ALI BENSSAM

Digital Cockpits and Decision Support Systems
Design of Technics and Tools to Extract and Process Data from Heterogeneous Databases

Mémoire présenté
à la Faculté des études supérieures de l'Université Laval
dans le cadre du programme de maîtrise en informatique
pour l'obtention du grade de Maître ès Sciences (M.Sc.)

FACULTÉ DES SCIENCES ET DE GÉNIE
UNIVERSITÉ LAVAL
QUÉBEC

2006

I dedicate this work to my parents, my sisters, brothers, and all my family

À mes parents, mes soeurs, mes frères, et à toute ma famille... Je dédie ce travail

Acknowledgment

I would like to sincerely thank my supervisors Dr. M.Debbabi and Dr N.Tawbi for their constant support, precious help and advice. I would only be restating the obvious when I say that they are great advisors. I have learnt a lot from them, and I consider myself truly fortunate to have them as my advisors and thank them for the opportunity to work under their guidance.

I would like to thank Dr. M.Mejri for his acceptance to evaluate my work and for his valued evaluation and remarks.

In these few lines, I should not forget my parents, sisters, brothers, and all my family for their continuous support.

Lastly, I thank all those who have, directly or indirectly, contributed in the achievement of this work and my success.

Résumé

Ce travail présente une nouvelle approche pour l'intégration des systèmes d'information. Cette approche permet d'intégrer des systèmes hétérogènes en matières de : modèles de données, systèmes d'operations, reseaux utilisés, etc. pour fournir enfin aux décideurs ou/et utilisateurs une information à jour et consistante qui sera la base de décisions correctes et fiables dans le processus decisionnel. Principalement, notre methodologie est une approche multi-couche: une couche intégration des différentes sources de données, une couche fournit un service de méssagerie qui envoie l'information provenant des différentes sources aux différents clients interessés dans cette information, et une autre consiste en une application client. La premiere couche vise à connecter les différentes bases de données en éliminant les différences spécifiques à chacune de ces dernières. La seconde, service de méssagerie, permet d'envoyer l'information aux multiple utilisateurs dans un mode asynhrone, ce qui libère l'application client de rester couplée avec la source de données et en même temps garantit la délivrance de cette information à ses abonnés. Pour la couche "application client", elle est responsable de : l'affichage et la présentation de l'information recue à partir de l'intergiciel basé sur le service de messagerie; de la mise à jour en temps réel de l'affichage en reflétant l'état présent des sources de données; ainsi que d'autres processus de control et d'optimisation.

Notre intergiciel, proposé au sein de cette thèse, est basé sur un ensemble d'APIs standards (surtout celles provenant du monde J2EE) ce qui lui offre une large interoperabilité et facilité d'extension.

Abstract

This work presents a new approach for integrating information systems. This approach allows connecting systems that are heterogenous in terms of: data models, operating systems, used networks, etc. It ultimately provides decision makers or/and users with up-to-date, consistent and well presented information which will be the basis of correct and reliable decisions in the decision making process. Primarily, our integration modus operandi is a multi-layer approach: an integration connection layer, a messaging service layer, and a client application layer.

The first one aims at connecting the various databases by eliminating the differences that are specific to each one. The second, messaging service, allows to push information incoming from the multiple sources to customers interested in this information in an asynchronous mode. It releases the customer application to remain coupled with the data source, and at the same time guarantees the delivery of this information to different subscribers. The third layer, client application, is responsible for: display and presentation of information received from the messaging layer; real-time updating of display to reflect the state of the data sources; and other control and optimization procedures.

Our middleware, proposed within this work, is platform, OS, and DBMS independent. It is based on a set of standard APIs (especially those coming from J2EE world) which offers broad interoperability and provides easy extensibility.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

In the last decade, tremendous advances occurred in computers performance, communication infrastructures, storage technologies, and middleware applications. All these achievements have made from information systems the cornerstone that shapes the present and the future of our economy and society. This omnipresence of computers in all aspects of our life has been fuelled by the explosion of Internet. This explosion with the reliability of communications were -and still- behind the new business initiatives such as e-business and e-commerce.

Today's enterprises and organizations use a large variety of networked computer systems and software applications to collect, process and produce large volumes of data. The ability to turn these islands of data into useful knowledge (information that can be used by decision makers) provides industries with a competitive advantage in their mission-critical situations. Decision makers, top management and leaders need robust, efficient and automatic tools to sense and respond to real-time changes.

However, such an undertaking is not a small task. Enterprise systems are disparate in terms of hardware, platforms, operating systems and software applications. Moreover, these systems extend beyond the firewall to include partners, suppliers, and customers. As a result, extracting critical nuggets of information from these heterogeneous, autonomous, and physically distributed systems presents a challenging problem for today's research communities and businesses.

Therefore, there is a big desideratum to provide software platforms that overcome the barriers revealed above. Such software platforms should be able to:

- Provide a structured, regular and real-time communication of fresh information inside and across the organization's boundaries.
- Keep synchronized and coherent multiple databases.
- Produce status and analysis reports on different activities/processes.
- Scrutinize to the desired level past, ongoing and future activities/processes.
- Handle the needed security services in terms of authentication, secrecy, authorization and integrity.
- Present to a given decision-maker or a principal the information in a graphical and user-friendly way.

As a downstream result, a decision maker will have a realtime big picture that integrates all the needed information to make an educated, solid and sound decision.

1.2 Objectives

The aim of this work is to propose the design of a distributed software platform (middleware and the underlying applications) that is referred to Digital Cockpit paradigm. The intent of the digital cockpit is twofold: At first, it achieves a synergistic integration of the various information systems. Secondly, the digital cockpit will display visual, structured, navigational and realtime big pictures, so that decision makers can drill down into the details and uncover relationships between information that might otherwise remain hidden. Consequently, the decision making process can be enhanced using the available information [7].

More explicitly, we can classify the objectives of our work in the following points:

- Identify the different information and service sources of the organization and the underlying databases, data models, formats and protocols.
- Explore the dependencies and relationships between these information sources.

- Integrate, in a synergistic way, all these sources of information in order to ensure a real-time availability of updated, consolidated, structured and unified data across the network.
- Elaborate a digital cockpit platform that will present the synthesized information through dynamic and real-time visual objects. Moreover, the digital cockpit should offer the possibility to customize the layout and the access privileges according to different user profiles.
- Propose a suite of procedures and tools that extract data from different sources in order to subject them to business intelligence on-line analytical processing (statistical analysis, graphical techniques, simulation of what-if scenarios, trend analysis, comparative analysis, etc.).

By building a digital cockpit that meets the aforementioned objectives, we anticipate that the decision makers will have the capability to access, analyze and visualize data and services in order to take judicious and consolidated decisions.

This work mainly describes the design and the implementation of a digital cockpit system; a prototype that shows the main functionalities of a decision making oriented middleware. Through this work, we will expose the different steps to build such a middleware, from the integration of simple data to visualization of critical nuggets of information as valuable business assets, that help significantly to enhance the decision making process.

1.3 Contributions

The following are the contributions of this work:

- Via our middleware, we are able to integrate a variety of data and services from heterogenous databases that are physically in different locations.
- Our system provides a function that transforms the integrated data into a visual big picture that can help in understanding data and support decision makers to take educated and sound decisions.
- The real-time capture of data changes in distant databases is indeed another functionality that can leverage managers with real-time monitoring of business events, and therefore react instantly accordingly.

As a whole, this work represents a new fashion for real-time information systems integration and decision support systems based on a set of standard technologies, which provide an interesting opportunity for an easier and economical extensibility and scalability to the system.

1.4 Document Structure

This document is composed of five chapters. In Chapter 2, we present Enterprise Application Integration (EAI) and Decision Support Systems (DSS). Different approaches of EAI, DSS and digital cockpit are introduced in this Chapter. In Chapter 3, we highlight the approach we are proposing to build a digital cockpit, as a new paradigm for real time information integration and decision making. In Chapter 4, we detail the design and implementation of the digital cockpit including the architecture of our proposed middleware. Finally, in Chapter 5, we conclude with a summary and provide directions to future work.

Chapter 2

Enterprise Application Integration and Decision Support Systems

2.1 Introduction

With the advent of the web and the increasing number of databases that each business or organization handles, the integration of heterogeneous, autonomous, and geographically distributed data sources has become a major concern of Information Technology (IT) community.

Statistics show that information integration is one of top priorities of Chief Information Officers (CIO) in almost every company [27]. Data integration systems aim to provide a uniform and transparent access to the aforementioned data sources. They differ according to the nature of the problem and the underlying field. Integration approach and used technologies are two main criteria that feature an integration solution. The first one is from a design stand point; it coins to the followed methodology and the content to be integrated. However, the last one is concerned with the applied products, as application programming interfaces (APIs), used to translate the above design to a concrete implementation.

In this chapter, we present first application integration. After, we discuss the main integration approaches and show where each approach fits better. We put more emphasis on information and service integration since they play a key role in our approach. Then, we highlight the messaging service as a communication style, detail existing messaging technologies, and describe Java Message Service (JMS) model, a Java-based standard messaging API. Afterwards, we present Decision Support Systems (DSS), their types and architecture. Finally, we introduce the "Digital Cockpit" concept as a new fashion to combine both of enterprise application integration and decision support realms. This combination of the above two fields allows real-time information systems integration with decision support capabilities.

