A MESSAGE-BASED MIDDLEWARE FOR ENTERPRISE APPLICATION INTEGRATION

SUJOY RAY

A THESIS
IN
THE DEPARTMENT
OF
COMPUTER SCIENCE AND SOFTWARE ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE
CONCORDIA UNIVERSITY
MONTRÉAL, QUÉBEC, CANADA

MAY 24, 2006

© SUJOY RAY, 2006

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-20781-9
Our file Notre référence
ISBN: 978-0-494-20781-9

NOTICE:
The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:
L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Abstract

A Message-Based Middleware for Enterprise Application Integration

Sujoy Ray

In this thesis, we address Enterprise Application Integration (EAI) through the presentation of a message-based middleware that we designed and implemented. Such a multi-tier middleware allows for the seamless integration of information sources and applications with monitoring capabilities over remote events. More precisely, we propose an end-to-end, loosely coupled message-based system that is not only capable of asynchronously integrating information but also allow local application components to be asynchronously notified of changes in distant sources with/without any user intervention. Furthermore, we have designed and implemented various decision support capabilities over this integrated platform that scrutinizes collected information and helps to take judicious decisions through optimization and simulation procedures. The thesis also elaborates a Java-based prototype implementation of the proposed concept, called "Digital Cockpit". Actually, Digital Cockpit implements a five-phase paradigm that integrates, displays, monitors, analyzes and controls information inside organization.

To, My Parents.

Acknowledgments

Several people helped me to make this research a success. First of all, I owe a great debt of gratitude to my supervisor Professor Dr. Mourad Debbabi. I consider myself fortunate working under his guidance and receiving affluent knowledge towards this research and discovery. I really feel privileged for his attention to complete a successful research.

The research described in this thesis is the part of a major project on the elaboration of Digital Cockpit for the Canadian National Defence. This project is a joint collaboration of Defence Research and Development Canada Valcartier (DRDC) and Computer Security Laboratory at Concordia University. I would specially like to thank DRDC researchers namely: Mr. Bernard Deschenes, Dr. Abdeslem Boukhtouta and Mr. Abderrazak Sahi, who provided ample opportunities for research. I do appreciate their guidance, discussion and scientific comments.

I would also like to express my gratitude to colleagues in Computer Security Laboratory in particular: A. R. Arasteh, A. Benssam, S. Dalouche, F. Guerroumi, O. El-Hajj, H. Issa, A. Venkataiahgari and H. Yahyaoui. They influenced me a lot in this research. This work would not have been possible without them. It is not only to provide a scope of work but also to encourage me day in and day out. Thank you all for your help and patience. Last but not the least, I remember my parents, sister, family and friends, all who inspire me throughout my career, concealing real odds of their lives.

Sujoy Ray
25.04.2006

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Nowadays, information systems have penetrated almost every aspect of our lives. Increasing trend towards economy and business automation has exhibited a major concern for producing, collecting, processing and storing large volumes of information continuously through computer-systems. These geographically distant systems are disparate in terms of hardware, platforms, operating systems and software applications. Such heterogeneity limits the scope of interchanging information inside and outside the organizational boundaries. In the last decade, the problem of sharing information and services has been significantly explored by the explosion of internet technology and the emergence of new business initiatives such as e-commerce. Furthermore, tremendous pace of evolution in the field of network centric communication and the omnipresence of communication devices have exposed several limitations of the existing information systems:

- *Poor information and application sharing:*

  Almost all enterprise application integration systems collect information through

request-response oriented connection. These technologies, over a user request, retrieve a static instance of a particular information from the resource. Therefore, such a retrieved information may not reflect its real state inside the source, in the case when data on this source change frequently. Moreover, there exists only a limited number of ad-hoc methods (e.g. polling, traps etc.) to keep displayed information synchronized to remote resources.

- *Poor sharing and presentation of information:*

  Displayed information does not reflect the actual state of the information because the information is previously retrieved by request/response process of information retrieval. Provision of sharing is also very low among data and service sources that may create inconsistency and limit the abilities of systems. Furthermore, presentation of fresh information is implemented by periodic polling from user's side and thereby creating huge network-load. Besides, user applications suffer from poor user-interfaces that present a pile of information without any meaningful insight. Moreover, menu-based user interfaces expect user to intervene for viewing information details. It is not always possible from the user-side, to understand a remote change in information. As a consequence of such sharing and presentation, these enterprise applications may react only after a significant delay.

- *Limited decision making abilities:*

  Some enterprise information systems are designed and implemented on top of data-warehouses. Data inside data-warehouses do not reflect real-time information due to long period of re-loading. Most of the state-of-the-art enterprise information systems use primitive decision support capabilities such as: Tables, graphs and curves. However, business process and analytic are more complex.

An enterprise information system uses a large variety of networked computer systems to collect, process and produce large volumes of information. The ability to transform these data into deeper insights may provide the institution with a competitive advantage in mission-critical situations. Decision makers, top management and leaders need robust and efficient automatic and/or systematic tools to sense and respond to real-time changes. Accordingly, there is an existing desideratum that consists in providing software platforms that:

- Provide structured, regular and real-time communication of fresh information inside and across the organizational boundaries.
- Keep multiple data and service sources synchronized and coherent in a loosely coupled asynchronous architecture that leverages notification to users in case of necessary changes of information in the sources.
- Present information to users, decision-makers in a graphical and user-friendly way.
- Analyze collected information through different algorithms/processes and thereby leverage generic structure for optimized decision support.
- Scrutinize to the desired level for past, ongoing and future activities/processes.
- Handle the required security services in terms of authentication, secrecy, authorization and integrity.

We approach this integration problem by a middleware solution that reduces the tight-coupling between user applications and remote resources. Consequently, it also improves the sharing of information and services. Actually, the incorporation of a middleware assures a stable yet flexible architecture on top of frequently changing business-needs. As a downstream result, users and decision makers obtain critical

insight of information by a real-time big picture from heterogeneous, autonomous and physically distributed information systems [3].

1.2 Background

Industrial and theoretical research towards a unified middleware-based information integration have been massively empowered by the large-cap corporations. A surge of interest has also lately been expressed separately in data integration and web services. In the year of 2002, a survey of CIO magazine on IT spending priorities revealed that 36% of the total software business budget went in integrated systems and processes while over 50% of the multi-billion dollar software market expenditure went to solve the legacy systems and data integration [23].

This example may better explain the problem of coupling in reality. Suppose that, a decision support system (DSS) requires weather service to decide its flight schedules. While an application requests current temperature or aviation information on a city to an information source, it receives a set of values that corresponds to the state of the information at the time of retrieval. However, these values change quite often and remains unnoticed to the user unless the application or the user invokes the same request recursively. This kind of limitation may critically affect flight operations. On the other side, from the user's perspective, there are two major challenges:

- First, it is hard for a user to judge the occurrence of a change inside a remote server without resorting to polling of requests. Thus, it creates unnecessary network load that is unacceptable.

- Second, complex client applications may require frequent distributed computation. Such distributed computation might need integration of remote services.

Therefore, it is clear that we need a software platform between user application and information sources, to notify users with up-to-date information. A middleware platform, in this regard, is very useful and convenient solution to integrate such data sources and services. In what follows, this thesis clearly states the primary objectives of our research.

1.3 Objectives

The aim of this work is to design a middleware platform for enterprise application integration. Accordingly, such a middleware should achieve a synergistic integration of various information systems. Moreover, it should facilitate writing of an application that displays visual, structured, navigational and realtime big pictures that allows decision makers to drill down into details, uncover relationships inside the information. More explicitly, the objectives of this research are [3]:

- Provide capabilities that facilitate the integration of information sources, ensuring real-time availability of updated, consolidated, structured and unified data across the networks.
- Elaborate a digital cockpit platform that presents synthesized information through dynamic and real-time visual objects with the possibility to customize the layout and the access privileges according to different user profiles.
- Understand the security requirements, such as authentication, integrity, privacy, non-repudiation, etc. to protect the middleware from possible security pitfalls.
- Propose a suite of procedures and tools that processes data from different sources in order to create business intelligence capabilities (statistical analysis,

graphical techniques, simulation of what-if scenarios, trend analysis, comparative analysis, etc.).

- Elaborate some optimization procedures that aim to enhance decisions of organizational processes and scenarios.

This thesis mainly elaborates the proposed solution to achieve the previously stated objectives assuming that information and service sources of the organization are identified with their underlying databases, data models, formats and protocols. Moreover, the dependencies and relationships among these information sources have been already explored. Our approach follows a five-phases paradigm:

- *Integration:* The main intent of this phase is to connect all the sources of information within and across the organization. This amounts to the integration of all these sources for the purpose of information sharing.

- *Display:* The main intent of this phase is to take the data from different sources, aggregate them and present the synthesized information into a meaningful, structured and big navigational picture that offers the ability to drill down into the details.

- *Monitor:* The main intent of this phase is the design and implementation of the capabilities that allows the active monitoring of the information system state for the purpose of testing the organizations assumptions, reactive and proactive measures, and response to dashboard thresholds etc.

- *Analyze:* The main intent of this phase is to bring the system to the business intelligence level i.e. to design and implement the capabilities for pattern and trend analysis, simulation of what-if scenarios etc.

- *Control:* The main intent of this phase is to design and implement optimization procedures that will enhance the used processes, methods and strategies.

Recall that, in the last section, we proposed that a middleware solution may better fulfill the aforementioned limitations. In what follows, we briefly study the possibility of a middleware solution to fulfill these objectives.

Actually, a middleware is a multi-tier software solution designed to manage the complexity and heterogeneity that is inherent to distributed systems. It provides a common programming abstraction between the layer of an application and operating system. Therefore, the burden of application programmers significantly reduces. It minimizes tedious programming and enforces implemented standards. Side by side, it generates a higher-level building block over error-prone low-level programming aspects of Application Programming Interfaces (APIs). Thus, a middleware meaningfully offers a unified framework over complex connections, that are required to integrate the multitude of data formats and services directly from client applications [10].

A middleware solution could be implemented in many ways, such as: The remote-process-call based middlware, message-oriented-middleware etc. It is also important to choose a particular middleware, solution for our purpose. A remote-process-call (RPC) based middleware tightly couples the response with the request from a user application. Therefore, when a client synchronously invokes complex processes and/or services at the server's end, the application stalls till the response arrives. Incorporation of object brokers such as CORBA: Common Object Request Broker Architecture from Object Management Group specification, solves only some of the related issues of deploying distributed programming object in the network [11]. However, taking

deceptive nature of internet into account, that often suffers from packet loss, we exclude the choice of an RPC-based middleware for our solution. This thesis proposes a robust message-oriented-middleware (MOM) model that decouples:

- A user application, from the remote data sources and services.
- A user interface, from the communication layer by providing a flexible and reliable messaging layer between them.

Therefore, the proposed integration model fulfills technological necessities of synchronous and asynchronous high-performance operations, as required. This solution leverages two major contributions:

- An asynchronous application/process does not wait for the response after sending a request.
- The application stays loosely coupled and asynchronously up-to-date with respect to remote sources i.e. it becomes capable of receiving notifications without any recursive requests.

Moreover, an integrated enterprise system always requires a front-end system that provides the users with a clear insight over up-to-date data. We have designed and implemented a user interface, called "Digital Cockpit" to exhibit some capabilities and features of the proposed middleware. Unlike presenting software features through menu-bar of traditional user interfaces, the digital cockpit offers users well-presented collections of highest level of information in hierarchy through a single layer and allows navigation in detailed information by clicking on the desired components. Furthermore, this real-time information display is empowered by a monitoring mechanism that runs in the background, listens to the changes in remote information sources and service and notifies the user when required. We have also designed and implemented

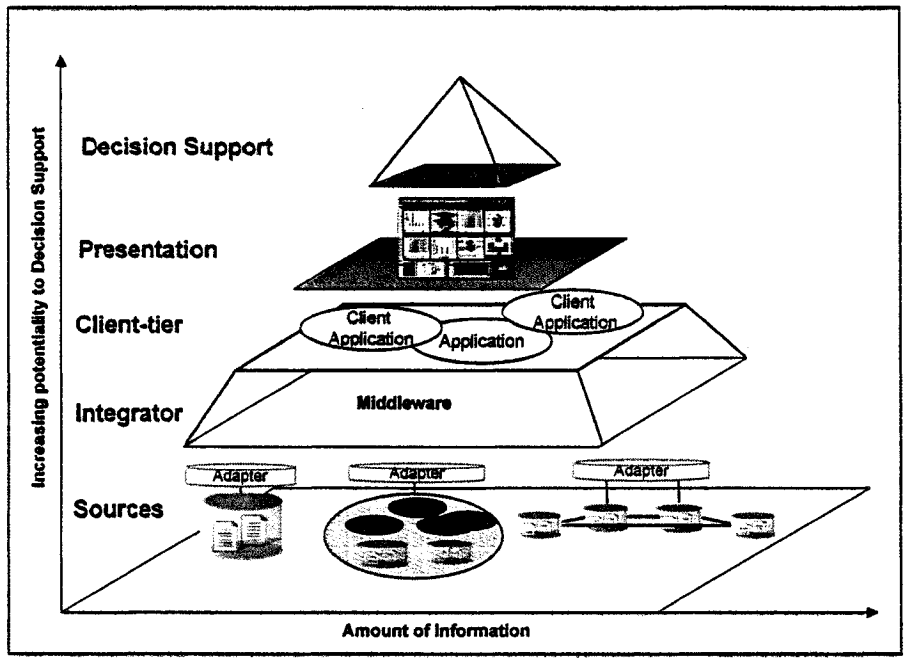


Figure 1: Architecture of a Typical Middleware

a set of lightweight libraries with this proposed solution that analyzes and simulates displayed scenarios dealing with dynamic data. The implementation is unique in a way, as it takes dynamic changes of aggregated information into consideration during analysis procedures.

Figure 1 represents the general notion of proposed middleware. It is expected that this middleware implementation will better serve users and decision makers to access, view, monitor and analyze data and services in order to take judicious and consolidated decisions.

To sum up, the main contribution of this thesis may be considered as the proposition of a loosely-coupled middleware solution that leverages an asynchronous mechanism to keep the displayed information consistent with the real state in the remote

sources. In this work, we address the architecture, the design and a prototype implementation of such a middleware. However, the discovery of the remote sources and services is beyond the scope of this thesis. The fine-grained monitoring of these integrated sources and services is also yet to be implemented. Moreover, the analysis module is kept simple according to the requirements of the project.

1.4 Thesis Organization

This thesis is organized as follows: Chapter 2 introduces the concept of information systems integration. Chapter 3 of this thesis presents the proposed approach i.e. a message based middleware for enterprise application integration. A validation of the proposed approach is presented in chapter 4 through a design and implementation of the so-called digital cockpit platform. Finally, chapter 5 summarizes our contribution, limitations of the approach and outlines future steps of research that may enhance and consolidate this approach.

Chapter 2

Information Systems Integration and Decision Support

2.1 Introduction

This chapter introduces the concept of information systems integration and decision support. The set of common enterprise applications use various off-the-shelf software packages over different heterogeneous and autonomous platforms. A flexible integration infrastructure is essential to these enterprise applications for retrieving information. The information is stored within and outside enterprise information systems. Such an integration infrastructure requires a flexible process-to-process communication to achieve adequate services from different domains. However, an efficient integration approach for performance critical distributed applications is one of the biggest challenges inside any large organization. This chapter presents the state-of-the-art in terms of integration approaches, underlying inter-process communication as well as the existing decision support techniques that may endorse consolidated decisions over integrated enterprise applications.

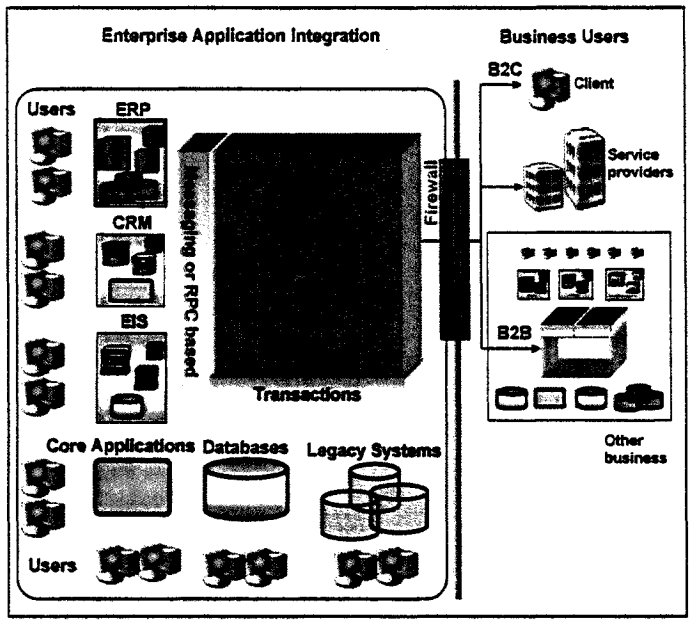


Figure 2: Enterprise Application Integration

2.2 Enterprise Application Integration

Enterprise Application Integration (EAI) pertains to the interconnection of the information systems, internal and/or external to the enterprise. The information systems refer to the sets of information sources and services that are scattered inside an organization. The interconnection of these systems aims to provide a better sharing of data and services among enterprise applications [11].

Figure 2 illustrates a clear example of an enterprise system. An enterprise system may have several applications (e.g. Customer Relationship Management (CRM), Enterprise Resource Planning (ERP), third party software, etc.). These applications run on different platforms and store data locally or globally within the domain of the enterprise. Each of them, in general, has a set of users working on it. As shown

in figure 2, EAI provides RPC-based and/or message-oriented transaction of the information among application components, enterprise data-sources, core applications, legacy systems, etc. Moreover, EAI leverages services that are stored in application servers to provide information to the outside enterprise applications.

2.3 Data Integration

Data integration is only concerned with data retrieval. It allows companies to combine data from disparate data sources. Within this approach, data sources are the principle points of connections [22]. The interconnection of these data sources is one of the longest standing problems in the industry. Previously, the data sources were integrated according to the business requirements using different vendor-based approaches [5]. However, these diversified and organization-specific approaches lack standard guidelines for the cross-vendor extensibility. Therefore, these custom-built integration approaches, commonly known as "in-house" development [9], are not scalable. Actually, they were produced as temporary, cost-effective and undocumented solutions. With the technological advancement, finally, they turned into legacy applications that require custom-built adapters with another phase of integration to be compliant with the standard solutions.

This section presents two standard solutions to the data integration problem: The data warehousing (materialized views) and the federated architecture (virtual views) [5, 34]. Both of these approaches take a set of the pre-existing data sources and develop a single unified (mediated) schema over the data sources. Then, a series of transformations or source mappings are specified to describe the relationship between each data source and the mediated schema [33]. The following part of this section details and compares these two approaches.

2.3.1 Data Warehousing

"A warehouse is a subject-oriented, integrated, time-variant and non-volatile collection of data in support of management's decision making process" [43]. The data warehousing solution requires an explicit database that copies data from multiple operational data sources including RDBMS, legacy systems, mainframes etc. Engineers use commercial ETL (Extract, Transform and Load) tools to load and remove data periodically after a certain interval [19]. ETL is a three-phase data loader. First, it retrieves data by copying them from the distant data sources. Second, it resolves multiple syntax and semantics associated with the metadata that conforms a unified schema of the target database. A series of transformation is also necessary to unify and validate the accurate information. Finally, the transformed data are written into a target database, a data warehouse, with appropriate summarization and aggregation. Figure 3 illustrates the logical view of a data warehouse. Data warehousing is a widely adopted information integration concept famous for the following features [43]:

- *Subject Oriented:* The data warehouse is always arranged around high-level entities of the business. As an example, a data warehouse of marketing contains subjects, such as: Customers, products, sales etc.

