Web Service, or in case the Server hosting the Web Service changes its IP, then the Web Service will not be in a position to find the destination Server and hence the Web Service will fail to execute.

```
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  <SOAP-ENV:Body>
   <JWSI: GetServiceFile xmlns:m= "217.5.155.230">
    <ServiceName>Hull Generation</ServiceName>
   </JWSI: GetServiceFile >
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 5.4 An Example of the contents of a SOAP Request in Java

5.2.2 UDDI IMPLEMENTATION

Before the JWSI on the client site can send a SOAP Request to the server to get the required data, it first needs to dynamically discover the Web Service that is hosted on the UDDI directory. The following section presents some examples of how this can be achieved using the Java language and the Java API for XML Registries (JAXR) and how the server JWSI can publish Java Web Services on the UDDI.

As already explained in Chapter 3, a Universal Description, Discovery, and Integration (UDDI) is an XML registry infrastructure that enables the building, deployment, and discovery of Web services. It is a neutral third party that facilitates dynamic and loosely coupled business-to-business (B2B) interactions. A registry is available to organisations as a shared resource, often in the form of a Web-based service.

Currently, there is a variety of specifications for XML registries. These include:

- The ebXML Registry and Repository standard, which is sponsored by the Organisation for the Advancement of Structured Information Standards (OASIS) and the United Nations Centre for the Facilitation of Procedures and Practices in Administration, Commerce and Transport (U.N./CEFACT)
- The Universal Description, Discovery, and Integration (UDDI) registry, which is being developed by a vendor consortium.

A *registry provider* is an implementation of a business registry that conforms to a specification for XML registries.

For the purpose of this thesis, the UDDI registry has been used along with the JAXR (Java API for XML Registries) package. JAXR enables Java software programmers to use a single, easy-to-use abstraction API to access a variety of XML registries. A unified JAXR information model describes content and metadata within XML registries [123]. Figure 5.5 illustrates the JAXR Architecture.

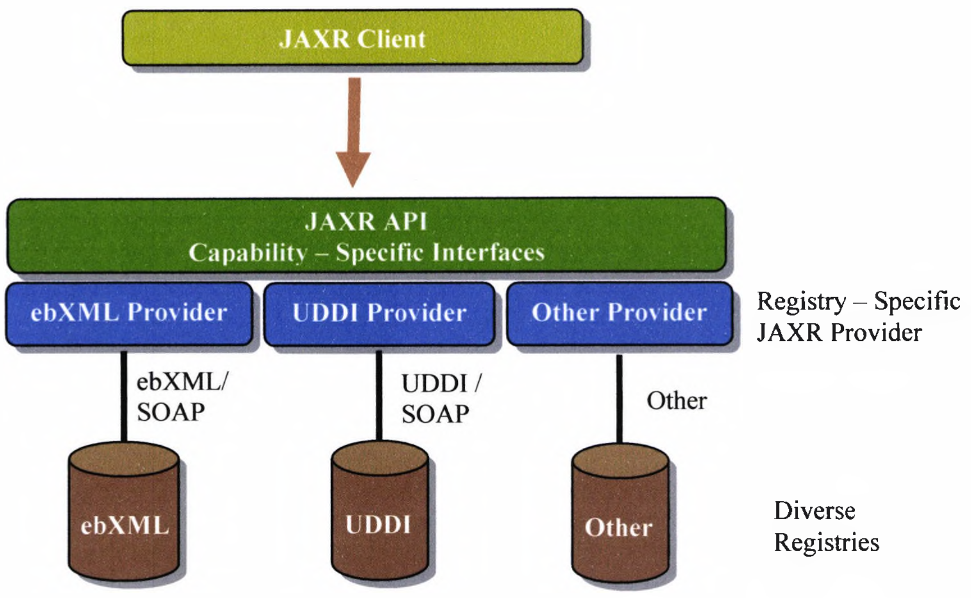


Figure 5.5 The JAXR(Java API for XML Registries) Architecture

In order for the server application to publish a Java Web Service on the UDDI or the client application to discover a Web Service, firstly the appropriate permissions must be obtained from the registry to access it. Following, is a list of public UDDI registries that the users of each application can apply to gain access to registry, hence publish and dynamically discover Web Services:

- **Microsoft :** http://uddi.microsoft.com
- **IBM :** http://uddi.ibm.com/testregistry/registry.html
- **SAP:** http://udditest.sap.com

After the users of the applications that need to be integrated have registered with one of the registry providers listed above, they can query the registries for publishing or discovering of Web Service.

One way for a client JWSI to use a registry is to query it for information about the organisations that have submitted data to it. The "*BusinessQueryManager*" interface supports a number of find methods that allow search for data using the JAXR information model. Many of these methods return a "*BulkResponse*" (a collection of objects) that meets a set of criteria specified in the method arguments. The most useful of these methods are:

- **FindOrganisations :** Returns a list of organisations that meet the specified criteria, often a name pattern or a classification within a classification scheme.
- **FindServices :** returns a set of Web Services offered by a specified organisation.
- **FindServiceBindings :** Returns the service bindings (information about how to access the service) that are supported by a specified service.

If a client JWSI has the appropriate authorisation, then it can submit data to a registry, modify it and remove it. It uses the "*BusinessLifeCycleManager*" interface to perform these tasks. Registries usually allow a client to modify or remove data only if the data is being modified or removed by the same user who first submitted the data.
Before it can submit data, the client JWSI must send its user name and password to the registry in a set of credentials. Figure 5.6 illustrates an example of a Java source code of how to send these credentials and how to create a registry entry and publish the details of the organisation prior to publishing the Java Web Services.

```
String username = "myUserName";
String password = "myPassword";

// Get authorization from the registry
PasswordAuthentication passwdAuth = new PasswordAuthentication(username,
password.toCharArray());

Set creds = new HashSet();
creds.add(passwdAuth);
connection.setCredentials(creds);
// Create organization name and description
Organization org = blcm.createOrganization("Shipping Services Ltd");
InternationalString s = blcm.createInternationalString("Provides web services for the
shipping sector.");
org.setDescription(s);

// Create primary contact, set name
User primaryContact = blcm.createUser();
PersonName pName = blcm.createPersonName("Stelios Christofi");
primaryContact.setPersonName(pName);

// Set primary contact phone number
TelephoneNumber tNum = blcm.createTelephoneNumber();
tNum.setNumber0044-0000-0000");
Collection phoneNums = new ArrayList();
phoneNums.add(tNum);
primaryContact.setTelephoneNumbers(phoneNums);
```

```
.. Continued
// Set primary contact email address
EmailAddress emailAddress =  blcm.createEmailAddress("something@mail.com");
Collection emailAddresses = new ArrayList();
emailAddresses.add(emailAddress);
primaryContact.setEmailAddresses(emailAddresses);

// Set primary contact for organization
org.setPrimaryContact(primaryContact);
// Set classification scheme to NAICS
ClassificationScheme cScheme = bqm.findClassificationSchemeByName(null,
"UDDITYPE");
// Create and add classification
Classification classification =  blcm.createClassification(cScheme, " Java Web Services",
"732213");
Collection classifications = new ArrayList();
classifications.add(classification);
org.addClassifications(classifications);
```

Figure 5.6 An Example of a adding an Organisation to the UDDI Registry

After the server JWSI successfully adds the organisation details to the UDDI registry, the Java Web Services can now be added. Figure 5.7 shows a Java code fragment on how to create a collection of services, add service bindings to a service and then add the services to the organisation.

```
// Create services and service
Collection services = new ArrayList();
Service service = blcm.createService("Hull Generation");
InternationalString is =  blcm.createInternationalString("The purpose of this service is
to generate the hull of the ship");
service.setDescription(is);

// Create service bindings
Collection serviceBindings = new ArrayList();
ServiceBinding binding = blcm.createServiceBinding();
is = blcm.createInternationalString("Hull Generation Binding Description");
binding.setDescription(is);

// allow us to publish the service URL without an error
binding.setValidateURI(false);
binding.setAccessURI("http://217.35.115.230:5151/JWSI/");
serviceBindings.add(binding);

// Add service bindings to service
service.addServiceBindings(serviceBindings);

// Add service to services, then add services to organisation
services.add(service);
org.addServices(services);
```

Figure 5.7 An Example of a adding a Web Service to the UDDI Registry

By completing the above steps, the Java Web Service is published on the UDDI registry and waiting to be dynamically discovered. Figure 5.8 shows one way to dynamically discover the Java Web Service for the purpose of dynamic Application Integration.

```
RegistryService rs = null;
BusinessQueryManager bqm = null;

// Get registry service and query manager
rs = connection.getRegistryService();
bqm = rs.getBusinessQueryManager();

// Define find qualifiers and name patterns
Collection findQualifiers = new ArrayList();
findQualifiers.add(FindQualifier.SORT_BY_NAME_DESC);
Collection namePatterns = new ArrayList();
namePatterns.add("%" + qString + "%");

// Find using the name
BulkResponse response = bqm.findOrganizations(findQualifiers, namePatterns, null, null, null, null);
Collection orgs = response.getCollection();

// Display information about the organisations found
Iterator orgIter = orgs.iterator();
if (!(orgIter.hasNext())) {
  System.out.println("No organisations found");
} else while (orgIter.hasNext()) {
  Organization org = (Organization) orgIter.next();
  System.out.println("Org name: " + getName(org));
  System.out.println("Org description: " +  getDescription(org));
  System.out.println("Org key id: " + getKey(org));

  // Display primary contact information
  User pc = org.getPrimaryContact();
  if (pc != null) {
    PersonName pcName = pc.getPersonName();
    System.out.println(" Contact name: " +  pcName.getFullName());
    Collection phNums =  pc.getTelephoneNumbers(null);
    Iterator phIter = phNums.iterator();
    while (phIter.hasNext()) { TelephoneNumber num =   (TelephoneNumber) phIter.next();
                               System.out.println("  Phone number: " +  num.getNumber());
    }
    Collection eAddrs = pc.getEmailAddresses();
    Iterator eaIter = eAddrs.iterator();
    while (eaIter.hasNext()) {
 EmailAddress eAd =  (EmailAddress) eaIter.next();
 System.out.println("  Email Address: " +  eAd.getAddress());
    }  }
  // Display service and binding information
  Collection services = org.getServices();
  Iterator svcIter = services.iterator();
  while (svcIter.hasNext()) {
    Service svc = (Service) svcIter.next();
    System.out.println(" Service name: " + getName(svc));
    System.out.println(" Service description: " +  getDescription(svc));
    Collection serviceBindings =     svc.getServiceBindings();
    Iterator sbIter = serviceBindings.iterator();
    while (sbIter.hasNext()) {
         ServiceBinding sb =  (ServiceBinding) sbIter.next();
         System.out.println("  Binding " + "Description: " +  getDescription(sb));
         System.out.println("  Access URI: " +    sb.getAccessURI());
    }
  }
}
```

Figure 5.8 An Example of a Web Service discovery on the UDDI Registry

5.3 PROTOTYPE EXECUTION

In this section, a typical scenario will be presented, demonstrating the functionality of the proposed system. Some screenshots of the system's user interface will be presented, together with the respective source code when appropriate, to better explain in detail the effort of the technical work involved behind the user friendly interface.

Scenario: A client application (Passenger Evacuation) requests to be interacted with a server application (Hull Generation Tool).

During the lifecycle of the design of the ship, a hypothetical Ship Designer (the client), wishes to test a Passenger Evacuation software from his company using Ship designs from other companies, in order to generate some Evacuation statistics to measure the performance of the software. The above software requires a Hull Generator Tool. This Tool creates basic Hull designs for ships in order for other software to get these Hulls and tests them prior to designing the final Hull form of a ship. In order to accomplish this task, the Ship Designer configures the client JWSI based on the input and output information of the above Passenger Evacuation software.

Then the JWSI contacts the appropriate UDDI to discover existing Java Web Services that offer this kind of service. When a Java Web Service is identified that closely matches the criteria of the client software, the client JWSI and the server JWSI establish between each other a Peer-to-Peer connection, in order to integrate the two ship applications.

The whole integration process is done dynamically without the intervention of the client or the server user. Once the Hull design data is downloaded to the client site, the Passenger Evacuation software runs automatically so as to generate the appropriate Evacuation statistics. By completing the above process, the two applications are integrated between them by only downloading or uploading the minimal necessary data, without the physical transfer of the whole software and its installation on the local client.

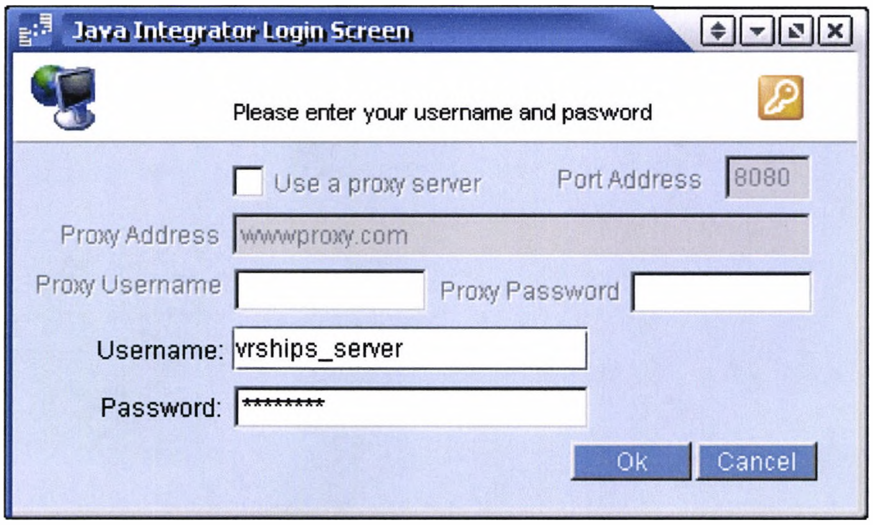
The following sections demonstrate in detail how the above scenario is achieved.

5.3.1 Server/Producer Execution

Taking into consideration the scenario described in section 5.3, this section demonstrates the functionality of the Java Web Service Integrator as well as the actions required to be performed by the user on the Server site. These actions consist of configuring the server application for publishing the newly created Java Web Service on the UDDI for the purpose of dynamic integration.

Before the integration process takes place, the user of the application server needs to create the Java Web Service. This is achieved by using the "Java Web Service Wizard" of the JWSI.

After running the JWSI on the server site, a window will appear requesting login information to be filled as shown in figure 5.9.


Java Integrator Login Screen

Please enter your username and pasword

☐ Use a proxy server

Port Address: 8080

Proxy Address: wwwproxy.com

Proxy Username:

Proxy Password:

Username: vrships_server

Password: ********

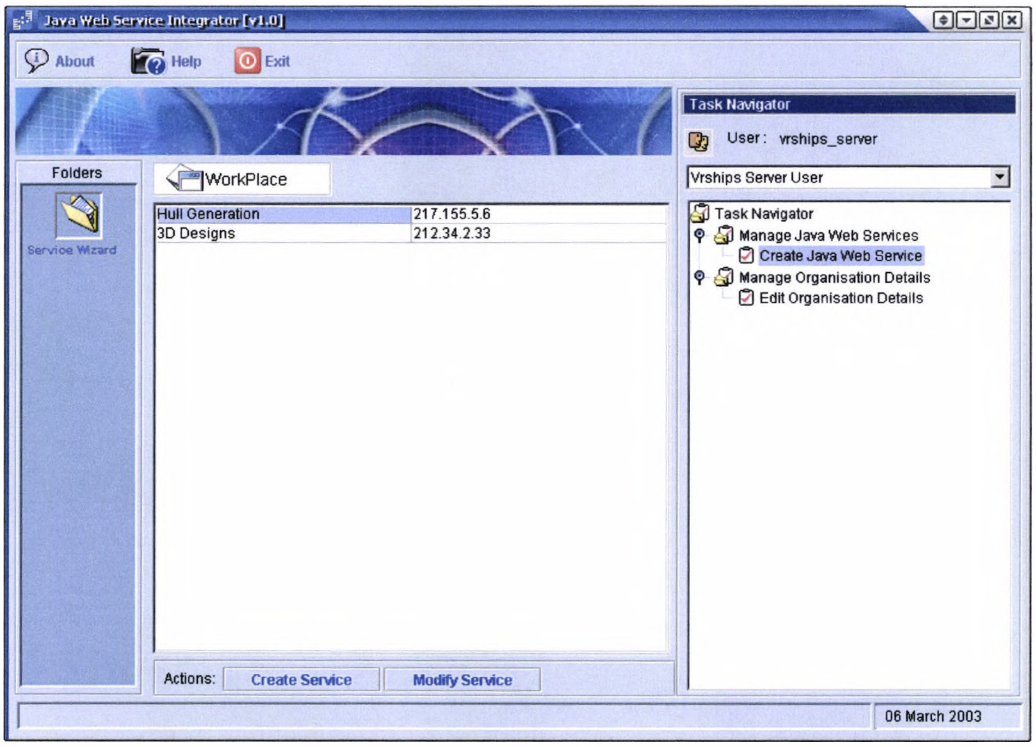
Figure 5.9 *The Java Web Service Integrator login screen – Server site*

The first five fields, i.e. proxy information, whose letters are illustrated by the light grey colour, indicate optional parameters. This means that some clients may require specifying a proxy server in order to have access to the Internet. These optional parameters are discussed in detail in section 6.3.5.

The following two fields, whose letters are illustrated by the black colour, indicate compulsory parameters. By providing a username and a password to the application, the

JWSI authenticates and connects to the local Java database server. It is assumed that the user of the application has already registered with the system and has received a username and a password from the system administrator.

After the user has successfully been granted access by the Java database server, the JWSI loads the main application window as shown in figure 5.10.



Hull Generation	217.155.5.6
3D Designs	212.34.2.33

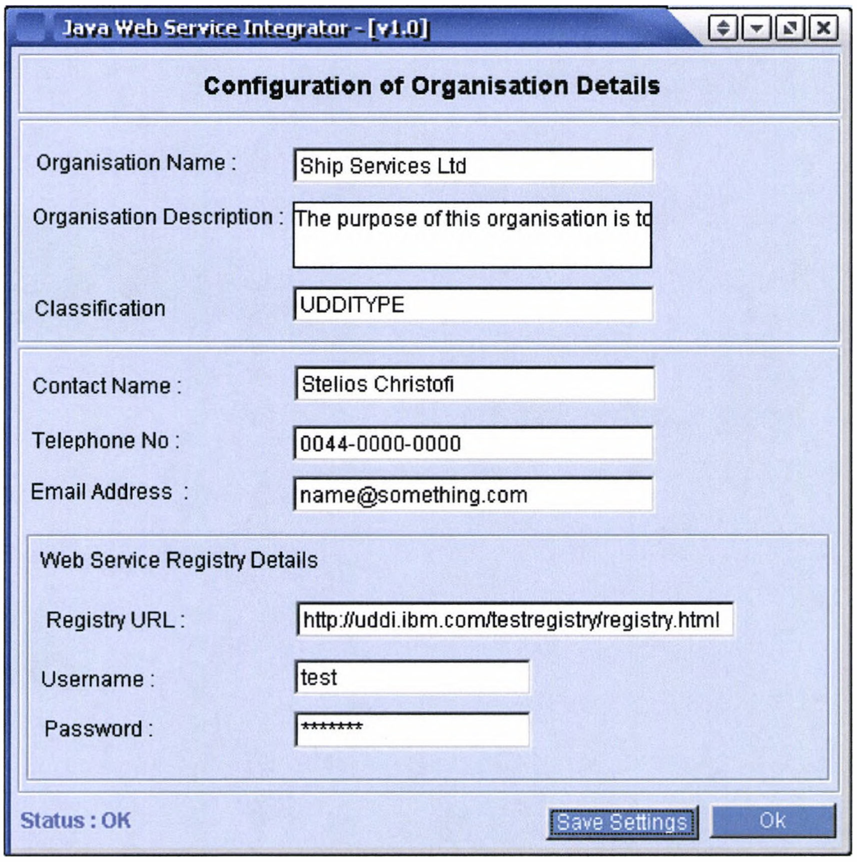
Figure 5.10 *Main window of the Java Web Service Integrator - Server site*

The above user interface takes into consideration issues like ease of use, ease to add dynamically services and ease to navigate with the help of a task-oriented paradigm. This paradigm helps the user to access business documents according to the user's task and role. So, usability is achieved by a close mapping between the system and the user conceptual model.

On the right hand side of this user interface, a task navigator menu appears. This menu corresponds to a list of tasks that the current logged-in user has been granted access by the Java database server. This means that different users may have different tasks under this navigation menu, according to their roles. By selecting a task, for this scenario the

"*Manage Organisation Details*" task, the application shows a list of associated sub tasks just below the selected task. In this case, there is only one sub task called "*Edit Organisation Details*". By selecting that sub task, the server application shows a list of associated folders on the left hand side of the application window. These folders help to divide the sub task into more meaningful distinct categories. For this scenario, the folder "Organisation Details" appears. By selecting that folder, a description of the business document appears in the middle of the main application window and simultaneously a list of actions "*Modify Organisation Details*" associated with that business document appear at the bottom of the application window.

Pressing the "*Modify Organisation Details*" button located at the bottom of this window, a new window is displayed requesting certain field names to be entered, as shown in figure 5.11.


Java Web Service Integrator - [v1.0]

Configuration of Organisation Details

Organisation Name :	Ship Services Ltd
Organisation Description :	The purpose of this organisation is tc
Classification	UDDITYPE
Contact Name :	Stelios Christofi
Telephone No :	0044-0000-0000
Email Address :	name@something.com

Web Service Registry Details

Registry URL :	http://uddi.ibm.com/testregistry/registry.html
Username :	test
Password :	*******

Status : OK

Figure 5.11 *Configuration of Organisation details of the Server JWSI*

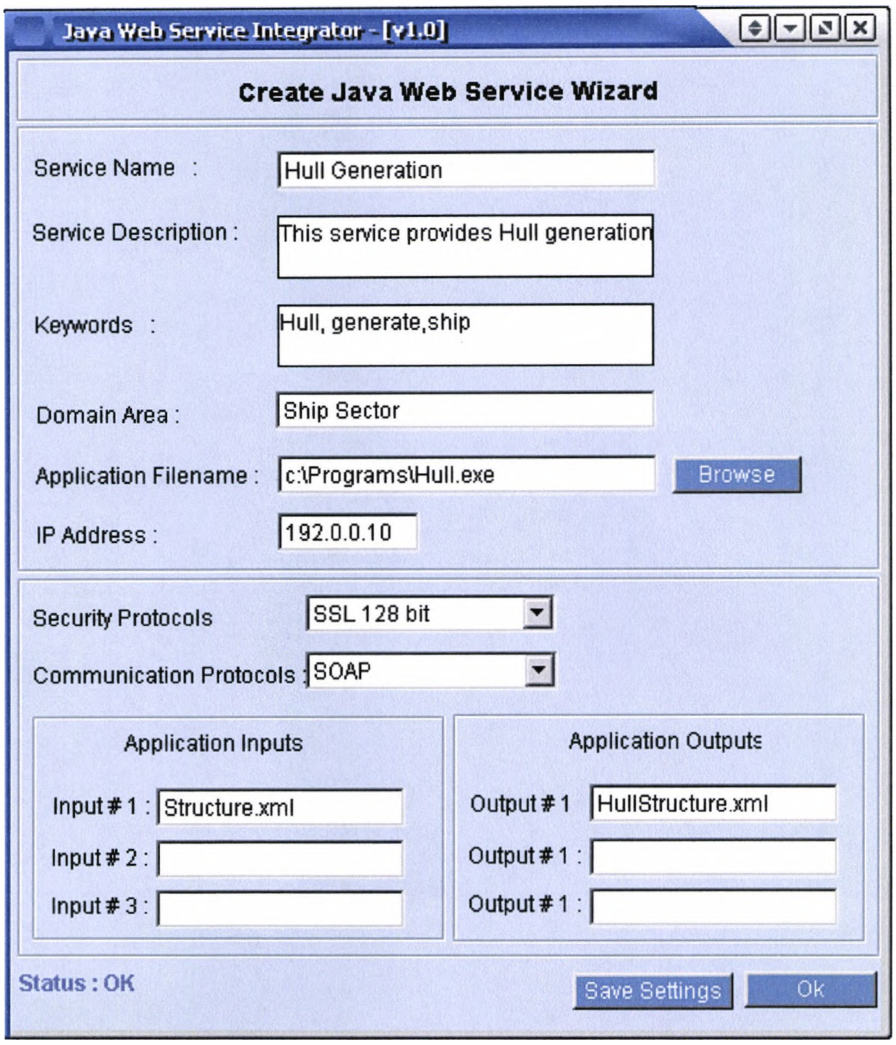
The user has to fill the organisation details form before any Java Web Services are published on the UDDI directory.

This organisation form includes field names, like the organisation name, organisation description and the classification, which will help during the discovery of the web service. Moreover, it includes the contact details of the company in charge of the web service, as well as the URL that the web services of the organisation will be registered. Finally, the username and password are very essential fields that need to be entered since any attempt to access the UDDI registry, requires authentication. Once all above field names have been entered, the user must save the organisation settings in the Java Database Server by pressing the "*Save Settings*" button.

After completing the previous task, the user may now continue to the next task for creating and publishing the Web Service. By selecting the task called "*Manage Java Web Services*", the application shows a list of associated sub tasks just below the selected task. In this case, there is only one sub task called "*Create Java Web Service*". By selecting that sub task, the server application shows the folder called "*Service Wizard*".

By selecting that folder, a description of the business document appears in the middle of the main application window and at the same time, a list of actions "*Create Service, Modify Service*" associated with that business document appear at the bottom of the application window.

By pressing the "*Create Service*" button located at the bottom of this window, a new window is displayed requesting certain field names for the web service wizard to be entered as shown in figure 5.12.


Java Web Service Integrator - [v1.0]

Create Java Web Service Wizard

Service Name : Hull Generation

Service Description : This service provides Hull generation

Keywords : Hull, generate,ship

Domain Area : Ship Sector

Application Filename : c:\Programs\Hull.exe

IP Address : 192.0.0.10

Security Protocols : SSL 128 bit

Communication Protocols : SOAP

Application Inputs

Input # 1 : Structure.xml

Input # 2 :

Input # 3 :

Application Outputs

Output # 1 : HullStructure.xml

Output # 1 :

Output # 1 :

Status : OK

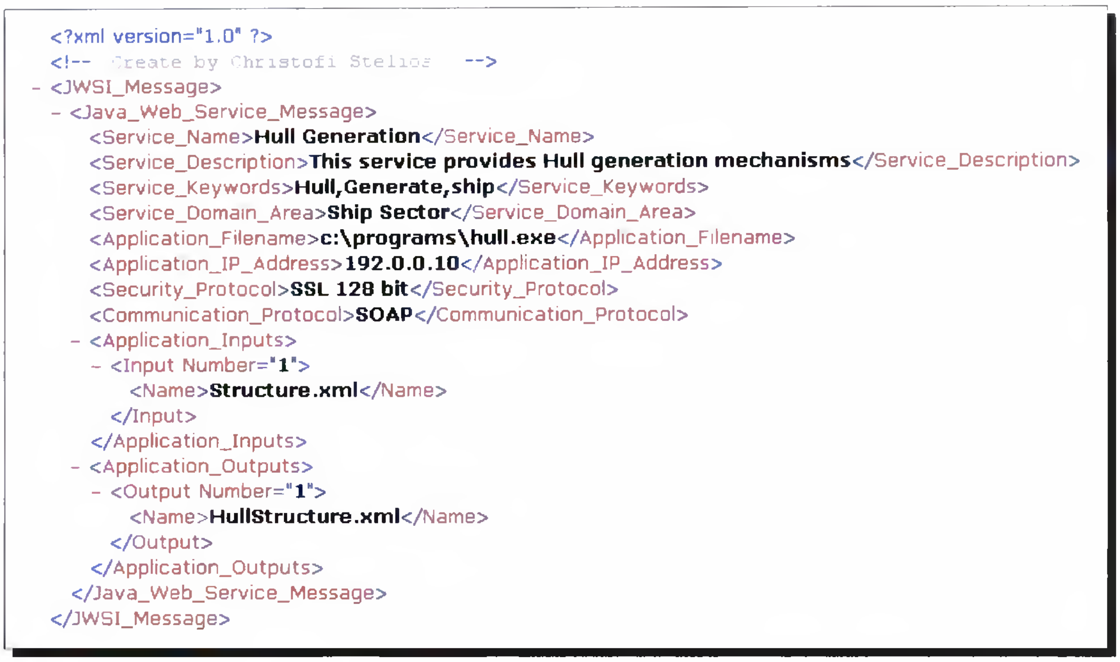
Figure 5.12 *Create Java Web Service Wizard of the JWSI*

In this wizard, the user needs to follow certain steps before the service is created and published on the UDDI registry. Some main parameters that have to be specified for this service are

- service name,
- service description,
- service keywords,
- the domain area of the service,
- the application that corresponds to this service,
- the IP address of the client running the application,
- available security and communication protocols, and finally,
- the input and output parameters for the application.

After these parameters have been specified, and after the user presses the "*Save Settings*", the *Publish Service Manager* component publishes the service on the UDDI registry. Figure 5.13 illustrates the specification of the Java Web Service that is dynamically created and published on the UDDI, in XML format.

Similarly to the above task, the user may edit an existing Web Service if he/she wishes to alter some properties of the Web Service on the UDDI.

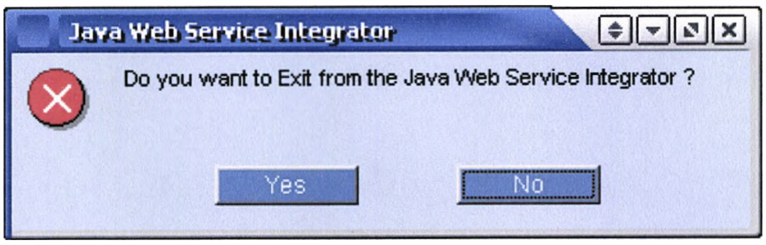