2.2 Application Integration

Enterprise Application Integration (EAI) pertains to the interconnection of information systems, internal and/or external to the enterprise. This interconnection aims to a better sharing of data and application services. As a downstream result, integration drives to an enhancement of information exchange, and therefore a real-time execution of business processes [45].

The connected systems may be an Enterprise Resource Planning (ERP), a Supply Chain Management (SCM), a Customer Relationship Management (CRM), or any other Enterprise Information System (EIS). Regardless of their heterogeneity, autonomy, and if they are on the same or on different machines; connecting these systems together will leverage users and especially decision makers with an easy and transparent access to all the information the enterprise retains. Figure [illegible] shows a typical integration problem.

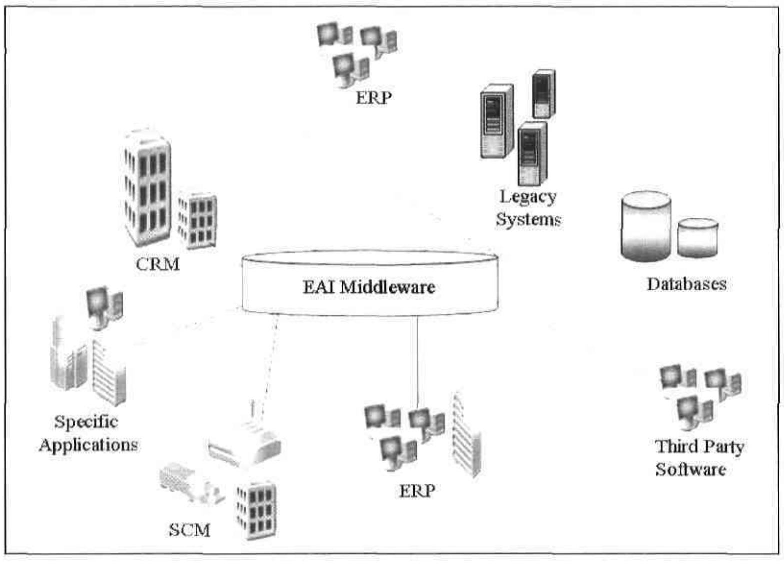


Figure 2.1: Enterprise Application Integration (EAI)

Enterprises needs are distinct in matter of integration. This distinctiveness generally reflects the business needs of each organization. Accordingly, any integration solution is business-requirements driven, and therefore, there is no universal and standard solution that can be applied for enterprises. Thus, each EAI solution pursues a specific approach. However, the same integration solution can combine several approaches, especially in big enterprises having complex and geographically distributed information systems. Among the most prevailing approaches we find: Information Integration, Business Process Integration, Portal Oriented Integration, and Service Integration [45]. In the following sections, we present these integration approaches, the advantages and limitations related to each one.

2.3 Information Integration

Information integration [20, 68] is limited only to data. This approach allows companies to combine data from disparate data sources which are considered as the main points of connection. Integration of data from multiple information sources is one of the longest standing problems facing the IT research community. In addition, being a problem in large corporations and organizations, research on this topic has been fuelled by the explosion of Internet. Therefore, for better information sharing, there is a real need to integrate the various and unlimited islands of data inside organizations and over the Web.

In this context, two main solutions [9, 67, 68] have been proposed for data integration: data warehousing (materialized views) and federated architecture (virtual views). Both approaches take a set of pre-existing decentralized data sources, and develop a single unified (mediated) schema over them. Then, a series of transformations or source mappings are specified to describe the relationship between each data source and the mediated schema.

2.3.1 Materialized Views

Materialized views or warehousing solution requires the building of a data warehouse and writing programs that load data from data sources to the warehouse periodically. This task is achieved by the use of Extract, Transform and Load (ETL) tools. One of the most widely accepted definitions of a data warehouse is that given by Inmon [93]; a pionner in data warehousing technology.

Inmon, in his book [93], provides the following definition: "a warehouse is a subject-oriented, integrated, time variant, and non-volatile collection of data in support of management's decision making process". First, the data warehouse is subject oriented means that it is organized around the high-level entities of the business. In marketing for example, subjects are customers, products, and sales. Second, it is integrated because it integrates data from different sources, and might be inconsistent, so these data must be stored in a consistent format (naming conventions, data constraints, etc.) to provide a unified view to the users. Time variant means data are valid at a point of time or during a time interval. Finally, non-volatile signifies that data do not change in real-time in the warehouse, but refreshed periodically from production's databases. In other words, a data warehouse is a separate database, able to support management decision-making and receiving data from multiple operational data sources. Figure 2.2 shows a simplified data warehouse architecture.

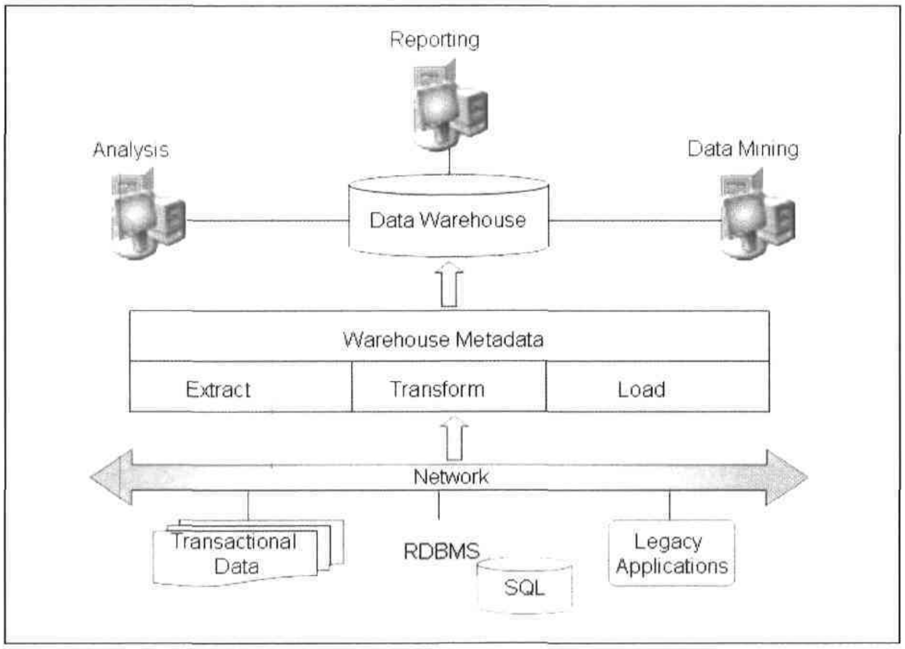


Figure 2.2: Data Warehouse Architecture

Operational Data come from a variety of sources including transactional data from mainframes, relational data from multiple RDBMS databases, and other production's databases. The ETL phase, called also data staging [64] aims at:

- First, pull out data from operational databases and place them into another database.

[1]Relational Database Management Systems

- Second, apply a series of transformations on the extracted data to check their validity and accuracy, and resolve differences in syntax and semantics to conform with the new target database.
- Third, once data are extracted and transformed, it is time to write the new data into the target data warehouse, after carrying out appropriate summarizations and aggregations.
- At the end of this step, data should be detailed (no summarization yet), historical (allows historical values), and normalized (3rd normal form or higher), in such a way that supports decision-making [64].

Afterwards, the data warehouse is used by Online Analytical Processing (OLAP) applications and tools to run complex queries of large multidimensional collections of data, with an intent to assist managers and decision makers. Other kinds of tools may be supported by data warehouse such as data mining and data analysis based tools.

Below, we present the major advantages and limitations of materialized views solution for data integration.

Advantages and Limitations

The data warehouse approach is relevant when data do not change frequently or when the integrated view does not need to be up-to-date. In addition, the warehouse is built specifically to enhance decision making, by crating a new database being accessed by different kinds of data analysis tools like OLAP and data mining tools. Hence, the warehouse represents the well appropriate approach for analysis of historical data, extrapolation, and supporting strategic decisions. Also, the separation of the warehouse from operational data gives a good performance for the analysis tools.

However, building, entertaining, and refreshing the warehouse -with ETL tools- is costly and time consuming [?]. The differed updating of the warehouse, also does not play in favor of this strong technology in a world attempting to reach real-time business.

2.3.2 Federated Architecture Solution

Alternatively, in the federated architecture approach, also known as mediation approach for data integration [43] (the mediator used as a connector between multiple databases in the federated approach for data integration), we define one or more mediated schemas, which are not used for storing data but only for querying it. When a query is issued to the system, it is translated into a set of sub-queries, over the data sources, having the same semantic as these target data sources. This approach keeps the local autonomy of data sources and creates a virtual repository enabling real-time on-demand data access. Moreover, it addresses the case when data change frequently, or when the global schema itself may change usually. Figure shows a simplified federated architecture for data integration.