- *Integrated view:* The data warehouse integrates the information from various sources as the copies of the existing data. A series of transformation confirms consistency (naming conventions, data constraints, etc.) and unified view to the user. Integrity of each dataset is checked prior to the loading of information into the warehouse.

- *Non-volatile:* Data are kept into the data warehouse for a certain period of time and the stored information is not rewritten during that interval.

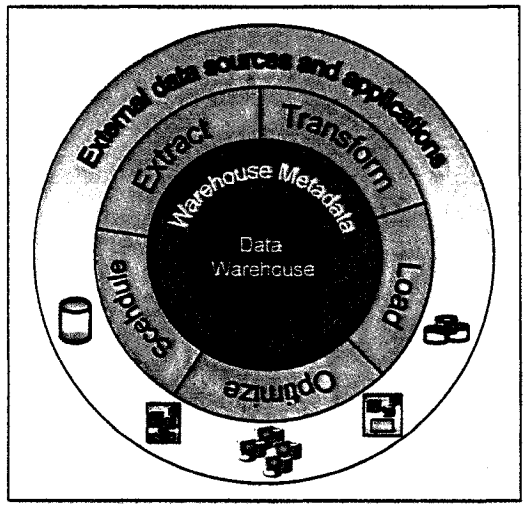


Figure 3: Data Warehouse: Logical View

- *Distinctiveness:* Distinct nature of the information integration perfectly reflects the organizational requirements through the data warehouse that also keeps historical information and always remain normalized in the third normal form or higher.

Data warehouse is typically used by the Online Analytical Processing (OLAP) applications and tools to run complex queries over a large multi-dimensional collection of data. It assists managers and decision makers to receive a global view on a particular topic. Data mining and data analysis oriented decision making procedures are also favored by this solution. Therefore several decision support systems are built on top of the data-warehouse. However, the state of the analyzed information always corresponds to the state of the data during last reloading. Thus, it reflects an older state of the original information residing in the remote information sources. Furthermore, the reloading of information to the data warehouse is typically expensive and slow [34]. Therefore, while the information changes frequently, a data warehouse approach is unable to reflect such changes to the user application.

2.3.2 Federated Architecture

The federated architecture is basically a mediator based data integration approach [29]. It uses one or more data connectors at the remote end of the operational databases. The mediator defines the schemas to represent each particular data format of the data sources. The aggregation of the mediated schemas creates the global schema. The federated architecture is different from data warehouse as it does not store data in a single storage. It queries directly the data sources from the application end [3]. A query may be translated into several queries and retrieves sets of information, on demand, without affecting the local autonomy of storage. Therefore, it always accesses the current state of information. The federated architecture is capable of responding to the frequently changing states of information or the global schema itself. Moreover, it offers seamless integration of data sources with a fair trade-off of additional connections.

In the past years, IBM and Sun Microsystems started this research on federated architecture with the collaboration of several academic research. In 1994, Stanford University introduced 'The Stanford-IBM Manager of Multiple Information Sources', a mediator based approach of integrating heterogenous, structured and unstructured data sources through a common object model [36]. However, the initial implementations of these academic approaches suffered from high-level vendor lock-in and inextensibility. Afterwards, Sun Microsystems successfully released Java DataBase Connectivity (JDBC), the de-facto standard Java API for database connectivity [24]. JDBC implements the federated architecture approach. In this model, once a synchronous connection is established, the fetched information is directly sent to its requestor through a tightly-coupled communication protocol. However, it was realized that data communication requires further flexibility from the coupling standpoint. In

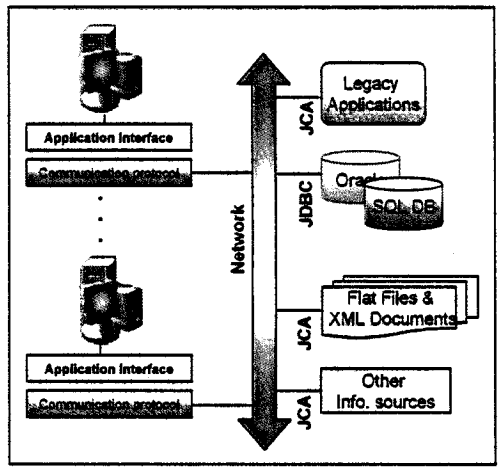


Figure 4: Data Integration: Federated Architecture

addition, the integration of the legacy systems turned into another major concern. Actually, 80% of the worldwide data is stored in mainframes. A report indicates that almost 50% of the software market expenditure goes to integrate legacy applications [23]. The flexible coupling in data retrieval was finally addressed by the release of the Java Message Service (JMS) [40], in 1999. Afterwards, the Java Community Process recently released J2EE Connector Architecture API (JCA) that is an open standard solution to access almost all kind of standard enterprize information systems (including legacy systems), in a tightly-coupled synchronous/asynchronous way [41]. Figure 4 illustrates a federated architecture of data integration.

The choice of an a integration approach always depends on organizational requirements. The proposed direction of our research aims towards an integration infrastructure that reflects the real-time change of information over frequently changing

business needs. Table 1 shows a comparisons among the data warehouse and federated architecture approaches based on certain criteria.

Approach	Federated Architecture Solution	Data Warehouse Solution
Event notification	√	-
Real-time changes in data sources	√	-
Strategic decision support	-	√
Easy extensibility	√	-
Easy access	-	√
Shorter interval of time-to-market	√	-
Economical solution	√	-

Table 1: Federated Architecture vs Data Warehousing Approach

We have chosen the federated architecture for the design of the proposed middleware. The federated architecture exhibits a more sophisticated approach to represent frequent changes of the remote information to the client side. However, the design of the data analysis and the decision support techniques are challenging in such a virtually integrated environment of multiple resources taking frequent changes of information into account. Furthermore, the scope of transforming data into a unified schema is limited inside a client application. Finally, the applications on top of such integration infrastructure should use lightweight decision support tools as the huge memory load on a client machine is unacceptable.

2.4 Service Integration

The retrieval of data is not enough to design an enterprise application integration infrastructure in the absence of the service sharing. It is not always possible to process a request by a single enterprise machine. Let us take an example. As we know, the

security is one of the major concerns of today's industries. It deserves explicit servers to authenticate the users and to encrypt or decrypt sensitive information. Similarly, current business logics have typical computation complexity. Suppose that, in an enterprise, the department of finance possesses certain service that calculates a specific output with respect to a request. Therefore, if the user of a remote enterprise application requests such an information, the remote application component must integrate that particular remote financial service to represent a meaningful view of its user's request.

In the past, the two-tier Electronic Data Interchange (EDI) was mostly used for many years to perform business transactions between the partners [14, 15]. The EDI transports data through formatted messages (based on defined standards) and proprietary network protocols. The companies were reluctant to invest in these technologies due to the large underlying investments that are required in terms of software, hardware and consultancy [14]. However, with the explosion of web technologies in early 90's, the communication within application components turned into an inseparable part in a distributed global environment. In the following, this section briefly illustrates few renowned process integration approaches.

2.4.1 Integration Approaches

Business Process Integration

The business process integration defines a common abstraction of a business model that spans over the choreography engines and internal processes of local information systems. These sub-systems satisfy various business requirements. The idea is implemented by a single business process that controls interactions and information flow between human and multiple systems inside the organization. Actually, this approach

allows a logical control layer that enables local systems to communicate through a single process for their business operations and objectives. This main process controls appropriate information, sequences, states, durability and exception handling on top of the sub-processes [11]. Moreover, it addresses sequences, hierarchy, events, logical execution and information transfer between systems within the same organization (EAI), and different organizations (Business to Business: B2B). As a result of the underlying independence between the main business process and the other sub-processes, the business process integration allows modification of processes and business rules without affecting the overall systems.

Web Portals:

The web portal integrates a multitude of enterprise systems, internal and/or external, through a single user interface, e.g. "Yahoo" portal. Actually, a portal is an interface to access back-end services inside an enterprise domain. The portal does not really integrate the services from distant servers, rather it provides a layer allowing authenticated clients to use several selected service/s. However, these services are unable to automatically exchange information between themselves through the portal. Moreover, the portal based integrations are mostly proprietary solution, achieved by using several technologies: Application servers and servlets, page servers (ASP, PHP, JSP, etc.). Presently, most of the B2B information flows through the portal-based user interfaces.

The web-portals are famous due to several advantages: It allows users to interact with company's internal systems through a user interface (generally a webpage). The web-portal is easily implementable than implementing sophisticated process integration where integrated processes communicate with each other. In addition, the portal

based integration approach can be easily deployed to existing systems in less duration of time without disturbing their functionalities [32]. The whole system follows a set of common business rules.

On the other hand, the web-portal approach presents limitations as information comes through a single server (or cluster). The incorporation of services are pre-designed and it requires human interaction to retrieve the information from them. Furthermore, the web-portal suffers from a single point of access. Besides, it uses request/response model of information interaction (tightly coupled to the application). Therefore, the complex interaction with data sources (using JDBC, JCA or other bridges) increases the delay to the main application [37].

Service Oriented Architecture:
The service oriented architecture (SOA) may be considered as the most sophisticated software integration technique, typically empowered by the industries [30]. The SOA is a software architectural concept based on agreed standards. The concept of the web services is coined to this architecture in order to represent the enterprise applications through a network independent service. The web services implement the inter-process communication using standard web protocols, naming conventions, XML protocols (SOAP), etc. In what follows, this section details a brief overview of the web services.

2.4.2 Service Oriented Architecture: Web Services

The service integration is an active research area that refers to the aggregation and composition of the remote services from a platform in order to fulfill the user's need. It defines a generic model of inter-process communication inside and outside organizational domain. The SOA exposes a flexible way of communication among a multitude

of enterprize systems and their users by standard implementation of web services. Each service is composed of few layers and each layer is independent of the others. This independence helps in the deployment of SOA in the commercial context of a heterogeneous e-business environment.

In the past, inter-process communications were limited by proprietary processes, messages and network protocols, examples: The Common Object Request Broker Architecture (CORBA) from OMG [42], the Distributed Component Object Model (DCOM) [7] from Microsoft, the Remote Method Invocation (RMI) from IBM and SUN Microsystems, etc. These technologies allow organizations to integrate applications in a distributed infrastructure using tightly-coupled remote process calls [14, 15]. However, the inter-operability between these RPC-based mechanisms is complex and limited. For instance, the CORBA and the DCOM cannot communicate directly without software bridges [42]. This limitation is due to the fact that each model uses its own communication protocol. CORBA uses Internet Inter-ORB Protocol (IIOP) while DCOM uses Object Remote Procedure Call (ORPC) and RMI relies on Java Remote Method Protocol (JRMP).

With the increasing dependency among application sharing in e-commerce, the companies were looking for a standard process to process communication model among business applications. Moreover, IT applications require automated, global communication without any prior agreement or static connections. The SOA leverages web services based architecture for e-business that successfully integrates these IT applications hiding connection complexities from the application users. The environment-Speak is the first implementation of SOA, introduced by HP in 1999

[30]. Shortly afterwards, many competing frameworks and proposals for the web services have been provided such as Microsoft's .Net, IBM's webspheres, SUNs J2EE application server, etc.

Web Services:

"A service is a self-describing, self-contained modular unit of application logic that provides some business functionalities to other applications through internet." Technically, web services use ancillary standard mechanisms, notations and naming conventions for describing, registering and requesting services (W3C- world-wide-web consortium) [11, 30]. The services are mostly written in the standard XML based languages.

The idea of using a more expressive markup language came from World Wide Web Consortium (W3C) as an extension to the Standard Generalized Markup Language (SGML). The XML allows the designer in customizing tags, enabling definition, transmission, validation, and interpretation of data among applications [1]. The XML's self-descriptive nature provides easy integrity to structured, semi-structured and unstructured data overcoming the limitations of HTML. The legal building blocks of an XML document are defined in: XML DTD (Document type Definition) and XML schema. DTD is either a world-wide standard document definition or a set of definitions agreed by a group of people in order to exchange information. XML Schema defines documents above and beyond the basic syntax constraints imposed by XML DTD definition itself. Schema may specify new data type of the elements and the content of documents of that type. Elements are written as compliant to that defined class. Furthermore, it may inherit and import elements from existing classes and schemas using namespaces [12].

The web services model takes advantage of XML through a layer based implementation of its architecture as described below. From the functional point of view, this model consists in three principle roles: The service provider, the service requestor and the service registry. A service provider is the owner of a service. The service provider describes the web services and publishes them in a host. A service requestor is either a human driven application or another service that receives a processed output (in XML format) in response to an XML request. Now in the global context, as the service requestor and the provider are unknown to each other. Therefore, they require a registry system that helps automatic binding of users to the pre-registered services according to their choices, business logic and other limitations. In the case of static binding, the service registry is optional because the provider and the requestor are known to each other. However, in the case of dynamic binding, the service requestor obtains the information about the services from a service registry and binds them to the appropriate web services at run time [18].

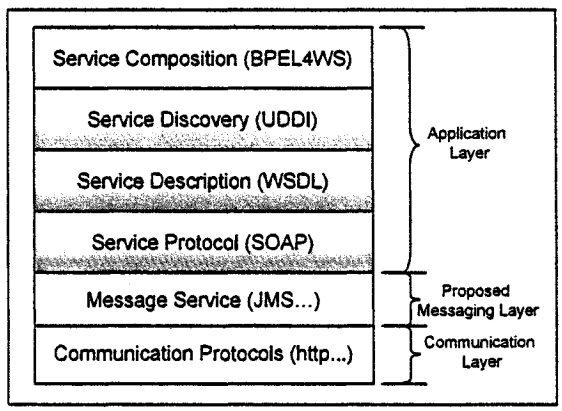


Figure 5: Stack of Web Services

In the following, this section elaborates the layers of the web-services architecture. Figure 5 describes different components of web services infrastructure.

- *Service Protocol:* The service protocol refers to a common standard application protocol to communicate the request and the response over standard network protocols. The specification of the implemented standard is defined as Simple Object Access Protocol (SOAP) by W3C. Actually, the SOAP requests are XML schema. It contains the envelope, the header and the body of a document-centric message. The SOAP binds this message with remote procedure calls using XML [18]. During the execution, the SOAP request binds with the methods of communication protocols such as: HTTP GET/POST, MIME, etc.

- *Service Description:* The service description elaborates the interface of an implemented service. It represents common business transactions (e.g. sending a purchase order), common data-interchange formats and mechanisms to negotiate business terms before commencing any transaction. As the services are defined in an XML-standard, the global consumers may interpret and request to a service automatically through this pre-defined service interface/s. The web services description language (WSDL) is the de-facto standard for service description, defined by the Organization for Advancement of Structured Information Standards (OASIS) [21]. As a whole, a WSDL document describes what a service can do, where it resides and how to invoke it.

- *Service Discovery:* As stated before, the discovery of the web services is implemented through a mechanism that lists organization's capabilities for the business transactions and provides an interface to lookup a company profile. The service discovery registry may use its own XML repository system (example: ebXML) or other repository (example: UDDI). Functionally, it binds

a producer with a consumer, while the requested features of a consumer perfectly matches to the provider's service. A standard based open specification for service description and discovery is achieved in UDDI (version 2) from OASIS.

The description of the web services is stateless as implemented in WSDL [21]. Therefore, it offers the execution of a whole service in response to the user's request. However, the human requests are often too complicated to be served by single service. Therefore, it needs the composition of multiple services to accomplish a required workflow. WSDL being stateless is unable to capture intermediate states of the workflow and a concurrent language turns up obvious on top of the WSDL for the execution of the service sharing. This necessity introduces another layer on the web services stack, called *Service Composition.* The software vendors implement a number of concurrent process execution languages. The most promising one is the Business Process Execution Language for Web Services (BPEL4WS), implemented by IBM, Microsoft and others [20].

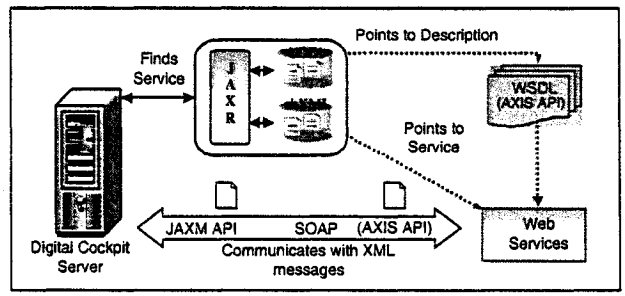


Figure 6: Architecture of Service Integration Model

The initiatives are also taken to integrate these pieces of standards of into a single standard infrastructure. OASIS has released the specification of Electronic Business XML (ebXML) towards this aim. The Java Community Process has also adopted

ebXML as an emerging standard for web service implementation in e-business applications. ebXML uses its own registry-repository system, CPP/A architecture for service description and ebXML MS messaging service to communicate between business service interfaces [6]. Figure 6 shows a general architecture of the web-service based service integration framework. As shown in the figure this thesis proposes the use of a messaging service by decoupling the SOAP and the communication protocols. Actually, the messaging of information holds the key of an efficient integration infrastructure. In what follows, we discuss the issue of a messaging service.

2.5 Messaging Service

The software enterprise system consists of a set of cooperating processes connected by the network. The unique challenges of EAI development include sharing information among these distant processes that are running in a heterogeneous environment. A middleware-based EAI system uses a variety of vendor-locked and home-grown solutions that are physically independent and mostly tightly coupled in nature. These tightly coupled remote procedure calls (RPC) block the requesting application until the remote procedure returns the control back to the caller. Such a deliberate synchronization results in an immediate impact over the performance of the whole system. The messaging service provides a better alternative in this aspect.

The enterprise applications consider messages as lightweight entities that consists of a header and a body. The header field contains the information related to the message routing and identification while the message body illustrates the data to be sent. By definition, a message is self-describing and must contain all necessary contexts to allow the recipients carrying out their work independently. These messages are generally created by an API as a payload (application data) on the communication

protocols with proper routing information. In addition, the messaging service layer decouples the user application from the remote data sources and services. A message oriented middleware exchanges the information in the form of messages. Therefore, it leverages the seamless integration of information sources and the services inside and outside the organization with a significant improvement over the quality of service [35].

2.5.1 Classification of Messaging Services

Enterprise messaging is not a new concept. There are several existing messaging products in the IT world, for example: IBM-MQSeries, Microsoft-MSMQ, TIBCO-Rendevous, Open Horizon-Ambrosia, Modulus-InterAgent, etc. Comparatively, newer generation of asynchronous messaging products are: Progress SonicMQ, Softwired iBus and FioranoMQ. In general, messaging services may be classified from three different perspectives:

- Messaging Architecture.
- Messaging Model.
- Messaging Style.

Messaging Architecture

The enterprise systems obey a distinguished framework. Some of them are centralized while others are decentralized or hybrid. The architecture of messaging systems may be classified according to these frameworks. In general, there are three messaging architectures [35].

- *Centralized Architecture:* The centralized messaging system relies on a single message server that routes messages and contains request/response brokers for the implementation details of its messaging clients. Here, clients always post messages to their associated server. A system, therefore, may add or delete a particular client of a server without impacting others in the same architecture. However, architecturally, a centralized server may be a distributed cluster working as a single logical unit.