```
  <?xml version="1.0" ?>
  <!--  Create by Christofi Stelios   -->
- <JWSI_Message>
  - <Java_Web_Service_Message>
      <Service_Name>Hull Generation</Service_Name>
      <Service_Description>This service provides Hull generation mechanisms</Service_Description>
      <Service_Keywords>Hull,Generate,ship</Service_Keywords>
      <Service_Domain_Area>Ship Sector</Service_Domain_Area>
      <Application_Filename>c:\programs\hull.exe</Application_Filename>
      <Application_IP_Address>192.0.0.10</Application_IP_Address>
      <Security_Protocol>SSL 128 bit</Security_Protocol>
      <Communication_Protocol>SOAP</Communication_Protocol>
    - <Application_Inputs>
      - <Input Number="1">
          <Name>Structure.xml</Name>
        </Input>
      </Application_Inputs>
    - <Application_Outputs>
      - <Output Number="1">
          <Name>HullStructure.xml</Name>
        </Output>
      </Application_Outputs>
    </Java_Web_Service_Message>
  </JWSI_Message>
```

Figure 5.13 *XML specification of the Java Web Service*

Following the above steps, the integration process can now be initiated. Before the user of the client application initiates the *Service Discovery Manager*, the JWSI needs to be configured. The configuration process is essential, since the JWSI must know the inputs and outputs of the local application needed to be integrated. This is important in order for the JWSI to know which dynamic Java Web Services are needed to be downloaded from the UDDI. This is accomplished by comparing the description, keywords and domain area of the service on the UDDI with the description of the local application. Not all Java Web Services found on the UDDI are downloaded since this is unnecessary and inefficient.

Finally, when the "Exit" button on the Main window of the JWSI is clicked, the user is prompted to select the "YES" button to close the JWSI server application. This is illustrated in figure 5.14.



Do you want to Exit from the Java Web Service Integrator ?

Figure 5.14 *Exit window of the Java Web Service Integrator*

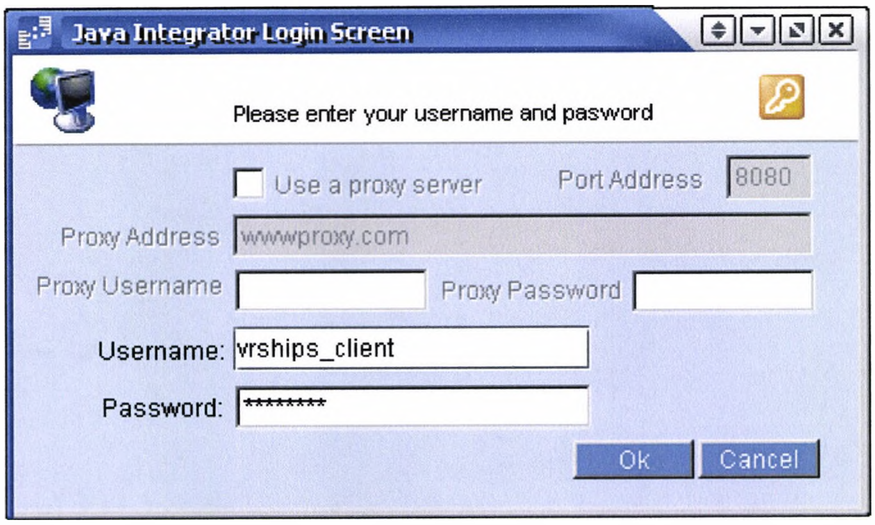
Once the JWSI server application is closed, then any clients wishing to dynamically integrate their application with the closed JWSI server application will be denied access, since the Java Web Service will not be in a position to process the incoming request with the JWSI application closed.

The following section describes in detail how the user of the client application can configure the client application and initiate the *Service Discovery Manager.*

5.3.2 Client/Consumer Execution

Taking into consideration the scenario described in section 5.3, this section demonstrates the functionality of the Java Web Service Integrator, as well as the actions required to be performed by the user to configure the client application so as to initiate the *Service Discovery Manager* for dynamic integration.

After running the JWSI on the client site, the user must enter a username and password to log into the JWSI as illustrated in figure 5.15.


Java Integrator Login Screen

Please enter your username and pasword

Use a proxy server Port Address 8080

Proxy Address wwwproxy.com

Proxy Username Proxy Password

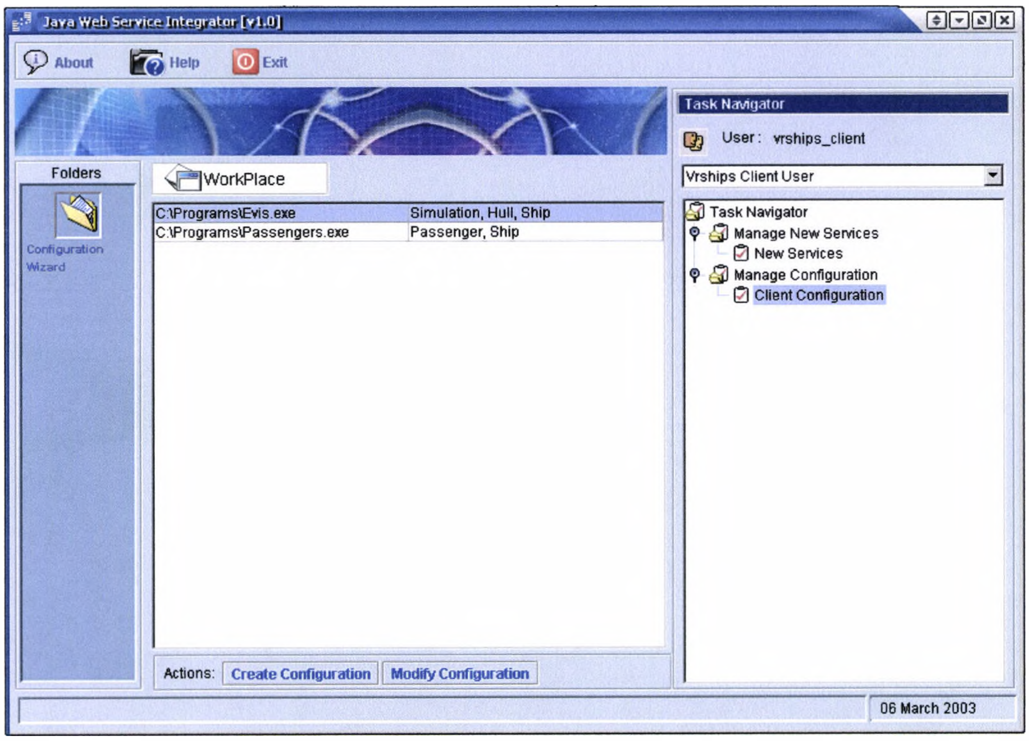
Username: vrships_client

Password: ********

Figure 5.15 *The Java Web Service Integrator login screen – Client site*

The login screen of the JWSI on the client site is the same as the login screen of the JWSI on the server site. The only difference is that both the client and the server users need to enter different usernames and passwords when using the JWSI in order to load different tasks for their needs. For a description of the fields that need to be entered in this screen, see section 5.3.1, figure 5.9.

After the user has successfully been granted access by the Java Database server, the JWSI loads the main application window as shown in figure 5.16.


Java Web Service Integrator [v1.0]

C:\Programs\Evis.exe	Simulation, Hull, Ship
C:\Programs\Passengers.exe	Passenger, Ship

Actions: Create Configuration Modify Configuration

Task Navigator

User: vrships_client

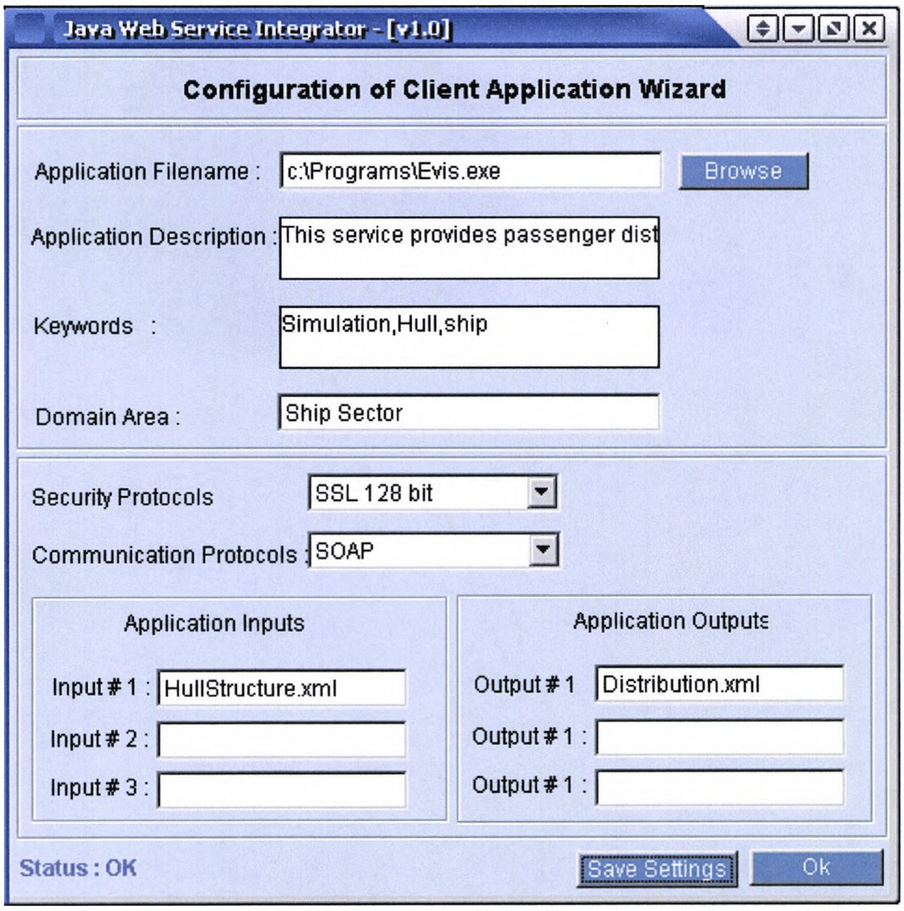
Vrships Client User

Task Navigator
- Manage New Services
  - New Services
- Manage Configuration
  - Client Configuration

06 March 2003

Figure 5.16 *Main window of the Java Web Service Integrator - Client site*

On the right hand side of this user interface, a task navigator menu appears similar to the one loaded on the server site of the JWSI as already discussed in section 5.3.1. This menu corresponds to a list of tasks that the client user has been granted access by the Java Database server. By selecting a task, for example, the "*Manage Configuration*" task, the application shows a list of associated sub tasks just below the selected task. In this case, there is only one sub task called "*Client Configuration*". By selecting that sub task, the client application shows a list of associated folders on the left hand side of the application window. These folders help to divide the sub task into more distinct categories. For this scenario, the folder "Configuration *Wizard*" appears. By selecting this folder, a description of the business document appears in the middle of the main application window and at the same time, a list of actions "*Create Configuration, Modify Configuration*" associated with that business document appear at the bottom of the application window.
By pressing the "*Create Configuration*" button located at the bottom of this window, a new window is displayed requesting certain field names to be entered, as shown in figure 5.17.


Java Web Service Integrator - [v1.0]

Configuration of Client Application Wizard

Application Filename : c:\Programs\Evis.exe

Application Description : This service provides passenger dist

Keywords : Simulation,Hull,ship

Domain Area : Ship Sector

Security Protocols SSL 128 bit

Communication Protocols : SOAP

Application Inputs

Input # 1 : HullStructure.xml

Input # 2 :

Input # 3 :

Application Outputs

Output # 1 Distribution.xml

Output # 1 :

Output # 1 :

Status : OK

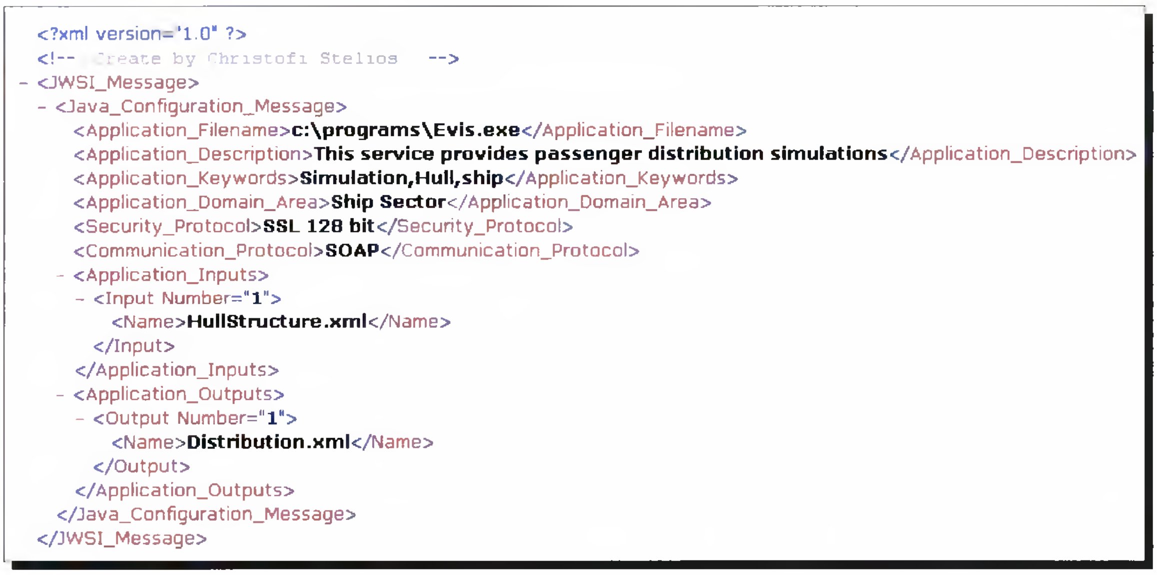
Figure 5.17 *Configuration of Client Application Wizard of the JWSI*

In this wizard, the user needs to follow certain steps before the configuration file is created and saved on the local Java Database server. Some main parameters that have to be specified in this configuration are

- application name,
- application description,
- application keywords,
- the domain area of the application,
- available security and communication protocols, and finally,
- the input and output parameters for the application.

The configuration process is essential, since the JWSI must know the inputs and outputs of the local application to be integrated. This is important in order for the JWSI to know which dynamic services are needed to be downloaded from the UDDI. This is accomplished by comparing the parameters of the service on the UDDI and the parameters of the configuration file of the local application. Not all Java Web Services found on the UDDI are downloaded, since this is unnecessary and inefficient.

After these parameters have been specified, and after the user presses the "*Save Configuration*", the Configuration file is created and stored in the local Java Database server. Figure 5.18 illustrates the specification of the Configuration file that will be used for dynamic discovery of Java Web Services on the UDDI, in XML format.