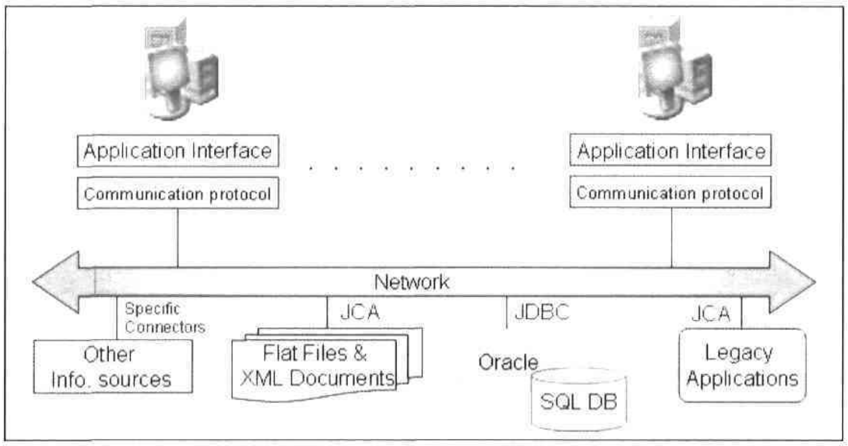


Figure 2.3: Federated Architecture for Data Integration

A surge of interest has been expressed in data integration by mediation approach, and many research projects have implemented this concept. Among the pioneer projects in this area we refer to TSIMMIS [74], Garlic [86], and DataJoiner [90].

First, TSIMMIS [35, 74] assumes a mediation approach to integrate both heterogeneous, structured and unstructured information from multiple data sources. It uses wrappers to convert information into a common object model. The intent of TSIMMIS is to provide decision makers with a tool that can get and fuse information from multiple sources on need and keeping data their consistency. TSIMMIS puts a translator on top of each information source. This wrapper converts the underlying data objects into a common object model. When receiving a query, the wrapper translates it into the common model that the source can understand and execute. Similarly, it converts the returned result into the common model. Figure is a simplified representation of TSIMMIS architecture for data integration.

[2]The Stanford-IBM Manager of Multiple Information Sources: a joint project between Stanford University and IBM Almaden Research Center

[3]A project developed by members of the database group in Computer Science, IBM Research Center to enable large-scale multimedia information integration

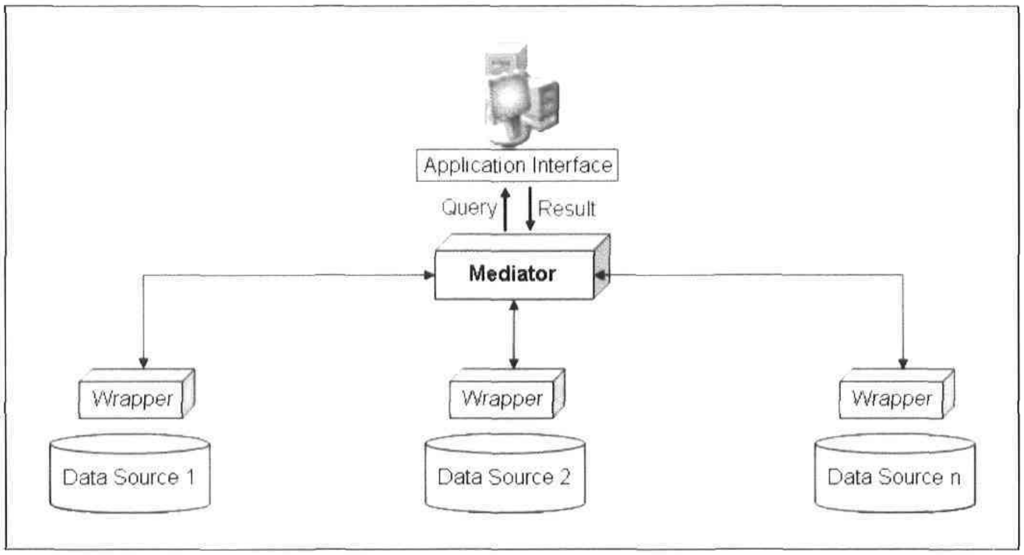


Figure 2.4: TSMMIS Architecture

Second, Garlic solution for data integration also relies on mediation approach and is developed in Garlic [86] project. Garlic aims to develop a multimedia information system that integrates data from multiple information sources. These data include different types such as text, images, audio, video and other types. Garlic provides an object-oriented schema to applications, interprets queries, defines execution plans and returns queries results to the applications.

Finally, at the same time, another project called DataJoiner was being developed by IBM research team. Its main objective was to develop robust and efficient queries over a set of relational data sources [90].

As cited previously, while DataJoiner was meant to integrate relational databases, Garlic's main goal was how to extend such a solution for a large set of heterogeneous information sources. At the end, both of these projects played a key role in defining functions to federate data sources with IBM DB2/DBMS [43].

For commercial tools in data federation, we can refer to DiscoveryLink from IBM[42] and Attunity Federate from Attunity [5], etc. DiscoveryLink allows visualization of multiple distributed data sources to provide a single virtual schema for use by the biologists. As for Attunity Federate, it is a federated solution that provides real-time access to heterogeneous data, both for querying and updating purposes. It also captures changes in data sources, so integration can be accomplished in near real-time (Attunity Stream).

To recapitulate, information integration through federated architecture is getting widely used especially with the increasing number of heterogeneous data sources and the need for real-time solutions for data integration. Hereafter, we present the advantages and limitations of this solution.

Advantages of Federated Architecture

The federated architecture for data integration possesses the following features:

- *Local autonomy of the integrated sources*: Allows to avoid problems related to management of these sources since their is no change in data sources management and administration.
- *Real-time integration*: Creates a virtual layer on top of data sources, which enables real-time and on demand access to up-to-date data. This feature is well suitable for today's business needs.
- *Transparency*: Provides the user with the query results in a transparent way. It masks all the data sources differences and complexities from the user.
- *Extensibility*: Allows easily adding of new data sources in a dynamic way to meet new business requirements.

Selecting Data Integration Approach

As stated above, the information integration is a complex problem for all IT community, and there is no standard and universal solution to this problem. Depending on the scope and the complexity of the integration problem (from simple connection of two local databases to complex integration of multiple heterogenous information sources geographically in different locations), there are many issues to consider and assess when selecting an integration approach [77].

- *Degree of data update or change*: What is the frequency of updating data, and does the data source updated by a single or multiple applications?
- *Latency*: How does data integration occur? Is it done periodically using batch operations such as in data warehouse, or in real-time as in federated architecture.

- *Degree of cleansing and transformation*: Are data clean and ready to be directly useful without or with minimum effort of cleansing and transformation?
- *Real-time or Strategic decisions*: Are data sources used to react in real-time to business events or to prepare strategic business decisions with an extensive use of historical data.
- *Budget and resource constraints*: Does the accorded budget for the integration problem abide with the creation of new physical databases or just for putting a wrapper over the existing data sources.
- *Time-to-market requirements*: The needs for data integration solutions differ according to time-to-market constraints. For example, building a data warehouse solution needs planning and building, contrary to federated architecture.
- *Transactions management*: Is transactions management important for the integration problem? If so and the data source can participate in transactions, the federated architecture may be considered.

In a nutshell, here are the most important parameters that influence the selection of a data integration solution: the number of data sources, their heterogeneity, the volume of data to integrate, the business requirements such as real-time or differed, the needs for a high level of transactions management, and the financial constraints.

In Table , we provide a summary of the main features and differences between materialized views and federated architecture solutions for data integration [77].

Information Integration Approach	**Federated Architecture**	**Data Warehouse**
Real-Time events notification	√	-
Frequent changes in data sources	√	-
Strategic decision support	-	√
Easy extensibility	√	-
Easy access	-	√
Short time-to-market	√	-
Economical solution	√	-

Table 2.1: Federated Architecture Vs Data Warehousing Approach

After this review of the state of the art in data integration, we present briefly below, the most important APIs provided by Java world.

Java Technology for Data Connection

Java products oriented data connection become defacto standard for almost all DBMS providers and large community of developers. Thus, Java Database Connectivity (JDBC) and J2EE Connector Architecture (JCA) are Java standard for connecting multiple information sources.

Java Database Connectivity (JDBC)

JDBC is an API provided by Java Sun within the Java Community Process (JCP) to leverage a standard API enabling connection of all relational databases and spreadsheets such as Ms Excel. Majority of application servers provide support for JDBC [81].

J2EE Connector Architecture (JCA)

The result of a fruitful collaboration of most active software vendors (Sun, Oracle, IBM, BEA, etc.) through JCP, the specification of JCA (Java Specification Requests (JSR) 16 and 112) provides a standard Java technology solution to the problem of connectivity between the many application servers and today's enterprise information systems. It enables vendors to create standardized connectors to EIS.

to sum up, federated approach is the most appropriate approach to achieve the integration of many data sources. Supporting real-time business events notification plays in favor of this approach in case of big numbers of data sources, and distributed computing environments where information sources are getting on and out. Therefore, in the rest of this document, we adopt the federated solution for data integration to connect the different databases related to our digital cockpit project.