- *Decentralized Architecture:* The decentralized architecture does not posses any message server. Each client is offered some server-like facilities (persistence, transactions, security, etc.) and the router implements some IP multi-casting functionalities to transport message requests to other messaging clients.

- *Hybrid Architecture:* The industrial messaging vendors often implement both of the aforementioned architectures in a single messaging system. The centralized multicast is mostly implemented in TCP protocol while its decentralized counterpart is implemented by IP multi-casting protocol.

The commercial messaging solutions are designed as per the average enterprise client's requirements. As of example, the messaging service of Sonic MQ series allows centrally managed distributed messaging components [39]. TIBCO Rendezvous explores a distributed architecture to eliminate bottleneck and single points of failure. Microsoft's "Distributed interNet Architecture" (DNA) architecture contributes MSMQ messaging service. MSMQ components are of two types: The MQ site controller (SC), that stores read-only copies of messages into a queue, located in client-side end. Second, the primary enterprise controller (PEC) that resides in the central MQ server side and keeps the broker and stores the queue details [28].

Messaging Model

The message information may be distributed according to two basic messaging models, namely: The Point-to-Point model and The Publish-and-Subscribe model [39].

- *Point-to-Point Model:* The point-to-point (P2P) model uses a connection component, called "queue", that allows one-to-one delivery of the message. More precisely, a message producer creates a message and puts it to a "queue" inside a messaging domain of the JMS server for a certain duration. Once the message is in the queue, the queue allows only one consumer at a time, who may receive that message from that queue. The point-to-point messaging confirms the loose coupling between a producer and a consumer at a time.

- *Publish-and-Subscribe Model:* The publish-and-subscribe model is a one-to-many message broadcasting model. The MOM uses this technique while one application sends message to multiple other applications. Each consumer receives a copy of the same message in nearly same time. In the publish-and-subscribe model, this virtual repository channel of the messaging domain is called the "topic". The publisher application puts the information into a topic (without any direct connection to the subscriber) and all the registered subscribers receive a message from the topic.

The auction sites, the stock quote services, the security services often require to push data to huge population of consumers that can be better implemented by the publish-and-subscribe messaging. Messaging vendors favor different messaging models. MSMQ supports queue based one-to-one messaging [28], while SonicMQ and TIBCO Rendezvous implements both of these messaging models [38].

Messaging Style

The significance of the messaging system lies in its capability to decouple applications while they share information between themselves. The model of the RPC is strictly synchronous. Messaging, in contrary, allows both synchronous and asynchronous communication, as required by enterprise systems [35].

- *Synchronous Messaging:* It tightly couples the messaging component and the receiving process. A synchronous consumer uses the "pull" technique to receive messages from the destination. It proves advantageous in the case of fail-safe communication and transaction processing. The receiver stays bound to the messaging server until it finishes the processing of the sender's request. However, the synchronous messaging requires both of the communicating processes available in the presence of an available network.

- *Asynchronous Messaging:* The asynchronous messaging style uses "store and forward" mechanism while a message producer sends the messages to the receiver. A message may traverse through one/many connection component/s between the sender and the receiver. An asynchronous consumer registers a message listener to a "destination" inside the server's messaging domain. This procedure allows the sender and the receiver not to block each other. As the message arrives, the listener model informs the registered consumer with the message. Therefore, the client application remains loosely-coupled to the messaging structure and any failure in a client application does not affect the whole infrastructure.

The business solutions favors asynchronous paradigm through a messaging solution. However, certain situations do require a synchronous solution, while an information is required to be compromised only between the sender and the receiver, such as in

transaction processing. Among several messaging vendors, TIBCO Rendezvous implements both of the point-to-point and publish/subscribe messaging in synchronous or asynchronous mode via WAN or the Internet. Similarly, IBM Websphere MQ series, Sonic messaging products also implement both notions in their messaging styles.

The next subsection details the standard implementation of a messaging service that includes all the aforementioned features. After a review of different messaging technologies and judging commercial messaging products, we have chosen Java Message Service (JMS) as the most prominent technology for this research.

2.5.2 Java Message Service

JMS is established as the Java messaging standard, released in 1999 and used by most of the leading industries including Sun Microsystems, Sonic MQ, IBM (Websphere), BEA (Weblogic). The aim of JMS consists of the following [40]:

- Illustrate a single unified messaging model that can be implemented as an API.
- Allow cross-vendor communication in the messaging service level with distant applications that does not use JMS but supports a similar specification.
- Integrate multiple heterogeneous sources of information and services (includes XML support) through a middleware based message communication of Java objects.
- Implement a layer between application program and communication protocols and thereby incorporate a magnitude of enterprise properties in the messaging such as: Security, portability, reliability, etc.
- Elaborate a new notion of asynchronism in case of intra and inter organizational business operations.

- Extend support to numerous Java based APIs, such as: Java DataBase Connectivity (JDBC), Java Transaction API (JTA), Java 2 Enterprise Edition (J2EE) and Enterprise Java Beans (EJB) components.

There are several JMS components associated with JMS messaging. All the messaging entities may be classified into four entities [4]:

- *Provider:* Each JMS application runs on a host application, called JMS provider, that leverages administrative, functional and control capabilities. A "provider" application runs inside the JMS server and hosts JMS components like a container. This container hosts JMS "broker" and other JMS components that hold the implementation details of a robust messaging service [27, 39].

- *Clients:* Clients are JMS applications that use the provider components to produce and consume the messages. A client may be synchronous or asynchronous to the JMS destinations for sending or receiving the messages. JMS is implemented in a way to support JMS and non JMS clients through Java applications or other native client APIs that are running inside the messaging system [35].

- *Messages:* A JMS message facilitates applications with data and event notifications. Unlike RPC, a message neither dictates the recipient nor it blocks the sender. A message object consists of message headers and the message itself, called "payload". Most of the message headers are automatically assigned as configured in the program, e.g. `JMSDestination`, `JMSDeliveryMode`, `JMSMessageID`, `JMSTimestamp`, `JMSPriority`, etc. Additional headers are also assigned sometimes to the application, the provider or the messaging service itself and are called "message properties". There is also a "message selector" mechanism to filter out specific messages as desired by consumer. The JMS message payload has the base interface defined in the Java library of

`javax.jms.Message`. The payload might be structured like `StreamMessage` or fairly unstructured as `TextMessage` [40].

- *Administered Object:* The administered objects are pre-configured, administrator-created JMS objects to control the message-based communication. The administrator binds these objects to the Java Naming and Directory Interface (JNDI) namespace prior to the messaging. The administered Objects are of two types: `ConnectionFactory` and `Destination`. Once created, they leverage an abstraction to the namespace by hiding the implementation complexities through a virtual interface. Therefore, the client program executes with a minimal programming that is required to lookup a specific JMS connection object. Moreover, administered objects always implement the standard-based profile and remain portable despite proprietary aspects of JMS providers [26, 35].

Apart from aforementioned components, JMS also uses a directory service for the storage of the implementation-specific settings in the distributed JNDI components. A directory service is a file system that helps client application for the lookup to a named, pre-configured `ConnectionFactory` or `Destination` as defined by the administrator. Most of the JNDI is implemented by the Lightweight Directory Access Protocol (LDAP) server to store and find the administered objects [8].

JMS Messaging

Figure 7 represents the JMS messaging models. JMS requires proper configuration of the administered objects in JNDI prior to messaging. In general, a directory service has its own domain, managed by a domain manager. The domain manager controls the administered objects within multiple virtual containers and allows communication between them [39]. First, the administered objects are required to be bound to JNDI using administrative tools. Once `ConnectionFactory` and `Destination` are

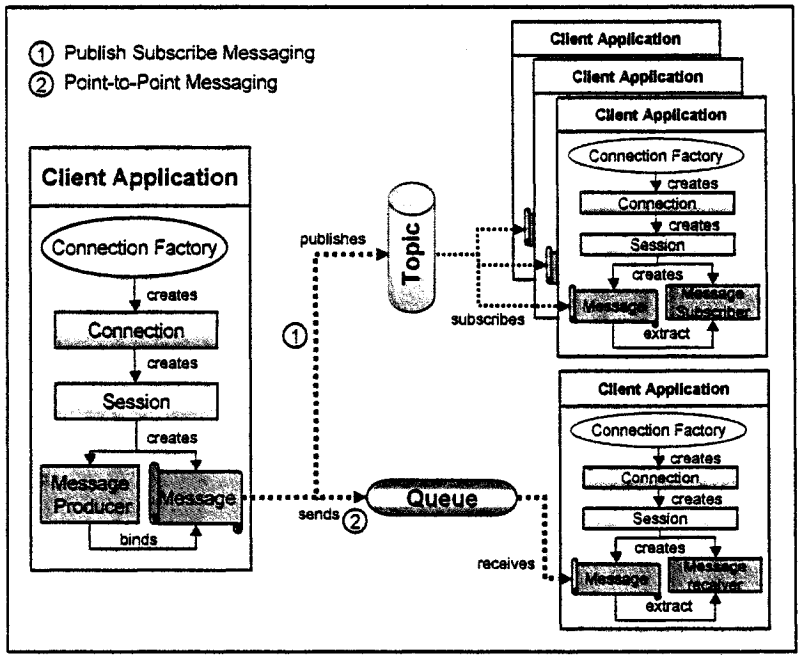


Figure 7: JMS Messaging Model

created in a JMS server and all its implementation-specific information are kept inside the system, JMS is ready to start the communication. A developer should create a JMS `Connection` initially from a valid and bound JMS `ConnectionFactory`. The `ConnectionFactory` may handle multiple connections simultaneously. JMS messaging is session-based. A `Session` object implements a particular valid `Connection`. The interface `Connection` is also capable of handling multiple sessions at a time. The objective of a session is to implement transactions in the synchronous or the asynchronous messaging. However, there are strict restrictions imposed on the concurrent access during the sessions. The JMS specification reserves the access to a session by a single message consumer. However, a sophisticated design may allow multiple sessions related to a single connection to process concurrent requests from the users. Each application process runs under a particular session. The JMS user applications are of two types: `QueueSender/QueueReceiver` and `TopicPublisher/TopicSubscriber`.

`QueueSender/QueueReceiver` application either sends message information to the queues or receives message information from them. Similarly, `TopicPublisher/ TopicSubscriber` either publishes message information to the topic or subscribes the message information from it. The JMS implements one-to-one and one-to-many messaging models through the same API. The JMS point-to-point messaging paradigm stores message in the queue that ensures single retrieval of message by a single receiver at a time. The queues are also used for load-balancing while a number of diverse systems share processing operations through a proper distribution of incoming messages. The JMS publish-subscribe messaging, on the other hand, is implemented by the JMS topic. A topic implements a multi-threaded architecture that is capable of notifying a message object all to its subscribers. Table 2 shows a piece of code for the JMS sender application that sends a message information to a JMS queue.

In the JMS, messages are communicated through a "store and forward" mechanism. However, there exists different binding techniques between the JMS servers and the clients. The message subscribers may be synchronous, asynchronous or durable [26, 35]:

- *Synchronous Consumer:* A synchronous subscriber waits for a message always or for a specified duration. As it receives the message, the JMS "listener" sends it for processing and then blocks to receive again. JMS "listener" implements the technique into an `onMessage()` method. The method calls are: `QueueReceiver.receive()` or `TopicSubscriber.receive()` for synchronous retrieval of message [35]. However, sophisticated programming may use dedicated onTimeOut() method to deactivate such a listener.

- *Asynchronous Consumer:* An asynchronous receiver receives the message when a message arrives but does not block the connection between the destination and

```
import javax.jms.*;
import javax.naming.*;

public class JmsQueue {
Context jndiContext=null;
QueueConnectionFactory qCF = null;
QueueConnection qC = null;
QueueSession qSess = null;
Queue queue = null;
QueueSender qS = null;
TextMessage message =null;

public qSend() {

//Initial lookup to Administered Objects assuming they are
already //created.
try { jndiContext = new InitialContext();
qCF = (QueueConnectionFactory)jndiContext.
      lookup(``QueueConnectionFactory");
queue = (Queue) jndiContext.lookup(``TestQ");
} catch (NamingException e) {}

//Creation of connection, session and message sender.
try { qC = qCF.createQueueConnection();
qSess = qC.createQueueSession(false,Session.AUTO_ACKNOWLEDGE);
qS = qSess.createSender(queue);
} catch (JMSException e){}

// Creation of message and sending it to the queue.
try {
message = queueSession.createTextMessage();
qS.send(message);
} catch (JMSException e) {}
} } }
```

Table 2: Java Code for a JMS "sender" Application

the application process. An asynchronous JMS message listener is programmed with event-based notification inside `onMessage()` method. The programming is done using `receiveNoWait()` method. Moreover, after receiving the message, it may or may not use a message-selector to filter the incoming messages from more than one topic. The loosely-coupled asynchronous consumer is more useful for free information exchange, except when there is a typical need for synchronous transaction.

- *Durable Consumer/Sunscriber:* Durable subscription frees the consumer from staying continuously connected to the JMS server. The "Store and forward" messaging mechanism also allows the server to store messages inside the server on behalf of a subscriber while client program is not available. It allows guaranteed messaging so that consumer receives all the messages at the re-connection, irrespective of the duration of staying disconnected [35]. A JMS durable-subscriber is defined by a `createDurableSubscriber()` method within a session.

2.5.3 JMS Properties

The Java messaging service leverages an established, secured, reliable loosely-coupled integration of the information sources and services based on a common messaging concepts. According to JMS specification, it inherits a large number of properties with enough scope of further development. The properties of the JMS API may be classified into two categories:

- *Architectural Properties:* The JMS is a standard-based, robust and resilient enterprise messaging system. Architecturally, it is centrally managed distributed sets of components, that are loosely coupled to each other. The JMS favors

significant performance improvement over the system using the asynchronous retrieval of message. The messaging technique is reliable and durable due to the "store and forward" mechanism of communication among the destinations. The JMS ensures once-and-only-once message delivery [40]. It also strives to maximize portability of messages within different cross-platform products inside same messaging domain. Besides, the JMS allows a seamless integration of applications with high scalability. It supports a big number of market available APIs (JDBC, Java Beans components, etc.) in different layers of the middleware and realizes asynchronous messaging of serialized data object.

- *Message Properties:* The Java messaging service allows more flexibility over communication among distributed components. The messages are inter-operable, structured, serialized object. The messaging properties admit two delivery modes: The persistent (the message delivered once and only once) and the nonpersistent (the message is delivered at most once, i.e. the message may be lost if JMS server fails). The JMS also offers configurable properties for message expiration, time-stamp, priority settings, etc. The Message properties helps the message selector to retrieve the message from multiple topic and run a filter program as specified by a developer. The JMS permits huge scope of programming. Furthermore, the messaging provides acknowledgement mechanism to assure delivery. A message may even be read-only. A fair amount of security policies is also deployed in relation to authentication, authorization and confidentiality on JMS destination components [35].

Actually, the specification of JMS does not cover all the high-level properties, such as load balancing, privacy, integrity, etc. The JMS is implemented as an open-standard. It inspires vendors with a big portion of implementation specific functionalities. There is no implemented system-message for error notification also. The JMS specification

does not define administered objects and JNDI components for the administration of messaging products. Actually, it is a high level specification that does not describe low-level wire protocols or storage type [35]. After analyzing all these aforementioned advantages and facilities we have chosen the Java message service to design and implement the proposed EAI integration infrastructure.

2.6 Decision Support Systems

The aim of this research work also includes the design and implementation of an indigenous user interface that exhibits certain decision support capabilities over the proposed integration infrastructure. As mentioned before, the idea of the so-called Digital Cockpit represents such a decision support system. Prior to the elaboration of this digital cockpit platform, this section presents the state-of-the-art of existing decision support systems.

In the past, several organizations used to run manual analysis and interpretation to turn data into useful knowledge, required to decision makers. However, the immediate drawback of this decision making procedure is the proof of correctness. Moreover, manual analysis is impossible while dealing with large databases of terabytes of information, example: In Europe Very Long Baseline Interferometry (VLBI) owns 16 telescopes, each of which produces 1 Gigabit/second of astronomical data over a 25-day observation session and requires in-depth analysis of the gathered information [17] . As a part of their research in University of California-Berkeley Professor, Peter Lyman and Hal R. Varian estimated that 5 exabytes (5 million terabytes) of new data was created in 2002. Therefore, the need of an automated system turns obvious for the true analysis of this huge information [16]. The concept of the DSS was first claimed by Peter Keen and Charles Stabell [13]. In the 70's, business journals

emphasized more on management decision systems, strategic planning systems and decision support systems. These first versions of DSS were developed to help decision makers to take educated decisions in complex fields such as financial management and strategic decision making. A few years after, DSS have been enlarged to support spatial, multidimensional and unstructured data, example: GEODATA (GADS). With the advent of web technologies, the data warehousing and the On-Line Analytical Processing (OLAP) solutions became famous in 90's. The OLAP software provides fast, consistent, interactive access to shared information and exhibits analytical views such as time series, trend analysis, etc.

2.6.1 Model of Decision Support Systems

Figure 8 represents a general model of the decision support systems. There are three main architectural components:

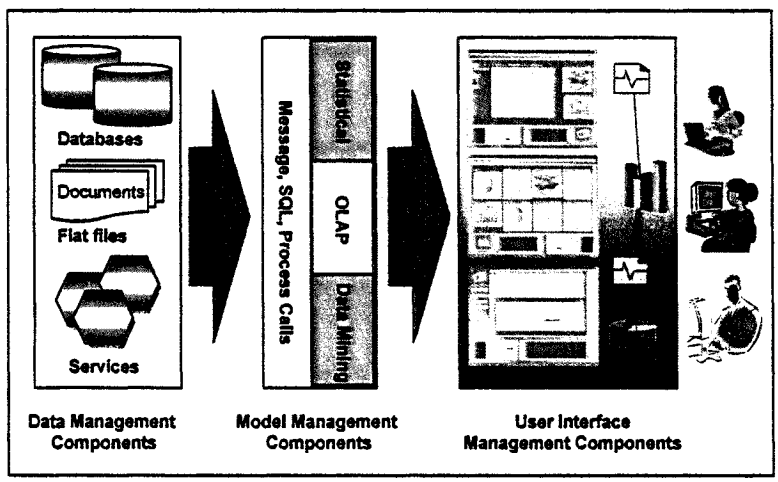


Figure 8: General View of a Decision Support System Model

- *Data Management Component:* The data management component stores and

maintains the information required for a standalone DSS system. This component integrates the organizational, personal, external information kept inside or outside the institution. Recall that, as discussed before, the proposed message oriented middleware system implements a message-based data management component that retrieves the information from the information sources and services in the network.

- *Model Management Component:* This component is responsible for the execution of the business logic. While a user requests a decision, the model management component calculates the merit of each possible solution and represents the optimal solution as the output. These optimization techniques include analytical processing like the OLAP, statistical analysis like the time-series, mission planning etc.

- *User-Interface Management Component:* A user interface is obvious that allows users to communicate with the DSS. Traditional user interfaces with piles of data and tables are frequently used in the industries although there exists several disadvantages associated to it.