```
  <?xml version="1.0" ?>
  <!-- Create by Christofi Stelios -->
- <JWSI_Message>
  - <Java_Configuration_Message>
      <Application_Filename>c:\programs\Evis.exe</Application_Filename>
      <Application_Description>This service provides passenger distribution simulations</Application_Description>
      <Application_Keywords>Simulation,Hull,ship</Application_Keywords>
      <Application_Domain_Area>Ship Sector</Application_Domain_Area>
      <Security_Protocol>SSL 128 bit</Security_Protocol>
      <Communication_Protocol>SOAP</Communication_Protocol>
    - <Application_Inputs>
      - <Input Number="1">
          <Name>HullStructure.xml</Name>
        </Input>
      </Application_Inputs>
    - <Application_Outputs>
      - <Output Number="1">
          <Name>Distribution.xml</Name>
        </Output>
      </Application_Outputs>
    </Java_Configuration_Message>
  </JWSI_Message>
```

Figure 5.18 *XML specification of the Configuration File*

Similarly to the above task, the user may edit an existing Configuration for altering some properties of the Application that needs to be integrated.

After the above steps have been completed, the integration process may commence. The Discovery Service Manager is initiated after some time interval and starts searching the UDDI for any new Java Web Services that closely match the specification of the pre-configured client JWSI, from which the local application requested to be integrated.

The Discovery Service Manager uses an advanced search technique that has been implemented specifically for this thesis, to intelligently identify and download the most relevant Web Services from the UDDI. Besides the usual methods of comparing the keywords and descriptions of the applications with the keywords and descriptions of the services found on the UDDI, the search technique uses the inputs and outputs of each application to more specifically pinpoint the Web Service.

This is accomplished by identifying the file format of each input and output of the application and by comparing these, with the file format of the application that the Web Service is linked. This ensures that only Web Services that match the exact specification of the Client application will be used for integration. It will be ineffective to download a Web Service for integration even if the service description matches the description of the potential integrated application, in case the output file format of the service is completely different or cannot be in any way mapped to the input file format of the potential Client application.

By using this intelligent technique, efficiency of the whole integration process is achieved and waste of communication bandwidth is kept to the minimum.

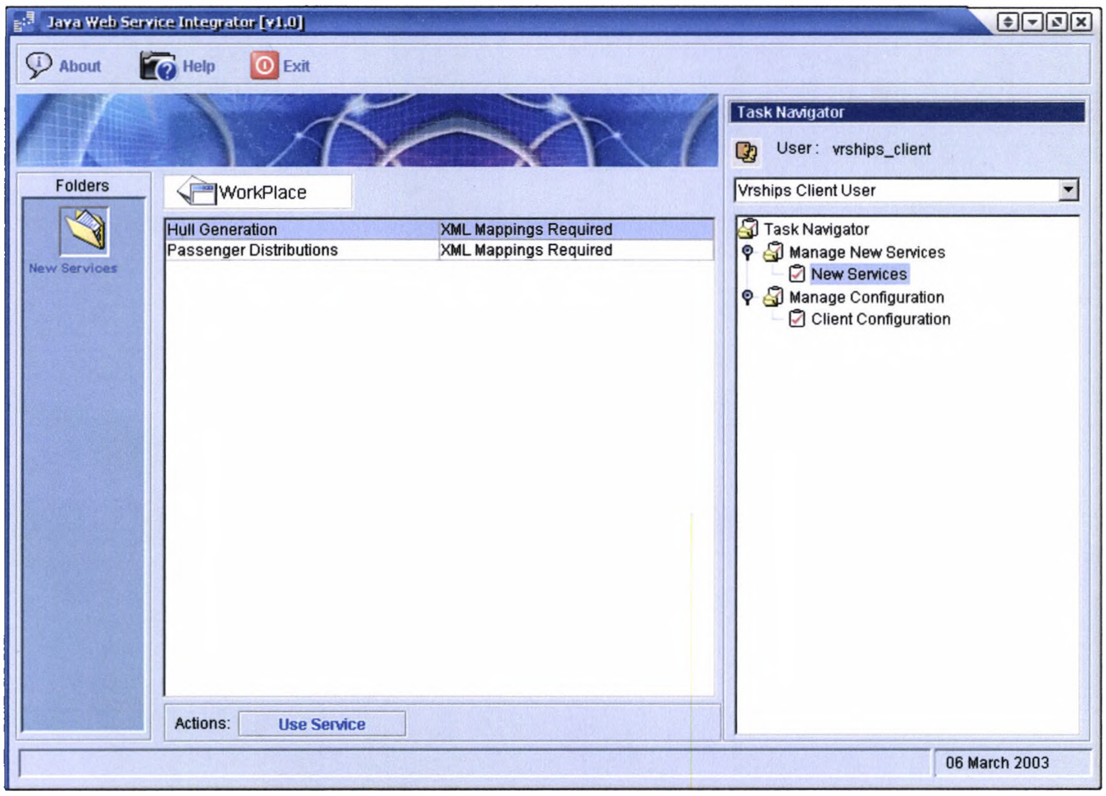
The above section demonstrated the steps required to be performed by the client using the JWSI, to configure the client application and initiate the *Service Discovery Manager* for dynamic integration. The following section describes in detail the steps involved when the actual integration takes place between the client and the server application.

5.3.3 Client/Consumer to Peer Execution

Taking into consideration the scenario described in section 5.3, this section demonstrates the steps involved during the integration process between the client and the server application and more specifically the tasks that are performed on the client site. As already stated in this thesis, the client or server application can act as peers in this integration process. Since the process of integrating the two applications can be very complicated, it has been decided to separate the whole integration process into two smaller processes. The first one will be referred to as the *"Client/Consumer to Peer Integration"* process and the second as the *"Server/Producer to Peer Integration"* process. This section concentrates in describing the former process.

After the steps in section 5.3.2 have been successfully completed, the integration process takes place. For the purpose of this scenario, it is assumed that a Web Service matching the specification of the Client application that needs to be integrated, already exists on the UDDI.

If a new service is dynamically discovered for the first time, then the schema of this service is downloaded and it is listed under the newly found service window. This window with the newly found Web Service is illustrated in Figure 5.19.



Hull Generation	XML Mappings Required
Passenger Distributions	XML Mappings Required

Figure 5.19 *Newly found service window in JWSI*

If the schema of the Java Web Service (JWS) indicates that the inputs of the Web Service requires XML mappings, then the user has to click on the "Use Service" button in order to create, for the first time only, an XSLT (eXtensible Stylesheet Language Transformations) between the schema of the JWS and the schema of the local application.

This is accomplished via the Java Web Service Integrator XML Mapper Tool (JWSIMT), where the user maps the fields of the XML schema of the Java Web Service with the fields of the schema of the local application. Figure 5.20 illustrates the XML Mapper Tool.

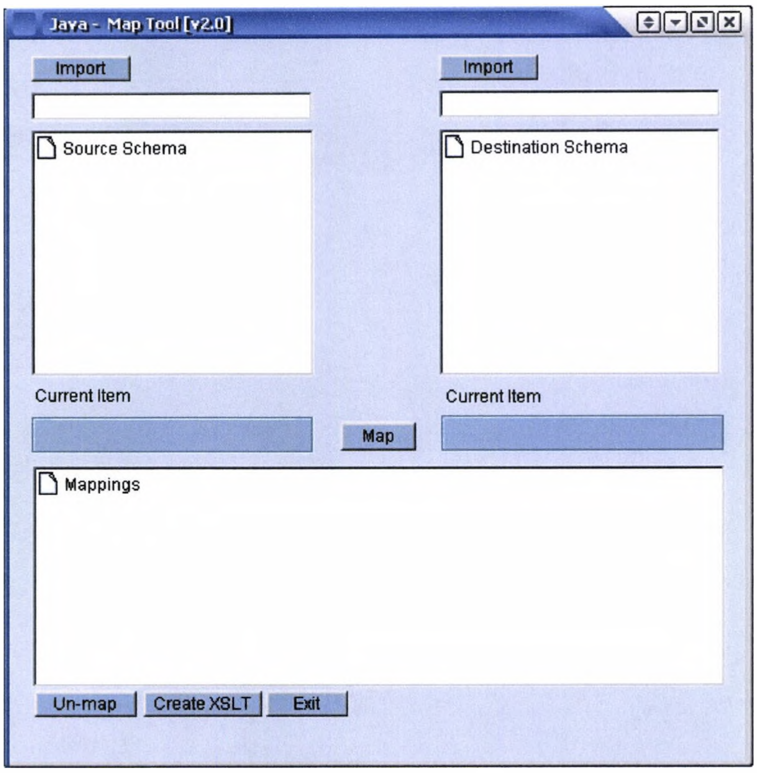


Figure 5.20 *XML Mapper Tool in JWSI*

The outcome of this process is an XSLT, which corresponds to the mappings of the two schemas.

If a new service is dynamically discovered and the XSLT for that Service already exists, then control is given to the Communication Manager which attempts to invoke the Web Service. In case the schema of the Java Web Service (JWS) indicates that the inputs of the Web Service do not require XML mappings, then the JWSI bypasses the XSLT creation process and proceeds directly to the invocation of the Web Service and the interchange of data between the two peers. This happens only when the application on the client site requires data from a Web Service on the server site that does not have to be transformed in any way. When the Service is invoked, the IP of the destination is resolved before any execution of the Service takes place on the server. If for any unforeseen reasons, i.e. temporary server failure, unavailability of the destination server, etc, the Communication Manager is re-scheduled to initiate a second handshake after a predefined interval of time. If after 10 re-tries the server is still unavailable then the Communication Manager marks the recent service as "*Not Available*" and returns control back to the Discovery Service Manger to search for any new Web Services on the UDDI.

If the server that hosts the Web Service is "*Available*" and therefore the Service is ready to be used, the two JWSI peers enter into an active state for the purpose of initiating the Dynamic Integration Process. This is done by attempting to negotiate the communication as well as the security protocols that are available between each other, in order to establish a secure and reliable connection. For this scenario, the available communication protocols that are listed on the envelope of the service are:

- HTTP,
- HTTPS,
- SMTP,
- FTP, and
- SOAP.

If all communication protocols are available, then the two peers need to decide which of them is more appropriate for use. The following decision rules are applied in order to select the appropriate communication protocol.

Rule # 1: Refers to the measurement of the communication bandwidth. This is accomplished by having the two Peers send some "*test data*" of different sizes to each other. By this way, the bandwidth and speed of the communication line is measured and subsequently, the selection of the available protocol is based on these results. Below, is a list of protocols and explanations of rules on where these protocols may be used for each specific scenario.

- **Simple Mail Transfer Protocol**

  The Simple Mail Transfer Protocol (SMTP) is likely to be used in the shipping context environment when some ship designers use the Dynamic Integration Platform from a ship that uses a low bandwidth but expensive satellite link as a means of communication. This kind of integration is applicable when the integrated data is not so confidential and does not require immediate response from the server.

- **File Transfer Protocol**

  The File Transfer Protocol (FTP) is used when the transfer of data is very large. This is essential when large volumes of data are interchanged between the two Peers in a timely manner.

- **Hyper Text Transfer Protocol or Simple Object Access Protocol**

  Hyper Text Transfer Protocol (HTTP) over Transmission Control Protocol/Internet Protocol (TCP/IP), in the shipping context environment, is likely to be used when ships are near shore and can use dial-up connections using Global System for Mobile Communication (GSM) or similar carrier. Hyper Text Transfer Protocol Secure (HTTPS) may be used if confidential information needs to be exchanged between the two legacy Peers. This is because it adds a layer of complexity, which may reduce the transfer rate of the data. In a general context environment, the SOAP protocol over the HTTP or the HTTP alone is likely to be used for almost all transactions that take place between the two peers. The SOAP protocol can be used together with Secure Socket Layer (SSL) to provide consistency and confidentiality of data.

After the most appropriate communication protocol is selected, based on the above criteria, the Security Manager takes place to negotiate the level of security that needs to be applied on the communication line. It is important to note that different security techniques may be applied on different communication protocols. In this scenario, the available security protocols are:

- Encryption over SMTP with different encryption algorithms i.e.3DES. 3DES is a cryptosystem which can encrypt and decrypt data using a single secret key.

- Encryption over FTP and HTTP with 40bit or 128 bit key. The SSL protocol provides data encryption, server authentication, message integrity, and optional client authentication for a TCP/IP connection.

- Digital Signatures. A digital signature is a 'message digest' (created by processing the message contents using a special algorithm) encrypted using the sender's private key. This is another security technology that can be use in order to securely interchange data between the two Peers.

- No Security. This is the default value of all Java Web Services and is used when security is not an issue and the information that needs to be interchanged is not confidential.

After the security protocol has been selected and before it is ready for use by the communication protocol, a "*validation check*" is performed to identify if the selected security protocol is compatible with the selected communication protocol. For example, if the selected communication protocol is HTTP and the selected security protocol is the "Encryption over SMTP" then the security protocol is discarded, since it cannot be applied over the HTTP protocol. In such a case, the Security Manager continues to the next available security protocol and the "*validation check*" is performed once again. For this scenario, the next available protocol is the "Encryption with 128 bit key", which is compatible with the HTTP protocol.

Once the communication protocol and the security protocols have been selected, the two peers are now ready to interchange the relevant data for the purpose of integrating the two applications. The operations that take place on the server site are detailed as described in section 5.3.4.

After the Web Service has been invoked and the transmitted data have reached the client site, the Communication Manager passes the data file to the XML Converter Manager.

In case the client application requires XML mappings, the XML Converter Manager converts the incoming XML file to an XML file readable by the client application. This is accomplished by the use of the XSLT Engine and the XSLT file that was created at the beginning of this scenario. Unless the client application requires the input file to be converted into a Comma Separated Values (CSV) file, the XML file is stored into a specific folder on the Computer in order to be accessed at a later stage by the client application. The conversion from an XML file to a CSV file is also handled by the XML Converted Manager.

In case the client application does not requires XML mappings, the incoming data file is directly saved into a pre-configured directory where it can be accessed by the client application for further processing.

By following the above steps, the dynamic integration process between the two peers is completed and the Communication Managers of both peers close the communication line. The client JWSI application then enters into the same stage as before, that is to monitor for new Java Web Services on the UDDI registry.

5.3.4 Server/Producer to Peer Execution

The above section demonstrated the procedure that takes place on the client site when the client JWSI dynamically finds a Web Service for the purpose of dynamic integration. This section describes the tasks that occur on the server site after the Web Service has been invoked and how the data are transferred back to the client for further processing.

Based on the scenario already described in section 5.3 and after the client JWSI found dynamically a new Web Service, the integration process is ready to be initiated. After the Web Service is invoked by the client Peer, the server Peer detected that a request is pending for execution. The server Peer handles this request by retrieving the specification schema of the requested Web Service. Inside the specification, a number of parameters are included describing specific functionality of the Service.

The first parameters that need to be retrieved are the parameters for describing the server application along with the appropriate input and output parameters of the application. Once the parameters have been retrieved, the server application is executed by the XML Converted Manager, for the purpose of producing the appropriate output. The output of the application, which can either be an XML file, an error message embedded into an XML file or any file type, is returned back to the XML Converter Manager.

If the output file is an XML file and needs XML mappings, then the appropriate XSLT file is loaded from the XSLT Repository so that it can be transformed to the destination XML file as requested by the Web Service. If the file does not require XML mappings or the file is not in XML format, the file is then passed to the Communication Manager in order to be transferred to the client. Before the actual file is transferred to the client, the two Peers need to decide the communication protocol for transferring the output file. The last parameters that have been retrieved from the Web Service specification are the parameters for describing the communication and security protocols.

After these parameters have been retrieved, the JWSI Peer starts the Communication Manager component. This component is responsible for negotiating a suitable and secure communication protocol before the server sends the requested data back to the client. Details of this negotiation are described in section 5.3.3. After the negotiation is agreed, the data are transferred securely to the client Peer so that they can be used by the client application.

By completing the above steps, the dynamic peer-to-peer application integration is achieved.

Due to the high capacity of the source code produced for the Java Web Service Integrator only the list of Java Classes is provided in this report. This is included in Appendix A.

5.4 IMPLEMENTATION SOFTWARE

The following section presents the implementation software that has been used, in order to achieve the development of this proposed system. It also explains why this specific software has been chosen for the implementation process and what the advantages of using such software are.

5.4.1 JBuilder V9.0

Borland JBuilder V9.0 [40,80] now known as Inprise JBuilder is a proprietary development tool marketed by Inprise Corporation, which enables the construction and development of applications in the JAVA programming language. The Java Web Service Integrator Adapter was developed using version 8.0 and 9.0 of this tool. Some features of the JBuilder used in this thesis are:

- **Web Development and Version Tracking**

  Provides expanded XML support with built-in tools for creating and manipulating XML files. Allows transfer of data between Java and HTML seamlessly. Provides revision support in every edition. This includes automatic backup copies of files.

- **User Interface**

  Provides a Graphical User Interface for fast form development. This includes the automatic code generation wizard, which is one of the most advanced features of this product.

- **Object Oriented**

  Provides an object Oriented environment to developers for creating robust and reusable components. It provides tools for easy packaging and deployment of applications.

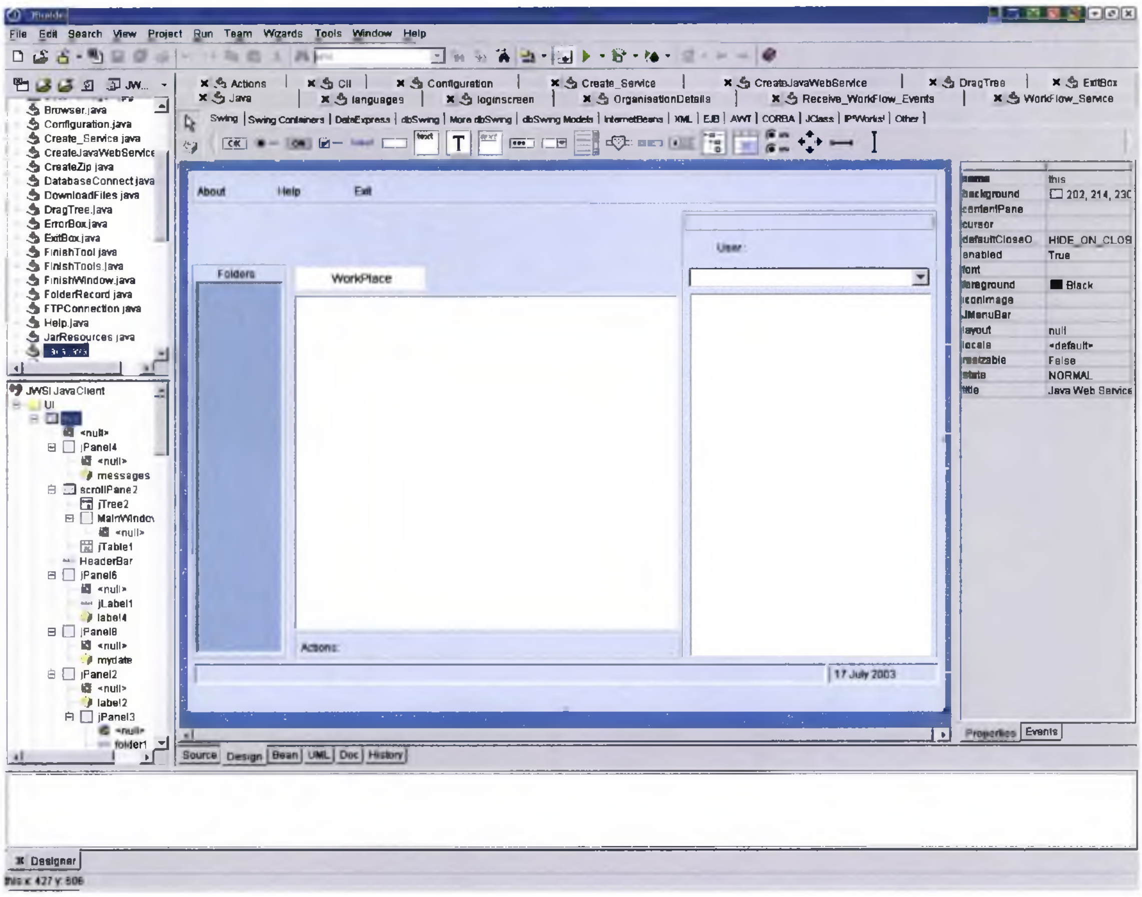
- **Standards**

  Support for the latest Java standards including Java 2, Java 2 JFC/Swing, XML, Java2D, Message Queue, Java collections, Accessibility API, Speech API and more.

- **UML Support**

  UML class and package diagram representation of code: Limited Class, Association Diagram, Reverse Class Association Diagram, Dependency Diagram, Reverse Class Dependency Diagram, and the Class Inheritance Diagram.

A screenshot of the Interface of the Borland JBuilder V8.0 is illustrated in figure 5.21.



name	this
background	202, 214, 230
contentPane	
cursor	
defaultCloseO	HIDE_ON_CLOS
enabled	True
font	
foreground	■ Black
iconImage	
JMenuBar	
layout	null
locale	<default>
resizable	False
state	NORMAL
title	Java Web Service

Figure 5.21 Interface of the Borland JBuilder V7.0

5.4.2 Borland JDataStore V6.0

Borland JDataStore V6.0 [9], is an object relational database management system written entirely in Java, that offers platform-independence, scalability and portability in a fully integrated development environment.

Some features of the JDataStore used in this thesis are:

- **Embedded and application-specific uses**

  Ease of installation, nearly zero administration and economical pricing model, makes JDataStore ideal for embedded and application uses.

- **Small and medium enterprise server applications**

  With its support for multiprocessor platforms and the high-end functionality that is required for sophisticated J2EE and Enterprise Java Beans (EJB) servers, the JDataStore database delivers the scalability required for mission–critical, multi-user applications serving the small and medium enterprise.

- **Platform independence**
  - Functions as a portable storage system
  - Platform-independent, highly scalable object relational database, written entirely in Java
  - Runs on any platform with Java Virtual Machine (JVM) installed

- **High-performance, scalable database engine**
  - Optimised for high transaction throughput
  - Built for intelligent query optimisation strategies
  - Designed to allow read-only transactions to operate at full speed without needing to acquire locks
  - Low-level locking for increased concurrency

- Includes separate local JDBC driver for high-speed in-process access
- Offers JDBC connection pooling with statement caching
- Provides high-speed crash recovery from system failures

❑ **JDataStore Visual Tools**

- Supports import and export from other JDBC databases for interoperability
- Create/restructure tables and indexes for better flexibility

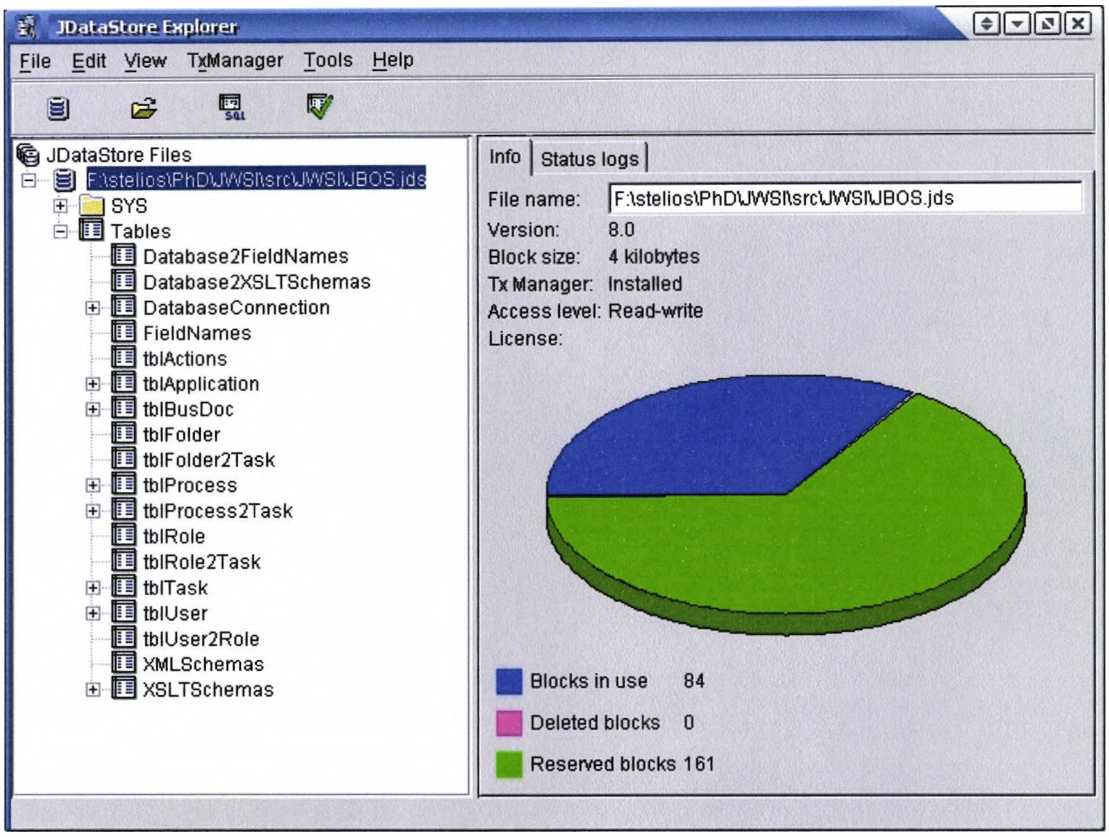
❑ **Development**

- Integrated with Borland JBuilder, the leading Java development solution, and Borland Enterprise Server, the leading enterprise platform for Web, CORBA, and J2EE deployment, for quicker development and deployment of applications.
- Unidirectional data replication for supporting a disconnected development model
- Security features for user authentication and database encryption

❑ **Deployment**

- Use a single jar file that can be used to deploy applications across all major platforms
- Take advantage of the portable file format; use database file and log files on any major platform
- Use read-only transactions for online backup for continuous database availability

A screenshot of the Interface of the Borland JDataStore V6.0 is illustrated in figure 5.22.



File name: F:\stelios\PhD\JWSI\src\JWSI\JBOS.jds
Version: 8.0
Block size: 4 kilobytes
Tx Manager: Installed
Access level: Read-write
License:

🟦 Blocks in use 84
🟪 Deleted blocks 0
🟩 Reserved blocks 161

Figure 5.22 Interface of the Borland JDataStore V6.0

Based on the above advanced features, it has been decided to use the JDataStore as the Database server of this Platform. The next section describes the Implementation models of the system.

5.5 IMPLEMENTATION MODELS

The implementation models are derived from the implementation view of the proposed system. The implementation view focuses on architectural decisions such as the organisation of the actual static software modules (source files, binaries and executables) within the development environment. The view of the architecture takes into account those requirements related to software management, ease of development, ease of use and constraints imposed by programming languages and development tools.

The results derived from the implementation view are the *Component* and *Deployment Diagrams*, which illustrate the dependencies between the subcomponents. Components represent software files that are contained in a package (subcomponent) and show the structure of the actual code. The classes identified in the logical view are components in the implementation view.

5.5.1 COMPONENT DIAGRAM

Component diagrams (see figure 4.10) provide a physical view of the current model. A component diagram shows the organisations and dependencies among software components, including source code components, binary code components, and executable components. These diagrams also show the externally visible behaviour of the components by displaying the interfaces of the components. Calling dependencies among components are shown as dependency relationships between components and interfaces on other components. Note that the interfaces actually belong to the logical view, but they can occur both in class diagrams and in component diagrams.

A *component* represents a piece of software code or file containing information (for example the XML specification of the Java Web Service). A component can also be a combination of other components, for example an application consisting of several executables. The component should implement the corresponding design correctly and fulfill the design specifications.
A high-level component diagram for the proposed architecture is illustrated in figure 5.23.

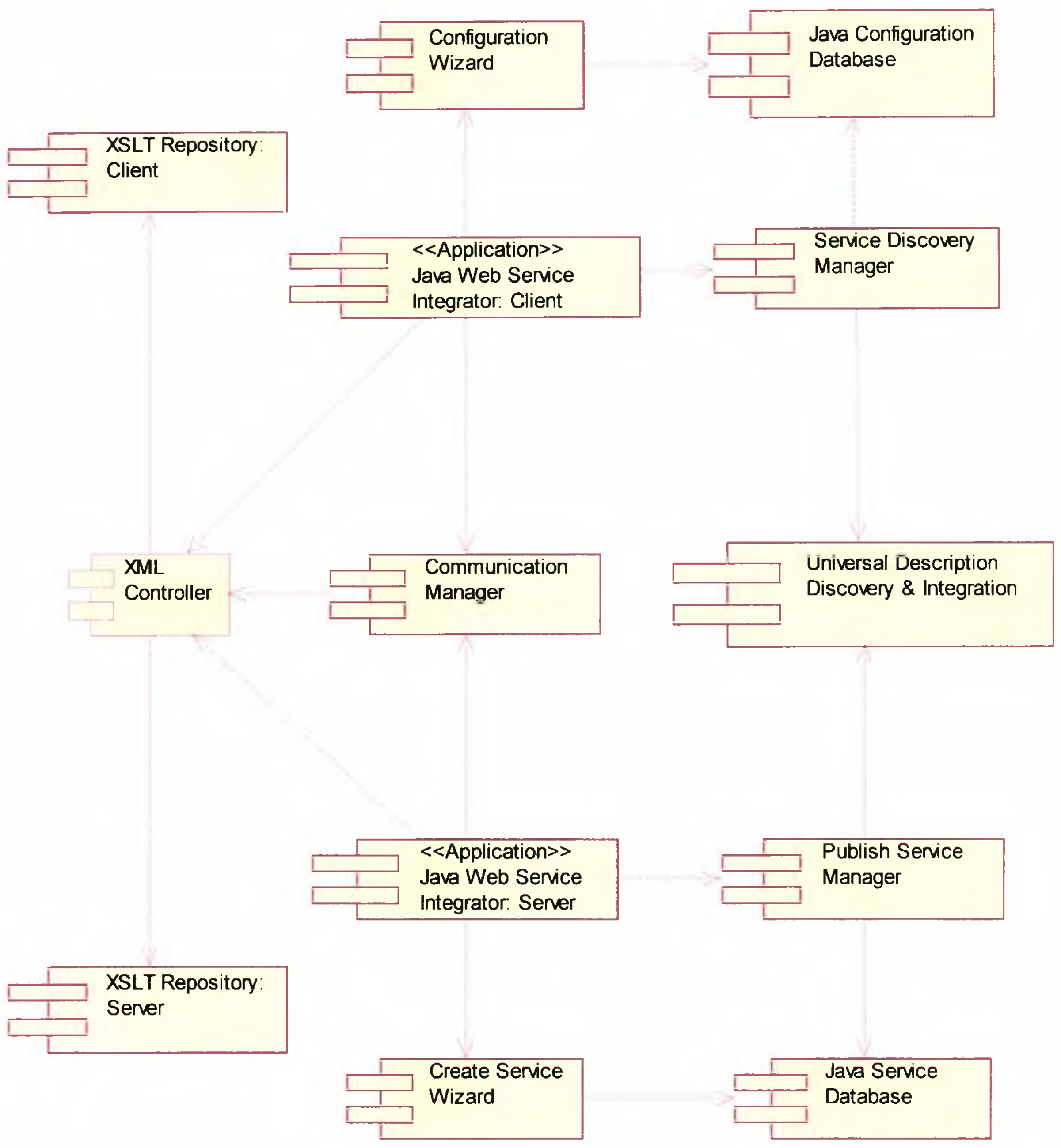

Configuration Wizard
Java Configuration Database
XSLT Repository: Client
<<Application>> Java Web Service Integrator: Client
Service Discovery Manager
XML Controller
Communication Manager
Universal Description Discovery & Integration
<<Application>> Java Web Service Integrator: Server
Publish Service Manager
XSLT Repository: Server
Create Service Wizard
Java Service Database

Figure 5.23 *Component diagram of the system*

5.5.2 DEPLOYMENT DIAGRAM

A *Deployment diagram* (see figure 4.11) shows processors, devices and connections. Each model contains a single deployment diagram, which shows the connections between its processors, devices, and the allocation of its processes to processors. Processor Specifications, Device Specifications, and Connection Specifications enable modification of the respective properties. The information in a specification is presented textually; some of this information can also be displayed inside the icons.

A Deployment diagram consists of a graph of nodes that are connected by communication associations. The nodes may contain component instances and components may contain objects. Components are connected through dashed-arrow dependencies.

The deployment diagram of the System is presented in figure 5.24.

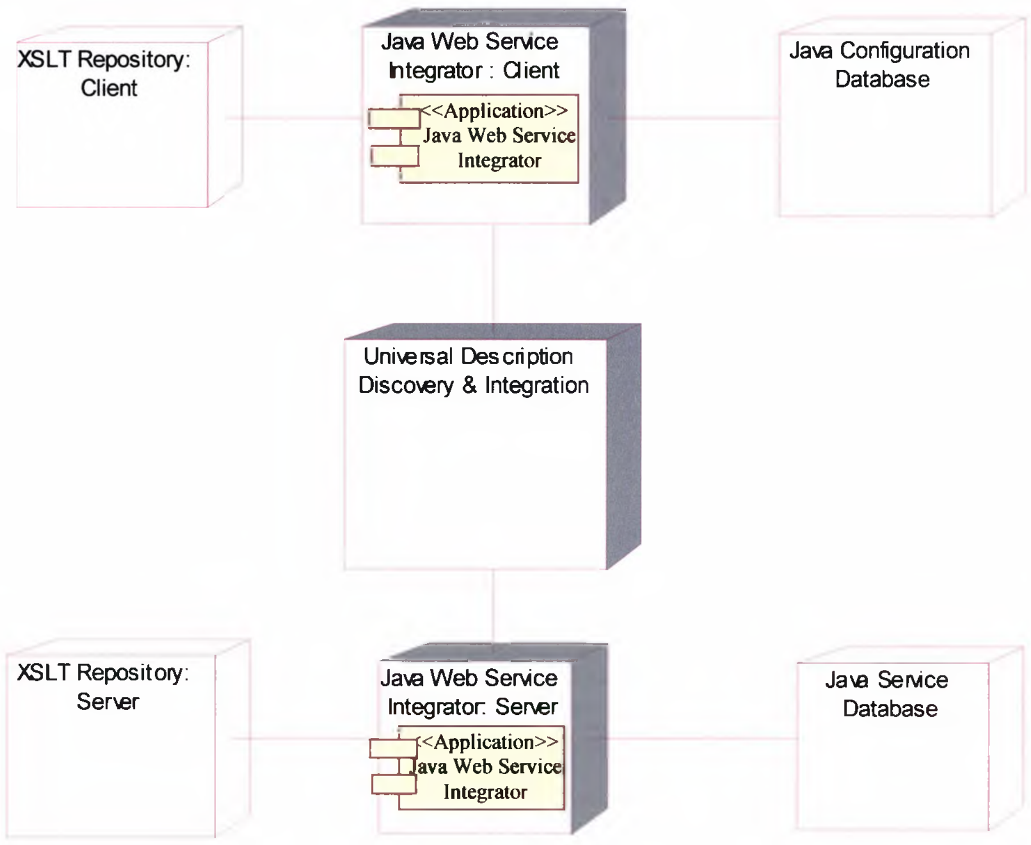

XSLT Repository: Client
Java Web Service Integrator : Client
<<Application>> Java Web Service Integrator
Java Configuration Database
Universal Description Discovery & Integration
XSLT Repository: Server
Java Web Service Integrator: Server
<<Application>> Java Web Service Integrator
Java Service Database

Figure 5.24 *Deployment diagram of the system*

5.6 CONCLUDING REMARKS

This chapter covered the various aspects of the system implemented according to guidelines set in the design stage of this thesis. The system's Web Service implementation protocols were presented by providing sample source codes. The functionality of the overall Peer-to-Peer platform, including the client/consumer and server/producer parts, were then demonstrated with the aid of some scenarios, where the implemented software was shown to perform the required tasks. Next, the implementation software that has been used was presented and the Chapter ends with a description of the component and the deployment diagram of the proposed platform.

Chapter 6

EVALUATION AND TESTING

6.1 INTRODUCTION

So far, the design and implementation phases of this thesis have been completed. In this Chapter the correctness of the software system that has been developed is evaluated and verified based on some criteria that will be discussed further in this Chapter.

Evaluation and testing is an important part of the system development process. It is undertaken to determine that the system achieved the requirements. The evaluation process will help to assess the strengths and limitations of this system. It is carried out in order to determine whether the system has achieved its objectives and aims. Feedback received from a group of users has also been evaluated and has provided guidance for further system improvement.

The scope of this Chapter is to discuss how the initial requirements are met and test whether the final system operates in accordance with the user needs.

6.2 EVALUATION

6.2.1 What is Evaluation?

"*Evaluation is the assessment of the achievement of the stated aims.*" [72]

Software evaluation is a process including the definition of assessment criteria, the specification of an evaluation plan, the collection of evaluation information and the analysis of the results in order to make necessary decisions about the software. Software evaluation can include any or a variety of different types of evaluation, such as for needs assessments, accreditation, cost/benefit analysis, effectiveness, efficiency, formative, summative, goal-based, process, outcomes, etc. The type of software evaluation that is undertaken to improve the software depends on what can be learnt about the software. The most important factor of software evaluation is to accurately collect and understand the information that has been collected [10, 45].

There are many different ways to conduct evaluation. Evaluation requires careful consideration of the questions that need to be answered, the type of software being evaluated and the ways in which the information generated will be used. It should provide valuable information about the system functionality that could contribute to system enhancements. To make the evaluation process successful, some measures of data gathering must be used. These measures are classified into two categories, which are:

1. **Quantitative:** Quantitative evaluation is an assessment process that answers the question, "How *much* did we do?"

   - Surveys and questionnaires

2. **Qualitative:** Qualitative evaluation is an assessment process that answers the question, "How *well* did we do?"

   - Interviews
   - Focus groups
   - Meetings

Since this system is partially funded by the European Framework V, regular meetings took place. Every three months, the members of the consortium met to discuss and refine the requirements of the system, as it was developing. This enabled the above-mentioned categories of data gathering to be implemented. The information collected was examined and proposals for further system improvement were considered. This included the identification of specific issues within the system including bugs, functionality shortcomings and dependability concerns particularly related to usability, security and performance.
Another evaluation technique that was used to evaluate the system according to the initial requirement was to actually distribute the Java Peers (including the Java Database) to different users and partners of the European project. This technique helped dramatically to identify any technical errors that were found during the installing, deployment as well as during the operation of the system. Moreover, a number of users suggested different ways on how to improve the Graphical Interface of the system for ease of use, as well as on ways to improve the whole dynamic integration process. All the above data gathering and comments from the users are listed in Appendix B.

The results and comments from the users evaluating the system have been collected and have helped to make the appropriate modifications and refinements in order to improve the system and gain the acceptance of the users.

The evaluation process was of great benefit to the system development, since the system was being continuously evaluated throughout its lifecycle. During the regular evaluation meetings, the modifications and refinements needed were reduced to the minimum, thus reducing difficulties to evaluate the whole system after the completion of the implementation phase.

6.2.2 Benefits of the System

One of the most important factors for the success or failure of a software system is the evaluation of the benefits of the system, whether the system has gained user acceptance and how this system can be used in order to solve the problems already outlined in Chapters 1 and 2.

The best evaluators of a system are the potential users of the system in their daily work. For the system developed in this Thesis, feedback has been received from the Consortium as these will be the end-users of the system.

The results and comments from the users evaluating the system have been collected and have helped to make the appropriate modifications and refinements in order to improve the system and gain the acceptance of the users. Appendix B presents the findings from the different methods used to collect the necessary requirements for the development of the system as well as collected significant data and comments received from the users of the system. The feedback that has been received by the users indicated that the benefits of designing and developing such a dynamic integration platform is of strategic importance, as this platform has not only increased the efficiency of their work dramatically, but has decreased the costs of integrating different ship design applications together.

Furthermore, since one of the initial requirements of this system was the easy integration of future software applications, more companies now can be easily connected to the platform in order to form a knowledge network for the purpose of sharing data and information related to the ship design process. This is of great importance, since according to the Consortium of the project a lot of companies were reluctant to spend time and money to make the necessary modifications on their software applications, which are usually enormous and cost a great deal of money, in order to communicate with other software applications.
In addition, the system helped dramatically in bringing companies together in a dynamic knowledge network where their ship applications will be able to find the relevant information on time, so as to accomplish specific tasks or processes.

Another importance benefit of the proposed solution is the cost of integration. Enterprise Application Integration costs come in three components [84]:

- architecture,
- integrations, and
- operations.

Architecture costs are costs related to the initial deployment such as integration development, execution and operations environments. They include the license cost that is negotiated with the vendor, the cost of new hardware required to develop, run and monitor integrations and the cost to implement architectural software and hardware. Roughly 80% of architecture costs are incurred within 6 months of implementation while additional expenses may be incurred for hardware or licenses as usage spreads [84]. Architectural costs are driven by the complexity of the integration software and the number of separate business entities to which it is deployed.

The advantage of the proposed solution is that it does not require any new hardware to be purchased by the companies that need to integrate their applications. The JWSI software of the dynamic integration platform can be installed in any machine easily since it has been developed using the Java language. No central server or extra hardware is required in order for the dynamic platform to function properly. Furthermore, it does not require any additional software development to be done, since the integration is achieved by following some easy-to-use integration wizards and no software code is required to be written. As far as the license of the proposed solution is concerned, this will be negotiated according to the Consortium of the project. Because of these reasons it can be concluded that the architecture costs of the proposed solution are minimal since no additional hardware or software implementation is required to achieve the dynamic integration.

Integration development costs are separate from the architectural costs. Integration costs are often related to the development of interfaces and collaborations between systems. Integration costs are variable and are driven by the number of interfaces that are developed. Integration costs with EAI are generally between 25% and 40% lower than with custom integration [84]. Development is less expensive because adapters come pre-built with the EAI architecture and the architecture provides a graphical interface in which to perform mapping as well as many pre-built functions.

The cost of the proposed solution in relation to the integration development costs is almost zero, since it is not required to be developed interfaces between the two applications. The JWSI peers will be responsible to send and receive the necessary data without having to learn or develop any new interfaces to communicate with the user's application.

Operating costs are expenses and include on-going operations and maintenance of the EAI system for architecture and integrations. Operating costs generally are driven by the number of interfaces that need to be maintained and rise as more interfaces are put into production. EAI generally provides a 50% to 80% reduction in application maintenance cost by reducing the number of interfaces that need to be maintained and offloading much of the costs of interface maintenance onto the EAI solution provider [84].

As already mentioned in the previous paragraph there is no need to implement any new interfaces and hence the cost of maintaining such interfaces is null.

To sum up regarding the total cost of the dynamic integration platform it can be concluded that this approach can be substantially cheaper than other enterprise integration packages. This is a huge advantage over other integration systems since the cost of integration is one of the most importance factors that influence companies to process or not to process with integration of their applications with other companies or within the same organisation. An estimated graphical comparison regarding the cost of integration between existing Integration systems and the proposed Integration system is shown in Figure 6.1.

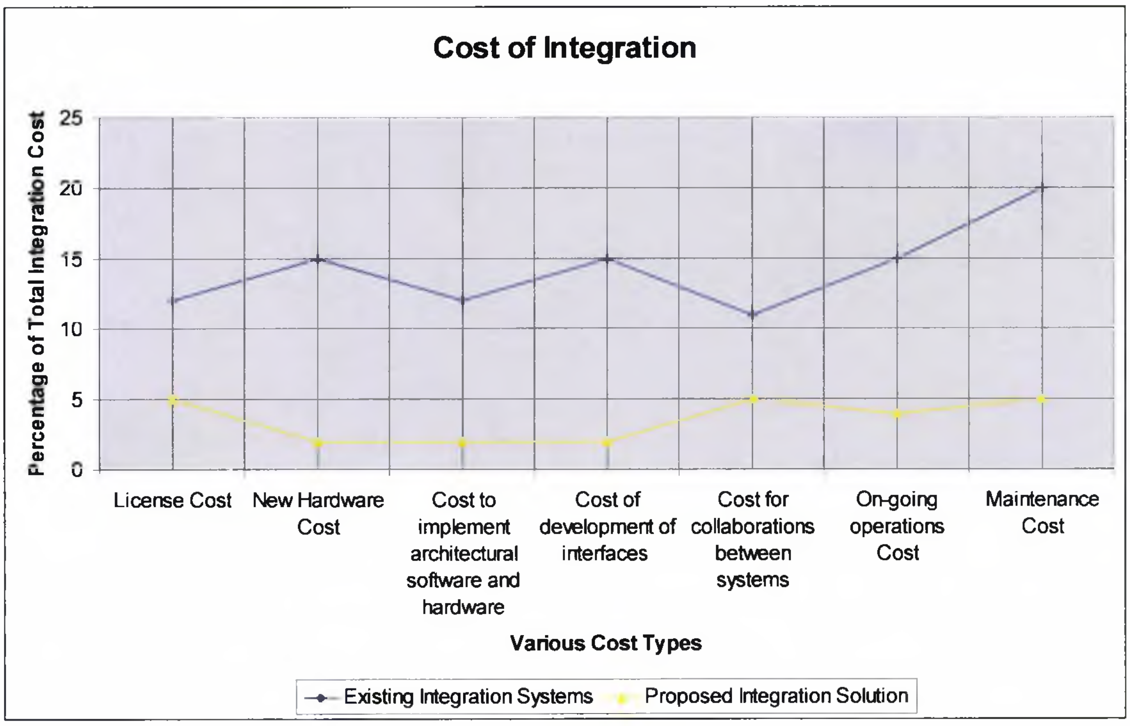


Figure 6.1 *Estimated graphical comparison regarding cost of Integration between existing Integration systems and the proposed Integration system*

As it can be seen from the Graph, the proposed solution is substantially cheaper in all cost types of an Enterprise Integration System as already explained in this section.

Last but not least, the companies have expressed their gratitude, since now they can expand into new areas of the shipping sector, which will lead them to the next level of advanced ship design. The next level will be a collaborative knowledge dynamic platform where a number of manufacturing companies may contribute to the design of large ship design projects in order to construct and deliver ships to their customers in a faster way.

To sum up, it can be deduced that the users that have evaluated the system were extremely satisfied with the results of the system and as stated earlier; this system will benefit not only the shipping industry but also other sectors that require application integration in their business processes.

6.2.3 Analysis of Evaluation

Based on the feedback obtained from various companies (see Appendix B), and the results of section 6.2.2 , it can be concluded that the system not only has dramatically increased the efficiency of the consortium work, but has also decreased the costs of integrating different ship design applications together. The efficiency of their work has been increased since they no longer have to manually transfer their ship models between the shipping companies and with the new proposed system they know the current progress of various ship models without having to manually communicate between each other.

Furthermore, using the peer-to-peer technology, the proposed system is faster than their existing systems and have helped enormously not only with the collaboration between the shipping companies but also enabled the easy and faster integration of their existing systems, which was very difficult to achieve using their existing manual methodologies. In addition, the continuing evaluation of the system by the consortium has enabled the proposed system to be adapted to their needs and as a result this has gained their acceptance. The easy maintenance, easy installation and easy integration are some additional factors of gaining user acceptance.

6.3 TESTING

6.3.1 Software Testing

"*The software testing process is the means by which people, methods, measurements, tools and equipment are integrated to test a software product.*" [69]

Software verification and testing are essential to minimise the risks of using technology. Verification and testing is best conducted well before the system is used for a "live" event (both on its own and in conjunction with associated hardware and communications). After successful testing, software systems will require appropriate maintenance to ensure they will perform effectively when needed.

The level of importance of the technology will impact on the degree of difficulty applied to verifying and testing software systems. For a peer to peer system, the degree of difficulty is considered to be high since there are a number of factors that are involved during various testing and verification scenarios.

6.3.2 Software verification

"*Verification, as defined by IEEE/ANSI, is the process of evaluating a system or components to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase*" [69].

Software verification tests or otherwise known as qualification tests could include:

- testing of software to ensure that appropriate standards are met and that the software performs its intended functions, including audits of code
- ensuring system documentation is adequate and complete
- verifying that systems are capable of performing under expected normal conditions and possible abnormal conditions

- ensuring that security measures are in place and that they conform to appropriate standards
- ensuring that appropriate quality assurance measures are in place

After software has been verified, it needs to be thoroughly tested to ensure that every component of the system is operating as it should, and that the system is performing exactly in accordance with the specific local requirements.

For the purpose of this system, a structured system testing program was established to ensure all aspects of a system are tested. Testing measures included the following:

- developing a set of test criteria
- applying functional tests to determine whether the test criteria have been met
- applying qualitative assessments to determine whether the test criteria have been met
- conducting tests over an extended period of time, to ensure systems can perform consistently
- conducting 'load tests', simulating as close as possible a variety of 'real life' conditions using or exceeding the amounts of data that could be expected in a real situation
- verifying that 'what goes in' is 'what comes out', by entering known data and checking that the output agrees with the input

Verification was achieved by applying a number of verification techniques at the very early stages of the system's development cycle. Some of these techniques are:

- requirements verification,
- functional design verification,
- internal design verification, and
- code verification.

6.3.3 Validation Testing

"*Validation, as defined by IEEE/ANSI, is the process of evaluating a system or component during or at the end of the development process to determine whether it satisfied specified requirements.*" [69]

Validation tests are tests that measure some aspect of an implementation's behaviour against an expectation and answer whether the software complies with the expectation. For example, a test that measures the time required to perform a specific task is not a validation test. A test that measures the time required to perform a specific task and compares it to a benchmark is a validation test.

Validation tests provide assurance that the software as written meets requirements and/or specifications. When validation testing becomes a fundamental part of the software development process, software is more likely to be completed, more likely to be fit for the intended purpose, is less expensive to build and maintain and takes less time to complete.

A common practice is to associate each module of software with a set of automated validation tests, often called 'unit tests'. These tests are run on a frequent basis, usually every night. Even if the code in the module has not changed, the module may depend on other modules that may have changed, so it is necessary to run the tests at least as often as code may change.

Validation normally involves executing the actual software or simulated prototype. Validation is a '*computer-based testing*' process. It actually exposes symptoms or errors. Some validation techniques that have been used in this thesis are:

- ❑ unit validation,
- ❑ integration validation,
- ❑ function validation, and
- ❑ system validation.

Several tests should be designed, built and executed in order to verify that the developed software system conforms to the initial requirements and specifications. These test scenarios are presented in the following section. In total, six test scenarios will be presented, which aim to provide a comprehensive test of the entire system and all its sub systems.

6.3.4 Server/Producer Testing

The testing of the Java Web Service Integrator application on the Server site aims to present the expected results as specified in the requirements phase of this thesis. It was decided to use three test scenarios that demonstrate the functionality according to the initial requirements of this system.

Scenario 1

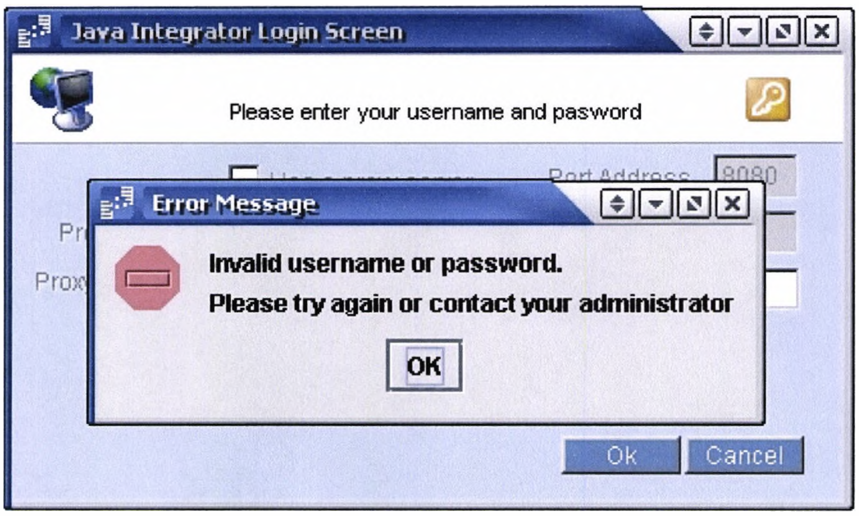
Testing the JWSI authentication mechanism

The following scenario illustrates the testing of the authorisation mechanism used to authenticate different users that will use the JWSI Server peer. As already discussed in Chapter 5, section 5.3.2, the user has to enter a username and a password in order to run the JWSI peer. Failure to enter a correct username and password would result in the specified user being denied access to the JWSI server peer application.

After completing the login form of the JWSI peer, the user may click on the "*OK*" button to initiate the login procedure and have access to the application. This validation is accomplished by a component searching the user's table in the JDataStore database, for the user name.

In case a username or the password for that username is incorrect, then an error message is passed from the JWSI peer to the user. For the purpose of testing, an invalid password has been supplied to check the authorisation mechanism.

Figure 6.2 shows that the login screen of the JWSI server peer denies access to the user by displaying in a pop-up window an error message and instructing the user to login again.


Java Integrator Login Screen
Please enter your username and pasword
Port Address
8080
Error Message
OK
Ok
Cancel

Invalid username or password.
Please try again or contact your administrator

Figure 6.2 *Error message returned by login screen of the Java Server Peer Client*

The above scenario has also been tested by using an invalid username. The results were successful and access to the particular user was denied.

The test results taken from the above scenario indicated that the authorisation mechanism is functioning according to the expected results, by not allowing any unauthorised users to access the server. This is of great importance since any unauthorized login to the system by users may result in abuse and subsequently to the compromisation of the entire platform.

Scenario 2

Automatic Java Web Service Creation

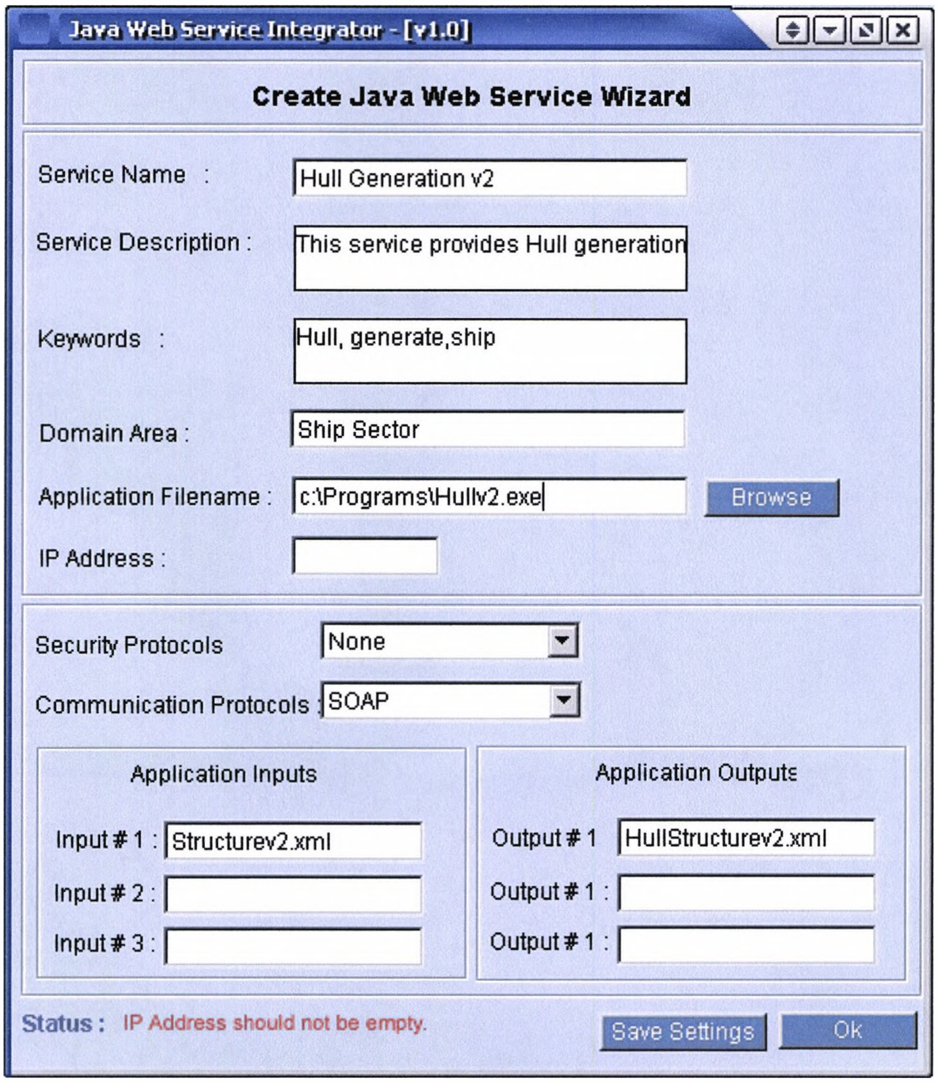
The following scenario illustrates the testing of the automatic creation of Java Web services.
In this scenario, the user needs to follow certain steps before the service is created and published on the UDDI registry. Some main parameters that have to be specified for this service are:

- service name,
- service description,
- service keywords,
- the domain area of the service,

- the application that corresponds to this service,
- the IP address of the client running the application,
- available security and communication protocols, and
- finally the input and output parameters for the application.

For the purpose of this scenario, specific values have been inserted into the system (see figure 6.2) in order to demonstrate that the Java Web service is correctly generated.

In case the user forgets to enter one parameter, the system prompts the user that all parameters are required to be entered and an error message is displayed on screen as illustrated in figure 6.3. In this example, the system returns the error message "*IP Address should not be empty*".
In order for the system to continue the client must enter a valid IP address.


Java Web Service Integrator - [v1.0]
Create Java Web Service Wizard

Service Name : Hull Generation v2
Service Description : This service provides Hull generation
Keywords : Hull, generate,ship
Domain Area : Ship Sector
Application Filename : c:\Programs\Hullv2.exe
IP Address :
Security Protocols None
Communication Protocols : SOAP

Application Inputs
Input # 1 : Structurev2.xml
Input # 2 :
Input # 3 :

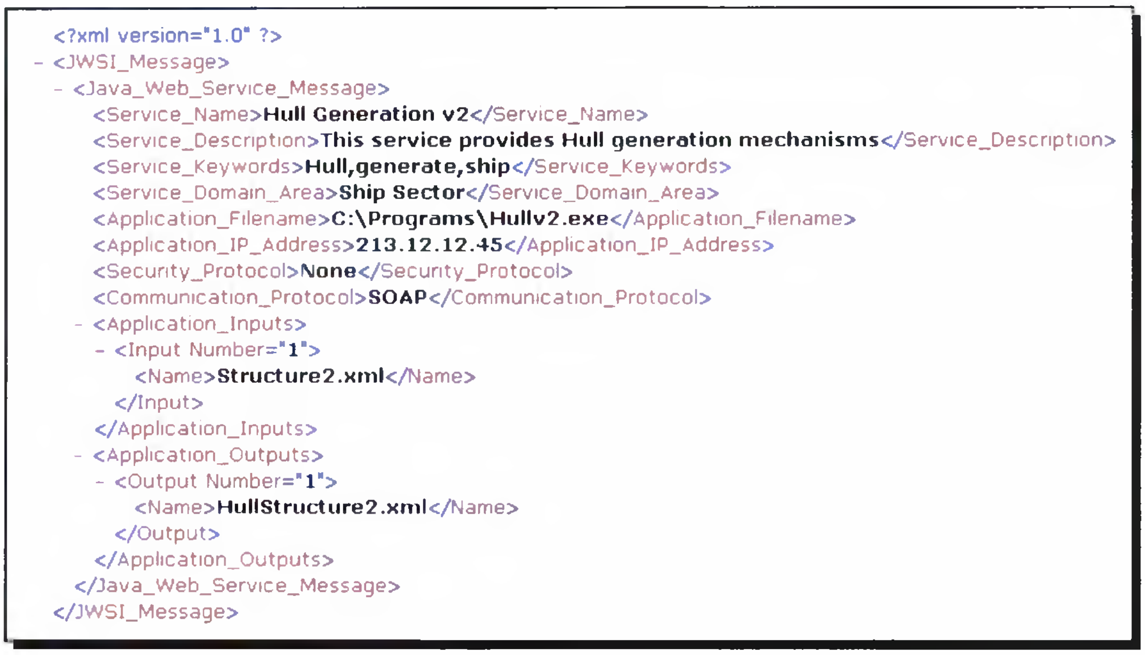
Application Outputs
Output # 1 HullStructurev2.xml
Output # 1 :
Output # 1 :

Status : IP Address should not be empty.

Figure 6.3 *Error message returned by the Create Java Web Service Wizard*

After all these parameters have been specified, and after the user clicks on the "*Save Settings" button,* the system component saves the Java Web Service specification in the XML database before the service is published on the UDDI registry. The automatic publishing of the Java Web Service is described in the following scenario.

By physically examining the XML database of the system, it can be seen that the correct XML specification of the Java Web Service is generated as illustrated in figure 6.4.