2.4 Business Processes Integration

In the previous section, we perceived that application integration oriented data is only meant for data sharing purposes. This certainly enhances the information exchange between enterprises and businesses. However, this approach does not deal with business processes. This means that the definition of information flow has to be taken care by another approach: Business Process Integration (BPI) [45]. Generally, the integrated systems are autonomous; thus they have their own process choreography engines and therefore, run internal business processes private to them [40].

The business process integration defines a general model for business processes that addresses the sequence, the hierarchy, the events, logical execution and information transfer between systems within the same organization (EAI), and between systems from different organizations (B2B) [34]. The main idea behind business process integration is to provide a single logical model that spans the multiple applications and data sources. Therefore, a single business process controls the interaction between humans and systems to satisfy business requirements [45].

The business process integration provides mechanisms that define and run the information flow over multiple systems [45]. It puts a logic control layer on the top of different integrated technologies. This enables to connect local systems into a single process that leverages the business operations and objectives. This integration approach should run this process in the correct order, with the appropriate information, with a control of sequences, keeping the state, durability, and possibility to handle exceptions [45].

2.4.1 Tasks Executed by an Internal Process

In general, an internal process executes many tasks to achieve the result expected by the general process. First, the local system sends an event to a business process engine. Second, this engine transforms the event to be conform to certain semantic standards and mechanisms of information processing (synchronous or asynchronous). Afterwards, it reacts to the transformed event by invoking other processes from other systems in favor of the execution of a model, through the invocation of B2B processes. Then, the new local system (target) reacts to its internal processes and sends the answer to the business process engine. Finally, the general model of the process controls the master process (on top of all other processes) in order to enhance the B2B communication.

2.4.2 Advantages of Business Process Integration

By defining a new layer above the source and target systems, the business process integration introduces many advantages:

- An instance of the new general process spans many instances of internal processes specific to local systems. This feature provides more visibility about the entire

[4]Business To Business

activity.

- The independence of business process integration from source and target systems allows modifying processes without affecting the aforementioned systems.
- The strategic approach of business process integration defines the business rules that determine the interactions between systems, in a common abstract business model.
- The definition of a common abstract business model provides the capability for real-time analysis of all aspects of business, and allows determining the state of the process at any time.
- The ability to redefine the process any moment to enhance the efficiency, hide the complexities of local applications, and allow users working with the same business semantic.

2.5 Portal Oriented Integration

The other approach in the application integration stack is Portal Oriented Integration.

2.5.1 Portals Role

Portals allow viewing a multitude of systems -both internal and/or external enterprise systems- through a single user interface or application. They extend the user interface of each system to a common aggregated user interface. Therefore, they connect multiple systems although they do not directly integrate the applications within or between the enterprises. Via portals, the user interacts with the back end systems through a user interface -such as a web browser-, rather than having the systems automatically exchange the information (as in data oriented integration). The integration using portals approach can be achieved along many technologies: application servers, page servers (ASP, PHP, JSP,etc.), and screens conversion technologies into HTML, etc. Today, more B2B information flow through the user interfaces (portals) than automatically through back end integration. The figure [illegible] shows how portals allow retrieving information from multiple applications in a unified user interface.

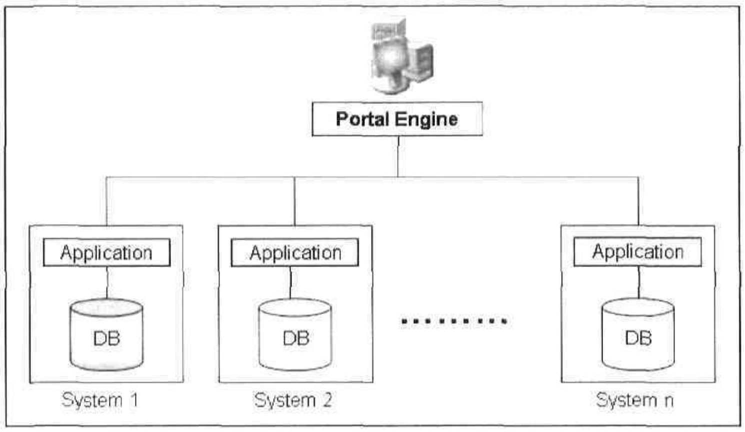


Figure 2.5: Portal Integration

2.5.2 Advantages and Limitations of Portal Oriented Integration

Portals allow users to interact with a company's internal system through a user interface (generally a web browser). They are typically much easier to build than sophisticated and real-time integration, the case of data and process integration. In addition, portal integration approach can be easily added to existing systems without disturbing the existing functionalities [44]. Moreover, adding a portal to existing applications can be achieved in a short time, whereas a full integration with other approaches is much more time consuming. Finally, portals enable human interaction, which is very useful in some situations where business rules are not well understood or not agreed upon.

However, the portal approach presents some limitations comparing to the other approaches. First, the information does not flow in real-time, thus the user has to interact with the user interface or a web browser to get the information he needs. As a result, systems do not react to business events within an enterprise. This deprives the approach to be event-driven, a key feature for automated business processes. Again, from a user stand point, the portal is one application at a time unlike other approaches enabling real-time integration [75], which represents a real handicap where the trend is towards real-time integration which enhances the real-time business.

We have perceived in the previous sections how information integration enhances the information exchange between systems inside and/or outside organizations. We have also seen that business process integration visualizes application integration as a high level of abstraction through the definition of a general process model. As for portal integration, it allows exposing the enterprize systems to external users via a single user interface. Hereafter, we present the last approach for application integration: Service Oriented Architecture.

2.6 Service Integration

Service Oriented Architecture (SOA) is not a new concept for the integration of services in distributed environments. Actually, it was achieved with many technologies such as Electronic Data Interchange (EDI), frameworks and distributed objects [45].

Initially, technologies like EDI have been used for many years to successfully perform business transactions between partners [22, 25]. EDI relies on pre-agreed formatted messages and proprietary network protocols for data transport. As a downstream result, companies have been reluctant to invest in these technologies, because of the large underlying investments that are required in terms of software, hardware and consultancy [22].

Afterwards, with the tremendous advances occurred in computers performance, communications infrastructure, and middleware applications, in the early 90s, more and more software systems have been built and lot of similar situations and patterns appeared. Therefore, there was a need to reuse functionalities of existing systems rather than build them from scratch, and consequently communication between them was required and considered as services achieved by a system to another one. This communication between software systems through achieving services marked the first days of Service Oriented Architecture (SOA).

In this field, many initiatives have been introduced to facilitate communication between application components in a distributed computing environment. Common Object Request Broker Architecture (CORBA) [82] from Object Management Group (OMG), Distributed Component Object Model (DCOM) [11] from Microsoft and Remote Method Invocation (RMI) from IBM and SUN Microsystems are examples of these initiatives. All these technologies allowed organizations to integrate applications in a distributed infrastructure using RPC-based mechanisms to bind application clients to a server [22, 25]. However, interoperability between these RPC-based mechanisms is still complex and limited. For instance, CORBA and DCOM cannot communicate easily and may need a bridge to allow communication. This limitation is due to the fact that each infrastructure uses its own communication protocol. CORBA uses Internet Inter-ORB Protocol (IIOP), DCOM uses Object Remote Procedure Call (ORPC) and RMI relies on Java Remote Method Protocol (JRMP). In addition, within these technologies, the application is statically bound to a single address and tightly coupled with request/response mode.

However, for the multiple limitations we discussed above, IT and business communities found out that providing a high level of interoperability between heterogenous systems will be the unique way to support web explosion and the new business initiatives. Such a solution should be independent of the underlying platform, language, data models and used protocols. The answer for this concern was the introduction of a new implementation of SOA: Web Services.

2.6.1 Web Services

The main idea behind the Web Services is how to provide a mechanism that allows enterprises and businesses to describ their services, publish them in centralized registries and/or repositories so that users can find and use them. This mechanism should achieve all these operations in a transparent way by masking all the complexities from the user. Among the first Web Services products, we refer to e-Speak [56], introduced by HP in 1999. Shortly afterwards, many competing frameworks and proposals for Web Services have been provided such as Microsoft .Net, IBM websphere and SUNs J2EE Web Services. They all share the basic definition and vision of Web Services.

A Web Service is a self-describing, self-contained, and modular unit of application logic, whose interface may be described in a standard machine-processable format (specifically Web Services Description Language (WSDL)). Inter-process communication with a Web Service is achieved using standard web protocols, notations and naming conventions, including XML Protocol (or until XML protocol is standardized, SOAP) [61].

In what follows, we first present briefly XML as a common standard used by Web Services components. Second, the Web Services model, and finally the Web Services stack including different layers: service description, service publication, service discovery, and service execution.

2.6.2 XML: A Big Step for an Unlimited Interoperability

In [92], eXtended Markup Language shortly **XML** is defined as: "a specification developed by the W3C. XML is a pared-down version of Standard Generalized Markup

Language (SGML), designed especially for Web documents. It allows designers to create their own customized tags, enabling the definition, transmission, validation, and interpretation of data between applications and between organizations".