2.6.2 Classification of Decision Support Systems

The decision support systems belong to two big categories: The data-oriented DSS and the model-oriented DSS. The data-oriented decision support systems focus on the databases, e.g. file-drawer systems, access data systems data analysis systems, etc. The model oriented decision support systems consist of vaious accounting, optimization and suggestion model that provides analytical capabilities, e.g. OLAP. However, note that the capabilities and objectives of DSS are different according to their use. On the other hand, the decision making procedure consists of four steps:

- *Intelligence:* The intelligence helps to automate discovery of the problem, the needs and the opportunity.
- *Design:* The design exhibits the ways of modeling all possible solutions of a problem.
- *Choice:* The key of the decision support is its way of choosing a particular solution by judging and comparing the merits the of each solution.
- *Implementation:* The implementation presents an optimal feasible solution and produces it in the output.

The decisions may also be classified as structured or unstructured. A structured decision indicates certain information in a specific way while an unstructured decision presents multiple correct decisions according to their ways of calculation. A decision may be a single decision, may be a crossover of combination of good outcomes in a specific way or even a randomly chosen mutation solution where each combination is evaluated. Decisions may also change recursively or infrequently.

2.7 Summary

This chapter presents the state-of-the-art of the notion of the enterprise application integration. Actually, an EAI infrastructure offers to retrieve and store the information kept in distant data sources and services so that the local application may process a request without paying much attention to the connection details. The integration approaches are discussed in two sections: The data integration and the service integration. The data integration aims to provide a uniform and transparent access to the enterprise data sources. The service integration, on the other side, includes communication between distant services inside or outside enterprises directly through

the web-based solutions. This chapter also includes a detailed discussion on information communication through a messaging service. The incorporation of the messaging service is the key idea for the proposed message-oriented-middleware. Finally, this literature review provides a general idea of a decision support system that represents the capabilities of the proposed integration solution. With knowledge of this literature review, the next chapter elaborates the notion of the proposed middleware followed by an implementation of this paradigm.

Chapter 3

Approach

3.1 Introduction

This chapter presents an end-to-end message-oriented-middleware solution for the integration of information sources and services. The intention of this proposed solution is threefold: First, it elaborates a multi-tier software platform that integrates heterogeneous and autonomously-administered information sources and services inside an enterprise. Second, it fulfills the long-standing organizational requirement for a reliable message based middleware system that represents dynamic, nearly real-time notification of changes in remote information systems. Finally, it elaborates a unified platform to implement the decision support capabilities by providing real-time information. After an introductory overview, this chapter presents a multi-tier message-based model of the integration infrastructure. The performance of this model is validated and compared with similar synchronous RPC-based middleware. In addition, this chapter describes an architecture to implement such asynchronous model for data integration. This architecture is later extended for the purpose of service integration and monitoring. Thus, the proposed solution succeeds to leverage a standard-based integration for the heterogeneous information systems that may help to take sound

and consolidated decisions by providing accurate and up-to-date information.

3.2 Overview

Several standard commercial software solutions implement tightly-coupled RPC-based middleware. However, this tight coupling between a request and a response restricts the system to receive up-to-date information without executing periodic process calls. Furthermore, the remote process call suffers from several limitations while it travels from one virtual machine to another. Let us take an example. Java servlet applications are often used for data retrieval from databases. While the user submits a request to a server, the servlet application invokes a remote process call as a payload over the communication protocol. This remote process call triggers a process in the remote data-source that collects the parameters, performs specific operations and finally responds to the requesting user through the server. The limitations of this RPC-procedure are: First, the information retrieval is always bound to the request from a user. Second, a process call blocks the caller application until it receives the response. This decreases the overall performance of the system. Third, the user application, the server and the information sources are required to be available in the network. Finally, the server requires detailed knowledge of the properties and distribution of the remote data sources and services. However, the information sources and the services are the properties of certain organizations or departments. These organizations are run by their own policy of information classification and distribution. Therefore, such sharing of knowledge is sometimes unacceptable due to privacy and security reasons.

The proposed model is carefully designed to overcome all the aforementioned

pitfalls that are associated with the existing technologies. This asynchronous multi-tier model uses a message service instead of directly invoking process calls. The use of the message, instead of process call, offers more flexibility to the solution. The proposed model offers the capabilities listed below [2]:

- *Integration of Information Sources:* This model elaborates an infrastructure that offers a user to request once for an information and subscribe to the state of the information as long the user is registered to the message server.
- *Real-time Notification:* Whenever an event or a change of information occurs in the information sources or services, this message-oriented-middleware model is capable to readily notify about such a change.
- *High Performance:* The underlying asynchronism of this model is the key to its higher performance over RPC-based middleware.
- *High Scalability:* This model supports the addition or removal of information sources and services by using less number of connections and sharing minimum knowledge about the sources.
- *Autonomy of Information:* This proposed middleware solution is flexible as organizations may independently apply their own rules and regulations to information sources and services.

In the following section, the proposed model is elaborated in three phases. Recall that, as mentioned in the previous chapters, this integration infrastructure is distributed among the information sources and services, message servers and user applications. However, during the implementation of this model, certain modules are reconfigured according to the implementation specific requirements. Actually, the implementation of the digital cockpit is one specific realization of this model.

3.3 Middleware Model

The presented message oriented middleware model is composed of four different tiers as shown in the figure 9: The resource tier, the integration tier, the acceleration tier and the presentation tier. The resource tier elaborates a unified framework over the heterogenous information sources and services. The integration tier represents a unique approach of using and extending JMS capabilities for the asynchronous communication of messages inside the middleware. The acceleration tier and the presentation tier are specific to a user. The acceleration tier includes a subscriber application that receives the requested information and the notification of changes. It also contains the monitoring mechanism. Furthermore, it includes the business policies through analysis and controlling methods that are applied on collected data as required in user system. Lastly, the presentation layer deals with the interface of the user application that shows the capabilities of this middleware through graphs and charts. Notice that the data flow, on the retrieval of a requested information, is shown by directed arrows. In what follows, we detail the proposed layer-based model in three sub-sections, namely: The unified framework, the integration infrastructure and the user application.

3.3.1 Unified Framework

A generic business model consists of heterogenous data sources and services in the resource tier. These data sources range from structural databases to unstructured legacy applications. Similarly, the services use various XML compliant formats to represent themselves. The data formats within these sources follow their own vendor-specific semantics. The proposed model uses "wrapper" applications over data sources and services that unify diverse structures of information into a homogeneous representation.

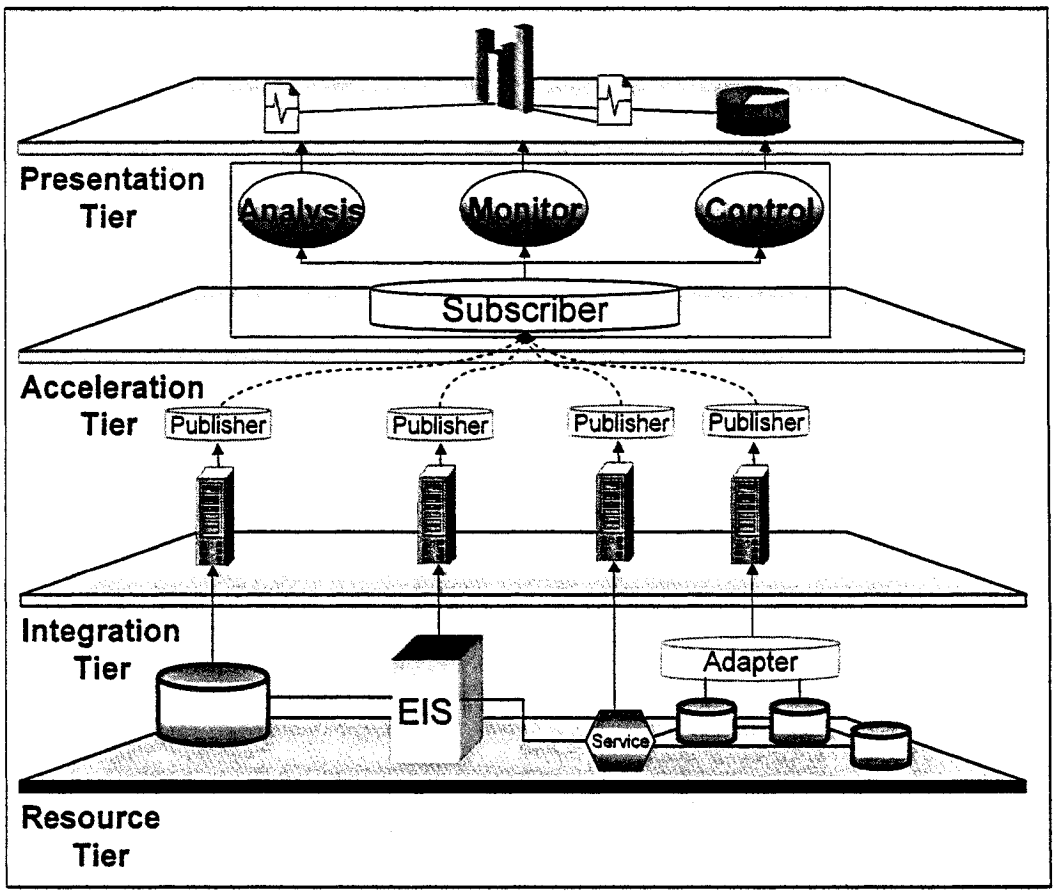


Figure 9: Layer-Based Integration Approach

The implementation of such a wrapper application is resource specific. A wrapper is kept local to the data sources. The user request comes to the information sources or services as an object message. The wrapper application receives the message and converts it into local process/service calls. Therefore, such calls are specific as required for the data sources or services such as: It may be an SQL query for a relational database that uses JDBC connection or it may be a SOAP request to a service. On retrieval of the information, the wrapper again transforms the response into a standard message format common to the proposed middleware system.

The contribution of the wrapper application is twofold in the model: First, it creates a uniform environment for message exchange on top of heterogeneous data sources and services. Second, it ensures the local autonomy of the sources. As the wrapper application resides in the information sources and services, the host may change any data distribution inside the data sources or services and compliantly map the changes to the wrapper application without affecting the whole infrastructure.

3.3.2 Integration Infrastructure

The presented model induces a flexible integration infrastructure that is based on the Java message service. Actually, the message service decouples an application from the remote source. The layer of the messaging service intercepts the user request, converts it into a serialized object message and adds implementation-specific headers to direct the message towards the target source. In the JMS server, a request-response broker extracts the connection details from this message object. The JMS broker stores this information through Java Naming and Directory Interface (JNDI). Such a stored connection helps to notify a registered user on the occurrence of an event.

This middleware uses point-to-point and publish-subscribe communication of information as required. The Java message service implements the point-to-point connection in the form of a queue while the publish-subscribe is implemented as a topic. In the beginning, a client system requires a collection of information to start the display of components. The message service provides a specific connection component, called queue, to retrieve such information. Once the client system starts displaying retrieved information, it only requires to gather real-time notification of changes. Figure 9 presents the data flow related to the propagation of changes into a user application. With the update of data, the underlying infrastructure automatically triggers a JMS publisher to publish information in a pre-existing JMS-Topic. The registered user applications listen to the updates published in a topic.

The integration infrastructure offers multiple features to the proposed middleware solution:

- *Asynchronous and Loosely Coupled:* The asynchronism allows a user application to request an information without waiting for its response. The user application may perform other tasks in this duration.
- *Real-time Notification:* The user application may bind itself to a JMS server and stay connected to the remote sources. Therefore, once the user requests to monitor a set of information it may automatically receive notification of changes on that remote information.
- *Flexibility:* The message service offers flexible communication of information

among the virtual machines. In contrast, remote process calls limits communication between processes in different virtual machines due to security restrictions.

- *Quality Control:* The message service provides more control on data communication. The developer may enforce several quality attributes such as reliability, priority, secured messaging, etc.

- *Easy Extensibility:* The message service allows easy extensibility and scalability of the infrastructure without affecting the performance of the system.

- *Message Storage:* The message service implements the "store and forward" mechanism. The information stays temporarily in the connection component that allows for durable subscription and may serve even a group of clients simultaneously requesting the same information.

The following section illustrates the design of a user application in relation to the proposed solution.

3.3.3 User Application

The user application of the presented middleware model depends on a unique subscribing component to receive asynchronous message objects inside user application. This component is referred to as the "subscriber" in this thesis. The subscriber plays two major roles: First, the subscriber sends the request to JMS server and registers to the request-response broker in an assigned namespace within the JMS server. Second, the subscriber invokes the JMS listeners to a particular JMS topic to receive notifications from the remote events [2].

Figure 9 shows the subscriber module at the basis of the acceleration layer inside the user system. When an application user expresses her interest on a certain set of remote information, the message service intercepts the request and creates message objects that may travel asynchronously. On the subscriber, these message objects triggers the subscription methods. First, the subscriber creates a JMS queue inside the JMS server to receive big amount of information from the remote information sources or services. Second, it creates a specific JMS topic, if such topic does not already exist. The subscriber component invokes JMS listeners to this topic. When a new message arrives in the middleware, JMS server publishes it into the corresponding topic and informs all subscribers, who are registered to those topics. Thus the listeners receive notification using a publish/subscribe mechanism while the remote sources are updated. Finally, it redraws the visualization of the respective visualization component with modified information. As discussed before, these subscribers can be synchronous, asynchronous or durable to the JMS topic.

Actually, the user application contains many other components like the monitor, the analysis library, the display APIs etc. The implementation of such components is based on the software requirements. This presented middleware model uses the model-view-control (MVC) and the observer design pattern. The presentation tier acts as a "view" while the acceleration tier requires a "controller" element in the form of "Monitor" module. The later is implemented as an observer that locally controls the threads in the background of the user application. It selects active JMS listener, sends requests to the subscriber component and finally observes, updates and filters the real-time notifications received through the subscriber. The user interface presents an interactive display of information through graphs and charts.

3.4 Performance Scenarios

This section illustrates the performance scenarios of the proposed integration approach. The proposed middleware improves the performance by three means: It consumes shorter time of the information retrieval. It allows single retrieval of the information in existence of identical user requests. Finally, it uses the notification mechanism, which reduces the network load. In what follows, we corroborate these claims by considering a high-level model of the presented middleware, as shown in the figure 10.

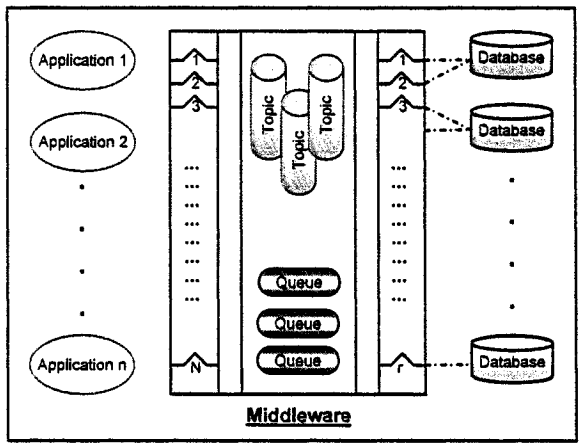


Figure 10: High-Level Model of Integration

Let us assume that the proposed message based middleware model is capable of invoking N threads at a time. In other words, it is capable of serving up to N simultaneous requests from user applications. We also assume that the middleware invokes at most r requests to the resource tier in a single execution. Now if w is the time required for the pre-request processing (i.e. queue and topic creation, etc.) of these N threads and t is the time taken by each request to retrieve the

information from information sources and services then the required time to retrieve all information in a single execution is approximately [31]:

$$\delta = Max_{k=1}^{r}(t_k)$$

The retrieval of the information is asynchronous. Therefore, it does not wait for the response in order to send another request to the information sources or services. In what follows, we verify the the performance scenarios based on this outcome.

Information Retrieval: The proposed middleware model is asynchronous. Therefore, it may receive requests from the user applications irrespective of providing responses to the preceding requests. It means that the middleware model continues to receive user requests independently. So, we assume that each single thread is serving c ($c \geq 1$) requests during the time taken by a single execution of N requests. Therefore, the total time to serve R users' requests is:

$$\mathsf{time}_{MOM} = \begin{cases} ((R \; div \, c \times N) + 1) \times \delta, \; if \; R \; is \; not \; divisible \; by \; N \\ (R \; div \, c \times N) \times \delta, \; if \; R \; is \; divisible \; by \; N \end{cases}$$

where, *div* is the integer division without remainder.

Temporary Storage of Information: The proposed middleware solution allows many user applications to request information from the information sources and services. In reality, most of these user applications contain similar application components being a part of a single enterprise solution. Therefore, it is expected that a group of users issues identical requests in a small duration of time. The presented

middleware model may store information temporarily through JMS-connection components i.e. the queue and the topic. Therefore, with an advanced implementation technique this stored information can be reused to respond to similar user requests. This technique leverages significant improvement in performance as the presented model saves the duration of the information retrieval for the common requests. If there exists g groups with q requests in each of them, on average, then for R user requests, the actual number of processed request is:

$$\textit{Number of processed requests} = (R - (g \times (q - 1)))$$

Furthermore, thanks to JMS, these messaging connection components are always updated to the current state of remote information. JMS guarantees that whenever a change of information takes place in the remote information sources and services, corresponding message will be automatically generated and published to the respective JMS topics.

Notification of Events: The presented middleware model leverages notification thanks to an asynchronous messaging mechanism that is implemented in the JMS topics. Recall that if a user subscribes to particular information, the server creates a JMS topic with respect to this information. This middleware solution publishes message notification on this respective topic if any update occurs in that remote information.

Let us consider the scenario of event notification that is initiated from the information sources or the services. Let us assume that such a source takes T_p time to process x events that occurred due to the change inside the source. Now, inside the middleware, we assume that it requires T_t to process the message information

and to put it in the respective JMS topics. Suppose that, the communication time between the source and the middleware is t_i and the communication time between the middleware and the user applications is τ_i for each i^{th} event. Therefore, the proposed middleware model notifies x events only after the time:

$$\mathsf{T}_{MOM} = T_p + Max_{i=1}^{x}(t_i) + T_t + Max_{i=1}^{x}(\tau_i)$$

3.4.1 RPC-Based Middleware versus Message-Based Middleware

The proposed solution yields better performance than the RPC-based middleware solution. A comparison is presented considering the same model in section 3.4. We compare the performance scenarios of this model for the remote process calls and the messaging service.

Retrieval of Information: In the case of RPC-middleware, each request waits for the response of the preceding one. Therefore, if a request from the middleware to the information sources or services takes t time, on average, to receive a response, an RPC-based middleware solution serves N simultaneous threads by r requests to resource and takes $(r \times t + w)$ time. So, as a whole the time taken by the system to serve R user requests is:

$$\mathsf{time}_{RPC} = \begin{cases} ((R \ div \ N) + 1) \times (r \times t + w), \ if \ R \ is \ not \ divisible \ by \ N \\ (R \ div \ N) \times (r \times t + w), \ if \ R \ is \ divisible \ by \ N \end{cases}$$

where, *div* is the integer division without remainder.

As defined earlier, the time taken by the proposed middleware to serve R requests is:

$$\text{time}_{MOM} = \begin{cases} ((R \; div \; c \times N) + 1) \times \delta, \; if \; R \; is \; not \; divisible \; by \; N \\ (R \; div \; c \times N) \times \delta, \; if \; R \; is \; divisible \; by \; N \end{cases}$$

where, $\delta = Max_{k=1}^{r}(t_k)$ and *div* is the integer division without remainder.

Hereby, it is clear that, with higher values of R, time_{MOM} is much less than time_{RPC}.