```
  <?xml version="1.0" ?>
- <JWSI_Message>
  - <Java_Web_Service_Message>
      <Service_Name>Hull Generation v2</Service_Name>
      <Service_Description>This service provides Hull generation mechanisms</Service_Description>
      <Service_Keywords>Hull,generate,ship</Service_Keywords>
      <Service_Domain_Area>Ship Sector</Service_Domain_Area>
      <Application_Filename>C:\Programs\Hullv2.exe</Application_Filename>
      <Application_IP_Address>213.12.12.45</Application_IP_Address>
      <Security_Protocol>None</Security_Protocol>
      <Communication_Protocol>SOAP</Communication_Protocol>
    - <Application_Inputs>
      - <Input Number="1">
          <Name>Structure2.xml</Name>
        </Input>
      </Application_Inputs>
    - <Application_Outputs>
      - <Output Number="1">
          <Name>HullStructure2.xml</Name>
        </Output>
      </Application_Outputs>
    </Java_Web_Service_Message>
  </JWSI_Message>
```

Figure 6.4 *Testing the XML specification of the Java Web Service*

The test results taken from the above scenario indicate that the automatic Java Web Service creation is functioning according to the expected results and according to the design requirements of the platform.

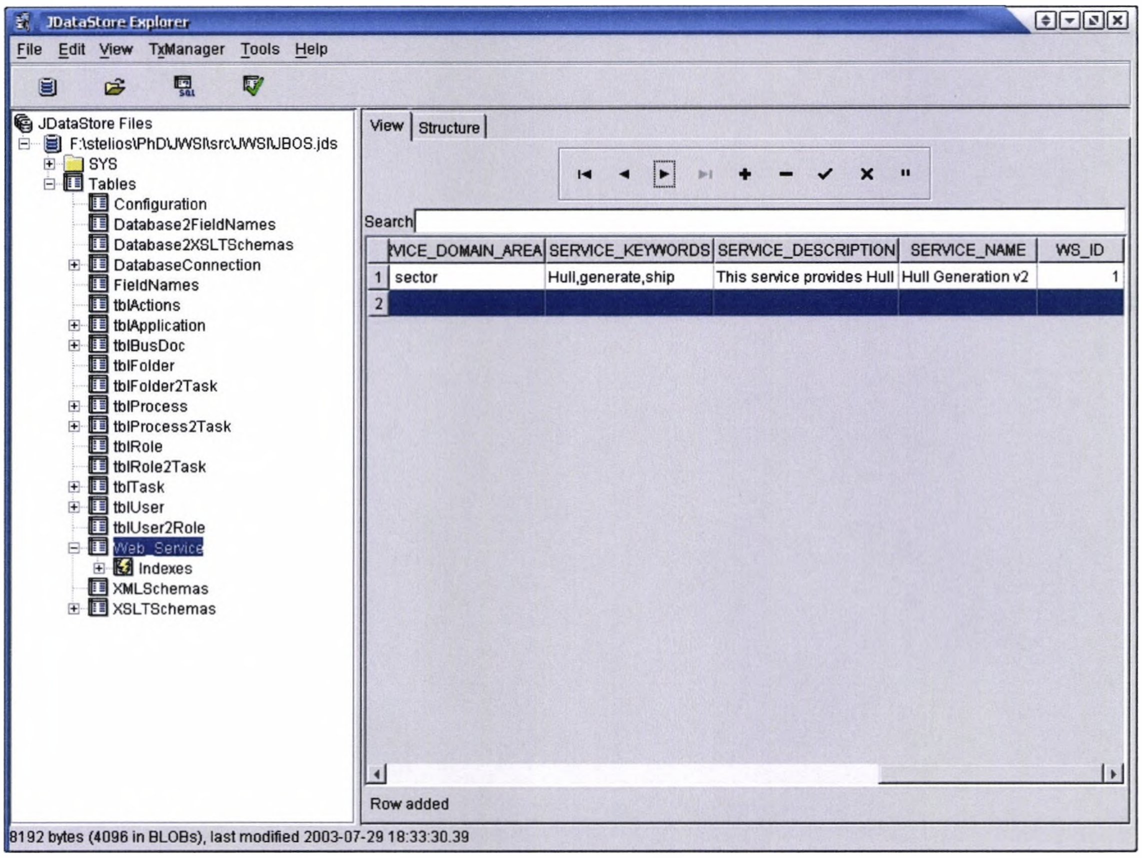
The next scenario illustrates how the newly created Java Web service can be published to the UDDI registry.

Scenario 3

Automatic Publishing of Java Web Services

The following scenario illustrates the testing of the automatic publishing of Java Web services to the UDDI registry.

After completing the creation of the XML specification of the Java Web Service and the user presses the "Save Settings" button, as illustrated in figure 6.5, the service is initially stored in the XML database and then it is published automatically on the UDDI.



JDataStore Files
- F:\stelios\PhD\JWSI\src\JWSI\JBOS.jds
  - SYS
  - Tables
    - Configuration
    - Database2FieldNames
    - Database2XSLTSchemas
    - DatabaseConnection
    - FieldNames
    - tblActions
    - tblApplication
    - tblBusDoc
    - tblFolder
    - tblFolder2Task
    - tblProcess
    - tblProcess2Task
    - tblRole
    - tblRole2Task
    - tblTask
    - tblUser
    - tblUser2Role
    - Web_Service
      - Indexes
    - XMLSchemas
    - XSLTSchemas

	VICE_DOMAIN_AREA	SERVICE_KEYWORDS	SERVICE_DESCRIPTION	SERVICE_NAME	WS_ID
1	sector	Hull,generate,ship	This service provides Hull	Hull Generation v2	1
2					

Row added

8192 bytes (4096 in BLOBs), last modified 2003-07-29 18:33:30.39

Figure 6.5 *Testing the publishing of the Java Web Service*

The test results taken from the above scenario indicate that the automatic publishing of Java Web Services is functioning as expected.

By completing the above three scenario, it can be concluded that the JWSI server peer is working according to the original requirements, design and implementation of the platform as well as according to user's feedback on the architecture of the platform. The next section describes the testing of the JWSI on the client site.

6.3.5 Client/Consumer Testing

The testing of the Java Web Service Integrator application on the Client site aims to present the expected results as they were specified in the requirements phase of this thesis. It was decided that three test scenarios would be adequate to thoroughly test the client part of this prototype.

Scenario 4

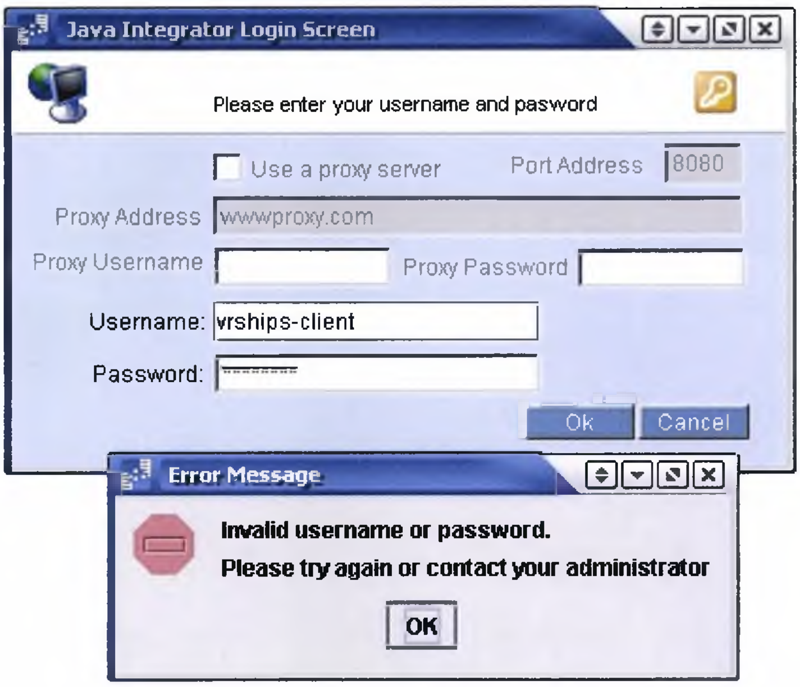
Testing the authorisation mechanism

The following scenario illustrates the testing of the authorisation mechanism used to authenticate the user that is using the Java Peer on the client site. Failure to enter a correct username or password would result in the specified user being denied access to the system.

As already discussed in Chapter 5, section 5.3.2, the client needs to enter a correct username and password. By entering a username or password, the Java Peer Client connects to the Java database stored locally, requesting that particular username to be validated against the database. This validation is accomplished by a component searching the user's table in the Java database, for the above username.

This process is of great importance since without this authorisation mechanism, any user that installs the system on his/her personal computer, could have access to the system and potentially misuse it. This process is a compulsory component of all software systems nowadays. Of course the authorization mechanism is no so advanced. An advanced version of the authorisation mechanism is discussed in Chapter 7 under section 7.3, future work.

In case a username or the password for that username is incorrect, the Java Peer displays a window indicating to the user that an incorrect username or password has been entered as shown in figure 6.6.



Username: vrships-client

Password: ********

Invalid username or password.
Please try again or contact your administrator

Figure 6.6 *Invalid username returned by the Java Peer Client*

By providing a correct username and password, the Java Peer grants access to the user.

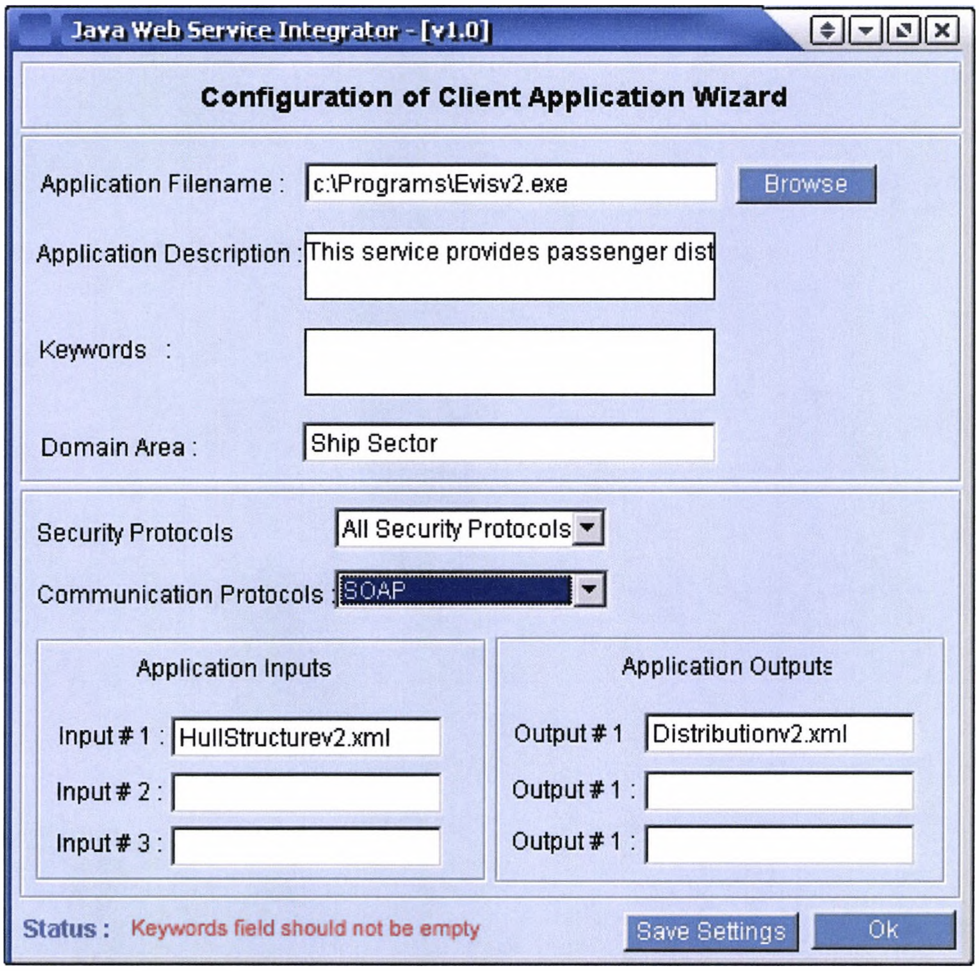
The test results obtained from the above scenario indicated that the authorisation mechanism is working according to the expected results, that is, any unauthorised users are not allowed to access the Java Client Peer. The same authorization mechanism exists for the Java Peer on the server site.

Scenario 5

Configuration File Generation.

The following scenario illustrates the testing of the creation of the configuration file. The configuration process is essential since the JWSI must know the inputs and outputs of the local application needed to be integrated.

This is important in order for the JWSI to know which dynamic services are needed to be downloaded from the UDDI. This is accomplished by comparing the parameters of the service on the UDDI and the parameters of the configuration file of the local application. Not all Java Web Services found on the UDDI are downloaded since this is unnecessary and inefficient. Figure 6.7 illustrates the testing of the creation of the configuration file.


Java Web Service Integrator - [v1.0]
Configuration of Client Application Wizard
Browse
Application Inputs
Application Outputs
Save Settings
Ok

Application Filename : c:\Programs\Evisv2.exe

Application Description : This service provides passenger dist

Keywords :

Domain Area : Ship Sector

Security Protocols All Security Protocols

Communication Protocols : SOAP

Input # 1 : HullStructurev2.xml

Input # 2 :

Input # 3 :

Output # 1 Distributionv2.xml

Output # 1 :

Output # 1 :

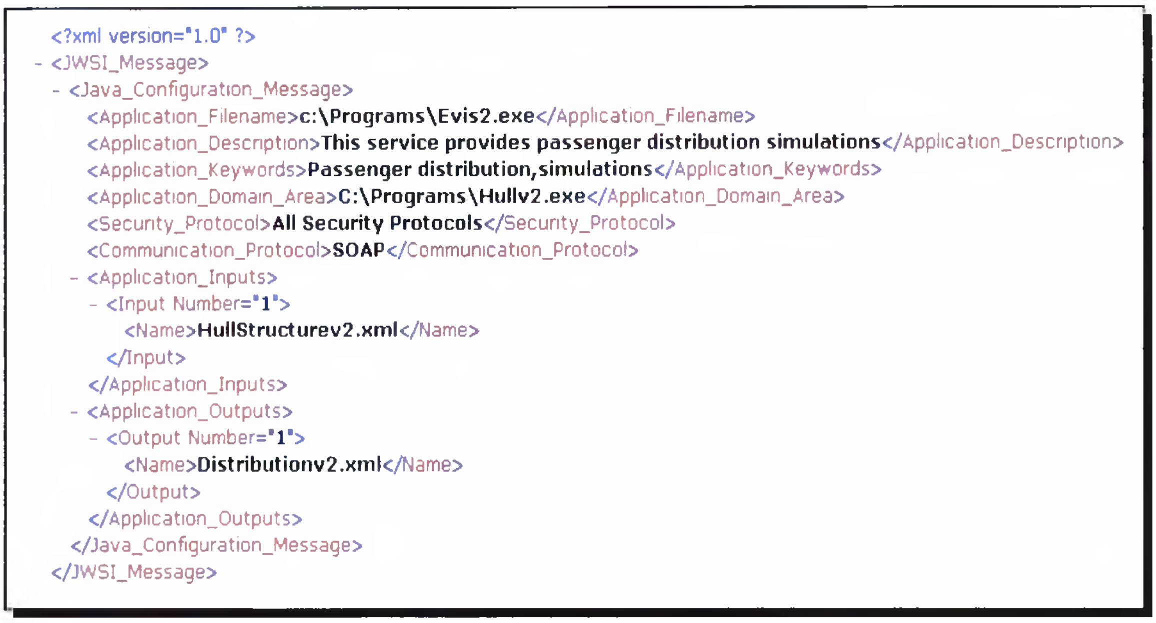
Status : Keywords field should not be empty

Figure 6.7 Testing the *Configuration of Client Application Wizard*

For the purpose of this scenario, specific values have been entered in order to test that the correct XML configuration file is generated. In case the user forgets to enter one of the above parameters, the system will notify the user that the XML configuration cannot be saved unless all the parameters have been entered. . In this example, the system returns the error message "*Keywords field should not be empty*". In order for the system to continue, the client must enter some keywords that describe the application.

After all parameters have been specified, and after the user presses the "*Save Configuration*" button, the Configuration file is created and stored in the local Java Database server. By physically examining the XML Database, it can be seen that the

XML configuration has been created successfully and according to the specifications of the platform. Figure 6.8 illustrates the specification of the Configuration file that will be used for dynamic discovery of Java Web Services on the UDDI.