Thus, XML [91] provides a standard format for data exchange, which helps to easily integrate structured, semi-structured and unstructured data through intranet and the web. In addition, the self-descriptive nature of XML provides easy integrity to structured, semi-structured and unstructured data outweighing the limitations of HTML. Legal building blocks of an XML document may be defined in two ways: XML DTD (Document Type Definition) or 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 in terms of constraints on the structure and content of documents of that type. Finally, the document elements can be written as to be compliant with those defined classes. Furthermore, it may inherit and import elements from existing element classes and schemas using namespaces.

2.6.3 Web Services Model

Web Services model takes advantage of these enormous capabilities offered by XML in a layer based architecture as described in figure 2.6 below. Three main roles [27, 36, 56] are defined: service provider, service registry and service requestor.

In a typical scenario, the service provider describes the Web Service and publishes it in a service registry to be used by a service requestor. Using a find operation, the service requestor retrieves the service description from the service registry and uses the service description to bind with the service provider and use the Web Service.

From a business stand point, a service provider is the owner of the service. However, from an architectural stand point, it coins to the platform that hosts this service. As for the service requestor, it points to the business user that requires certain functions to be satisfied. Architecturally, it represents the application looking for and invoking or initiating an interaction with a service. Whereas the service registry represents a searchable registry or repository of services descriptions published by service providers. Requestors of services find and obtain binding information for static or dynamic binding. For statically binding, the service registry is optional since the service provider sends directly the service description to the requestor. However, for dynamic binding, the service description is obtained from a service registry by the requestor which is bound

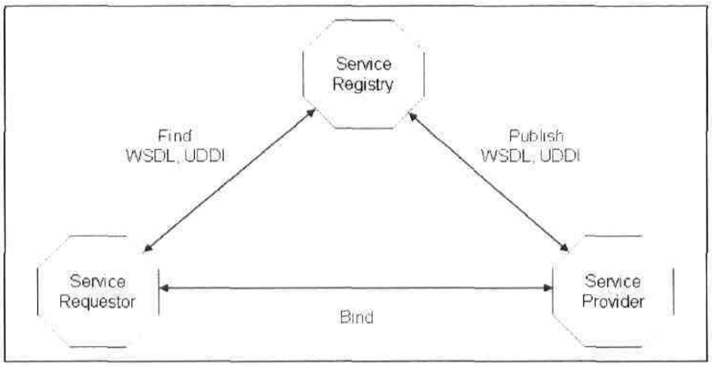


Figure 2.6: Web Services Roles

to the Web Service at run time.

2.6.4 Web Services Stack

To achieve the interaction between the three roles shown in figure 2.6, in an interoperable manner, many organisms and consortiums including the principle software and hardware vendors define a Web Services stack built on well established standards, particularly: Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Universal Description, Discovery, and Integration (UDDI) [59]. As shown in Figure 2.7, this stack consists mainly of the following layers: service protocol, messaging service, service description, service publication, service discovery, and service composition.

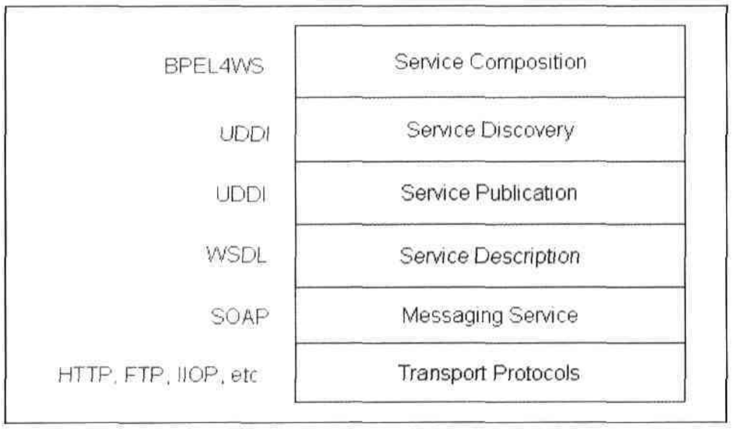


Figure 2.7: Web Services Stack

- *Service Protocol:* Service Protocol defines a common standard application level protocol to transfer information as a load over communication protocol. Because of its incontrovertible popularity and ubiquity, HTTP is the de facto standard transport protocol for Web Services. HTTP does not exclude the other Internet protocols, therefore many others can be supported, including FTP and SMTP. For Intranet (inside the organization), communications can use reliable messaging and call infrastructures [36] like Java Messaging Service (JMS), CORBA, etc.

- *Messaging Service:* The next layer, XML-based messaging based on Simple Object Access Protocol (SOAP), a standard defined by W3C. SOAP acts as the envelope for XML-based messages. SOAP provides standard mechanisms for enveloping, communicating document-centric messages, and remote procedure calls using XML [36]. Moreover, SOAP messages support different operations to describe, discover and use the services (publish, find and bind).

- *Service Description and Publication:* Service Description defines the interface of an implemented service. It signifies common business transactions (e.g. sending a purchase order), common data-interchange formats and mechanisms to negotiate business terms among organizations before commencing transactions. Since the service is defined in a an XML-standard. Any global consumer can automatically interpret a service and use it through defined interface to suffice his/her own interests. Web Service Description Language (WSDL) is the de-facto standard for service description. A WSDL document simply describes what the service can do, where it resides and how to invoke it through the binding with SOAP1.1, HTTP GET/POST, etc. Once the service provider describes the service, it is time to publish it in a service registry or repository. The most dominant standards for service publication are UDDI [59] and eXtensible Markup Language (ebXML) [89]
.

- *Service Discovery:* As stated before, Service Discovery is implemented in a mechanism that lists organization's capabilities for business transactions and provides a look up facility to company profile. Service Discovery uses its own XML repository system (ebXML) or other repository (UDDI). Once agreed, service discovery system binds the producer and consumers. A standard based open specification for Service Description and Discovery is achieved in Universal Description, Discovery and Integration (UDDI). However, Java based implementation of UDDI uses JAXR (from Sun Microsystems) interface to unify and speed up transactions from different repository systems.

The aforementioned description of Web Services is stateless as implemented in WSDL. Therefore, it offers execution of a whole process in response to customer's request. However, human requests are often too complicated to be served by a single service. It requires composition of multiple services to accomplish the required work flow. As WSDL being stateless and unable to capture intermediate states; a language, on top of WSDL, that deals with work flow becomes a necessity. This necessity introduces another Web Service layer, called **Service Execution**. Software vendors implemented a number of process execution languages. The most promising one may be considered is Business Process Execution Language for Web Services (BPEL4WS) provided by many industry leaders such as BEA Systems, IBM, Microsoft, SAP AG, Siebel Systems, etc.

2.7 Messaging Service

For information sharing purposes, any EAI solution requires a communication layer [71] between different connected systems called middleware. This layer enables transparent interaction between applications by masking the complexities related to each one.

2.7.1 Types of Middleware

A middleware comes under various forms: Transactional Processing Monitors (TP), Remote Procedure Call (RPC), and Messaging Oriented Middleware (MOM).

First, Transactional Monitors technology [8, 17] emerged in the beginning of the 1980's to balance the processing load problems on the mainframes, by splitting complex applications into small pieces of code called transactions. This technology performs business logic and manages database transactions. It is used in data management, network access, security systems and delivery order processing . Second, RPC-based middlewares [80] allow access to remote servers by using special function calls embedded within the client side of the client/server application program. When a client program is compiled, a local stub for the client side and another stub for the server side are created. When the application requires a remote function and typically support synchronous calls between clients and servers, these stubs are invoked. Tightly coupled, remote procedure call (RPC) puts serious handicap in system-to-system processing due to distinguished frameworks. Moreover, architecturally, CORBA, Microsoft's DCOM, and Java RMI maintain many-to-many connection framework based on their own synchronous communication protocol. A caller is blocked until the procedure completes remotely and returns control to the caller. Consequently a distributed middleware follows highly interdependent nature, where one failure on a system has immediate and deliberating impact on other systems. Finally, MOM, which we will present in detail hereafter, represents a better alternative for TP and RPC-based middleware.

2.7.2 Messaging Oriented Middleware

A Messaging Oriented Middleware [69] is a software that resides in both sides of a client/server architecture and typically supports asynchronous communications between the client and server tiers. Hereafter, we highlight this communication middleware, its architecture, messaging models, styles, and follow up by detailing the standard Java messaging service, one of the most widely accepted and used messaging services.

For an enterprise application, a message is a lightweight entity that consists of a header and a body. The header contains fields used for message routing and identification. However, the body contains the application data being sent.

A Message Oriented Middleware exchanges information in a message format, generally created by an API, places it as a payload (application data) on the network protocols, and assigns the routing information. In addition, the messaging service decouples the user application from data or service sources and therefore allows seamless integration of resources inside and outside the organization with significant improvement in quality of service .

2.7.3 Classification of Messaging Service

Message Oriented Middlewares exist in several industrial products. IBM MQSeries (become IBM WebSphere MQ) [88], Microsoft MSMQ [49], TIBCO Rendezvous [85], Modulus InterAgent [57], etc. are examples of these products. These MOM products can be classified according to their messaging architecture, model, and style.