Existence of Common Requests: The RPC based middleware uses tightly-coupled point-to-point connection. Such a process call does not compromise the information irrespective of the existence of common requests. So, in reference to our proposed middleware model, RPC-based integration middleware would process all the R requests separately while the message oriented middleware processes a much lower number of requests to retrieve information from the sources. Therefore, the proposed message-based middleware model improves the performance scenario as well as reduces the possibility of bottleneck around the middleware.

Event Notification: The RPC-based middleware does not allow a notification mechanism from the side of information sources and services. Therefore, a user application requires the polling of requests to receive information about the changes. As discussed in the previously presented model, let us assume that an RPC-based middleware receives x requests from the users and processes them in T_r time. As a result, the middleware creates x' process calls to the information sources and services. Now, if these calls takes $T_{p'}$ time, in total, in order to generate x' responses, the RPC-based middleware model receives notification after at least T_{RPC} time, where T_{RPC} may be defined as:

$$\mathsf{T}_{RPC} = 2 \times \sum_{i=1}^{x}(\tau_i) + 2 \times T_r + 2 \times \sum_{i=1}^{x'}(t_i) + T_{p'};$$

Notice that, similar to the previous approach, the communication time between the source and middleware is t_i and the communication time between the middleware and the user applications is τ_i for each i^{th} request. Furthermore, if we consider that a successful change of information takes place after each P poll times, the network will suffer a bottleneck of $P \times (x + x')$ requests.

3.5 Data Integration Architecture

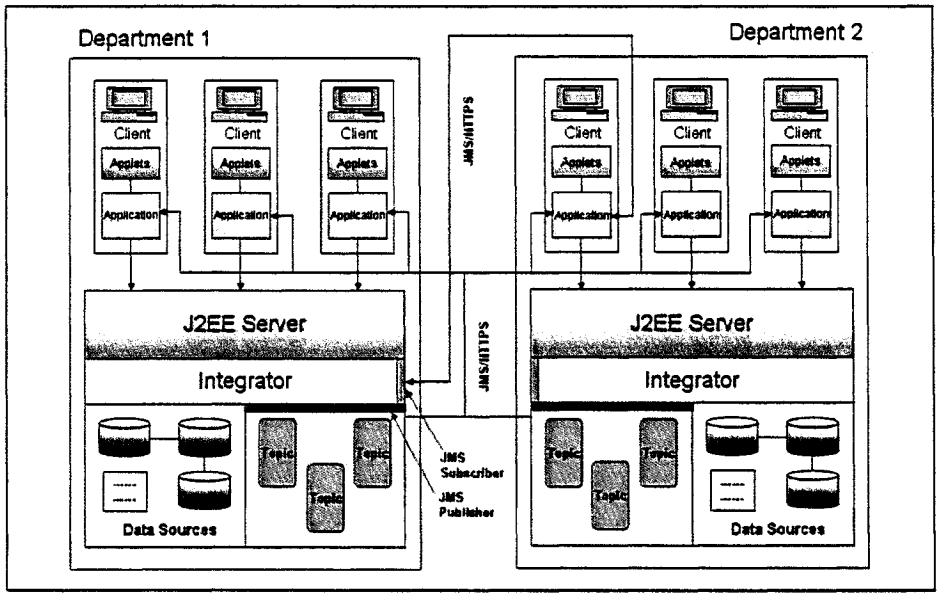


Figure 11: Architecture of Data Integration

Figure 11 depicts the operational view of the proposed middleware architecture integrating data sources. This model illustrates transfer of information between two departments maintaining users and databases that are local to the autonomous departmental environments. The departmental user applications are integrated and controlled through application server/s that are situated inside the department. This architecture brings the enterprise system into a peer-to-peer asynchronous framework

to share information from the sources. Hereafter, this section details the components and the flow of information inside this architecture. However, this discussion is limited to the retrieval of information from data sources. Service integration is detailed in the next section.

More precisely, the middleware architecture includes a message server inside application server/s. An implementation of it could be realized using the Java message service and a JNDI namespace inside the server components. Each server contains an "integrator" module that creates, updates and deletes JMS connection components (i.e. the queue and the topic), which is necessary for information sharing. The user applications are allowed to retrieve information locally or remotely according to their privileges. Initially, when a user application starts, it requires a huge amount of data to start each application component. This infrastructure provides a direct point-to-point communication mechanism between the users and the server/s. The retrieval of information is favored by the asynchronous architecture that might be implemented through a JMS queue. As the user system starts displaying retrieved information, it only requires to gather real-time notifications of remote changes. The integrator module creates a JMS topic component for every requested application component in the users' side. An improvement has been achieved by optimizing creation of such topics by judging dissimilarity in client request. It means that the similar users' requests may receive notification from a single topic. Afterwards, with the updates of information, the integrator module automatically triggers a "publisher" method to publish information on specified topic/s. Whenever a new event arrives to the topic, all its registered subscribers receive notification by their own listening processes. As discussed before, these listeners are implemented as JMS listeners. Finally, the display module of the user application updates the visualization of information on the

user's side.

3.6 Service Integration Architecture

This section details a service oriented architecture to integrate and monitor remote services into the proposed middleware model. The proposed architecture, so far, is only suitable for integrating heterogenous data sources. Service integration and composition is much more complex. Moreover, designing a monitoring mechanism, to receive notifications from remote services, is challenging. This is true because most of the service execution languages implements a one-time execution of web services. The proposed service integration architecture focusses on two main concerns: First, it allows synergistic integration over various services through an asynchronous loosely-coupled architecture. Second, it allows for a notification mechanism to monitor events from remote services, which in turn recomputes the middleware's own business process integration plan to keep users always updated with "fresh" information. Figure 12 presents the underlying detailed architecture. In what follows, we present a brief elaboration of the service integration components followed by a presentation of a request life-cycle.

3.6.1 Integration Components

Integration Container

The need of an integration container raises due to the limitations in the present technologies that are used to compose web services. Web services are composed and executed in one-time fashion. Therefore, the user cannot be notified of any change that takes place after she receives a response from the target service. The present solution to this problem is to keep on requesting the services constantly to display

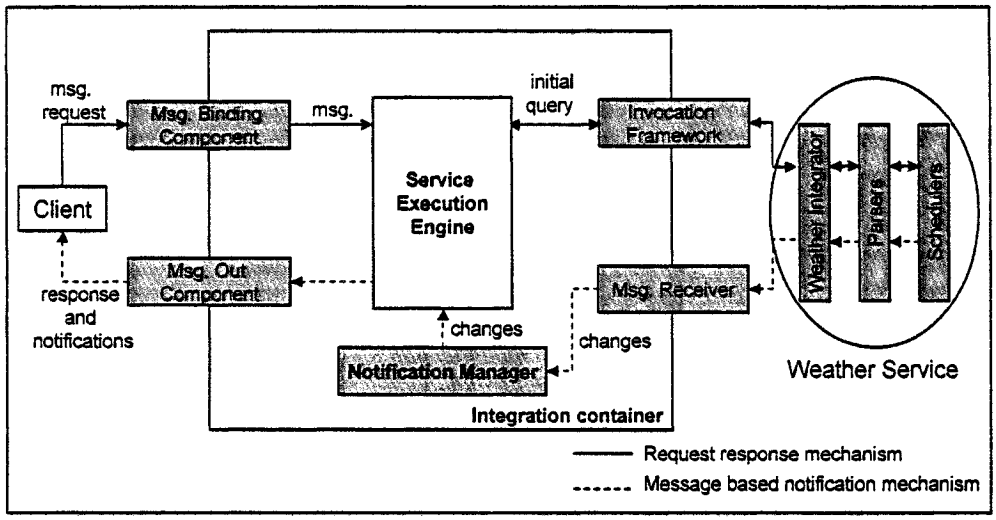


Figure 12: Architecture of Service Integration

updated results. The proposed architecture includes an "integration container" to solve this problem. The integration container ensures that the user interacts only once with the system indicating her interest in some services. After this point, the user is automatically guaranteed to receive the result, followed by all the remote changes on those services whenever they occur. Such an integration infrastructure helps to plug various decision support capabilities and analytical methods over accurate and up-to-date information. In addition, the integration container allows for the execution of sophisticated business rules to re-execute the main service optimally that retains the previous states of operations. Therefore, in case of any remote change in service output, the integration container does not require the re-execution of the entire global process.

Notification Manager

Once a user sends a request to the service(s), the integration container registers its notification manager to all the participating services that are identified during the

design of the main business process execution plan. It is assumed here that services exhibit a scheduling mechanism, which informs the notification manager that a change occurred in its previously requested information set. The notification procedure can be better implemented using the message service. After receiving this notification, the notification manager injects this information in the execution manager, which in turn will initiate another re-execution of certain processes to update the output of the global business process. Finally, this new global business process updates the user again by using a message service based notification mechanism. Actually, the notification manager is the component of the integration container that frees the user from the burden of periodic polling to observe changes of data corresponding to a service.

Execution Engine

The execution engine resides inside the integration container that contains business logic of the service composition framework. Once the integration container receives a service-request from a user, it passes the request to the execution engine. At this point, the engine initiates a global planning of service composition though a main process. The global planning identifies the sequence and the parallelism among the target services. It may also decompose the user's request into many smaller requests, if needed. Afterwards, it executes the service composition plan.

3.6.2 Request Life-Cycle

The newly added integration container complements the existing architecture by providing means for service integration and monitoring. This container is the core, providing the ability to detect and reflect real-time changes without user intervention. The process of service integration is dynamic and initiated by a user request.

For the sake of illustration, a single user and a remote "weather service" is considered for the integration and monitoring purposes. The message request comes to a binding component and is translated into a normalized message, as accepted by the service execution engine. The engine starts a business process in response and decomposes the request into one or many standard queries according to implemented logic. These queries travel to the input of standard remote services and retrieve the result in response. The business process captures the result in different states of its logic and composes them towards the solution of client's request. Finally, the output is sent as a normalized message that is again converted to a message response by the binding components and reaches to the user. Furthermore, the information at the end of a remote service often changes with time. The proposed middleware ensures to reflect this changed output through the active display component. The process of notification, inside the container, is integrated into the service execution engine by a notification manager. This component starts listening to the remote services during the initial execution of the business process and notifies the business process with updated information in an acceptable format. The retrieval of the notification is assumed to be event-based and originating from the service. In figure 12 the weather service runs scheduler/s that checks for any change of weather information and produce with an asynchronous message to a "message receiver" binding component for any successful update. This component, in reply, informs the notification manager that decides on the state of re-computation required in the business process and finally the process notifies the user by partial regeneration of the output. It should be noted that inner/outer interaction between the system components and the services is modeled asynchronously using messages.

3.6.3 Summary

In this chapter, we presented the model underlying the proposed middleware solution. This message-based solution allows for the independent integration of data and services across heterogeneous platforms and network, with ample scope of adopting organizational policies. The intended message-based middleware structure allows flexible integration of multiple sources of information and services as the messaging service provides a uniform environment of information sharing. Unlike traditional solutions, this model supports user applications always up-to-date in terms of the state information into the remote sources. The proposed model is proven advantageous than the mostly deployed RPC-based middleware. Furthermore, this chapter included two integration architectures for data integration and service integration, deduced from the presented model. The architectures are generic and efficient. However, they are described from a very high level without much elaboration of their technologies and limitations. The details of the integration procedures are presented in chapter 4.

Chapter 4

Design and Implementation of Digital Cockpit

4.1 Introduction

The digital cockpit project is a proof of concept that implements the proposed message-based middleware model, guided by specific organizational requirements. The main objective of the digital cockpit solution is to enhance enterprise applications by executing high performance decision support techniques based on accurate and up-to-date information through a message-based and integrated middleware platform. The design and implementation of the digital cockpit is presented in three sections: The first section elaborates on the software requirements, as have been exposed in the requirements specification (SRS) document of the project. It presents the high-level domain model and use case diagrams of this project. The second section details on the software design phase. It presents the main approach, architecture, class and sequence diagrams and the methodology to program such a system. The final section details on the implementation. It includes short comparisons of different commercial off-the-shelf technologies that may implement different modules of the

digital cockpit. Finally, this chapter presents few screenshots of the digital cockpit user interface.

4.2 Software Requirements

This section presents the requirements of the digital cockpit project after an analysis of the identified functionalities and aspects. We used the "Six Sigma" methodology to analyze the identified requirements into a "Quality Functional Deployment" diagram. The figure 13 presents the "Quality Functional Deployment" diagram. It presents the importance of different quality attributes. A considerable concern has been expressed for a secured infrastructure. This requirements are further mapped through a high level domain model of the system. Actually, as exposed, the digital cockpit solution is an independent piece of software that runs on its own enterprise resources. The user application of the digital cockpit solution is capable of integrating organizational data sources and services. A sophisticated user interface is also required for such a user application to reflect the accurate information of various decision parameters. More precisely, this user interface presents an overview of the "combined picture" ranging from monitoring the changes of data in real-time to perform information analysis and optimization procedures. On the other side, the server applications of the digital cockpit solution are integrated by a peer-to-peer asynchronous middleware that follows the proposed paradigm. These servers allow to perform independent and inter-dependent tasks.

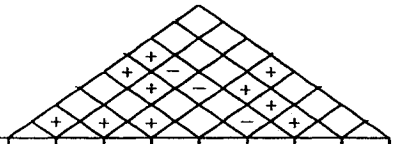
4.2.1 Domain Model

Recall that the intention behind the digital cockpit solution is to develop a real-time monitoring mechanism that displays various kinds of information from diverse remote

● Strong interrelationship (value = 5)

■ Medium interrelationship (value = 3)

▲ Weak interrelationship (value = 1)



		Direction of improvement			↑	↑	↑	↑	↑		↑	
		Customer Requirements / Technical Requirements		STD	Design				Complexity			
			Customer importance	Standard-based	Architecture	Object oriented analysis	Graphical User Interface	Use of Avaliable APIs(JCA, JMS..)	J2EE support	DataBase support (Oracle, Sybase, etc.)	IDE and management console packages	Total
Technical Criteria	Runtime Q. A.	Performance	5	●	●		■	●	●	●		140
		Security	5	●	●			●	●	■		115
		Availability	4		●			●	●			60
		Functionality	4	■		●			▲			36
		Usability	5				●	●	●		■	95
	Non-Runtime Q.A.	Modifiability	4	●	▲	●		▲	■			60
		Portability	2	▲						●		12
		Inter-operability	4	▲				●	●	■		56
		Integratability	3		■			●	●			39
		Reusability	3	▲		●				●		33
Business Q A		Documentation	3	■	■	■	■				●	51
		Cost	2	●	●	■	■					32
		Project life-time	3	●	▲	▲	▲			■		33
		Training	3				●				●	30
		Total		125	105	73	73	134	146	86	45	

Figure 13: Quality Functional Deployment of the Digital Cockpit

information sources and services. By hosting this Digital Cockpit solution on the customer's systems, users will be able to access heterogeneous databases and services seamlessly, monitor data in a real-time mode and perform various data analysis and optimization procedures. These customer's expectations may be realized by providing such functionalities through the distinguished packages inside a domain model. The designed domain model is divided into seven components.

- *Integration module:* Provides a uniform access to local and remote data as needed by the digital cockpit system.
- *Display module:* Visualizes digital cockpit components with a group of features, e.g. "drill-down", zoom etc.
- *Subscription module:* Handles the subscription to and un-subscription from data and services according to interest of the user.
- *Monitor module:* Helps to identify active elements on the user interface and retrieves their information changes.
- *Analysis module:* Analyzes displayed information to find trends, what-if scenarios, cause and effect and even probabilistic operations to compare information to the past data.
- *Control module:* Uses specific methods to optimize and compare possible analyzed scenarios and thereby supports accurate and consolidated decisions.
- *Security module:* Allows user authentication and authorization with necessary event logging.

These modules together provide for all the required features. The figure 14 represents inter-relationships among these packages.

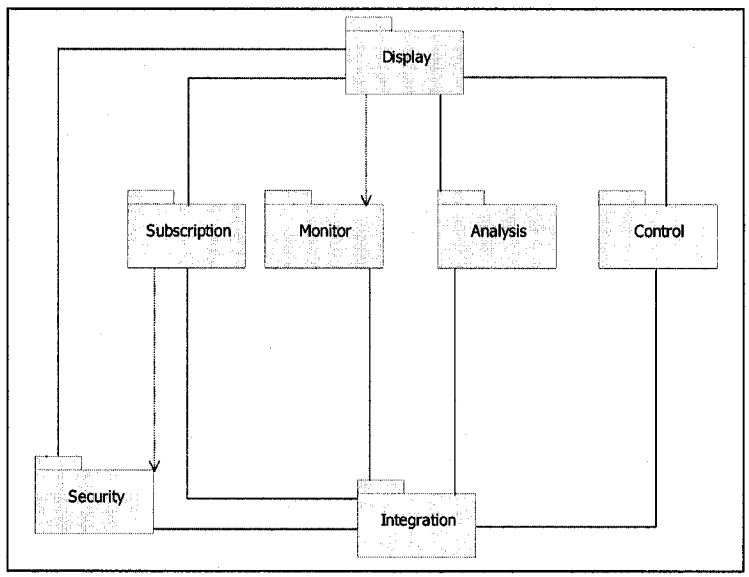


Figure 14: Domain Model of the Digital Cockpit

4.2.2 Use Case Model

The users of the Digital Cockpit system are of three types, as proposed. The first type of users is the "normal user" who use the system as per their privileges. They are capable of monitoring, analyzing and controlling different views generated from the available data sources and services in their domain. The second type of users is called the "power user". The power users are able to track the previous and present operations of the normal users based on their own privileges. A power user also has all the privileges of a normal user. Finally, there is an "administrator" for the whole cockpit system who is capable of exploiting all the features of the digital cockpit solution including the creation of a connection to a new set of data sources and services. Furthermore, a digital cockpit system may also connect and use the information and services from another remote digital cockpit system by proving their identity to the system. Figure 15 represents the overall use case model that illustrates

the capabilities of the users on the digital cockpit system.

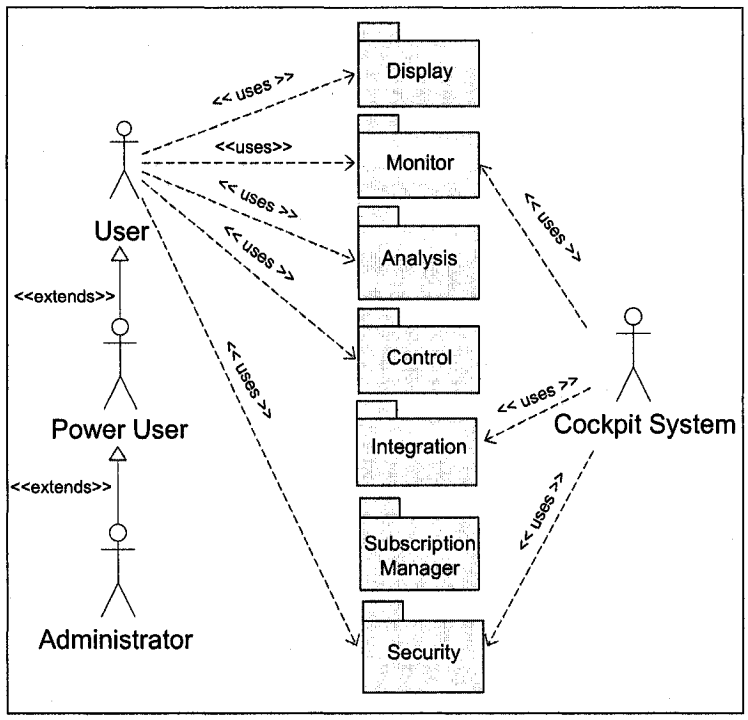


Figure 15: Use Case Model of the Digital Cockpit

For the lack of the space, it is impossible to report the domain model and the use case model of each module. However, a further refined vision of this model may be extracted from the class diagram. The names of all available use cases are listed in the appendix. In the following, we attempt to explain an example sequence diagram [c.f. Figure 16], to represent a possible interaction between two domain model components. This sequence diagram presents the initialization of a display component in the cockpit after verification and confirmation of the user privileges and the subscription to that service. The system establishes a session with each service provider and starts interacting with it. Afterwards, the system starts real-time monitoring of the target data set. On successful execution of the information, the digital cockpit

system presents such an information through an application component.