```
  <?xml version="1.0" ?>
- <JWSI_Message>
  - <Java_Configuration_Message>
      <Application_Filename>c:\Programs\Evis2.exe</Application_Filename>
      <Application_Description>This service provides passenger distribution simulations</Application_Description>
      <Application_Keywords>Passenger distribution,simulations</Application_Keywords>
      <Application_Domain_Area>C:\Programs\Hullv2.exe</Application_Domain_Area>
      <Security_Protocol>All Security Protocols</Security_Protocol>
      <Communication_Protocol>SOAP</Communication_Protocol>
    - <Application_Inputs>
      - <Input Number="1">
          <Name>HullStructurev2.xml</Name>
        </Input>
      </Application_Inputs>
    - <Application_Outputs>
      - <Output Number="1">
          <Name>Distributionv2.xml</Name>
        </Output>
      </Application_Outputs>
    </Java_Configuration_Message>
  </JWSI_Message>
```

Figure 6.8 *Testing the XML specification of the Configuration File*

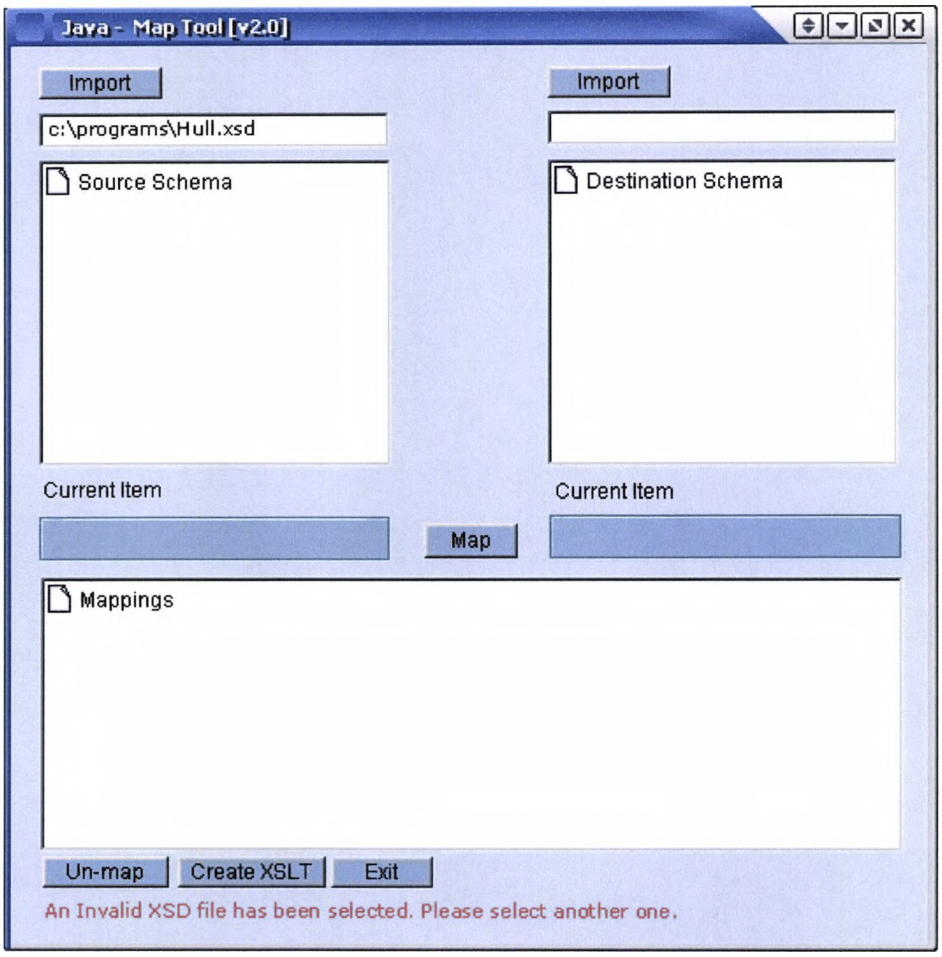
The test results taken from the above scenario indicate that the creation of the configuration file is functioning as expected.

By completing the above three scenarios, it can be concluded that the JWSI client peer is working according to the original requirements, design and implementation of the platform as well as according to user's feedback on the architecture of the platform.

Scenario 6

Testing the XML mapping Tool

This section addresses the testing of the XML mapping tool for cases when the XML transformation cannot be initiated and the user manual interaction is required. For the purpose of testing the XML mapper, an invalid XSD document is loaded in this component. The testing file has been deliberately tampered to contain invalid parameters and tags in order to demonstrate that the file will not be accepted by the component and an error message will be displayed to the user. Figure 6.9 shows the loading of an invalid XSD document.



c:\programs\Hull.xsd

An Invalid XSD file has been selected. Please select another one.

Figure 6.9 *Error message – XML mapping Tool*

As can be seen from the above figure, the XML mapping tool displays an error indicating that an invalid XSD document is loaded into the tree window. The user has to select another valid XSD document before the mapping takes place. This will guarantee that the XSLT created will be valid and compatible according to the system's specifications.

The above six scenarios present a comprehensive testing of the main features of the Java Peer component for both the client and the server site. All tests that have been performed in this section were successful.

6.3.6 Simulation Testing

In order to better test the robustness of the platform, several tools have been used to simulate a real environment where the two Java peers would have been communicating. One of the main tools used is the "PureLoad".

PureLoad is a load testing tool that simulates hundreds of users executing requests against applications. PureLoad can be used to verify that an application will meet the expected performance criteria. PureLoad reports quality and performance problems as well as detailed statistics gathered during a load test. PureLoad also includes extensive support for easy recording and testing of web based applications [89].

Some features of the PureLoad software are listed below:

- **Ease of use**

  PureLoad is designed to provide ease of use. A test session is managed using the PureLoad Console which is the central point of control. Using the console it is easy to define the virtual users, design the different simulations scenarios, and execute and evaluate the results.

- **Web testing**

  "PureLoad" can be used to test a wide range of applications but offers additional support to make it easier testing web applications.

  The PureLoad Web Recorder captures all requests between a web browser and the web application. These requests are then transformed into PureLoad scenarios for use in a load test session.

  The majority of web and application servers have been verified with PureLoad, including BEA WebLogic, Apache, Oracle 8i IAS, i-Planet, Microsoft IIS.

- **Extensive Reporting**

  PureLoad reports response time, failing requests, bytes transferred, and other parameters, in chart or text format. The resulting information from several load sessions can easily be visualised and compared in the PureLoad Result Comparer.

- **Fully distributed and platform independent**

  All components in PureLoad are platform independent and the runtime architecture is fully distributed. This powerful combination enables the use of single and multi CPU machines all mixed in a distributed environment using different OS systems.

Using the above simulation software it can be concluded that the system has been tested extensively and behaved as expected and according to simulation scenarios.

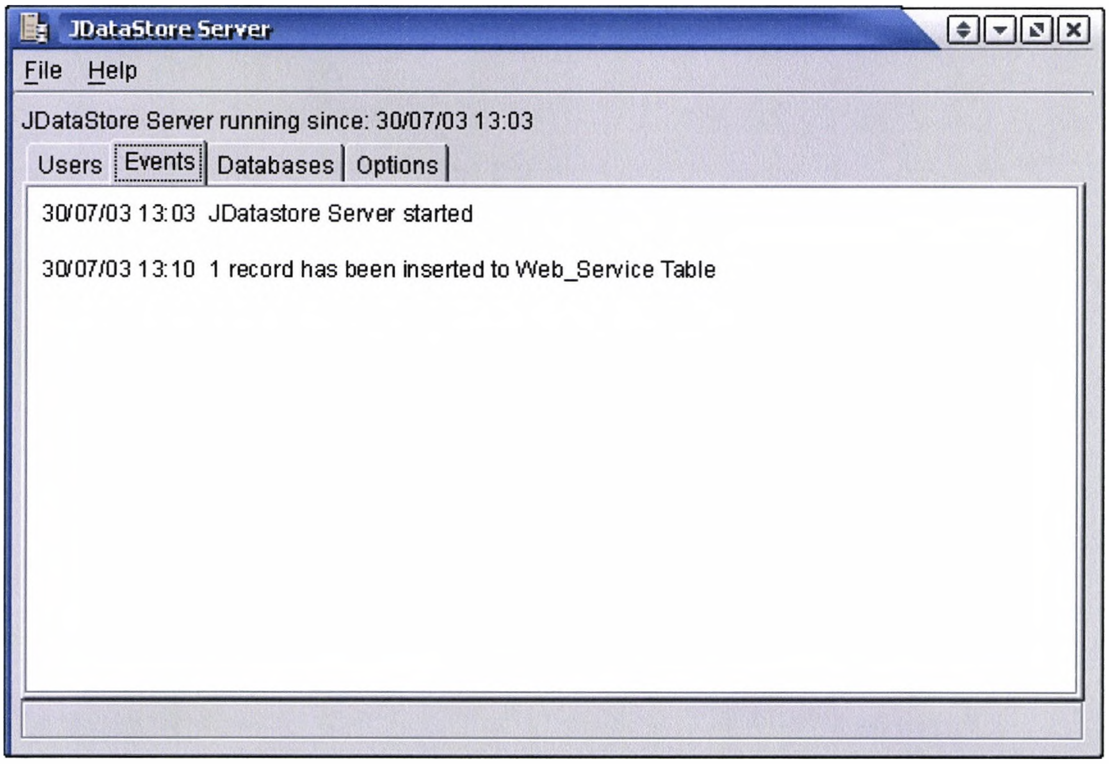
6.3.7 Additional Testing

Both JWSI applications on the server and client site were tested for load balancing and performance. These two factors are essential as they may compromise the integrity and robustness of the overall system. They are difficult to be performed since there are currently no standard methods that would guarantee the integrity and robustness of any system. For the purpose of testing the above two factors, different techniques have been used.

A number of JWSI client applications have been executed simultaneously, both on the same and on different Computers. This is done to simulate an environment of a number of users (i.e. 20) concurrently logged into the JWSI application. Then, an attempt was made by each JWSI client application to send different file types to the JWSI on the server site for the purpose of dynamic integration.

The results taken from this test scenario were successful since the JWSI on the server site was able to handle all these transactions simultaneously without compromising the integrity of the platform. The robustness of the system was verified by ensuring that no errors were received from both peer applications. This indicated that the JWSI peers could handle all transactions even at high volumes of data being sent via the communication lines.

In addition, the *JDataStore* database has also been tested. This has been achieved by checking the records that were inserted into the database every time a new service was dynamically discovered. Moreover, the robustness of the database has been tested by monitoring, via the JDataStore Server tool, the incoming and outgoing events. This is illustrated in figure 6.10.



JDataStore Server running since: 30/07/03 13:03