Messaging Architecture

Each enterprise system follows an architecture which is centralized, decentralized or hybrid [69]. Implementing a messaging system for information exchange is not an exception.

- *Centralized architecture:* Centralized messaging systems rely on a single message server that routes the message and contains the request response broker that holds implementation details of its messaging clients. Clients are added to the server and post messages to their server. Therefore, any client can be added or deleted without impacting the whole system. A centralized server may be a distributed cluster working as a single logical unit.

- *Decentralized architecture:* Decentralized architecture does not rely on a single server. Each client is given some server-like facilities (persistence, transactions, security) and the router implements IP multi-casting functionalities to allow a message requests to other messaging clients.

- *Hybrid architecture:* Messaging vendors generally implement both of the above-mentioned architectures. Centralized multicast is generally implemented on TCP/IP communication protocol, while the decentralized counterpart is on IP multicasting protocol.

Existing messaging products follow the aforementioned architectures as per their requirements. Messaging service of SonicMQ [79] allows centrally managed messaging components. TIBCO Rendezvous utilizes a distributed architecture to eliminate bottlenecks and single points of failure. MSMQ [87], a part of the Microsoft DNA architecture, has two components: MQ site controller (SC), that store read-only copies of messages in a queue, located in client-side end; and a primary enterprise controller (PEC) that resides in the central MQ server side, that keeps the broker and stores the queue details.

Messaging Model

A MOM follows one of two modes: Point-to-Point (P2P) model and Publish-and-Subscribe (Publish/Subscribe) model [79].

- *Point-to-Point model:* P2P messaging model uses a connection component that allows one-to-one delivery of messages. More precisely, a message producer creates a message and put it in a 'queue' inside the 'messaging domain'. Afterwards, a consumer can receive that message from the same queue. Point-to-point messaging confirms loose coupling between single producer and single consumer at a time.

- *Publish-and-Subscribe model:* Publish-and-subscribe is a one-to-many broadcasting model of messaging. Message based middleware systems use this technique to allow an application sending the same request to multiple other applications. Each consumer receives a copy of the same message in nearly same time. The virtual repository channel of the messaging domain is called 'topic'.

Messaging model defines handling of messages sent to or from a particular site. Auction sites, stock quote service, or security services often require to push data to huge population of consumers. Different vendors approach different styles. For example, MSMQ from Microsoft supports queue based one to one messaging, while the present versions of SonicMQ and TIBCO Rendezvous work with both aforementioned models.

Messaging Style

An important feature of messaging service is its ability to decouple two applications to share information or processes. The basic infrastructure of RPC [76] is strictly synchronous . However, messaging [69] allows both synchronous and asynchronous communication as required in the enterprise systems.

- *Synchronous messaging:* Synchronous messaging tightly couples messaging component (called destination) and receiver of messages. It provides an advantage for fail-safe communication and transaction processing. However, the receiver is bound to destination until it finishes the processing of the sender's request. Network and both communicating processes must be available during the communication. A synchronous consumer [14] uses a pull technique to receive messages from a destination .

- *Asynchronous messaging:* Asynchronous Messaging style uses 'store and forward' mechanism. Design of this messaging allows one/many connection component(s) between the sender and receiver. An asynchronous consumer registers a message listener to a destination. These components allow the sender and receiver non-blocking to each other. When a message arrives, the implemented listener model informs the consumer with a message. Consequently, the client frees the messaging structure and failure of a client does not affect the whole system.

Most of the business solutions introduce the asynchronous paradigm in their messaging solution. However, certain situations require synchronous solutions, while the information is required to be compromised only between the sender and receiver for transaction processing, for instance. TIBCO Rendezvous provides support for request/reply, publish/subscribe, and synchronous/asynchronous mesaging. Similarly, IBM Websphere MQ series, Sonic messaging products implements both notions in their messaging styles with limited variations.

2.7.4 Java Message Service

According to our objective, the proposed middleware system requires both; peer-to-peer communication and broadcasting of messages for enterprise-wise integration of information systems. After an intensive research on different technologies and assessing the available messaging products, Java Message Service (JMS) has been chosen as the most prominent technology in our design and implementation. JMS is the Java messaging standard, released in 1999 [51, 52] and used by most of the leading industries including, Sun Microsystems, Sonic MQ, IBM Websphere, BEA Weblogic, etc. It leverages a union of all the existing features of messaging service and includes functionalities of sophisticated enterprise applications in portable messaging applications. The main objectives behind JMS are the following:

- Provide a single unified message model that can be implemented as an API (JSR 914).
- Allow cross vendor communication in the messaging service level with applications that do not use JMS but support similar specifications.
- Integrate multiple heterogeneous sources of information and services, with XML support, through communication of Java objects.
- Implement another layer between application and communication layers and thereby enforcing magnitude of enterprise properties like security, portability, reliability etc.
- Implement a new notion of asynchronism, much needed for integration of both intra and/or inter organizations.
- Extend support to numerous Java based APIs: Java DataBase Connectivity (JDBC), Java Transaction API (JTA), and Enterprise Java Beans (EJB) components.

Hereafter, we present various JMS components associated to the JMS messaging.

JMS API Architecture

A JMS application is composed of the following parts [33]: a JMS Provider, Clients, Messages, and Administered objects.

- *Provider:* Each JMS application runs on a host application, called JMS provider. A JMS provider implements the JMS interfaces and is responsible for administrative functions and control capabilities. Inside Java virtual machine, a distributed component resides within a 'provider' application. This container hosts 'broker' and other JMS components that hold the messaging implementation details [41, 79].
- *Clients:* Clients are JMS applications that use the provider components to produce and consume messages. A client can be synchronous or asynchronous to the JMS destinations to send or receive messages. JMS is implemented in a way to support JMS and non JMS clients through Java application or other client APIs [69].
- *Messages:* A JMS message carries application data and event notifications in flexible and dynamic way. Unlike RPC, messages neither dictate the recipient nor block the sender. A `Message` object consists of a message header and the message itself, called payload. Most of the message headers are automatically assigned and determine `JMSDestination`, `JMSDeliveryMode`, `JMSMessageID`, `JMSTimestamp`, `JMSPriority`, etc. A `Message` object contains a built-in facility for supporting application-defined property values. Actually, this provides a mechanism to add application-specific header fields to a message. Properties allow an application, via 'message selectors', having a JMS provider select, or filter, messages on its behalf using application-specific criteria. A message selector mechanism is used by the consumer to filter out its specific messages. JMS message payload has the base interface defined in `javax.jms.Message`. Payload might be structured like `StreamMessage` or fairly unstructured as in `TextMessage` [51].
- *Administered objects:* Administered objects are pre-configured JMS Objects created by an administrator to control the message-based communication. The administrator binds these objects inside the Java Naming and Directory Interface (JNDI) namespace prior to the messaging. Administered objects are of two types: `ConnectionFactory` and `Destination`. Once, created they provide abstraction to the namespace by hiding implementation complexities through this virtual and dynamic interface. Therefore, a client program executes with minimal programming required to 'lookup' a specific object. Administered objects always imple-

ment standard based profile, and therefore remain portable despite proprietary aspects of JMS providers [24, 69].

Apart from the aforementioned components, JMS uses a directory service for the storage of implementation specific settings of distributed JNDI components. Directory service is a file system that allows clients to look up a named connection factories or destinations defined by administrators. Generally, it uses Lightweight Directory Access Protocol (LDAP) [21] server to store the administered objects.

JMS Model

The basic building blocks of a JMS application consist are: Administered objects, Connections, Sessions, Message producers, Message consumers, and Messages (Figure 2.8).

In general, a directory service has its own domain managed by a domain manager. The domain manager [79] controls administered objects within multiple virtual containers and allows communication between them. JMS messaging requires proper configuration of administered objects in JNDI prior to messaging. First, administered objects are required to be bound to JNDI using administrative tools. Once `ConnectionFactory` and `Destination` are created in the JMS server and all of its implementation specific information are kept within the system, and JMS is ready to start communication. A developer should create a `Connection` initially from a valid and bound JMS ConnectionFactory. As it allows multi-threading, a ConnectionFactory may handle multiple connections simultaneously. JMS messaging is session based. A `Session` object implements a particular valid `Connection`. A `Connection` is capable also to handle multiple sessions at a time. However, there are strict restrictions imposed on concurrent access into Sessions. The objective of a session is to implement transactions in synchronous or asynchronous messaging. Therefore, JMS specification reserves the access to a session by a single message consumer. However, sophisticated design may allow multi-session related to a single connection to process concurrent requests from users. Each application process runs under a particular session. JMS user applications are of two types: `QueueSender/QueueReceiver` and `TopicPublisher/TopicSubscriber`. `QueueSender/QueueReceiver` application sends and receives messages to/from a Queue, while `TopicPublisher/TopicSubscriber` publishes and subscribes information to/from a Topic. JMS send/receive (in JMS Queue) and publish/subscribe (in JMS Topic) [51, 69] are detailed in the next subsection . The piece of code illustrates a simple JMS sender application as described above.

```
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) {} }
                      }
```