4.3 Software Design and Methodology

This section details the software design of the digital cockpit system. First, it presents the assumptions and policies of the software design based on the aforementioned requirements. Second, it includes a five-phase implementation guidelines of our methodology followed by the design considerations and the class diagrams.

4.3.1 Assumptions and Policies

Architecturally, the proposed middleware deploys a distributed structure on top of an existing client-server architectures which are common to intra-departmental network. The design of the system is based on certain user assumptions and policies, such as:

- The software is designed assuming the existence of two kinds of different data sources, one kept in the Oracle and other in the Sybase database. The service integration is implemented by integrating a commercially available free weather-service information developed in DWML format.

- This software is meant to provide a prototype of the proposed solution. Each part may be further refined for operational deployment. The recommended technologies are also mentioned in the appendix that can better fit for each of these modules.

- The digital cockpit solution is designed in a way that all its corresponding modules can be implemented using the standard based APIs. These APIs that are well established in terms of technology, performance and other software quality attributes.

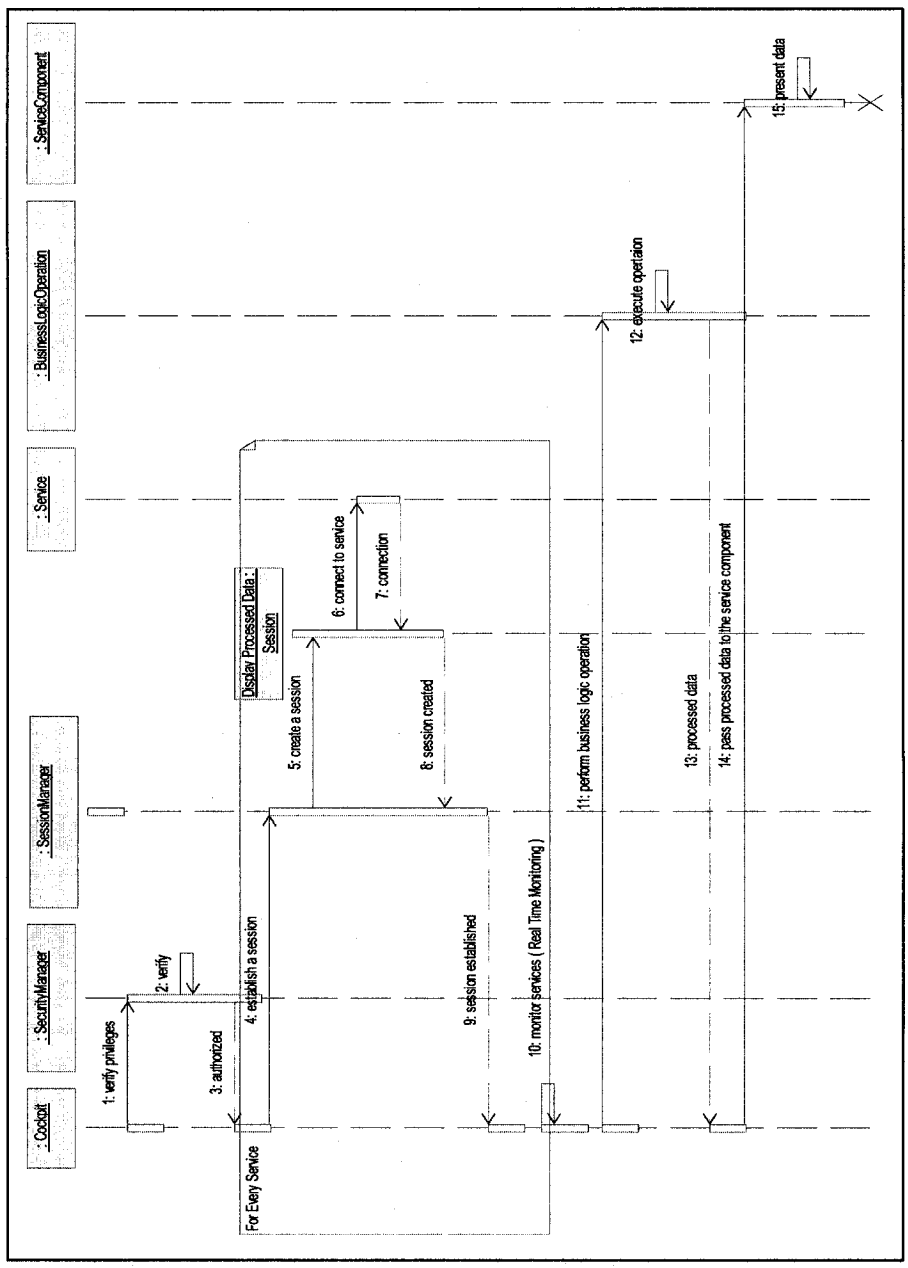


Figure 16: Sequence Diagram of Display Module

- The software design and implementation are provided considering known static sources of information. The dynamic integration of data sources and services is beyond the scope of this implementation.

- The digital cockpit graphical user interface is quite different from traditional graphical user interfaces. It shows most of the required information from a single screen as well as allows users to navigate in to the information details by clicking on a particular component. It monitors and represents changes of the information under real-time inspection, without any user intervention.

- According to the requirements, the digital cockpit prototype supports mathematical analysis such as: statistical operations, time-based analysis, simulations, etc. The implementation of a full-scale decision-support model is yet to be realized inside the scope of implementation.

- The whole architecture of the system is planned keeping the security constraints in mind. The TLS/SSL over HTTP is used as a cryptographic protocol for the secure transfer of information as well as authentication.

- Being a prototype, this solution hardly has any scope to be widely tested and verified for its performance properties.

The intention behind the digital cockpit paradigm is to display and to monitor remote information sources and services directly from a software interface that also enables performing various data analysis and optimization procedures to leverage better decision support.

4.3.2 Methodology

Recall that, the high level design methodology of the digital cockpit project is realized by a five-phase guideline as follows:

- *Integration*: to connect all information sources and services within and across the organization, for information sharing purposes.
- *Display:* to take the data from different sources, aggregate them and present the synthesized information into a meaningful, structured and big navigational "picture" that offers the ability to drill down into the details.
- *Monitor:* to design and implement the capabilities that allow for the active monitoring of the information system state for the purpose of testing the organization's assumptions, reactive and proactive measures, and response to cockpit thresholds, etc.
- *Analysis:* to insert the required business logic to the integrated system i.e. to design and implement the capabilities for time and trend analysis, simulation of "what-if" scenarios, etc.
- *Control:* to optimize procedures, events and scenarios that may enhance the used processes, methods and strategies.

This five-phase design methodology provides a cost-effective, standard based solution that is scalable, efficient and secure. Table 3 briefly illustrates these advantages.

4.3.3 Class Diagrams and Technology

The digital cockpit offers users an unprecedented level of information hierarchy through a layer-based approach. It allows the access to the real-time information by readily accepting changes in the remote information sources and services. The overall architecture of the software is designed through three modules: The integrator, the monitor, and the display modules. The integrator module implements the digital cockpit server on top of the JMS enabled enterprise application server of the enterprise. This module deploys a namespace using JMS server that contains JMS

Cost-effective	The proposed solution is mostly based on free technological components.
Secure	The proposed solution is designed with the information security in mind.
Standards-Based	The proposed solution is based on published, well-accepted and widely-adopted standard APIs.
Scalable	The proposed solution is designed to meet the current organizational requirements and also scalable to support new functionalities in the system.
Efficient	The proposed solution is efficient since it uses performance-proven technological components.
Proven	The proposed solution is proven since it has already been partially demonstrated.

Table 3: Advantages of Digital Cockpit Solution

connection components and helps clients to store necessary information and updates during the process run. The digital Cockpit addresses two major concerns of information sources and service integration. First, it achieves a synergistic integration of the various data/service sources in an asynchronous loosely-coupled architecture. Second, this platform uses a notification mechanism from the integrated sources that publishes the effect of changes of the information sources and services directly in the JMS components. The monitor module, on the other hand, resides in the user's side that runs in the back-end and implements the JMS point-to-point and publish-subscribe mechanisms with the server. As of its functionalities, the monitor module sends the users' request through the subscriber component and persistently monitors changes that are published to the topics inside the server. The display module is local to each client and is treated as a desktop application. Once the information changes are received, this module ensures the redrawing of the particular digital cockpit user-interface components that are currently showing such information. Moreover, the

design of the display module also includes analysis and optimization procedures that take a set of the retrieved information as a parameter and process them to the user desired outputs. A set of high level class diagrams is cited in the appendix.

4.4 Implementation

The use of efficient technologies is particularly crucial for a decision support system that constantly evolves with ever-changing organizational information and business rules. This section demonstrates precise comparisons among the market-available APIs for each design module. Such a comparison offers a clear guideline, in terms of technical choices, for further large-scale commercial implementations.

4.4.1 Data Integration

The middleware based data integration is already achieved using remote procedure calls that are implemented in synchronous architectures. The digital cockpit proposes an asynchronous middleware paradigm that implements message-based information integration. More precisely, the software aims a semblance of end-to-end inter-organizational communication as supported by the Java message service (JMS). The information retrieval inside the data-sources is implemented by a wrapper application through JDBC and/or JCA APIs. J2EE Connector Architecture (JCA) and the Java database connectivity (JDBC) Java APIs implement the querying of information sources. Actually, JDBC and JCA are the de facto standard API from the Java community. The JDBC is used for accessing relational databases. JCA is suggested mainly to access legacy systems (e.g. mainframes), semi-structured (e.g. web pages) or unstructured (e.g. flat files) information sources. The rationale behind this choice of JDBC is for its merits in terms of performance. As of JCA, it grants

an asynchronous mode of fetching information and offers strong pooling mechanism that significantly improves the scalability of the data integration with numerous resources. Furthermore, both of these APIs (JDBC and JCA) are freely available as reference implementations. This is another strong argument that supports the cost-effectiveness of the proposed solution.

The JMS, on the other hand, allows the asynchronous exchange of these retrieved information to the distant users and the other interested digital cockpit systems. The choice of JMS is also motivated by: The cost-effectiveness (freely available API from JCP); standard implementation (Java Specification Request 914); reliability (guaranteed delivery), performance (outperforms other communication solutions such as RPC), etc. In particular, we used the JMS version 1.1 provided by Sun Microsystems with J2EE software bundles.

4.4.2 Service Integration

The implementation of the service integration is comparatively complicated as implemented in this project. The procedures are hereby described with the choices of reference technologies.

Framework

The local communication between the classes for service integration is achieved by interface contracts and the wiring is achieved using an "Inversion Of Control" (IoC) container called Spring Framework. The IoC is an emerging paradigm that greatly improves the reusability, flexibility, maintainability and unit-testability of components. The localisation and instantiation of the classes are transparently achieved

through the "Singleton and Factory Design Patterns". This allows for the full decoupling of the components and system events. There exists numerous open source IoC containers available including PicoContainer, Avalon, NanoContainer, Excalibur and HiveMind. However, "Spring" stands as the best candidate since it has rapidly become the de-facto standard for enterprise application wiring and lightweight J2EE development. Additionally, the Spring Framework provides helper classes that ease J2EE development. Furthermore, Spring integrates with Java Connector Architecture (JCA), Java DataBase Connectivity (JDBC) and other persistent frameworks to provide declarative local or distributed transaction demarcation.

Integration Container

The Digital cockpit implements a standard-based integration using the Java Business Integration (JSR 208) API [25]. The JBI container accomplishes the access to the service in a transport independent manner. Therefore, the integration may take the full advantages of the asynchronous features that may be built into BPEL4WS. The use of this container further improves the necessary asynchronism that are typically absent in a classical SOAP over HTTP approach. The JBI standard being relatively new, there are few Open Source JBI implementations available, e.g. CodeHaus's ServiceMix and Sun's Reference Implementation (RI). Sun's RI provides a more restrictive license than ServiceMix's Apache license. It also has fewer community support and provides a very limited set of Binding Components (BC). On the other hand, the ServiceMix provides the support for Spring as well as JSR 208 deployment unit. It allows with a number of BC and embeds Fivesight's BPEL Process eXecution Engine. Furthermore, it is integrated with other CodeHaus projects such as ActiveMQ-JMS, Jencks-Java Connector Architecture, etc. However, the ServiceMix documentation is still in its infancy whereas the RI's one is a little more complete. Since the BPEL4WS

support was necessary, as well as a JMS BC, the ServiceMix has been chosen.

Business Process

The execution engine runs the business processes that are capable of composing multiple services from distant locations. The "Business Process Execution Language for Web Services" (BPEL4WS) provides a high-level language to compose web services. This proposed middleware uses BPEL4WS as it is a competing standard implementation for the service composition. There are several competing BPEL engines. The most popular open source alternatives include ActiveBPEL, Apache Twister, Fivesight PXE and IBM BPWS4J. The BPWS4J has never been updated and its community support is nearly non-existing. The Apache Twister is still in its infancy and does not provide a complete implementation of the BPEL standard yet. ActiveBPELs documentation and community support is excellent, but it provides no other transport than SOAP over HTTP. Therefore, ActiveBPEL can not be easily integrated inside a JBI container. Finally, Fivesight PXE has been chosen because of its integration inside ServiceMix. It can be embedded as a Service Engine (SE) inside a JBI container using its extensible architecture. The integration procedure is also available with ServiceMix.

JMS Binding Components

The ServiceMix provides support to a number of binding components (BC) including ActiveMQ-JMS, Jencks-JCA etc. The implementation of the asynchronous distributed communication is achieved using ActiveMQs JMS implementation that is usually considered as the most flexible open source JMS broker. The Spring framework provides helper classes for the JMS templates which help to avoid programming and to maintain the ever-needed initialization and cleanup code. Furthermore, the

ServiceMix supports JMS binding component. However, there is a limitation in the current implementation of Springs JMS template regarding asynchronous reception of messages which can be easily circumvented by using Springs JCA support together with ActiveMQs JCA resource adapter. The JCA resource adapter offers asynchronous receiving of the JMS messages.

Service Notification

The proposed integration approach encourages the use of a "notification manager" inside the JBI container that asynchronously informs the business process instances of the newly available information from the remote services. It should be mentioned that this approach assumes statically known locations of the remote services. An extension of this approach could be realized to find the services dynamically by integrating with an outside XML registry,. However such a discussion is out of the scope of this thesis.

In particular, the digital cockpit project yields an infrastructure of decision support through the integration of a frequently changing weather service. This weather service is composed of a set of schedulers and parsers which aggregate weather data coming from various sources and provided under both XML and legacy formats. The jWeather package has been used to parse Meteorological Aviation Routine Weather Report (METAR) legacy format, whereas the Java API for XML Processing (JAXP) has been used to parse XML weather data available from USA's National Oceanic and Atmospheric Administration (NOAA). The integration container receives changes in remote services data using internal schedulers and notifies the user though the asynchronous JMS notifications. As a consequence, it re-calculates the necessary part of the relevant business process, if required. Finally the integration container

and informs the user by binding component. the user application instantaneously reflects the changes through the GUI. The Java3D and JFreeChart libraries are used for visualizing weather information in a user-friendly way.

4.4.3 Display and User Interface

The digital cockpit solution leverages an indigenous graphical user interface (GUI) to its users. It presents the integrated and synthesized information in a meaningful, structured, navigational and graphical way. The display phase is in charge of providing the dynamic and graphical views to the users decision-makers that may help them to quickly grasp the meaning of the presented data. Furthermore, a permitted user is offered with the capabilities to click on high level summary of information and to view more details of it. The user-interfaces are implemented by dynamically changing graphical objects and representations such as charts, curves, knobs, histograms, reports, animated maps, etc. The dynamic aspect of these views makes the digital cockpit display module much more complex than a simple elaboration of a conventional and static graphical user interface (GUI). As proposed earlier, this display module is much more sophisticated as the decision maker requires to view almost real-time reflection of the changes in the organizational key indicators through the relevant graphical objects. Technologically, such a display module possesses several challenging and interesting problems such as: The listening to the messaging components, triggering and delivery of remote events, binding graphical objects to threads, thread management, animation and rendering of graphical objects based on remote events, etc. Furthermore, it also provides a lightweight library of analytic procedures to analyze information [c. f. figure 17] and to simulate on such analyzed scenarios [c. f. figure 18].

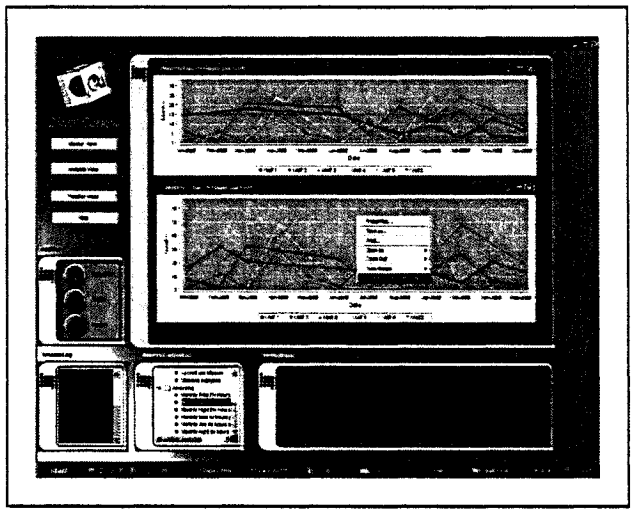


Figure 17: Time-Based Analysis of a Set of Information

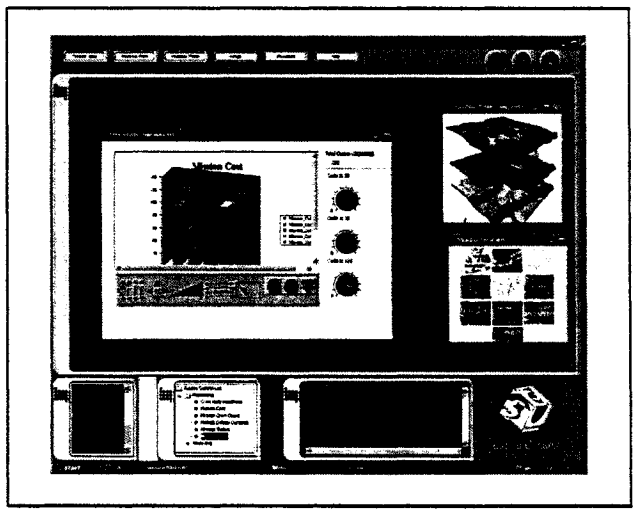


Figure 18: Simulation of an Analyzed Scenario.