```
30/07/03 13:03  JDatastore Server started

30/07/03 13:10  1 record has been inserted to Web_Service Table
```

Figure 6.10 *Events under the JDataStore Server Tool*

To sum up, it can be seen that by completing the test in section 6.3.7, the testing process was completed and the system was working according to expected results.

6.4 CONCLUDING REMARKS

In this Chapter, the various methodologies used for the evaluation and testing phase of the prototype system have been discussed. It can be concluded that the evaluation and testing phase was successful with a significant amount of information collected as feedback, which subsequently enabled relevant modifications to the system.

Taking into account the complexity and the size of this prototype, it has been decided that the test scenarios discussed in this Chapter, for both JWSI peer sites, were adequate to determine the quality of the system produced.

Chapter 7

CONCLUSIONS AND FURTHER WORK

7.1 THESIS SUMMARY

The research presented in the preceding Chapters has basically covered all aspects needed to be taken into consideration in order to develop an innovative and universal system architecture. The system architecture was used for the development of an open technology platform, which supports through the latest Peer-to-Peer and Java Web Services technology dynamic application integration. The primary accomplishments and contributions of the research are discussed in the current Chapter.

7.2 RESEARCH CONTRIBUTIONS

i. **Dynamic Creation of Java Web Services**

   A novel Java Component has been developed, which is one of the most important contributions of this Thesis. This innovative component is capable of creating a definition for Java Web Services (JWS) and it is able to dynamically publish it on the Internet in order to be discovered by other applications. Other software packages need to develop software code in order to produce web services for integration. In this Thesis, the creation of a Java web service can be done by anybody using an easy-to-use wizard and does not require any technical knowledge or software code to be written. This has been achieved using the latest state-of-the-art Internet technologies and protocols to give a new meaning to the application integration problem.

ii. Dynamic Discovery and Publishing of Java Web Services

Another equally important contribution of this Thesis is the dynamic discovery and publishing of Java Web Services, since the proposed system, in order to provide a P2P architecture, should be able to dynamically discover and publish JWS using the Universal Description, Discovery and Integration (UDDI) registry. This is important because in order for a P2P system to function properly, it requires a global registry where all Java Web Services will be registered so as to be discovered dynamically by Java peers. The innovation of this architecture is the publishing and subsequently, the discovery of Java Web Services, based on specific criteria and techniques as explained in Chapter 3. Only the JWS that fully comply with the parameters of the requested application are downloaded by Java peers, in order to be executed and used during the integration process. This has been implemented so as to avoid redundant Web Services to be downloaded on the client site that are not compatible with the application requesting to be integrated.

iii. Dynamic Application Integration

Another contribution of the Thesis is the achievement of dynamic application integration. In the context of this Thesis, dynamic means that any application may be integrated and thus interchange data with another application on the Network or Internet, dynamically, without prior knowledge of the underlying infrastructure or data structure of the other application. This is accomplished with the help of the innovative Java Web Service Integrator (JWSI) peer (agent) that has been specifically developed for this Thesis. The JWSI generates Java Web Services on the fly, that is dynamically, while the potential applications to be integrated are running. To establish a communication link between the two dynamic integrated applications, peers are required to act as agents on the client's and server's machine, to send and receive requests from both locations. These peers are universal enough and can be installed in different machines under different operating systems because they are developed using the universality accepted platform independent Java language.

iv. Provide Peer to Peer Technology

This is also one of the most important contributions of this Thesis, since the architecture of the platform that was developed is based on the peer-to-peer technology. Using the platform derived from this Thesis, a P2P enabled application is capable of locating other peers in the network. Once an application is able to locate other peers, it can communicate with them using messages. Once the communication is established with other peers, the application can receive and provide information that was extracted, either from the database or the application itself. The advantage of using a P2P architecture is the fact that the two applications requesting to be integrated are linked with each other directly without any intermediate Integration servers like the "*hub-and-spoke*" architecture, as illustrated in figure 1.1, Chapter 1.

v. Provide dynamic XSLT transformation

Another contribution of this Thesis is the dynamic XSLT transformation mechanism. Occasionally, the web services are not capable of handling all the relevant information needed to be exchanged between two peers. In such a case, a mechanism is required to dynamically convert the data structure from the source peer to the data structure of the destination peer. This is achieved via the use of the XSLT language. This language creates the mappings of two different data structures by describing a template that needs to be applied to the given data structure. The user of each application usually creates these mappings manually. The innovation and contribution of this Thesis is to dynamically generate these mappings using the XML Mapper tool and hence dynamically apply them to transform the given data format to the destination.

In addition, no mechanisms currently exist for agreeing on message formats. More importantly, there are no Java Web Services in place that supply application-independent functionality, such as supporting security, transactions, authentication or message transformation. Such issues have also been addressed in this Thesis.

vi. Dynamic Database Integration

Another contribution of this Thesis is the dynamic database integration. Frequently, applications may use databases to store their information and subsequently, these data must be exported in a format that is readable by the requested peer. This has been also achieved using the XSLT generator, which transforms the data stored into the database to XML and vice-versa.

Moreover, a web service may be connected directly to a database engine without an intermediate application, for the purpose of retrieving some information. By this way, the dynamic database integration is achieved, since different database sources can be synchronised easily.

vii. Provide SOAP and WSDL specification compatibilities

Another contribution of this Thesis is the ability to generate dynamic Java Web Services, which comply with the Simple Object Access Protocol (SOAP) and the Web Services Description Language (WSDL) specification. This contribution is significant in cases where new or external systems may wish to discover and use the already published Java Web Services produced by the system. This makes the system extremely flexible and adaptable to future collaboration with other systems, which are already compatible with these standard specifications. A well-designed dynamic integration platform should have this functionality since newly developed systems have the need to integrate with other systems to provide a total solution to the client.

viii. Provide transfer and security protocols mechanisms

Another important contribution of this Thesis is the dynamic selection and negotiation of different transfer and security protocols between the two peers. Firstly, this is accomplished by having the two peers negotiate the appropriate transfer protocol based on the bandwidth availability of the communication between them. Secondly, based on some parameters found on each Java Web Service, the security protocol is dynamically selected, in case the information sent is confidential, and based on the availability of that protocol between the peers. Sometimes, the information sent is not so confidential and the two peers may decide not to use any security protocols for that session. This is important as it saves bandwidth, since sometimes the communication link, for example a satellite link between the ships and the shore offices, is very expensive especially for large data transfers.

From the above achievements and contributions it can be concluded that the developed proposed system fulfils all the initial requirements that have been identified during the analysis phase of this Thesis. Some minor issues, related to the implementation restrictions of each programming language have been identified. These issues were corrected by combining the advantages of more than one software technology and/or programming language together, to give a new meaning to the software development process and yet at the same time to accomplish the development of an innovative approach for dynamic application integration that will benefit the entire software industry.

However, in order for the full potential of the developed system to be realised in real life applications and to become accepted by all industries, some enhancements may be considered for future work. These recommendations are discussed in detail in section 7.

7.3 RECOMMENDATIONS AND FURTHER WORK

A number of feasible enhancements and extensions may be considered as candidates for extending the system developed by the reported research. These are discussed below:

- **An advanced authentication technique**

  Currently, the authentication technique that was specifically developed for this Thesis is not so advanced and secure. This is because the author wanted to prove a concept and not to concentrate on the authentication process. The user of the JWSI is authenticated based on a username and password that exists in the Java database.

If in case an unauthorised user attempts to manipulate the system, can easily extract with the aid of some technical knowledge of the Java database engine, the username and password from the Java database, then the system's security is compromised. This component can be improved significantly by the following recommendations:

- Firstly, the username and password can be stored in the Java database as encrypted text and only the system will be able to decrypt these confidential values based on an encryption algorithm. Currently, the username and password are stored as plain text.
- Secondly, it would be preferable for the system to be activated after it is installed on the client's machine. This can be achieved by providing an activation code based on the hardware of the installation machine. This will ensure that the system will not be copied to any machine without the prior notification of the platform's supplier.

❑ **Database connectivity**

Currently, the system supports only one database driver, the SQL driver. Some companies may have information and data in other database engines. The system can be extended so that more database drivers are connected to the system. The ability to connect to a wider range of database vendors will enable more companies to use the proposed architecture since the system will be able to read information from databases that are currently being used by companies that wish to use the dynamic integration platform. As long as drivers exist for each database vendor, then each driver can be easily integrated to the dynamic platform.

❑ **Ontology based discovery of web services**

Currently, the platform has been designed in such a way so as to discover new Java web services from the UDDI, the system should perform a comparison between the configuration file of the local application with the description of the Java web service that is registered on the UDDI. However, this kind of comparison is no that advanced, since many people may use different words to describe similar web services. In such a case, the mechanism for discovering the web services will not be in a position to find the correct web service for the purpose of dynamic integration. This design limitation is due to the fact that the author wanted to prove the dynamic discovering concept and not to implement an advanced technique for precise web service discovery, as this was out of the scope of this Thesis.

However, to improve this limitation, the architecture may be extended to support dynamic discovery with the help of Ontologies. Ontology is an explicit conceptualisation that describes the semantics of the data. Ontologies provide a shared and common understating of a domain that can be communicated across people and application systems. Hence, Ontologies may play an important role in discovering new Web Services which have similar functionality in the same domain area [47].

- **Enhanced XSLT transformations**

  Currently, the XSLT transformations can handle only the top level of XML data. That is, if more nested levels of XML tags exist in an XML document, then these cannot be transformed properly and as a result, the whole dynamic integration process may collapse. This also applies when the information is retrieved from the database and the corresponding XML files are created. A more complicated XSLT transformation mechanism could be developed to accommodate an unlimited number of data levels. This can be achieved by storing temporarily the "*level*" position of each XML tag, so that it can be used during the dynamic generation of the XSLT. By developing this functionality, the dynamic integration platform could handle any type and any level of XML tags.

In conclusion, the overall aim of this thesis was achieved to the maximum by developing successfully a system that will allow the dynamic integration of different software applications and information sources, including legacy systems, using the latest state of the art Internet technology called "*Peer-to-Peer*".

Appendix A

Java Classes

To describe all elements of the source code that was written for the purpose of developing the peer-to-peer platform described in this Thesis, is neither practical nor useful, so it has been decided that only the abstract classes would be included in this section.

The following classes have been used to implement the Java Web Service Integrator.

Class LoginScreen

This class is responsible for validating users based on their account on the Java Database Engine as well as to retrieve the appropriate tasks and actions for each user.

```
package JWSI;

/*
 * Title:          Java Web Service Integrator
 * Description:    Java Web Service Integrator
 * Copyright:      Copyright (c) 2003
 * @author         Christofi Stelios
 */


// Imports
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import javax.swing.border.*;
import javax.swing.tree.*;
import java.io.*;
import java.net.*;
import java.beans.*;
import java.net.URLEncoder;
import com.borland.datastore.jdbc.*;
import java.sql.*;


public class loginscreen extends JFrame {


 // Constructors
 public loginscreen() { }


// Methods
 protected void processWindowEvent(WindowEvent e) {  }
 public static void main(String[] args) {  }
 private void jbInit() throws Exception { }
 void button_cancel_actionPerformed(ActionEvent e) {  }
 void username_textValueChanged(TextEvent e) {  }
 void button_ok_actionPerformed(ActionEvent e) {  }
 void Check_Password()  {  }
 void Check_Username()  {  }
 void Load_Roles(int intUserID)  {  }
  void Load_Tasks()  {  }
 void username_keyPressed(KeyEvent e) { }
 void password_keyPressed(KeyEvent e) { }
 void this_keyPressed(KeyEvent e) {  }
 void password_keyReleased(KeyEvent e) {  }
 void proxy_itemStateChanged(ItemEvent e) {  }


} // End of class loginscreen
```

Class JavaIntegrator

This class is responsible for loading the main window of the Java Web Services Integrator and allows the user to navigate through the various business documents using the different tasks.

```
package JWSI;


/*
 * Title:          Java Web Service Integrator
 * Description:    Java Web Service Integrator
 * Copyright:      Copyright (c) 2003
 * @author         Christofi Stelios
 */


// Imports
import com.borland.datastore.jdbc.*;
import java.awt.*;
import javax.swing.*;
import javax.swing.tree.*;
import javax.swing.border.*;
import java.awt.event.*;
import javax.swing.event.*;
import java.net.*;
import java.text.DateFormat;
import java.util.*;
import java.io.*;
import org.w3c.dom.*;
import org.w3c.dom.Document;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.sql.ResultSet;
import javax.swing.table.*;
import javax.swing.table.DefaultTableCellRenderer;
import javax.swing.table.JTableHeader;
import javax.swing.table.TableCellRenderer;
import javax.xml.parsers.*;
import org.xml.sax.SAXException;
import org.xml.sax.*;
import com.borland.dx.dataset.*;
import com.borland.dbswing.*;
import com.borland.dx.sql.dataset.*;
import com.borland.internetbeans.*;


public class JavaIntegrator extends JFrame{


 // Constructors
 public JavaIntegrator() { }


// Methods
 protected void processWindowEvent(WindowEvent e) { }
 private void jbInit() throws Exception {
 protected static void visible(boolean value) { }
 void loadcurrentroles() { }
 void jTree1_mousePressed(MouseEvent e) { }
 void showmainwindow(String selectedfolder) { }
 void showactions() { }
 void folder1_mousePressed(MouseEvent e) { }
 void folder2_mousePressed(MouseEvent e) { }
 void folder3_mousePressed(MouseEvent e) { }
 void folder4_mousePressed(MouseEvent e) { }
 void folder5_mousePressed(MouseEvent e) { }
 void folder6_mousePressed(MouseEvent e) { }
 void button1_actionPerformed(ActionEvent e) { }
 void action1_mousePressed(MouseEvent e) { }
 void action2_mousePressed(MouseEvent e) { }
 void action3_mousePressed(MouseEvent e) { }
 void action4_mousePressed(MouseEvent e) { }
 void performaction(String BusDoc,String selection) { }
 void view_label_mousePressed(MouseEvent e) { }
 void jPanel3_mouseEntered(MouseEvent e) { }
 void action4_mouseEntered(MouseEvent e) { }
 void action3_mouseEntered(MouseEvent e) { }
 void action2_mouseEntered(MouseEvent e) { }
 void action1_mouseEntered(MouseEvent e) { }
 void view_label_mouseEntered(MouseEvent e) { }
 void view_label_mouseExited(MouseEvent e) { }
 void about_label_mousePressed(MouseEvent e) { }
 void about_label_mouseEntered(MouseEvent e) { }
 void about_label_mouseExited(MouseEvent e) {  }
 void Existing_Service_MainWindow(String title)  { }
 void Business_Document(String Title,String[] FieldNames,String[] DisplayNames,int
FieldNumber,String sql)  { }
 void New_JavaWebService_MainWindow(String title)  { }
 public InputStream GetResource(String filename)  {  }
 void readfromXMLfileToTree(String filename1){ }
 void checksettingschild(Node child1)  {   }
 void exit_label1_mouseClicked(MouseEvent e) {  }
 void exit_label1_mouseEntered(MouseEvent e) {  }
 void exit_label1_mouseExited(MouseEvent e) {  }
 void exit_label1_mousePressed(MouseEvent e) {  }
 void map_label_mouseClicked(MouseEvent e) {  }
 void map_label_mousePressed(MouseEvent e) {  }
 void map_label_mouseReleased(MouseEvent e) {  }
 void map_label_mouseEntered(MouseEvent e) {  }
 void map_label_mouseExited(MouseEvent e) {  }
 void RoleList_itemStateChanged(ItemEvent e)  {  }
 void jTree2_mouseMoved(MouseEvent e) {  }
 void jTree2_mousePressed(MouseEvent e) { }
 public class CustomTableCellRenderer extends DefaultTableCellRenderer  {  }
 class JComponentCellRenderer implements TableCellRenderer  {}


} // End of class JavaIntegrator
```

Class Actions

This class is responsible for executing the relevant actions that are selected by each user.

```
package JWSI;


/*
 * Title:          Java Web Service Integrator
 * Description:    Java Web Service Integrator
 * Copyright:      Copyright (c) 2003
 * @author         Christofi Stelios
 */


// Imports
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
import java.net.*;
import java.util.Date;
import java.text.DateFormat;
import java.lang.Long;

public class Actions  extends JDialog  implements ActionListener
{

  // Constructors
   public Actions ()  {  }

 // Methods
  protected void processWindowEvent(WindowEvent e) {  }
  public void addNotify ()   {  }
  public void actionPerformed ( ActionEvent event)   {   }
  private void jbInit() throws Exception { }
  void upload_ok_actionPerformed(ActionEvent e) {  }
  void execute(String BusDoc,String action,String settingfile,JTree jTree1) {  }
  void viewschema()  { }
  void viewsettings(String setfile,JTree jTree1) {  }
  void startuploads(String setfile,JTree jTree1) {  }
  String onlyfile(String temp1) {  }
  void startdownloads(String setfile,JTree jTree1)  {  }
  void createservice()  {  }
  void addrules()  {  }
  void viewservice()  {  }
  void uploadimages(String setfile,JTree jTree1)  {  }
  void databasewizard(String action)  {  }
  void WorkFlowService(String action)  {  }
  void KBosWorkFlowService(String action) {  }
  void ClientConfiguration(String action)  {  }
  void CreateJavaWebService(String action)  {  }
  void CreateOrganisationDetails(String action)  {  }
  void CreateNewServices(String action) {  }

} // End of class Actions
```

Class DownloadFiles

This class is responsible for downloading any files from the server peer to the client peer in order to be used by the requesting application to be integrated.

```
package JWSI;


/*
 * Title:         Java Web Service Integrator
 * Description:   Java Web Service Integrator
 * Copyright:     Copyright (c) 2003
 * @author        Christofi Stelios
 */
```

```
// Imports
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
import java.net.*;
import java.util.Date;
import java.text.DateFormat;
import java.lang.Long;


public class DownloadFiles  extends JDialog   implements ActionListener
{

   // Constructors
   public DownloadFiles ()  {  }

  // Methods
  protected void processWindowEvent(WindowEvent e) { }
  public void addNotify ()  {  }
  public void actionPerformed (  ActionEvent event)  {  }
  private void jbInit() throws Exception  {  }
  void upload_ok_actionPerformed(ActionEvent e) {  }
  void stop_button_actionPerformed(ActionEvent e) {  }
  void start_button_actionPerformed(ActionEvent e) {  }

  //Class that implements the downloading of xml file
  public class  downloadXML extends Thread
  {

  public void run() {  }
  public downloadXML() { }

  } //end of downloadXML

  void downloadxmlfile(String save_file,String filename) {  }


} // End of class DownloadFiles
```

Class MapTool

This class is responsible for the XML mappings between the client peer and the server peer. The output of this class is an XSLT file.

```
package JWSI;

/*
 * Title:          Java Web Service Integrator
 * Description:    Java Web Service Integrator
 * Copyright:      Copyright (c) 2003
 * @author         Christofi Stelios
 */

// Imports
import com.borland.datastore.jdbc.*;
import java.sql.*;
import javax.xml.transform.Templates;
import javax.xml.transform.Transformer;
import javax.xml.transform.stream.StreamSource;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.OutputKeys;
import org.xml.sax.SAXException;
import org.xml.sax.*;
import javax.xml.parsers.*;
import org.w3c.dom.Document;
import org.w3c.dom.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
import java.awt.datatransfer.*;
import java.awt.dnd.*;
import java.awt.dnd.peer.*;
// JDOM classes used for document representation
import java.util.Iterator;
import java.util.List;
import java.net.URL;
import javax.swing.tree.*;
import java.io.*;
import java.util.zip.*;

public class maptool  extends JDialog   implements ActionListener
{

  // Constructors
   public maptool ()   {  }

  // Methods
  protected void processWindowEvent(WindowEvent e) {  }
  public void addNotify ()  {  }
  public void actionPerformed (  ActionEvent event)  { }
  private void jbInit() throws Exception { }
  void create_XSLT(String XSLT_file) { }
  void button1_actionPerformed(ActionEvent e) {  }
  void jTree11_mouseMoved(MouseEvent e) {  }
  void jTree21_mouseMoved(MouseEvent e) { }
  void jTree11_mousePressed(MouseEvent e) {}
  void jTree21_mousePressed(MouseEvent e) { }
  void loadschemasource(String source)  {  }
  void loadschemadestination(String destination)  {  }
  protected static void execute_map_button()  {  }
  void map_button_actionPerformed(ActionEvent e) {  }
  void parseschema(DefaultMutableTreeNode tempmainRoot,String txt_schema)  {  }
  void import_button_actionPerformed(ActionEvent e) {  }
  void import_button1_actionPerformed(ActionEvent e) {  }
  void button2_actionPerformed(ActionEvent e) { }
  void Remove_mapping_actionPerformed(ActionEvent e) {  }
  void create_XML_actionPerformed(ActionEvent e) { }
  void exportToTxt(String xml_file) {  }
  int checkchild(Node child1,String temparray[])  {  }
  void exportToXML(String txt_schema)  {  }
  void transformxml(String sourcexml,String xslt_file,String outfile)  {  }
  void Help_button_actionPerformed(ActionEvent e) { }
  void create_ZIP_actionPerformed(ActionEvent e) { }
  void create_ZIP1_actionPerformed(ActionEvent e) {  }

} // End of class maptool
```

Class CreateJavaWebService

This class is responsible for creating and publishing the Java Web Service.

```
package JWSI;


/*
 * Title:          Java Web Service Integrator
 * Description:    Java Web Service Integrator
 * Copyright:      Copyright (c) 2003
 * @author         Christofi Stelios
 */


// Imports
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
import java.net.*;
import java.util.Date;
import java.text.DateFormat;
import java.lang.Long;

public class CreateJavaWebService  extends JDialog  implements ActionListener
{

  // Constructors
  public CreateJavaWebService ()  {  }

 // Methods
 protected void processWindowEvent(WindowEvent e) { }
 public void addNotify ()  {  }
 public void publish( ){  }
 public void actionPerformed ( ActionEvent event)  {   }
 private void jbInit() throws Exception  { }
 void upload_ok_actionPerformed(ActionEvent e) { }
 void CreateService_ok_actionPerformed(ActionEvent e) {  }
 void browse_button1_actionPerformed(ActionEvent e) {  }

} // End of class CreateJavaWebService
```

Class OrganisationDetails

This class is responsible for saving the organization details to be published along with the Java Web Service.