JMS messaging implements one-to-one and one-to-many messaging models in the same API. As mentioned earlier, JMS destination objects are of two types: Queue and Topic. A queue allows exchange of message objects through a virtual channel, that follows *First In First Out*(FIFO) model. Despite existence of multiple receivers waiting, a queue ensures the retrieval of message by a single queue's receiver at a time. A consumer

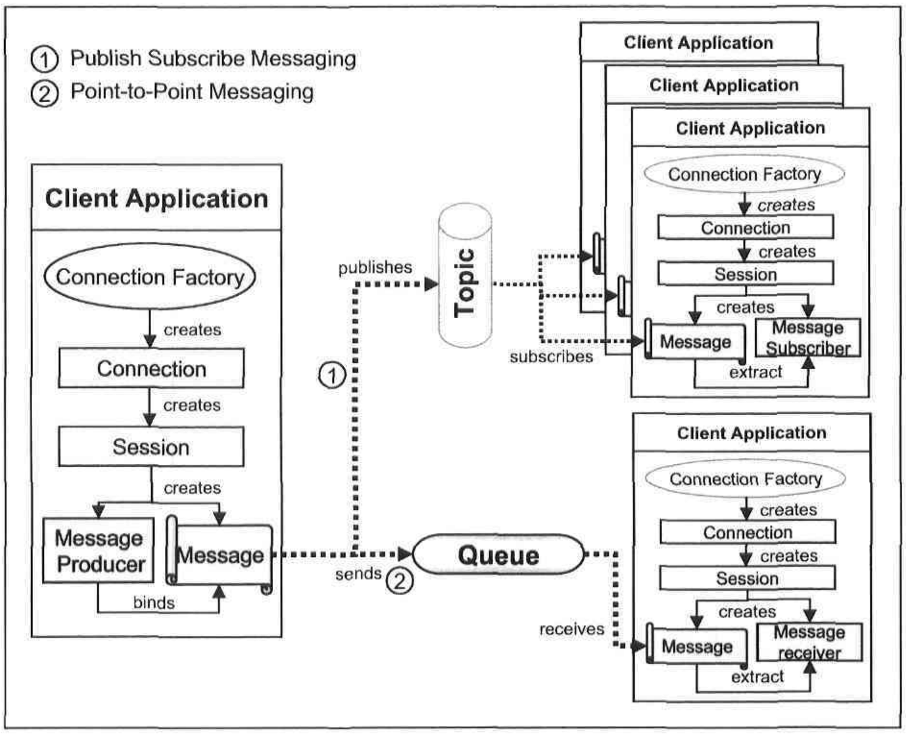


Figure 2.8: JMS Messaging Model

receives the messages in the same order as they are placed in the queue inside message server. JMS point-to-point messaging paradigm uses a storage inside each server node that keeps the messages for certain duration as configured. Queues are also used for load-balancing as well when many diverse systems share processing operations through a proper distribution of incoming messages. The JMS publish-subscribe messaging is implemented through a connection component, called JMS Topic. A topic implements a multi-threaded architecture that is capable of publishing a message object to its subscribers. A JMS publisher application places a single message into the message server and the later multicasts copies of the same object and distributes to all subscribers simultaneously. Figure [illegible] represents both of the JMS messaging models. However, subscribers may be synchronous, asynchronous or durable. A durable subscription accepts message for a subscriber even while the application is not connected. JMS also allows to configure various properties to improve the performance and reliability of the system.

The most important aspect of JMS messaging system lies in the loose-coupling between message producer and message consumer. Messaging service outweighs temporal dependency on the command based tightly-coupled process communication mechanism as in Remote Procedure Calls. According to JMS specification, the producer releases a message to the connection component irrespective of the availability and processing interval of receiving application. On the other hand, the consumer receives the message from the respective JMS component of the server in its side. Using Java message driven beans, a producer often creates temporary topic with `JMSReplyTo` header to put messages asynchronously in the destination. Messages are communicated through "store and forward" mechanism. In the following, we describe various binding technique related to the retrieval of messages from a JMS destination component [24, 69].

- *Synchronous consumer:* A synchronous subscriber, waits for a message always or for a specified duration (if the program implements `onTimeOut()method` to deactivate the listener). As it receives the message, the listener sends it for processing and then blocks to receive again. JMS 'listener' provides an `onMessage()` method. In the programming level, the functions `QueueReceiver.receive()` and `TopicSubscriber.receive()` for synchronous retrieval of message [69].

- *Asynchronous consumer:* An asynchronous consumer receives the message when a message arrives but does not block the connection between the destination and application process. An asynchronous JMS message listener is programmed with event based notification inside `onMessage()` method. Implementation is done using `receiveNoWait()` method. However, as it receives the message, it may or may not use message selector to filter the incoming messages from more than one topic. Loosely coupled asynchronous consumer is more useful for free information exchange, except there is a typical need for synchronous transaction.

- *Durable consumer/subscriber:* Durable subscription frees the consumer from staying continuously connected to the JMS server. "Store and forward" messaging mechanism also allows server to store messages inside server on behalf of a subscriber while client program is not available. It allows guaranteed messaging so that the consumer receives all the messages at re-connection, irrespective of the duration of staying disconnected [69]. A JMS durable-subscriber may be defined through `createDurableSubscriber()` method within a session.

2.7.5 JMS Properties

According to its specification, JMS inherits a large number of features with enough scope of further development. The properties of JMS API may be classified into two categories: architectural properties and message properties.

- *Architectural properties:* JMS is standard-based, robust and resilient enterprise messaging system. Architecturally, it is a distributed set of loosely coupled components that are centrally managed. JMS favors significant performance improvement over the system using asynchronous retrieval of messages. Messaging technique is reliable and durable, due to the "store and forward" mechanism of communication among destinations. JMS ensures once-and-only-once message delivery [51]. It also strives to maximize portability of messages within different cross-platform products within the same messaging domain. Loosely coupled nature of JMS allows seamless integration of applications with high scalability. Moreover, it supports a big number of market available APIs (JDBC, JCA, Java Beans components etc.) in different layers of the middleware and adds asynchronous messaging of serialized object data.

- *Message properties:* Messaging service allows more flexibility over communication among distributed components. Messages are inter-operable structured serialized object. Messaging properties ensure two delivery mode: persistant (message delivered once and only once) and nonpersistent (at most once, message may be lost if JMS server fails). JMS also offers properties of message expiration, time-stamp, priority settings etc. Message properties implement message selector to retrieve message from multiple topic and run a filter program as specified by a user. JMS permits huge scope of programming using the API. Messaging is provided with acknowledgement mechanism to assure delivery. A message may be read-only. A fair amount of security policies are also associated in relation to authentication, authorization and confidentiality on the JMS destination components [69].

Java Message Service is implemented as an open standard. It inspires with a big portion of implementation specific functionalities for its vendors. As an example, the specification of JMS does not cover load balancing, privacy, integrity etc. properties, however well implemented some vendor specific solutions like SonicMQ [79]. Also, there is no implemented system message for error notification. Furthermore, JMS specification does not define administered objects and JNDI components for the administration of messaging products. JMS messaging is free to implement except following the above-mentioned high-level properties. The architecture empowers the possibility of seamless integration, although practically, there is an obvious limit of extensibility, if the send rate far exceeds the receive rate, the queue or topic memory will be overflown with leakage of message.

A number of MOM vendors participated in the specification request of JMS . Sun Microsystems was the spec-lead, however it is more an industry effort than a single company-wise initiatives. The choice of the best JMS server is difficult and driven by the environment of the middleware solution. Commercial JMS implementation [13, 32] of SonicMQ, TIBCO-Rendezvous are better in performance than first generation of JMS implemented by Sun, IBM webshpere.

However, for the middleware we are highlighting architecture and implementation in Chapters 3 and 4, we adopted Java Message Service version 1.1, freely available with J2EE application server solution from Sun Microsystems.

To conclude with Enterprise Application Integration, besides being a big challenge in large corporations, this topic has been fuelled by the explosion of the web, the high degree of heterogeneity in existing systems, and continuously increasing number of data and service sources.

The first solutions related to this problem are time and money consuming, having limited capabilities and dealt only with data. However, emergence of new business initiatives over the web, have made from EAI a big concern for both community research and business world. Therefore, the concept of standards started to emerge during the last decade; infrastructures for distributed systems, data connectors, communication middlewares are just few examples. Java technology represent an important part [66] (more than 70%) from EAI standard solutionsand supported by almost all active software vendors: SUN, IBM, Oracle, BEA, HP, Sybase, Attunity, and many others.

Recently, Web Service concept has been introduced as a promising standard approach over the web that increase reusability of services and applications sharing. The Web Services stack is based on a set of standards: SOAP, WSDL, UDDI, ebXML, etc. and all of them based on XML. To support these satandards, a set of organizations and concortiums appeared such as Java Community Process (JCP), World Wide Web Consortium (W3C), Organization for the Advancement of Structured Information Standards (OASIS), RosettaNet, and many others.