The digital cockpit introduces a unique notion of displaying the synthesized information into a meaningful, structured and big navigational picture that offers the ability to drill down into the details. A traditional menu-based user-interface is not enough to represent the real-time changes of information on every application component under active display. This research favors a single layer of the user interface to present the information through various application components. Each application component contains graphs and charts that meaningfully represent a huge set of information in a small place. An interested user may view the details of a particular set of information by clicking on these charts inside the graphical component. There are several available libraries of charting, for example: EsspressChart, JCLASS, and JFREECHART. The EsspressChart is a Java-based charting toolkit that allows the users to build charts easily, interactively and programmatically. JCLASS, on the other hand, is a suite of Java components that help to build applications quickly and easily with charts, tables and reporting/printing features. JFREECHART is a Java open source library that is meant for the generation of charts (e.g. pie charts, bar charts, line and area charts, scatter plots, bubble charts, time series, candle stick charts). There exists some general-purpose visualization tools also in the software market. The examples of these tools are: ADVIZOR, OpenViz, NetChart Reporting Suite, Xcelsius, etc. The ADVIZOR toolkit provides a presentation of business data along with a query capability. The OpenViz is a suite of software components that enables the development and deployment of interactive 2D and 3D visual presentations. The NetCharts suite allows the embedding of charts and graphs into a web-based applications. The Xcelsius suite is a windows application that allows users to create dynamic, interactive reports based on Excel spreadsheets. After a careful review on all aforementioned charting solutions, we have chosen JFREECHART and Quadbase EsspressChart APIs for the implementation of our proposed user-interface.

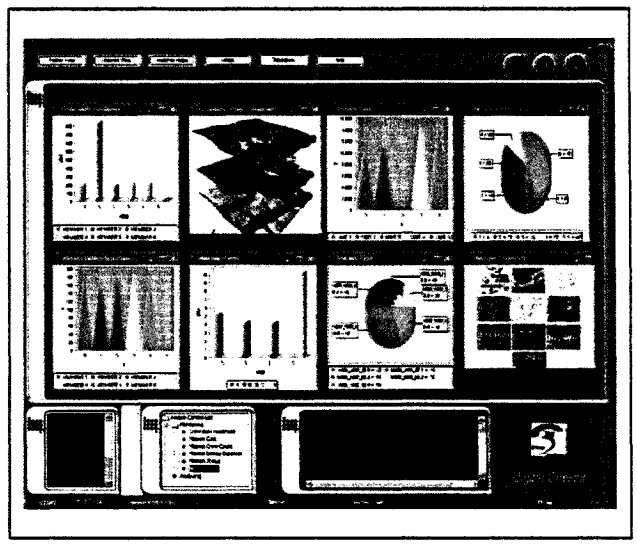


Figure 19: Main User Interface of Digital Cockpit System

In addition, the scope of the display module literally depends on the clients requirements. As for this related project, it implements a big variety of military applications that represent their resources, planning for further missions, GIS maps with required information for military wings, etc. [c. f. figure 19]. Besides, the digital cockpit means for displaying service monitoring as proposed during the implementation. The screenshot presents a weather service integration scenario, in figure 22, that integrates XML weather data, freely available from USA's National Oceanic and Atmospheric Administration (NOAA). A mission planning tool has been shown in the figure 24 that shows how the solution decides on the feasibility various missions. As presented in the user interface, the "plus" sign shows positive feedback over the execution of a mission while the "minus" sign means such a mission is undergoing with high risks. The "exclamation" sign demonstrates the indecisiveness of the tool that

may occur due to less information. It should be mentioned that all this representation is generated in almost real-time and changes with the fluctuations at remote end.

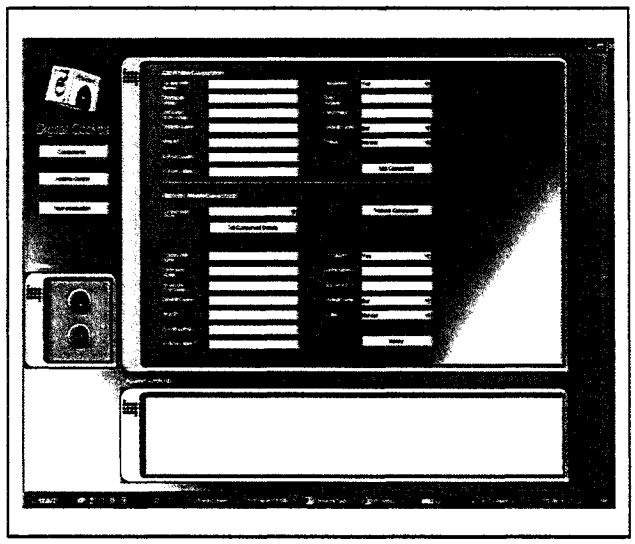


Figure 20: Administrator's User Interface of Digital Cockpit System

The development of this solution strictly follows an "Observer-Observable pattern" inside the Monitor module so that any change of information triggers an update to the corresponding display component. The digital cockpit user interface also contains some typical GUI utilities that are common to the screen elements, such as: Saving, zooming, navigating, etc. Moreover, digital cockpit user interface is customizable for the visualized cockpit components. The decision makers and the users may customize the active display components among the available cockpit-components. Such a customization also depends user's preferences and permission, as set by an administrator [c. f. figure 20]. We used EspressChart API from Quadbase Inc., Java 3D API from Sun Microsystems and JFreeChart API from SourceForge project. The Espresschart API is specially useful for dynamic rendering. The Java3D provides a

high-level programming interface for rendering three dimensional scenes whereas the JFreeChart is the most widely used open source charting library. Some snapshots of the user interfaces are presented from the digital cockpit project in this section.

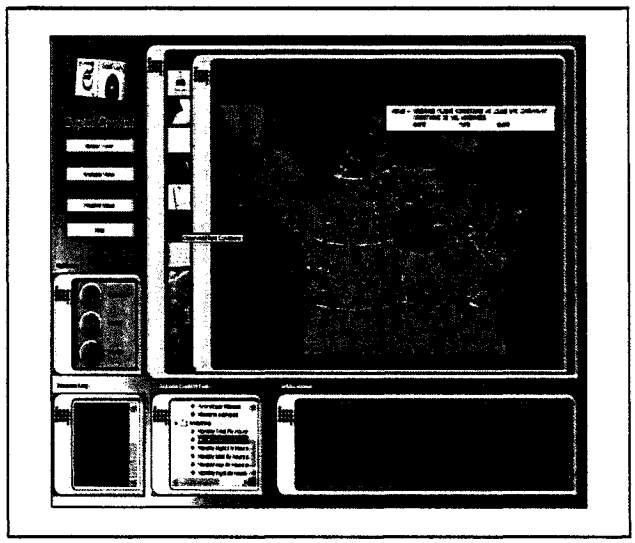


Figure 21: Overall View of Weather Scenario

Figures 21, 22, 23 represent a 'drill down' scenario through the digital cockpit implementation. An integration scenario of the weather service, in figure 21, shows an overall weather map representing certain weather information. An interested user may focus on a specific region of interest and request more information as shown in figure 22. The sufficient service integration successfully enables the platform to represent bulk of information from the remote services. Finally, the figure 23 shows a specific wind condition, that presents the lowest layer of drilled down information.

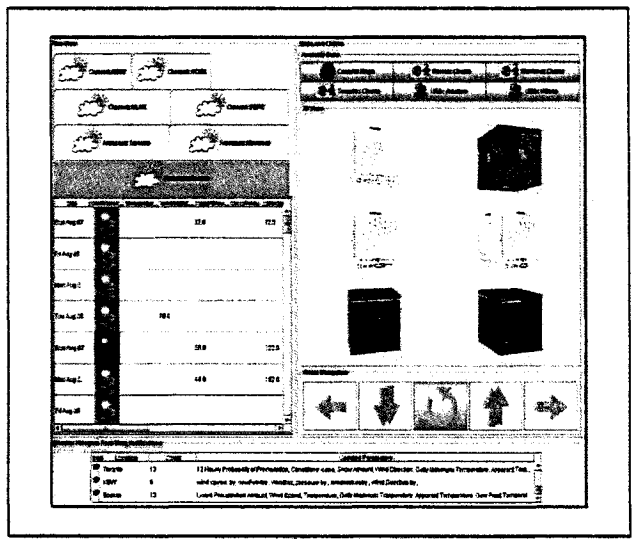


Figure 22: "Drill-Down" Feature for Detailed Information

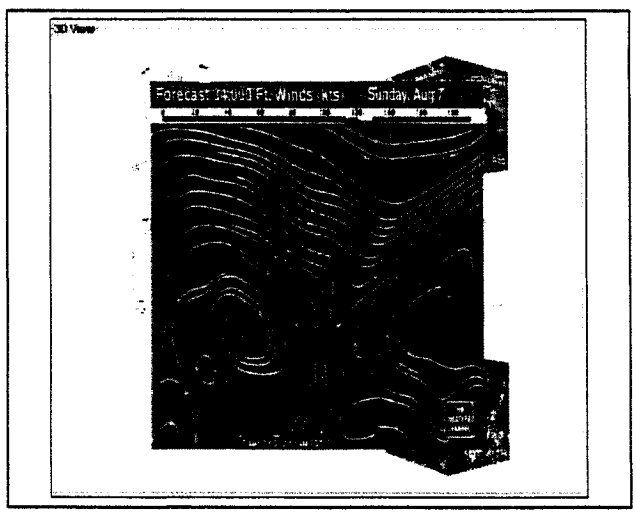


Figure 23: Specific Weather Component: Wind Forecast

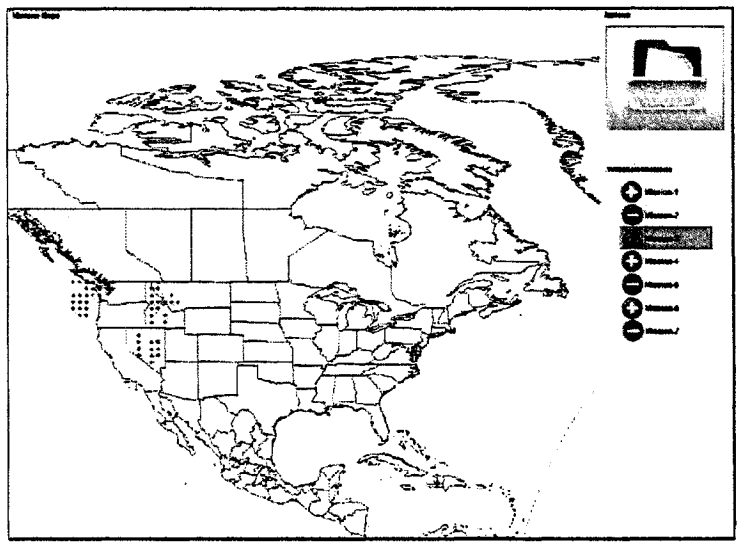


Figure 24: Optimization Scenarios of Mission Planning

Chapter 5

Conclusion

The demands of the software industries are proven in the matter of middleware based enterprise application integration. However, most of the large-cap corporations still use RPC-based middlware for the information retrieval from the data sources. These middleware infrastructures are unable to reflect the real state of the retrieved information, especially when the original information changes in the source after the retrieval. Similarly, the present service integration approaches use a one-time execution of a multi-task business process. So, the business process languages compose remote services but the end-result, once again, corresponds to a static instance of the information received during the time of retrieval. This technology exhibits a crucial drawback since the infrastructure lacks the guarantee on the freshness of data. Therefore, decision support systems (DSS), that are implemented on top of such integration infrastructure, often provide wrong judgements. Our research explores a message-based, asynchronous middleware solution that may improve the integration of the remote information sources and the services by constantly providing up-to-date information to the users. Such a solution will better help to take accurate decisions in an opportune time.

The main contribution of this thesis may be considered as the proposition of a loosely-coupled middleware solution that implements an end-to-end integration platform for heterogeneous information systems. The middleware model leverages an asynchronous mechanism to keep the displayed information consistent with the real state in the remote sources. In this work, we address the architecture, the design and a prototype implementation of such a middleware. As of the validation of the capabilities of such a middleware system, we jointly programmed the Digital Cockpit software solution that relies on the proposed message oriented middleware and explores some of its capabilities.

The monitoring of information systems is a necessity to gather accurate information and to take decisions that span over a long time. This thesis presents an interesting concept of monitoring remote information sources and services. The present RPC-based middleware infrastructures "refresh" information using polling of requests. The presented message-based middleware is a better alternative to provide the solution that accepts real-time notification of remote changes during a process run. Consequently, the output complies with up-to-date scenario of remote information systems. Furthermore, this thesis identifies the scope of the re-computation independently from the integration style. This re-computation mechanism is necessary to deliver up-to-date information. However the details of the re-computation mechanism is complex especially in the context of the service monitoring. The elaboration of these re-computation procedures is beyond the scope of this thesis.

The notion of decision support systems is brought to elaborate the underlying potential of the asynchronously integrated platform to take consolidated, quick and correct decisions using available information and services. Actually, this thesis does

not cover all the possible aspects of the decision support techniques. Furthermore such a decision support system is typically implementation specific. We implemented the digital cockpit solution with some decision support techniques that covers time and trend analysis of collected information, weather scenario, mission planning, etc. However, this thesis contributes a direction of further research to improve the present strategic systems by reflecting more accurate decisions with "fresh" information.

The implementation of the digital cockpit includes an innovative graphical user interface to display information. Instead of using traditional menu-driven user interface the digital cockpit graphical user interface presents the information in a priority-based hierarchy from several sub-components. Such a graphical user interface allows users to see all the desired information without a lot of interaction. Furthermore, the users may view the fluctuations of the presented information due to changes in the remote sources with/without their intervention. Besides, the users may view detailed information from any sub-component by clicking on its information set. The information sets are shown by graphs and charts to present the values. The graphs and charts may meaningfully represent piles of information through a small component.

The digital cockpit software solution still requires fine-grained monitoring of information and automatic discovery of sources and services. The analysis module also needs more statistical, analyzing and optimizing algorithms to be incorporated into it. One possible direction of the future work may also include investigating each aspect of the integration in details, especially from the qualitative point of view. We are still working on the composition and re-computation logic of data sources and services that may provide up-to-date and accurate notification faster in less duration. Second, the idea of decision support system requires further enhancement to deal with

the real complex enterprise situations. A direction of future work could be oriented towards a more formal definition of the system components and use of theoretical artifacts to optimize the goal. Finally, the digital cockpit is a robust and distributed solution. Therefore, the information sharing should be protected from unauthenticated and unauthorized users inside the enterprise. The detailed understanding of the security requirements for such asynchronous and distributed digital cockpit platform will certainly bring more focus on large-scale, real-life implementation of such systems.

Bibliography

[1] Claudio Sacerdoti Coen, Paolo Marinelli, Fabio Vitali. Schemapath, a minimal extension to xml schema for conditional constraints. In *Proceedings of 13th international conference on World Wide Web*, Session: XML, pages 164–174. International World Wide Web Conference, ACM Press, 2004. ISBN-1-58113-844-X.

[2] A.Benssam, A.Boukhtouta, M. Debbabi, H. Issa, S. Ray and A. Sahi. A Message-Based Middleware for Asynchronous Operations: Issues and Experience. In *Proceedings of NOTERE-'05*, pages 51–60, Ottawa, Canada, August-September 2005.

[3] A.Benssam, S.Ray, A.Boukhtouta, F.Guerroumi, C.Assi and M.Debbabi. A new Paradigm for Information Systems Integration. In *Montreal Conference Proceedings on eTechnologies*, pages 63–72, Montreal, Canada, January 2005.

[4] A.Boukhtouta and M.Debbabi and N.Tawbi. A new paradigm for decision making: a synergy between business intelligence and digital cockpits. In *10th ISPE International Conference on Concurrent Engineering: Research and Application*, Portugal, 2003.

[5] Alon Halevy. Data Integration: A Status Report. In *Proceedings of the German Database Conference, BTW-03*, 2003. Invited paper.

[6] Business Process Project Team. ebXML Business Process Specification Schema 6 Version 1.01. OASIS, [online], http://www.ebxml.org/specs/ebBPSS.pdf, May 2001.

[7] C.Davis. Distributed objects and components. UCL Computer Science, [online], http://www.cs.ucl.ac.uk/staff/W.Emmerich/lectures/3C05-02-03/aswe19-essay.pdf, May 2003.

[8] Daniel Drasin. Get the message? *IBM developerWorks*, February 2002.

[9] DataMirror Corporation. Benefits of Transformational Data Integration. [online], http://www.grcdi.nl/benefits.pdf, 2000.

[10] David E. Bakken. Middleware. [online], http://moab.eecs.wsu.edu/~bakken/middleware.pdf.

[11] David S. Linthicum. *Next generation for application integration: from simple information to web services.* Addison Wesley publications, August 2003.

[12] Dianne Kennedy. From SGML to XML; From DTD to Schema; Making the Choice. *XML 2002 Proceedings by deepX*, 2002.

[13] D.J.Power. A Brief History of Decision Support Systems. [online], http://DSSResources.COM/history/dsshistory.html, May 2003.

[14] E.Cerami. *Essentials of web services.* OReilly publications, 2002.

[15] F.Coyle. *XML, web services and the data revolution.* Addison-Wesley Information Technology Series: Addison-Wesley Professional, 2002.

[16] Fred M. Heath. ACLS Commission on Cyberinfrastructure for the Humanities and Social Sciences. Technical report, University of Texas at Austin, Baltimore, Maryland, October 2004.

[17] G.Piatetsky-Shapiro. Machine Learning and Data Mining: Course Notes, KDnuggets. [online], http://www.kdnuggets.com/dmcourse/data_mining_course/course_notes.pdf, 2003.

[18] Heather Kreger. Web Services Conceptual Architecture. IBM Software Group, [online], http://www-306.ibm.com/software/solutions/webservices/pdf/WSCA.pdf, May 2001.

[19] Hyperion Solutions Corporation. An Overview of Hyperions Data Warehousing Methodology. [online], http://dev.hyperion.com/resource_library/white_papers/data_warehousing_methodology.pdf, October 2001.

[20] IBM, BEA Systems, Microsoft, SAP AG, Siebel Systems. Business Process Execution Language for Web Services version 1.1. Specification Release, [online], ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf, May 2003.

[21] IBM Research, Microsoft. Web Services Description Language (WSDL) 1.1. Specification Release, [online], http://www.w3.org/TR/wsdl, March 2001.

[22] IPEDO Inc. Guide to Enterprise Information Integration (EII). [online], http://www.dmreview.com/whitepaper/WID1011596.pdf, August 2004.

[23] Jack McCarthy. Special Report: Six great myths of IT. Technical Report Issue 33, InfoWorld, August 2004. Page 35.

[24] Java Community Process. Java™DataBase Connectivity 3.0 [jsr 054]. [online], http://jcp.org/en/jsr/detail?id=54, 2002.

[25] Java Community Process. Java™DataBase Connectivity 3.0 [jsr 208]. Specification Release, [online], http://jcp.org/en/jsr/detail?id=208, 2005.

[26] Jim Farley. Java Enterprise Breakthroughs, Part 1. *ONJava.com:O'REILLY*, May 2002.

[27] Lev Kochubeevsky. Generic Request Response Broker for JMS. *JAVA DEVELOPER'S JOURNAL*, 10(4), April 2005.

[28] Liviu Tudor. MSMQ Part 1/2: Architecture and Simple Implementation Using VB. Dev Articles™, [online], `http://www.devarticles.com/c/a/Visual-Basic/MSMQ-Part-1-Architecture-and-Simple-Implementation-Using-VB/`, March 2002.