```
package JWSI;

/*
 * Title:          Java Web Service Integrator
 * Description:    Java Web Service Integrator
 * Copyright:      Copyright (c) 2003
 * @author         Christofi Stelios
 */

// Imports
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
import java.net.*;
import java.util.Date;
import java.text.DateFormat;
import java.lang.Long;

public class OrganisationDetails  extends JDialog   implements ActionListener
{

 // Constructors
   public OrganisationDetails ()   { }

  // Methods
  protected void processWindowEvent(WindowEvent e) {  }
  public void addNotify ()  {  }
  public void actionPerformed (  ActionEvent event)  {  }
  private void jbInit() throws Exception {  }
  void Save_ok_actionPerformed(ActionEvent e) {  }
  void upload_ok_actionPerformed(ActionEvent e) {  }

} // End of class OrganisationDetails
```

Class SOAPClient

This class is responsible for initiating the Java Web Service via the SOAP protocol once it has been discovered.

```
package JWSI;


/*
 * Title:          Java Web Service Integrator
 * Description:    Java Web Service Integrator
 * Copyright:      Copyright (c) 2003
 * @author         Christofi Stelios
 */

// Imports
import javax.wsdl.Definition;
import javax.wsdl.Port;
import javax.wsdl.WSDLException;
import javax.wsdl.extensions.ExtensibilityElement;
import javax.wsdl.extensions.soap.SOAPAddress;
import javax.wsdl.factory.WSDLFactory;
import javax.wsdl.xml.WSDLReader;
import javax.xml.namespace.QName;
import javax.xml.rpc.ServiceException;
import java.net.MalformedURLException;*/
import java.net.*;
import java.io.*;
import java.rmi.RemoteException;
import java.util.*;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.*;


public class SOAPClient {

  // Methods
  public byte[] PopMessagemarshall()  {  }
  public byte[] AcknowledgeReceiptmarshall(String refID)  {  }
```

```
   public byte[] NotifyForProcessingCompletionmarshall(String refID)  {  }
   public String postPopMessage(String FunctionName)   {  }
   public String postAcknowledgeReceipt(String FunctionName,String refID)   {   }
   public String postNotifyForProcessingCompletion(String FunctionName,String refID)   {   }
   public void ParseXml(String xmldata)   {      }
   public String PopMessage()   {   }
   public void discover (){  }
   public String AcknowledgeReceipt(String RefID)   {   }
   public String NotifyForProcessingCompletion(String Message)   {   }

} //End of class SOAPClient
```

Class Configuration

This class is responsible for the configuration of the application that needs to be integrated.

```
package JWSI;



/*
 * Title:          Java Web Service Integrator
 * Description:    Java Web Service Integrator
 * Copyright:      Copyright (c) 2003
 * @author         Christofi Stelios
 */


// Imports
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
import java.net.*;
import java.util.Date;
import java.text.DateFormat;
import java.lang.Long;

public class Configuration  extends JDialog   implements ActionListener
{
  // Constructors
  public Configuration ()  {  }

  // Methods
  protected void processWindowEvent(WindowEvent e) { }
  public void addNotify ()  { }
  public void actionPerformed ( ActionEvent event)  {  }
  private void jbInit() throws Exception { }
  void CreateConfiguration_ok_actionPerformed(ActionEvent e) { }
  void browse_button_actionPerformed(ActionEvent e) {  }
  void Save_ok_actionPerformed(ActionEvent e) {  }

} // End of class Configuration
```

Class UploadFiles

This class is responsible for uploading any files from the client peer to the server peer for integration process to take place.

```
package JWSI;


/*
 * Title:          Java Web Service Integrator
 * Description:    Java Web Service Integrator
 * Copyright:      Copyright (c) 2003
 * @author         Christofi Stelios
 */

// Imports
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
import java.net.*;
import java.util.Date;
import java.text.DateFormat;
import java.lang.Long;
```

```
public class UploadFiles  extends JDialog   implements ActionListener
{

 // Constructors
  public UploadFiles ()   {  }



// Methods
 protected void processWindowEvent(WindowEvent e) {  }
 public void addNotify ()   {  }
 public void actionPerformed (   ActionEvent event)   {  }
 private void jbInit() throws Exception {  }
 void upload_ok_actionPerformed(ActionEvent e) { }
 void stop_button_actionPerformed(ActionEvent e) {  }
 void start_button_actionPerformed(ActionEvent e) { }

 public class  checkfolder extends Thread
 {

 public void run() {  }
 public checkfolder() {  }
 public void checkcommand()  {  }

 } //end of checkfolder

 //Class that implements the sending of xml file
 public class  sendxml extends Thread
 {

 public void run()  {  }
 public sendxml() {  }

 } //end of sendxml

 void sendxmlfile(String filename) {  }


} // End of class UploadFiles
```

Class DatabaseConnect

This class is responsible for connecting to different database vendors and extract information that will be sent to any application requesting to be integrated.

```
package JWSI;

/*
 * Title:          Java Web Service Integrator
 * Description:    Java Web Service Integrator
 * Copyright:      Copyright (c) 2003
 * @author         Christofi Stelios
 */


// Imports
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
import com.borland.datastore.*;
import java.text.DateFormat;
import java.util.Date;
import java.sql.*;
import javax.sql.*;
import sun.jdbc.rowset.CachedRowSet;
import javax.swing.border.*;
import oracle.xml.sql.query.OracleXMLQuery;
import oracle.xml.sql.query.*;
import oracle.jdbc.driver.*;
import java.lang.*;
import org.w3c.dom.*;

public class DatabaseConnect  extends JDialog   implements ActionListener
{


  // Constructors
  public DatabaseConnect (String action)   {  }
// Methods
 protected void processWindowEvent(WindowEvent e) { }
 public void addNotify ()  {  }
 public void actionPerformed ( ActionEvent event)  { }
 private void jbInit() throws Exception { }
void button1_actionPerformed(ActionEvent e) { }
void CheckAction()  { }
void checkmappings()  { }
void help_button_actionPerformed(ActionEvent e) {  }

class Connect {

      public Connect() {}
      public void run() { }

 } // End of connect class

void Start_button_actionPerformed(ActionEvent e) { }
void exportToText()  {  }
void exportToXML()  { }
void Connectdb_actionPerformed(ActionEvent e) {  }
void SelectTables() { }
void Select_Tables_actionPerformed(ActionEvent e) {  }
void fieldnames_itemStateChanged(ItemEvent e) { }
void LoadSchemaValues() { }
void LoadValues()  { }
void Delete_Database_Connection(int Action)
void Update_Database_Connection()  {    }
void Save_Database_Connection()  {   }
void exporttype_itemStateChanged(ItemEvent e) {  }
void exporttype1_itemStateChanged(ItemEvent e) {  }
void Select_Mappings_actionPerformed(ActionEvent e) {  }
void save_button_actionPerformed(ActionEvent e) { }
void this_focusGained(FocusEvent e) { }
void this_windowActivated(WindowEvent e) { }


} // End of class DatabaseConnect
```

Class Map_Database

This class is responsible for creating the mappings between the database and the XML file. Then, these mappings will be converted to a format readable by the requesting application in order to achieve dynamic integration.

```
package JWSI;


/*
 * Title:         Java Web Service Integrator
 * Description:   Java Web Service Integrator
 * Copyright:     Copyright (c) 2003
 * @author        Christofi Stelios
 */


// Imports
import com.borland.datastore.jdbc.*;
import org.apache.xalan.processor.TransformerFactoryImpl;
import java.sql.*;
import javax.sql.*;
import javax.xml.transform.*;
import javax.xml.transform.Templates;
import javax.xml.transform.Transformer;
import javax.xml.transform.stream.StreamSource;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.OutputKeys;
import org.xml.sax.SAXException;
import org.xml.sax.*;
import javax.xml.parsers.*;
import org.w3c.dom.Document;
import org.w3c.dom.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
import java.awt.datatransfer.*;
import java.awt.dnd.*;
import java.awt.dnd.peer.*;

// JDOM classes used for document representation
import java.util.Iterator;
import java.util.List;
import java.net.URL;
import javax.swing.tree.*;
import java.io.*;
import java.util.zip.*;


public class Map_Database  extends JDialog   implements ActionListener
{


  // Constructors
   public Map_Database ()  {  }


 // Methods
  protected void processWindowEvent(WindowEvent e) { }
  public void addNotify ()  {  }
  public void actionPerformed ( ActionEvent event)  {  }
  private void jbInit() throws Exception {  }
  void ClearjTrees()  {  }
  protected static void create_XSLT(String XSLT_file)  {  }
  void jTree11_mouseMoved(MouseEvent e) { }
  void jTree21_mouseMoved(MouseEvent e) {  }
  void jTree11_mousePressed(MouseEvent e) {}
  void jTree21_mousePressed(MouseEvent e) { }
  void loadschemasource(String source)  { }
  void loadschemadestination(String destination)  {  }
  void loadselectedfieldnames()  {  }
  void loaddestinationfieldnames(int xmlSchemaID)  { }
  void LoadMappings(int DatabaseID)  {  }
  void checkmappings()  {  }
  void execute_map_button()  {  }
  void map_button_actionPerformed(ActionEvent e) {  }
  void parseschema(DefaultMutableTreeNode tempmainRoot,String txt_schema)  { }
  void import_button_actionPerformed(ActionEvent e) {  }
  void import_New_Apply_Button_actionPerformed(ActionEvent e) {  }
  void button2_actionPerformed(ActionEvent e) {  }
  void Remove_mapping_actionPerformed(ActionEvent e) {  }
  void Apply_button_actionPerformed(ActionEvent e) {  }
  public void createXMLfilefromservice()  {  }
  public void createXMLfile()  {  }
  void exportToTxt(String xml_file)  {  }
  int checkchild(Node child1,String temparray[])  {  }
  void exportTxtToXML(String txt_schema)  {  }
  protected static void transformxml(String sourcexml,String xslt_file,String outfile)  { }
  void Help_button_actionPerformed(ActionEvent e) {  }
  void create_ZIP_actionPerformed(ActionEvent e) {  }

  public class  Create_XML extends Thread
  {

  public void run() {    }
  public void execute() {  }
  public void process_xml()  {  }
  public Create_XML()  {}

  } // End of Create_XML

  public void exportDatabaseToXML(String filename,int writer)  {  }
  void this_focusGained(FocusEvent e) { }
  void New_Apply_Button_actionPerformed(ActionEvent e) {  }
  void button_apply_actionPerformed(ActionEvent e) { }



} // End of class Map_Database
```

Class MessageBox

This class is responsible for creating popup windows and displays any messages to the user of the JWSI.

```
package JWSI;


/*
 * Title:         Java Web Service Integrator
 * Description:   Java Web Service Integrator
 * Copyright:     Copyright (c) 2003
 * @author        Christofi Stelios
 */



// Imports
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class MessageBox   extends JDialog   implements ActionListener

{

  // Constructors
  public MessageBox ( Frame parent)   { }



 // Methods
  public void setVisible ( boolean b)   { }
  public void addNotify ()   { }
  public void actionPerformed (  ActionEvent event)   { }
  public void setText ( String sMessage)  { }
  private void jbInit() throws Exception { }