[29] L.M.Haas,E.T.Lin and M.A.Roth. Data integration through database federation. *IBM Systems Journal.*

[30] M. Myerson. Web service architectures. Technical report, Technical Teport, Tect©, 29 South LaSalle St.Suite 520, Chicago, Illinois 60603 USA, 2002.

[31] Brian Maso. JMS: A Solution in Search of a Problem? Blumenfeld & Maso Inc., [online], `http://archive.devx.com/java/free/articles/MasoJMS02/Maso02-5.asp`.

[32] Microsoft Corporation MSDN Library. Integrating Layer: Portal Integration. [online], `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/archprocessintegration.asp`.

[33] R.Hull. Managing semantic heterogeneity in databases: a theoretical perspective. *Proceedings of PODS*, pages 51–61, 1997.

[34] R.Hull and G.Zhou. A framework for supporting data integration using the materioalzed and virtual approaches. *Prac. Of SIGMOD*, pages 481–492, 1996.

[35] Richard Monson-Haefel, David Chappell. *Java Message Service.* O'Reilly & Associates Inc., Java Series, 1st Edition edition, December 2000.

[36] S.Chawathe, H.Garcia-Molina, J.Hammer, K.Ireland, Y.Papakonstantinou, J.Ullman and J.Widom. The TSIMMIS Project: Integration of Heterogeneous Information Sources. *Proceedings of IPSJ Conference*, pages 7–18, October 1994.

[37] S.Haag, M.Cummings and Donald J McCubbrey. *Management Information Systems for the Information Age.* McGraw Hill, 4th Edition edition, 2004.

[38] Sonic Software Corporation. JMS Performance Comparison: Publish Subscribe Messaging. Whitepaper, [online], {`http://www.sonicsoftware.com/products/whitepapers/docs/jms_comparison_tibco.pdf`}, year = 2003, month = November,.

[39] SonicMQ, Sonic Software Corporation. Getting Started with SonicMQ®V6.1. Documentation, [online], `http://www.sonicsoftware.com/products/documentation/docs/mq_getstart.pdf`, September 2004.

[40] SUN Microsystems. Java™Message Service [jsr 914]. Specification Release, [online], `http://java.sun.com/products/jms/docs.html`, 2000.

[41] Sun Microsystems. J2EE™Connector Architecture [jsr 016]. [online], `http://jcp.org/aboutJava/communityprocess/final/jsr016/index.html`, 2001.

[42] S.Vinoski. CORBA: Integrating diverse applications within distributed heterogeneous environments. *IEEE Communication Magazine*, 35(2), February 1997.

[43] W.H.Inmon. *Building the Data Warehouse.* Wiley, 2nd Edition edition, 1996.

Appendix A

Use Cases of Digital Cockpit

- Display Module:
  - Use Case: Initialize components in the Digital Cockpit.
  - Use Case: Add new component(s) to the Digital Cockpit.
  - Use Case: Move component(s) on the Digital Cockpit.
  - Use Case: Change properties of the Digital Cockpit.
  - Use Case: Change data representation inside a Digital Cockpit component.
  - Use Case: Drill into component details.
  - Use Case: Drill out of component details.
  - Use Case: Data import.
  - Use Case: Data export.
- Integrator Module:
  - Use Case: Access local data.
  - Use Case: Access local service.
  - Use Case: Access remote data.

- Use Case: Access remote service.
- Use Case: publish data.
- Use Case: Subscribe data.
- Use Case: Retrieve data for remote requestor.
- Use Case: Add data hook (pre-defined access permission to remote data sources)
- Use Case: Add service hook.
- Use Case: Remove data hook.
- Use Case: Remove service hook.
- Use Case: Save to local information source.

- Analysis Module:
  - Use Case: Request data for analysis
  - Use Case: Pattern identification analysis.
  - Use Case: What-if analysis.
  - Use Case: Probability and statistical analysis.
  - Use Case: Cause and effect analysis.

- Control:
  - Use Case: Optimize data

- Monitor:
  - Use Case: Monitor real-time changes(local/ remote).
  - Use Case: Notify update.

- Subscriber:
  - Use Case: Subscribe information and service.
  - Use Case: Unsubscribe information and service.
- Security:
  - Use Case: Log-on/Authenticate user.
  - Use Case: Log user / administration operations.
  - Use Case: Log system events.
  - Use Case: Privileged update of user information.
  - Use Case: Update of User privileges.
  - Use Case: Retrieve logged events.

Appendix B

Class Diagrams

INTEGRATOR MODULE

CLASS COMPONENTDATAMESSAGE, CLASS DATA_CONN, CLASS DCCLOSETHREAD, CLASS DCSERVERTHREAD, CLASS FILEREADER1, CLASS INTEGRATOR1, CLASS MSGRDR, CLASS OBJECTLISTENER, CLASS READ_MMA, CLASS REQUESTOR, CLASS RESULTSETSENDER, CLASS SERIALIZEOBJ, CLASS SERVICEPROVIDER, CLASS SERVICEPUBLISHER, CLASS SHUTDOWN, CLASS TOPICQUEUEMANAGER, CLASS UPDATETRACKER, CLASS XLIST.

MONITOR MODULE

CLASS COMPONENTCONNECTIONCLOSE, CLASS COMPONENTCONNECTIONREQUEST, CLASS DCMONITOR, CLASS MONITORCOMPONENTTHREAD, CLASS QUEUELISTENERTHREAD, CLASS TOPICLISTENERTHREAD.

DISPLAY MODULE:

CLASS CHARTDATA, CLASS COMPONENTPAGE, CLASS DRDCOMP, CLASS DCPLIBRARY, CLASS DISPLAYMODULE, CLASS EXPORTTOEXCEL, CLASS IMPORTEXCEL, CLASS INTERNALLISTEN, CLASS LOGIN, CLASS MONITORTHREAD, CLASS NUI, CLASS OBSERVABLEOBJECT, CLASS PREFERENCES, CLASS SERIALIZEOBJ, CLASS SERIALIZEDRESULTSET, CLASS TREE, CLASS TREEDATA, CLASS USER, CLASS XLIST.

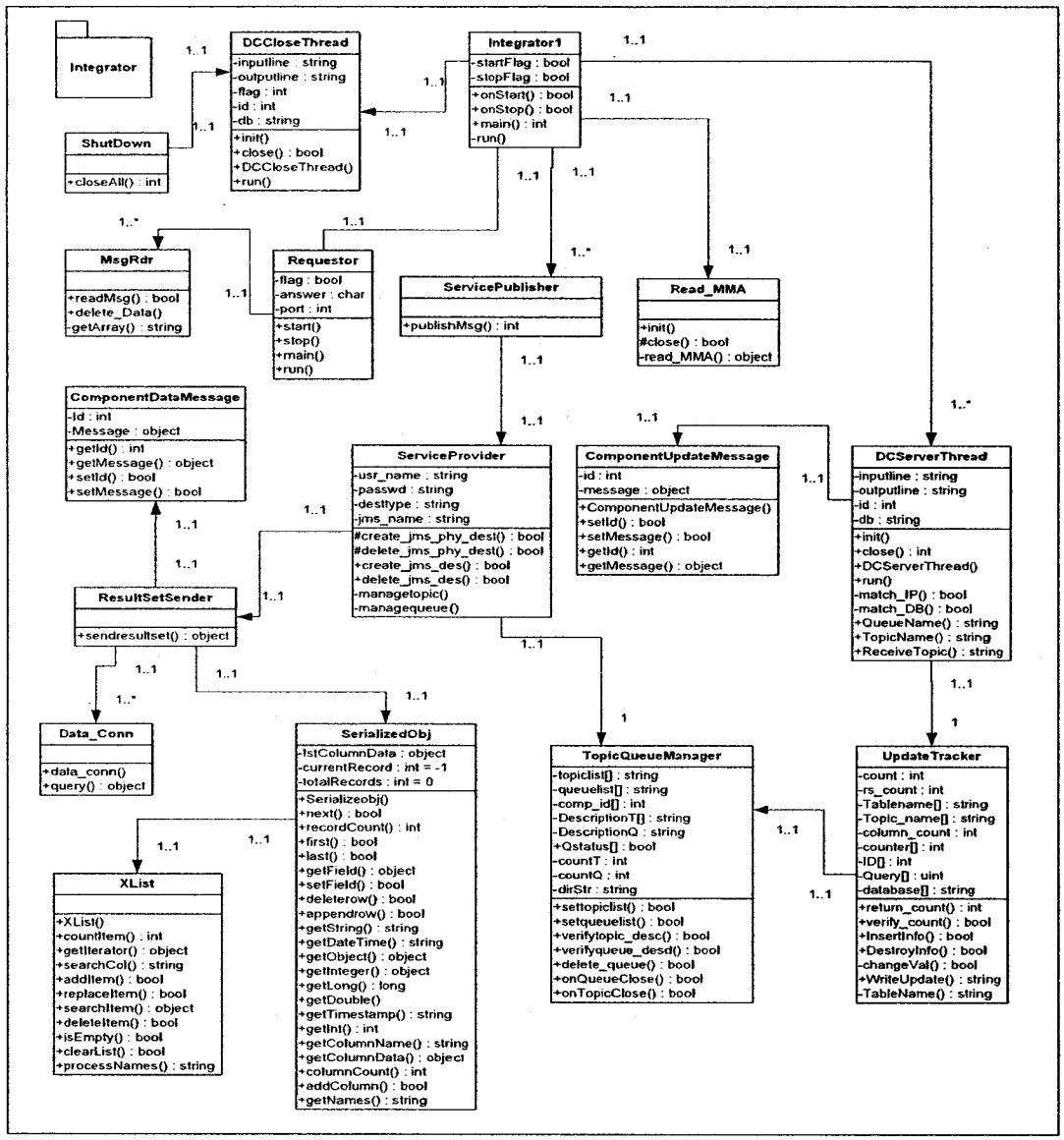


Figure 25: Class Diagram: Integrator Module

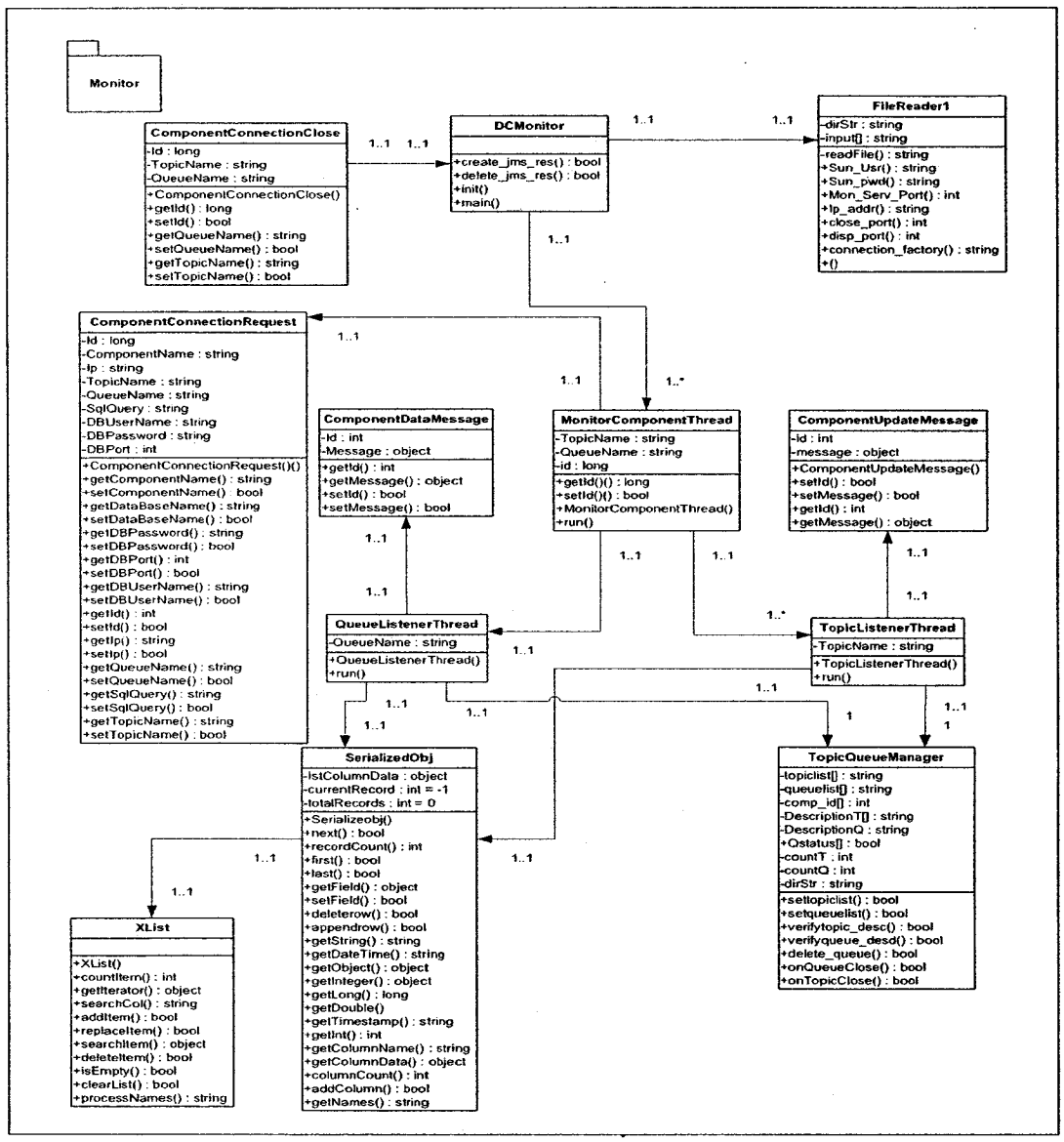


Figure 26: Class Diagram: Monitor Module

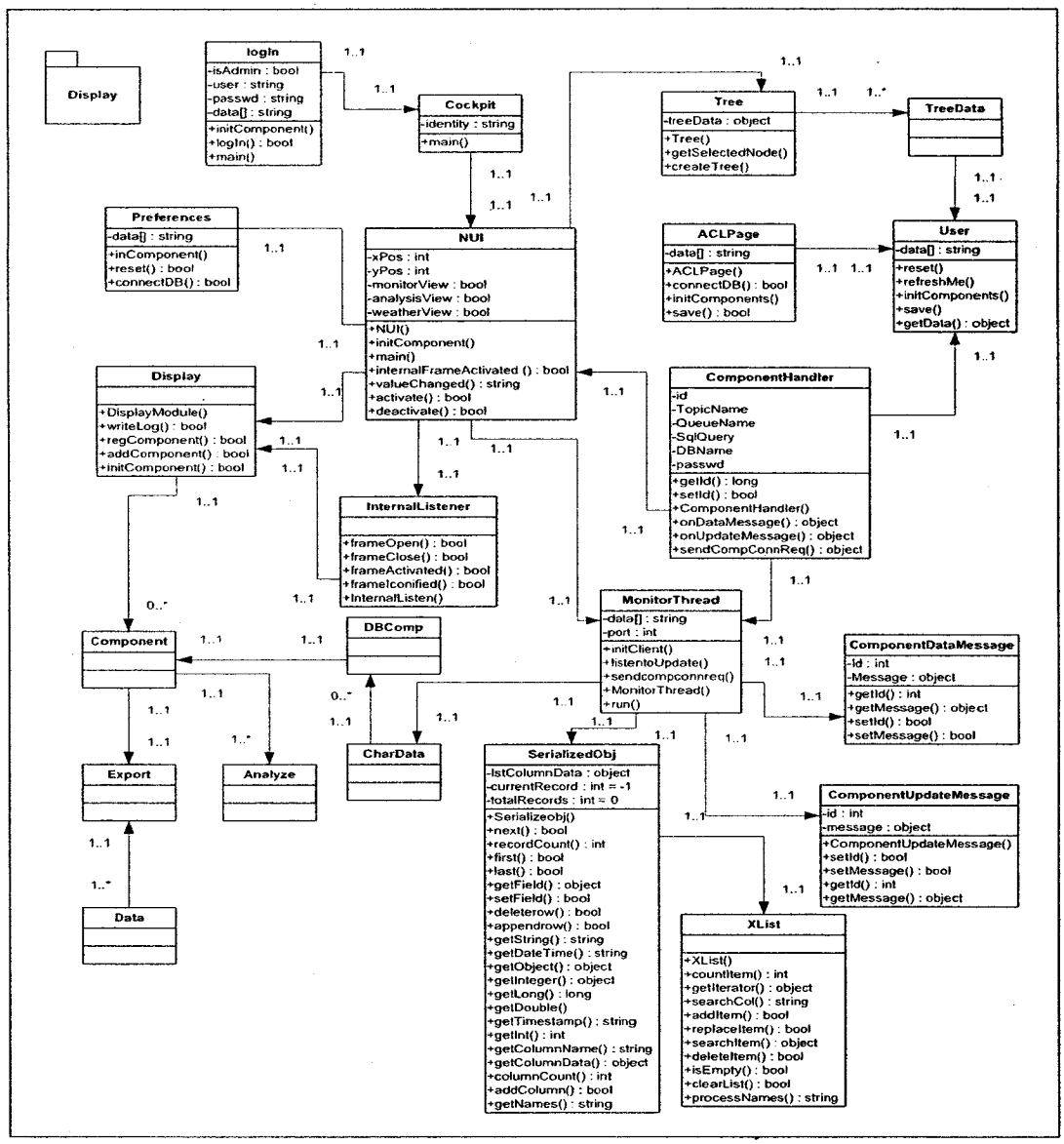


Figure 27: Class Diagram: Display Module

Appendix C

Technology Stack

Modules	Challenges	Technologies
Information sources and services	• Heterogeneity of sources. • Distant location of storage of information. • Absence of an integrated platform.	• Data Integration. – Oracle database. – Sybase database. – Legacy System. • Service Integration. – WSDL description. – XML schema. * Digital Weather Markup Language (DWML).

Table 4: Technology Stack of Digital Cockpit Implementation-I

Modules	Challenges	Technologies
Information sources and services		• Service Integration. – XML schema. * METeorological Aerodrome Report (METAR) data.
Integraion of information sources and services	• Choice of application server. • Distant location of information and service sources. • Establishing an asynchronous paradigm through JMS. • Lack of multi-purpose integrated middleware.	• Application Server. – Sun J2EE. – Apache Gerenimo. • Data integration – J2EE API from Sun Microsystems. – JCA API from Sun Microsystems. – ActiveMQ JMS. • Service Integration. – BPEL4WS implementation: Fivesight PXE – JBI implementation for Integration Container. – ServiceMix implementation from Codehaus.

Table 5: Technology Stack of Digital Cockpit Implementation-II

Display and Monitoring	• Graphical representation. • Real-time view and notification. • Performance.	• Spring lightweight container : Inversion of Control (IoC) paradigm • AXIS and XFire SOAP • Weather METAR / TAF (Legacy Data) Parser. • JFreeChart, Java3D and Espresschart (Quadbase Inc.) • JMS API: Msg. comm.
Modules	**Challenges**	**Technologies**
Analysis and Control	• Understanding Scenarios. • Suitable analysis and optimization algorithms.	• JFREEChart analysis lib. • Jakarta POI. • Game theory based decision support mechanism.
Security and Reliability	• Distributed nature of applications. • Military applications. • Traceability.	• Network and authorization protocols. • Crypto-protocols. • JSSE API.

Table 6: Technology Stack of Digital Cockpit Implementation-III