} // End of class MessageBox
```

Appendix B

Data Gathering

The following methods have been used in order to collect the necessary requirements for the development of the system as well as significant data and comments have been received from the users of the system in order to evaluate and hence improve the system so that it can meet their needs.

Questionnaires

The following section lists some of the questionnaires that have been received from various users of the Consortium before, during and after the development of the system.

During Requirement Capturing

Company Name: NAPA Oy
Country: FINLAND
Tool Name: NAPA
Tool Description: **NAPA**, the Naval Architectural PAckage, is a CAE system for initial and basic ship design, comprising, among other things, hull surface definition, production-level fairing, definition of the ship's compartmentation and naval architectural calculations. Geometry definitions are based on a product model created by the system. NAPA can be used both for preliminary design and for the production of final delivery documents. It covers all applications needed in ordinary naval architectural design and can handle floating structures of any kind. **Features** - 3D modeling of the entire ship. - Standard naval architectural calculations. - Hydrodynamic calculations. - Report generator and drawing functions. - Communication with other design systems through numerous links and interfaces.
Operating System Supported: Windows (2000, NT, XP), Unix, Sun and H-P platforms
Number of Input Files: 1
Number of Output Files: 1
Actual Filename: Napa.exe
Comments: None

Company Name: SU-SSRC/ University of Strathclyde
Country: UNITED KINGDOM
Tool Name: EVE/ EVI
Tool Description: **EVE** is a passenger evacuation software which takes as input 3D modeling designs of a ship and distribute the passengers in order to find any discrepancies of the ship design. **Features** - Passenger evacuation simulation software. - Uses General Arrangement data.
Operating System Supported: Windows (2000, NT, XP)
Number of Input Files: 1 or 2
Number of Output Files: 1
Actual Filename: Eve.exe or Evi.exe
Comments: None

Company Name: TBS / TRIBON Solutions AB
Country: SWEEDEN
Tool Name: TRIBON
Tool Description: The Tribon Product Information Model (PIM) holds information and documentation about the ship design and about the manufacturing of the ship. Tribon M2 Data Management is an embedded and adapted data management functionality fully integrated with the Tribon PIM. Tribon M2 Data Management focuses on: • Control of bject status in the Tribon PIM • Revision handling for drawings • Access control of data and functions **Features** - Commercial software covering initial, basic and detailed design as well as production engineering and production.
Operating System Supported: Windows (2000, NT, XP)
Number of Input Files: 1
Number of Output Files: 1
Actual Filename: Tribon.exe
Comments: None

During System Development - Prototype

Company Name: SU-SSRC/ University of Strathclyde				
Country: UNITED KINGDOM				
Please put a tick ✓ where applicable.				
Usability of the system:	**Poor**	**Good** ✓	**Very Good**	**Excellent**
Interface of the system:	**Poor**	**Good**	**Very Good** ✓	**Excellent**
Functionality of the system:	**Poor**	**Good**	**Very Good** ✓	**Excellent**
Robustness of the system:	**Poor**	**Good** ✓	**Very Good**	**Excellent**
Please cross where NOT applicable.				
Did you have any problems during the installation of the system: ~~Yes~~ / No If Yes please specify				
Does the system satisfy your needs: Yes / ~~No~~ If No please specify why				
Do you need extra functionality: ~~Yes~~ / No If Yes please specify				
Is it compatible with your Operating System: Yes / ~~No~~ If No please specify the operating system you are using				
Have you noticed any bugs of the system: Yes / ~~No~~ If Yes please specify The system is slow when it is run on a computer with 128Mbytes Memory				
Please enter any other comments that you have regarding the system: None				

Company Name: NAPA Oy

Country: FINLAND

Please put a tick ✓ where applicable.

Usability of the system:	**Poor**	[✓ **Good**]	**Very Good**	**Excellent**
Interface of the system:	**Poor**	**Good**	**Very Good**	[✓ **Excellent**]
Functionality of the system:	**Poor**	**Good**	**Very Good**	[✓ **Excellent**]
Robustness of the system:	**Poor**	**Good**	[✓ **Very Good**]	**Excellent**

Please cross where NOT applicable.

Did you have any problems during the installation of the system: ~~Yes~~ / No If Yes please specify

Does the system satisfy your needs: Yes / ~~No~~ If No please specify why

Do you need extra functionality: Yes / ~~No~~ If Yes please specify
To be able to specify multiple input and multiple output files for a specify Design Tool.

Is it compatible with your Operating System: Yes / ~~No~~ If No please specify the operating system you are using

Have you noticed any bugs of the system: ~~Yes~~ / No If Yes please specify

Please enter any other comments that you have regarding the system:
The overall functionality of the system is very good.

Company Name: Ship Design and Research Centre				
Country: POLAND				
Please put a tick ✓ where applicable.				
Usability of the system:	**Poor**	**Good**	**✓ Very Good**	**Excellent**
Interface of the system:	**Poor**	**Good**	**Very Good**	**✓ Excellent**
Functionality of the system:	**Poor**	**Good**	**Very Good**	**✓ Excellent**
Robustness of the system:	**Poor**	**Good**	**Very Good**	**✓ Excellent**
Please cross where NOT applicable.				
Did you have any problems during the installation of the system: ~~Yes~~ / No If Yes please specify				
Does the system satisfy your needs: Yes / ~~No~~ If No please specify why				
Do you need extra functionality: Yes / ~~No~~ If Yes please specify To be able to accept connections from various Design Tools at the same time. This connection may be over Secure Socket Layer (SSL)				
Is it compatible with your Operating System: Yes / ~~No~~ If No please specify the operating system you are using				
Have you noticed any bugs of the system: ~~Yes~~ / No If Yes please specify				
Please enter any other comments that you have regarding the system: None				

After System Development – During real life testing

Company Name: SU-SSRC/ University of Strathclyde				
Country: UNITED KINGDOM				
Please put a tick ✓ where applicable.				
Usability of the system:	**Poor**	**Good** ✓	**Very Good**	**Excellent**
Interface of the system:	**Poor**	**Good**	**Very Good** ✓	**Excellent**
Functionality of the system:	**Poor**	**Good**	**Very Good** ✓	**Excellent**
Robustness of the system:	**Poor**	**Good** ✓	**Very Good**	**Excellent**
Easy integration with tools:	**Poor**	**Good**	**Very Good** ✓	**Excellent**
Please cross where NOT applicable.				
Does the transfer of files done correctly? Yes / ~~No~~ If No please specify the error messages:				
Does the system satisfy the initial requirements of the project: Yes / ~~No~~ If No please specify why:				
Does the system satisfy your needs: Yes / ~~No~~ If No please specify why				
Do you need extra functionality: ~~Yes~~ / No If Yes please specify				
Is it compatible with your Operating System: Yes / ~~No~~ If No please specify the operating system you are using				
Have you noticed any bugs of the system: ~~Yes~~ / No If Yes please specify				
Please enter any other comments that you have regarding the system: None				

Company Name: NAPA Oy				
Country: FINLAND				
Please put a tick ✓ where applicable.				
Usability of the system:	**Poor**	**Good** ✓	**Very Good**	**Excellent**
Interface of the system:	**Poor**	**Good**	**Very Good**	✓ **Excellent**
Functionality of the system:	**Poor**	**Good**	**Very Good**	✓ **Excellent**
Robustness of the system:	**Poor**	**Good**	**Very Good**	✓ **Excellent**
Easy integration with tools:	**Poor**	**Good**	**Very Good**	✓ **Excellent**
Please cross where NOT applicable.				
Does the transfer of files done correctly? Yes / ~~No~~ If No please specify the error messages:				
Does the system satisfy the initial requirements of the project: Yes / ~~No~~ If No please specify why:				
Does the system satisfy your needs: Yes / ~~No~~ If No please specify why				
Do you need extra functionality: ~~Yes~~ / No If Yes please specify				
Is it compatible with your Operating System: Yes / ~~No~~ If No please specify the operating system you are using				
Have you noticed any bugs of the system: ~~Yes~~ / No If Yes please specify				
Please enter any other comments that you have regarding the system: This version of the system is much better than the previous one in terms of robustness and functionality.				

Company Name: TBS / TRIBON Solutions AB				
Country: SWEEN				
Please put a tick ✓ where applicable.				
Usability of the system:	**Poor**	**Good** ✓	**Very Good**	**Excellent**
Interface of the system:	**Poor**	**Good** ✓	**Very Good**	**Excellent**
Functionality of the system:	**Poor**	**Good**	**Very Good** ✓	**Excellent**
Robustness of the system:	**Poor**	**Good**	**Very Good** ✓	**Excellent**
Easy integration with tools:	**Poor**	**Good**	**Very Good** ✓	**Excellent**
Please cross where NOT applicable.				
Does the transfer of files done correctly? Yes / ~~No~~ If No please specify the error messages:				
Does the system satisfy the initial requirements of the project: Yes / ~~No~~ If No please specify why:				
Does the system satisfy your needs: Yes / ~~No~~ If No please specify why				
Do you need extra functionality: ~~Yes~~ / No If Yes please specify				
Is it compatible with your Operating System: Yes / ~~No~~ If No please specify the operating system you are using				
Have you noticed any bugs of the system: ~~Yes~~ / No If Yes please specify				
Please enter any other comments that you have regarding the system: The robustness of the system is outstanding.				

Consortium Meetings

Below is a list of all tools that has been agreed by the consortium to be used for the Integration System during the various Consortium meeting of the project lifecycle.

Tool Name	Partner	Description
NAPA	Napa Oy	- 3D modeling of the entire ship. - Standard naval architectural calculations. - Hydrodynamic calculations. - Report generator and drawing functions. - Communication with other design systems through numerous links and interfaces.
SDL (Surface Design Library)	NTUA-SDL	- B-Spline Bezier curve/surface modeler. - Can communicate with CATIA, TRIBON and NAPA.
AVPRO	Principia	-This tool executes a design loop in the conceptual or initial phase of the design. - Define the main particulars. - Define the hull form (in a crude preliminary but simple -definition). - Check hydrostatics. - Check stability. - Check scantlings. - Define propulsion arrangement. - Define a preliminary GA. - Check weights and CG. - Estimate performance.
LBR5	Principia	- LBR5 carries out structural analysis, and scantling optimisation. - Especially dedicated to ship structure assessment, but can address any kind of stiffened structure.
MECHANICAL DESKTOP	UNEW	- Uses the AutoCAD core technology. - Developed for mechanical designers who prefer to create 3D designs in native AutoCAD software environment.
EVE / EVI	SU-SSRC	- Passenger evacuation simulation software. - Uses General Arrangement data.
POLYCAD	SU-SSRC	- PolyCAD is a geometry-editing tool developed with a lean towards Naval Architecture and Ship Design.
PROTEUS	SU-SSRC	- Damage Survivability Simulation
PARALLAX	SU-SSRC	- Damage Survivability Visualisation (from PROTEUS)

CFAST	SU-SSRC	- CFAST is a zone model that predicts the effect of a specified fire on temperatures, various gas concentrations and smoke layer heights in a multi-compartment structure.
MOCKUP-2002i2	UNIPATRAS	- A VR software platform based on CAD engineering to support the development of customised applications through its Development Toolkit.
BAL.DIS Catalogue	BALANCE	- Supply Chain e-Catalog system.
WITNESS (suite)	IZAR	- MATFLOW – Material flow planning system. - WITNESS – Simulation software. - WITNESS OPTIMIZER. - WITNESS VR – Virtual Reality Software. - Ability to construct a computer model of virtually any existing or proposed process, simulate its functions, and analyse its functions. - Virtual Prototyping.
DENEB / QUEST	IZAR	- Digital Manufacturing environment for modelling, analysing, validating and visualizing facility layout and process flow. - Analyse and optimise the throughput of the line, factory or even the extended production environment, including the suppliers
TRIBON	TBS	- Commercial software covering initial, basic and detailed design as well as production engineering and production.

Furthermore, during the consortium meetings several comments and suggestions have been received regarding the functionality and usability of the system in order to adjust it according to the user needs.

MSN Meetings

MSN meetings have been setup several times in order to check the robustness and usability of the system in real time from 4 different countries: United Kingdom, Athens, Germany and France. The following section lists some of the comments that have been received during these meetings.

Meeting # 1	
Partner	**Comments**
NAPA Oy	1) The system can not start. It returns an error saying that the password is incorrect. 2) Now the system is working. I have put an invalid password.
SU-SSRC	1) My screen resolution is 640x480 and I can not see the whole application. 2) Now I can see the application since I have increase my screen resolution to 800x600.
TBS	1) The system is working and it says that is ready to receive and send new files.
Ship Design and Research Centre	1) I have problems finding the application. 2) I found the applications, before I was login as a different user.

Meeting # 2	
Partner	**Comments**
SU-SSRC	1) The system can not send files. It says “Socket Error”. 2) Now the system can send files. Before my Internet connection with the ISP was cut out.
TBS	1) I have received a message on my screen that the file has been send correctly and is listening for incoming events. 2) The system behaves as expected with not errors.
NAPA Oy	1) The system has send and received three files without crashing. 2) If the system was running as a Windows service it would be fantastic.

Meeting # 3	
Partner	**Comments**
NAPA Oy	1) The system is running as a Windows service and is much better since when the Operating System starts I do not forget to initiate the integration system. 2) The system helps dramatically our integration processes and we save a lot of time in comparison with the manual integration.
SU-SSRC	1) The system is robust enough to handle any file size and is not limited to the number of concurrent connections from various Tools vendors.
TBS	1) The system is working fine with not bugs.
Ship Design and Research Centre	1) The system is communicating with the various design tools with no problems. 2) Since the system is running now as a Windows service it helps a lot the whole integration process.

The above comments and suggestions have helped dramatically to improve the overall system in order to satisfy the user needs.

References

[1] Adler, S., "*Extensible Stylesheet Language (XSL) Version 1.0*", W3C, 2001. Available at: http://www.w3.org/TR/xsl/ (accessed on 05.11.2001).

[2] Alhir, S. S., "*UML in a nutshell: a desktop quick reference*", Beijing, Cambridge : O'Reilly, 1998.

[3] Allen, D. W., "*Establishing an EAI Architecture*", EAI Journal, June 2001.

[4] ASC X12, "EDI Standards", ASC X12, 2002, Available at: http://www.x12.org (accessed on 20.07.2002).

[5] Atkinson, B., Della-Libera, G., Hada, S., Hondo, et al., "*Web Services Security (WS-Security) Version 1.0*", IBM, Microsoft and VeriSign, April 2002. Available at: ftp://www.software.ibm.com/software/developer/library/ws-secure.pdf (accessed on 05.04.2003).

[6] Balen, H., "*Deconstructing Babel: XML and application integration*", ADT ,December 2000.

[7] Baum, D., Dessaux, C., Talukdar, N., "*e-Business Integration*", Oracle Corporation, 2001. Available at: http://www.oracle.com/oramag/oracle/ 00-sep/o50cov.html (accessed on 10.12.2001).

[8] Bernd, O., "*Developing software with UML*", Addison-Wesley, 1999.

[9] Borland Software Corporation, "*Borland JDataStore*", Borland Software Corporation, 2003. Available at: http://www.borland.com/jdatastore/previous /index.html (accessed on 03.04.2003).

[10] Bourne, K. C., "*Testing client/server systems*", New York; London : McGraw-Hill, 1997.

[11] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., et al, "*Simple Object Access Protocol (SOAP) 1.1*", W3C Note, May 2000. Available at: http://www.w3.org/TR/SOAP/ (accessed on 14.06.2002).

[12] Bpmi.org, "*Business Process Modeling Language (BPML)*", Bpmi.org, 2002. Available at: www.bpmi.org/specifications.esp (accessed on 20.06.2002).

[13] Bradley, N., "*The XSL companion: styling XML documents*", London/New York: Addison-Wesley/Pearson Education, May 2000.

[14] Bray, T., et al, "*Extensible Markup Language (XML) 1.0 (Second Edition)*", W3C, 2000. Available at: http://www.w3.org/TR/REC-xml#dt-doctype (accessed on 17.12.2000).

[15] Bussler, C., *"P2P in B2BI",* Proceedings of the 35th Hawaii International Conference on System Sciences, 2002.

[16] Bussler, C., "*Semantic B2B Integration Server Technology as Infrastructure for Electronic Hubs*", Proceedings of First International Workshop on Electronic Business Hubs: XML, Metadata, Ontologies, and Business Knowledge on the Web (WEBH2001), Munich, Germany, September 2001.

[17] Cagle, K., **"***XSD Schema Tricks and Tips",* Wrox conferences, 2001. Available at: http://www.vbxml.com/xml/articles/xsd/ (accessed on 17.12.2000).

[18] Casati, F., et al., "*eFlow: A platform for developing and managing composite web services*" in Proc. AIWoRC'00, Buffalo NY, April 27–29, 2000, IEEE Comp. Society.

[19] Chang, R., "*Application Integration*", TechTarget, July 2001. Available at http:///www.searchWebServices.com (accessed on 20.10.2002).

[20] Chester, M., "*Middleware Cross-Platform Integration with XML and SOAP",* IT Professional, September 2001.

[21] Christensen, E., Curbera, F., Meredith, G., Weerawarana, S., "*Web Services Description Language (WSDL) 1.1*", W3C Note, March 2001. Available at: http://www.w3.org/TR/2001/NOTE-wsdl-20010315 (accessed on 15.06.2002).

[22] Christofi, S., "*A Reference System Architecture and Technology Platform for the Shipping Sector*", M. Phil Thesis, UMIST, Manchester, November 2001.

[23] Christofi, S., Karakostas, B., et al, "*XML based Architecture for Shipping Application Integration*", IEEE proceedings conference, Toronto, Canada, May 2001.

[24] Christofi, S., Karakostas, B., Turega, M., "*An XML and Java Based Approach to Application Integration in Shipping*", Proc. of the Third International Conference on Information Integration and Web-based Applications & Services (IIWAS 2001), Vol. 150, Vienna, OCG, pp. 119-125, 2001.

[25] Clark, D., "*Face to Face with Peer to Peer Networking",* Computer, January 2001.

[26] Clark, J., "*XSL Transformations (XSLT) Version 1.0*", W3C, 2001. Available at: http://www.w3.org/TR/xslt (accessed on 12.04.2001).

[27] Coad, P., "*Java modeling in colour with UML : enterprise components and process*", Upper Saddle River, N.J.; London : Prentice Hall, 1999.

[28] Compaq, "*Enterprise Application Integration: Compaq NonStop™ Solutions Integrator*", Compaq Computer Corporation, 1999. Available at: http://himalaya.compaq.com/view.asp?IOID=76#17 (accessed on 11.11.2000).

[29] Connolly, D., "*Extensible Markup Language (XML)",* W3C, 2001. Available at: http://www.w3.org/XML/ (accessed on 17.12.2000).

[30] Connolly, D., Henry, Thompson., "*XML Schema*", W3C, April 2000, Available at: http://www.w3.org/XML/Schema (accessed on 17.12.2000).

[31] Cover, R., **"***The XML Cover Pages, Extensible Markup Language (XML)",* Oasis, July 2001. Available at: http://www.oasis-open.org/cover/ xml.html#overview (accessed on 17.12.2000).

[32] Curran, T., "E-P2P: the new middleware?", Groove Networks, Inc., from Volume 15, Report 1, February 2001.

[33] Draluk, V., "*Discovering Web Services: An Overview*", In Proceedings of the 27th International Conference on Very Large Data Bases, Rome, Italy, September, 2001.

[34] Duchame, B., "*XSLT, Comments and Processing Instructions*", O'Reilly & Associates, September 2000. Available at: http://www.xml.com/pub/a/2000 /09/13/ xslt/index.html (accessed on 15.02.2001).

[35] ebXML, "ebXML SPECS", ebXML, 2002. Available at: http://www.ebxml.org /specs/index.htm#white_papers (accessed on 08.05.2002).

[36] Emmerich, W., Ellmer, E., Fieglein, H., "*TIGRA - an architectural style for enterprise application integration*", Proceedings of the 23rd international conference on Software engineering, Canada, May 2001.

[37] Farley, J., Loukides, M., "*Java Distributed Computing", O'Reilly Java,* January 1998.

[38] Fattah, H., "*How Peer-To-Peer Technology Is Revolutionizing the Way We Do Business***"**, Dearborn Trade, January, 2002.

[39] Fauvet, D., Paik, Y., "*Peer-to-peer traced execution of composite services*" in Proceedings of TES 2001, Rome, Italy, 2001, pp. 103–117.

[40] Flanagan, D., "*Java Enterprise in a nutshell: a desktop quick reference*", O'Reilly, 1999.

[41] Fowler, M., "*UML distilled: applying the standard object modeling language*", Harlow, Addison-Wesley, 1997.

[42] Fremantle, P., "*Applying the Web Services Invocation Framework – Calling Services Independent of Protocols*", IBM DeveloperWorks, June 2002. Available at: http://www.ibm.com/developerworks/webservices/library/ws-appwsif.html (accessed on 10.06.2002).

[43] Gisolfi, D., "*Web Services Architect Part 1: An Introduction to Dynamic e-Business*", IBM, April 2001. Available at: ftp://www.software.ibm.com /software/developer/library/ws-arc1.pdf (accessed on 10.5.2002).

[44] Gnutella.com, "*Gnutella*", Gnutella.com, 2002. Available at: http://www.gnutella.com (accessed on 20.07.2003).

[45] Goglia, P. A., "*Testing client/server applications*" / Patricia A. Goglia.. - Boston; London: QED, 1993.

[46] Gonsalves, A., "*IBM, Extricity Integrate B-to-B Products*", TechWeb News, 2000. Available at: http://www.techweb.com/wire/story/ TWB20000912S0018 (accessed on 08.01.2002).

[47] Gruber R., "What is an Ontology", Stanford University, 1993. Available at: http://www-ksl.stanford.edu/kst/what-is-an-ontology.html (accessed on 30.07.2003).

[48] Herring, C., Milosevic, Z., "*Implementing B2B Constructs using BizTalk*", Hawaii, IEEE Computer Society, January 2001.

[49] Homayounfar, H., Wang, F., Areibi, S., "*Advanced P2P Architecture Using Autonomous Agents*", CAINE, San Diego California, pp:115-118, November, 2002.

[50] Hondo, M., Nagaratnam, N., Nadalin, A., "*Securing Web Services*", IBM Systems, Journal, Vol. 41, No. 2, 2002. Available at: http://www.research.ibm.com/journal/sj/412/hondo.pdf (accessed on 06.02.2003).

[51] Hu, J., Grefen, P., "*Component Based System Framework for Dynamic B2B Interaction*", 26th Annual International Computer Software and Applications Conference, p557, Oxford, England, August 2002.

[52] IBM, "*WebSphere Application Server*", IBM, 2000. Available at: http://www-4.ibm.com/software/webservers/appserv (accessed on 10.12.2001).

[53] IBM, "*WebSphere B2B Integrator*", IBM, 2000. Available at: http://www-4.ibm.com/software/webservers/btobintegrator (accessed on 11.12.2001).

[54] IBM, "*WebSphere Host Integration Solution*", IBM, 2000. Available at: http://www-4.ibm.com/software/webservers/hostintegration (accessed on 09.12.2001).

[55] *IBM, "Web Services",* IBM, 2003. Available at: http://www-3.ibm.com/software/solutions/webservices/uddi/ (accessed on 10.06.2002).

[56] IBM, "*Web Services Demos: Learn by Example – GUI, Code and Documentation*", IBM DeveloperWorks. Available at: http://www.ibm.com /developerworks/offers /wsdemos.html (accessed on 10.7.2002).

[57] IONA Technologies, "*End to Anywhere is everything*", IONA Technologies, 2002. Available at http://www.iona.com/ info/aboutus/strategy.htm (accessed on 15.10.2002).

[58] IONA Technologies. "*IONA E2A Web Services Integration Platform*", IONA Technologies, *Product Brief*, January 2002.

[59] IONA Technologies, "*Orbix E2A Web Services Integration Platform*", IONA Technologies, 2002. Available at http://www.iona.com products/webserv.htm (accessed on 15.10.2002).

[60] IONA Technologies. "*White Paper, IONA Orbix E2A CORBA Technology*", IONA Technologies, Appril 2002.

[61] Jacobson, I., Grady, B. J., "*The unified software development process*", Harlow : Addison-Wesley, 1999.

[62] Jewell, T., Chappell, D.," *Java Web Services*", O'Reilly, March, 2002.

[63] Kao, J., "*Best Practices. Web Services – Technical Overviews*", Sun Microsystems, August 2001. Available at http://dcb.sun.com/practices/ webservices/overviews/overview_wsdl.jsp (accessed on 15.12.2002).

[64] Karakostas, B., Christofi, S., Dahanayake, A., Gerhardt, W., "*An Approach to Web-Based Application Integration Using Java Adapters and XML, In Web-Enabled Systems Integration: Practices and Challenges*", Idea Group Publishing, 2003.

[65] Kato, D., "*GISP: Global Information Sharing Protocol - a distributed index for peer-to-peer systems*", Computer Science Department, Stanford University, Proceedings of the Second International Conference on Peer-to-Peer Computing, 2002.

[66] Ken, R., "*Creating Value from Business to Business Integration*", Extricity Software, Inc. (2000). Available at: http://www.ascet.com /authors.asp?a_id=129 (accessed on 12/11/2002).

[67] Khriss, I., Brassard, M., Pitman, N., "*GAIL: The Gen-It (r) Abstract Integration Layer for B2B Application Integration Solutions*", 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems (TOOLS39) , p 073, Santa Barbara, California, August 2001.

[68] Kim, S., Graupner, S., "*A Secure Platform for Peer-to-Peer Computing in the Internet*", 35th Annual Hawaii International Conference on System Sciences (HICSS'02)-Volume 9, p. 304, Hawaii, January 2002.

[69] Kit, E., "*Software testing in the real world: improving the process*", Wokingham : Addison-Wesley, 1995.

[70] Knutson, J., Kreger, H., "*Web Services for J2E*E, Version 1.0", IBM, April 2002. Available at: http://www.ibm.com/software/solutions/webservices/pdf/ websvcs-0_3-pd.pdf (accessed on 11.08.2002).

[71] Kobielus, J. G., "*BizTalk : implementing business-to-business e-commerce*", Upper Saddle River, N.J.; London : Prentice Hall PTR, 2000.

[72] Krawczyk, H., "*Analysis and testing of distributed software applications*", Baldock : Research Studies Press, 1998.

[73] Lander, R., "*Deriving DTDs and Data from Schemas with XSLT*", Microsoft Corp., January 2001. Available at: http://msdn.microsoft.com/library/ default.asp?URL=/library/techart/transschema.htm (accessed on 14.02.2001).

[74] Lange, B., Oshima, M., "*Programming and Deploying Agents with Java*", Addison Wesley, Reading, MA, 1998.

[75] Linthicum, D., "*Making EAI Scale*", CMP Media LLC, April 2001. Available at: http://www.intelligenteai.com/feature/010416/linthicum.shtml (accessed on 25.10.2002).

[76] Madron, T., "*Peer-to-peer LANs: networking two to ten PCs*", Wiley, Chichester,1993.

[77] Mandava, R., "*Java Web Services Developer Pack Part 1: Registration and the JAXR API*", Sun Microsystems Inc, February 2002. Available at http://developer.java.sun.com/developer/technicalArticles/WebServices/WSPack /#uddidesc (accessed on 14.11.2002).

[78] Mayank, B., Cooper, B., Arturo Crespo, et al, "*Peer-to-peer research at Stanford*", ACM Press , August 2003, USA .

[79] M. Bawa, Garcia-Molina, G., Motwani, R., "*Estimating aggregates on a peer-to-peer network*", Technical report, Computer Science Dept., Stanford University, 2003.

[80] Mannion, M., "*Rapid Development for the Web: Exloiting Java, XML and XSLT*", 101 Communications, Java Report, pp. 24-30, LLC, USA, 2000.

[81] Markatos, E., "*Tracing a Large-Scale Peer to Peer System: An Hour in the Life of Gnutella*", 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02), p.65, Germany , May 2002.

[82] McCall, K., "*BizTalk Server 2000 – Business Process Orchestration Server*", Microsoft Corp., 2000. Available at: http://www.microsoft.com /Seminar/Includes/Seminar.asp?Url=/Seminar/1033/20000926BizServerKM1/Po rtal.xml (accessed on 20.12.2000).

[83] Metis Technologies Inc, "*The Metis Collaboration Platform (MCP)",* Metis Technologies Inc, *June 2002.* Available at http://www.metistech.com (accessed on 30.09.2002)

[84] Michael, L. J., Bass, C., "*A Case for Enterprise Application Integration*", EarthWeb, November 2001. Available at: http://intranetjournal.com /articles/200111/eai_11_28_01a.html (accessed on 12.06.2003).

[85] Microsoft Corporation, "*Microsoft BizTalk Server*", Microsoft Corporation, 2001. Available at: http://www.microsoft.com/biztalk (accessed on 12.02.2001).

[86] Microsoft Corporation, "*Microsoft BizTalk Server: System Requirements*", Microsoft Corporation, 2001. Available at: http://www.microsoft.com /biztalk/evaluation/sysreqs/default.asp (accessed on 15.07.2001).

[87] Mills, A., "*BizTalk and the Architecture of Tomorrow*", Microsoft Corporation, 2000. Available at: http://www.microsoft.com/Seminar/1033/ 19990916TEComm3 / Seminar.htm (accessed on 19.12.2000).

[88] Minds, H., "*Peer to Peer Application Development: Cracking the Code*", Dreamtech Software, November 2001.

[89] Minq Software AB ,"*Pureload, Java Load Stress Testing Tool",* Minq Software AB, 2003. Available at http://www.codework.com/pureload/ (accessed on 20.07.2003).

[90] Mohan, C., "*Dynamic e-Business: Trends in Web Services*", 3rd VLDB Workshop on Technologies for E- Services (TES'02), Hong Kong, China, August 2002.

[91] Moore, D., Hebeler, J., "*Peer-To-Peer : Building Secure, Scalable, and Manageable Networks*", McGraw-Hill Osborne Media, November 2001.

[92] Morrison, M., Brownell, D., Boumphrey, F., "*XML Unleashed*", SAMS, December 1999.

[93] Napster.com, "*Napster*", Napster.com, 2002. Available at: http://www.napster.com (accessed on 19.07.2002).

[94] Open Applications Group, "*OAGIS Browser for OAGIS 8.0*",Open Applications Group, 2002. Available at: http://www.openapplications.org/ (.09.07.2002).

[95] Oracle, "*Creating the Integrated E-Business with Oracle Integration Server*", An Oracle Technical White Paper, November 1999. Available at: http://www.oracle.com/ebusiness/integration/integration14.pdf (accessed on 15.12.2000).

[96] Oracle, "*Overview of Oracle Integration Server*", Oracle, 2000. Available at: http://technet.oracle.com/docs/products/oracle8i/doc_library/817_doc/ois.817/a8 3729/adois03.htm#998274 (accessed on 17.12.2000).

[97] Oram, A., "*Peer-to-peer: harnessing the benefits of a disruptive technology*", O'Reilly, Cambridge, 2001.

[98] O'Reilly & Associates, Inc, "*Agents as Peers*", O'Reilly & Associates, Inc, 2003. Available at: http://www.openp2p.com/pub/t/69 (accessed on 22.05.2002).

[99] O'Reilly & Associates Inc, "*Web Services Tutorials*", O'Reilly & Associates, xml.com.Available at: http://www.xml.com/pub/rg/ Web_Services_Tutorials (accessed on 12.07.2002).

[100] Özalp, B., Meling, H., Montresor, A., "*Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems of Agent-Based Peer-to-Peer Systems*", 22nd International Conference on Distributed Computing Systems (ICDCS'02) pp. 15, Institute of Electrical and Electronics Engineers, Austria, July 2002.

[101] Pankaj, K., Hsueh-Ieng, P., "*Perspectives of XML in E-Commerce*", IRTORG, December 2000. Available at: http://tech.irt.org/articles/ js215/#xml_ec_advantages (accessed on 17.12.2000).

[102] Vishnu S. Pendyala, Simon S.Y. Shim, Jerry Z. Gao , "*An XML Based Framework for Enterprise Application Integration*", IEEE International Conference on E-Commerce, p.128, Newport Beach, California, June 2003.

[103] Philippe, Le H., "*Document Object Model (DOM)*", W3C DOM Working Group, 2003. Available at: http://www.w3.org/DOM/ (accessed on 15.02.2003).

[104] Pooley, R. J., "*Using UML : software engineering with objects and components*", Harlow : Addison-Wesley, 1999.

[105] Project JXTA, "*Project JXTA Technology Overview*", Sun Microsystems Inc., Available at: http://www.jxta.org/docs/jxta_overview_2003.pdf (accessed on 20.05.2002).

[106] Qiming, C., Parvathi, C., Umeshwar, D., Meichun, H., "*Dynamic-Agents for Dynamic Service Provisioning*", Third International Conference of Cooperative Information Systems, p 95, New York, August 1998.

[107] Rational Software Corporation, "*The Unified Modeling Language (UML)*", Rational Software Corporation, 2001. Available at: http://www.rational.com /uml/gstart/faq.jsp (accessed on 22.01.2002).

[108] Roger, C., "*Privacy Implications of Digital Signatures*", The Australian National University, March 1997. Available at: http://www.anu.edu.au /people/Roger. Clarke/DV/DigSig.html *(accessed on 20.10.2002).*

[109] RosettaNet, "*RosettaNet Implementation Framework*", RosettaNet, 2002. Available at: http://www.rosettanet.org/rnif (accessed on 14.05.2002).

[110] Rowstron, A., Druschel, P., "*Pastry: Scalable, de-centralized object location and routing for large-scale peer-to-peer systems*", In Proceedings IFIP/ACM Inter-national Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, November, 2001.

[111] Samtani, G., Sadhwani, D., "*B2Bi and Web Services. An Intimidating Task?*", Web Services Architect, 2002. Available at: http://www.webservicesarchitect. com/content/articles/samtani02.asp (accessed on 10.10.2002).

[112] Samtani, G., Sadhwani, D., "*EAI and Web Services. Easier Enterprise Application Integration?*", Web Services Architect, 2002. Available at http://www.webservicesarchitect.com /content/articles/samtani01.asp (accessed on 11.12.2002).

[113] Samtani, G., Sadhwani, D., "*Web Services and Peer-to-Peer Computing.*", Web Services Architect, 2002. Available at http://www.webservicesarchitect.com /content/articles/samtani05.asp (accessed on 11.10.2002).

[114] Schlosser, M., Sintek, M., Decker, S., Nejdl, W., "*A scalable and ontology-based P2P infrastructure for semantic web services*", In Proceedings of the 2nd International IEEE Conference on P2P Computing, Linkoping, Sweden, September 2002.

[115] Seltzsam, S., Borzsonyi, S., Kemper, A., "*Security for distributed web service composition*" in Proceedings of TES 2001,pp. 147–162,Rome,Italy,2001.

[116] Sharon, A., et al, "Extensible Stylesheet Language (XSL) Version 1.0", W3C, 2001 Available at: http://www.w3.org/TR/xsl (accessed on 11.03.2003).

[117] Shohoud, Y., "*Introduction to WSDL*", LearnXmlWS, October, 2002. Available at http://www.vbws.com/tutors/wsdl/wsdl.aspx. (accessed on 22.10.2002).

[118] Siu, L., Sai, K., "*Interoperability of peer-to-peer file sharing protocols*", ACM SIGecom Exchanges, Volume 3, Issue 3, p25-33, ACM Press, USA, June 2002.

[119] Snell, J., "*Securing Web Services*", IBM, May 2002. Available at: http://www.ibm.com/software/solutions/webservices/pdf/wp_securingws.pdf (accessed on 12.07.2002).

[120] Stephansen, S., "*The Benefits of a Peer-to-Peer Architecture*", ITQuadrant, Inc., 2001. Available at http://e-serv.ebizq.net/p2p/ stephansen_1.html (accessed on 15.09.2002).

[121] Stoica, I., Morris, R., Karger, D., Kaashoek F.M., Balakrishnan, H., "*Chord: A scalable peer-to-peer lookup service for internet applications*", In Proceedings ACM SIG-COMM, San Diego, California, Aug. 2001.

[122] Sun Microsystems Inc, "*Enterprise Java Beans TM Specification, Version 2.1*", Sun Microsystems, California, August 2002.

[123] Sun Microsystems, Inc, "*Java API for XML Registries (JAXR)*"", Sun Microsystems, Inc, May 2003. Available at: http://java.sun.com/xml/jaxr/ (accessed on 23.7.2003).

[124] Sun Microsystems Inc., "*Java Servlet Technology* ", Sun Microsystems Inc, 2003. Available at: http://java.sun.com/products/servlet/ (accessed on 29.03.2003).

[125] Sun Microsystems Inc, "*Sun[tm] ONE Integration Server, EAI Edition*", Sun Microsystems, 2002. Available at http://wwws.sun.com/software/products/ integration_srvr_eai/home_int_eai.html (accessed on 24.10.2002).

[126] Sun Microsystems Inc, **"***The Java language, an overview***"**, Sun Microsystems, October 2000. Available at: http://java.sun.com/docs/overviews/java/java-overview-1.html (accessed on 17.12.2001).

[127] Sutherland, J., Heuvel, W., "Developing and integrating enterprise components and services: Enterprise application integration and complex adaptive systems", Communications of the ACM , ACM Press, USA, October 2002.

[128] SWIFT, "*SWIFTNet Migration*",SWIFT, 2003. Available at: http://www.swift.com/index.cfm?item_id=7506 (accessed on 25.09.2002).

[129] Thatte, S., " XLANG *Web Services for Business Process Design*", Microsoft Corporation, 2001. Available at: http://www.gotdotnet.com/team/ xml_wsspecs/xlang-c/default.htm (accessed on 10.12.2001).

[130] Themistocleous, M., Irani, Z., O'Keefe, R., Paul, R., "*ERP and Application Integration Issues: An Empirical Survey*", Hawaii IEEE Computer Society, January 2001.

[131] Tibco Software Inc., "*TIBCO ActiveEnterprise™*", Tibco Software Inc., 2002. Avalaible at: http://www.tibco.com/solutions/products/default.jsp (accessed on 10.06.2002).

[132] Truelove, K., Shirky, C., Gonze, L., Dornfest, R., "*P2P Networking Overview. The Emergent P2P Platform of Presence, Identity, and Edge Resources*", O'Reilly, October 2001.

[133] Tsalgatidou, A., "An Overview of Standards and Related Technology in Web Services", University of Athens, Department of Informatics & Telecommunications, 2003, Available at: http://cgi.di.uoa.gr/~afrodite /PADP2002.pdf (accessed on 14.02.2003).

[134] Uddi.org, "*UDDI Executive White Paper* ", uddi.org, November 2001. Available at: http://www.uddi.org/pubs/UDDI_Executive_White_Paper.pdf (accessed on 01.04.2003).

[135] Uddi.org, "*UDDI Technical White Paper*", Uddi.org, November 2001. Available at: http://www.uddi.org/pubs/UDDI_Technical_White_Paper.pdf (accessed on 01.04.2003).

[136] Vinoski, S., **"***White Paper, IONA and CORBA***"**, IONA Technologies, July 2002.

[137] Versata Inc., "*Versata*™", Versata Inc., 2003. Available at: http://www.versata.com (accessed on 17.01.2003).

[138] W3C, "*Extensible Markup Language (XML)*", W3C, 2003. Available at http://www.w3.org/XML/ (accessed on 14.02.2003).

[139] W3C, "*Web Services Description Language (WSDL) 1.1*", W3C, March 2001. Available at http://www.w3.org/TR/wsdl. (accessed on 12.11.2002).

[140] WebV2 Inc, "*The WebV2 PeerBeans Solution: A Conceptual Overview, A WebV2 Whitepaper*", WebV2 Inc , San Francisco, May 2002.

[141] WebV2, Inc , "*WebV2 Solutions*", WebV2, Inc, 2002. Available at: http://www.webv2.com/solutions.html (accessed on 10.09.2002).

[142] Wilkes, L., "*Application Integration*", Butler Direct Limited, June 1999.

[143] Williams, J., "*Keys to Enterprise Application Integration*", The Proceedings of the: Technology of Object-Oriented Languages and Systems, p399, IEEE, *USA, 2000.*

[144] Yugyung L., Changgyu O., Eun P., "*XML schemas: integration and translation: Intelligent knowledge discovery in peer-to-peer file sharing*", Proceedings of the eleventh international conference on Information and knowledge management, Pages: 308 - 315 , ACM Press, USA, November 2002.