Messaging service enables enterprise application to be loosely coupled rather than tightly coupled API, such as Remote Procedure Call. Therefore, applications can send information to another and continue to operate without waiting for an immediate response. JMS is the standard and most prominent messaging service widely used in application integration.

2.8 Decision Support Systems and Digital Cockpit

In this section, we provide an overview of decision support systems: evolution of DSS, data explosion phenomenon illustrated by some examples, types of DSS, and DSS architecture. Afterwards, we present a new wave of DSS meant for real-time information systems integration and decision making.

2.8.1 Decision Support Systems

Like any information system, a decision support system deals with collection of data, their organization and storage. However, in a decision support system, the needed data can be quickly retrieved and transformed into information for a decision making purpose. A decision support system highlights decision making effectiveness and decision making efficiency [29] rather than efficiency alone; which is the case of information systems. DSS effectiveness means what should be done in a decision making situation and ensuring that the chosen criteria are relevant, whereas decision-making efficiency means minimizing time, costs, or effort.

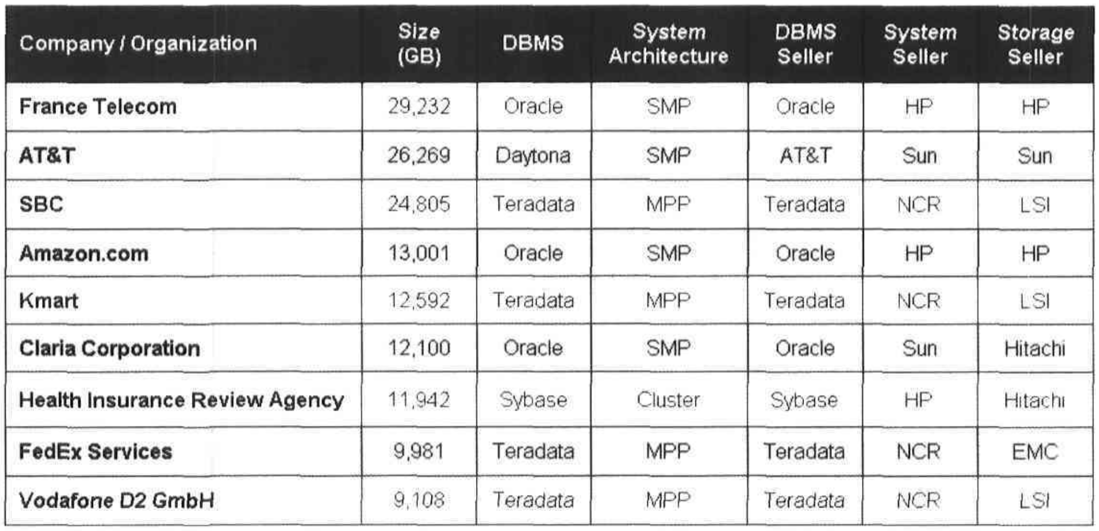
Traditional methods to turn islands od data, generated from infinite connected systems and from the web, into useful knowledge lie on manual analysis and interpretation. For instance, a specialist in Customer Relationship Management (CRM), analyzes periodically data relative to customers, prepares a detailed report and then transmits it to marketing service to take the appropriate measures and decisions. This report becomes the basis of future decisions in this domain. This analysis approach is applicable with almost other domains of science and business (health care, commerce, space, etc.).

The aforementioned approach of data analysis may be suitable for small businesses where the analyst handles few data. However, this is no longer the case nowadays, databases evolved from many hundred records to many thousands and millions, according to two dimensions: the number of records in the database and the number of attributes within an object. Therefore, new generation of tools and techniques to extract nuggets of knowledge from the enormous amounts of data is getting more imperative than any time before. Digital cockpits are among these new systems especially for real-time integration and user friendly graphical presentation.

2.8.2 Data Explosion Phenomenon

Nowadays, databases contain billions of records and thousands of attributes. Figure , taken from a survey achieved by the company Winter Corp's in 2003 [48], illustrates data explosion phenomenon.

This survey shows that the biggest database used in decision support approaches 30 tera bytes (30.000 Giga Bytes!). Another example, the Europe's Very Long Baseline Interferometry (VLBI) has 16 telescopes; each one generates 1 Gigabit / sec of astronomic data, during 25 days of the observation session! Another study estimates that between 1 and 2 exabytes -10^9 Giga bytes or 10^{18} bytes- of data are produced per year [31].



Company / Organization	Size (GB)	DBMS	System Architecture	DBMS Seller	System Seller	Storage Seller
France Telecom	29,232	Oracle	SMP	Oracle	HP	HP
AT&T	26,269	Daytona	SMP	AT&T	Sun	Sun
SBC	24,805	Teradata	MPP	Teradata	NCR	LSI
Amazon.com	13,001	Oracle	SMP	Oracle	HP	HP
Kmart	12,592	Teradata	MPP	Teradata	NCR	LSI
Claria Corporation	12,100	Oracle	SMP	Oracle	Sun	Hitachi
Health Insurance Review Agency	11,942	Sybase	Cluster	Sybase	HP	Hitachi
FedEx Services	9,981	Teradata	MPP	Teradata	NCR	EMC
Vodafone D2 GmbH	9,108	Teradata	MPP	Teradata	NCR	LSI

Figure 2.9: Data Explosion Example
Source: Winter Corp's Survey, November, 2003. www.eweek.com

From the above estimates, the basic question a decision maker might raise that is it possible for humans to analyze and interpret these islands of data to enhance the decision making process? The legitimate answer would be that the human capabilities cannot address this kind of problems. Therefore, information systems and decision support systems have been developed. In the following section, we will reveal the evolution of decision support systems, and the most important research that relate to this area.

Evolution of DSS

In the 50's, computers were luxurious and capabilities limited. Therefore, it was costly to build information systems. In the late 60's, with the development of more powerful computers, such as IBM system 360, information systems era had more scope in many of large corporations. Little after, a new category of information systems became practical: Model-Oriented DSS or Management Decision Systems. The concept of decision support system was first claimed by Peter Keen and Charles Stabell [18]; two DSS pioneers, in the 60's. In the 70's, business journals started to publish articles on management decision systems, strategic planning systems and decision support systems.

The first decision support systems were simple interactive information systems developed to help decision makers taking educated decisions in complex fields such as financial management and strategic decision making (military field for example). These systems use semi-structured data. A few years later, DSS have been enlarged to support spatial, multi dimensional and unstructured data. As example of these systems, we can refer to GADS[5] [23]. Those systems support many models such as optimization, simulation and statistical packages [18]. In the 70's, financial planning became popular DSS tools, used by some executives to build models autonomously. IFPS[6] was one of the famous planning languages at that time [28]. In the 80's, many works and researches have been done in the aim to develop a new category of software tools that can support group decision making . We find in this category Mindsight from Execucom Systems, developed at the University of Arizona and the SAMM system developed at University of Minnesota [55].

At the beginning of 90's [18], data warehousing and On-Line Analytical Processing (OLAP) started to emerge as a new generation of decision support systems. The warehouse and its related tools provided a big potentiality to enterprise information systems (EIS), evolving them from single user and model-driven to multi-user and data-driven respectively.

Lately [7], driven by data explosion problem as stated above, a need for a new generation of tools that are capable to analyze online data became crucial. Therefore, new kinds of tools supporting real-time business started to appear. Digital cockpits is an example of these tools that represent a new paradigm for information integration and decision making.

[5]Geodata Analysis and Display System

[6]Interactive Financial Planning System

Below we provide the nain types of decision support systems.

Types of Decision Support Systems

In literature, there is an agreement that DSS can be divided in several groups. This affiliation (affectation) to a group reflects the functionality for which the DSS has been developed. In decision making process, a manager can face many kinds of decisions, depending on the nature of the affronted problem. These problems can be classified into many categories from a simple database query to data analysis and optimization. According to each situation, there is a matching between the problem and a type of DSS. The type of DSS reflects what a system does and not how it was built. The classification of DSS into many types was originally used by Alter [72]. A DSS falls in one of two big categories: data-oriented or model-oriented DSS. A data-oriented DSS focuses principally on the database and comprises file drawer systems, access data systems, and data analysis systems. The problem with this category of systems is that they do not show what really the system does, and do not outline the business situation. However, a model-oriented DSS provides some analysis capabilities, and it consists of accounting models, representational models, optimization models, and suggestion models.

On the other hand, a decision support system can be for personal, group, or organizational use. The first focuses on an individual user and the achieved task is generally independent of other tasks. The second, focuses on a group of users, each one involved in a separate task but in relation with other tasks. This category is often called groupware. Finally, the last category focuses on organizational activities involving a sequence of operations, different functions and locations [30].

Decision Support System Architecture

As any information system, a DSS architecture reflects the environment in which the DSS runs. This architecture defines which approach, hardware and software will be used by the DSS. The DSS runs on the top of this architecture, which is actually a key factor of a DSS accomplishment and responsiveness.

Figure 2.10 outlines a generic DSS architecture. It consists of the following components: data management module, model management module, and user interface module.

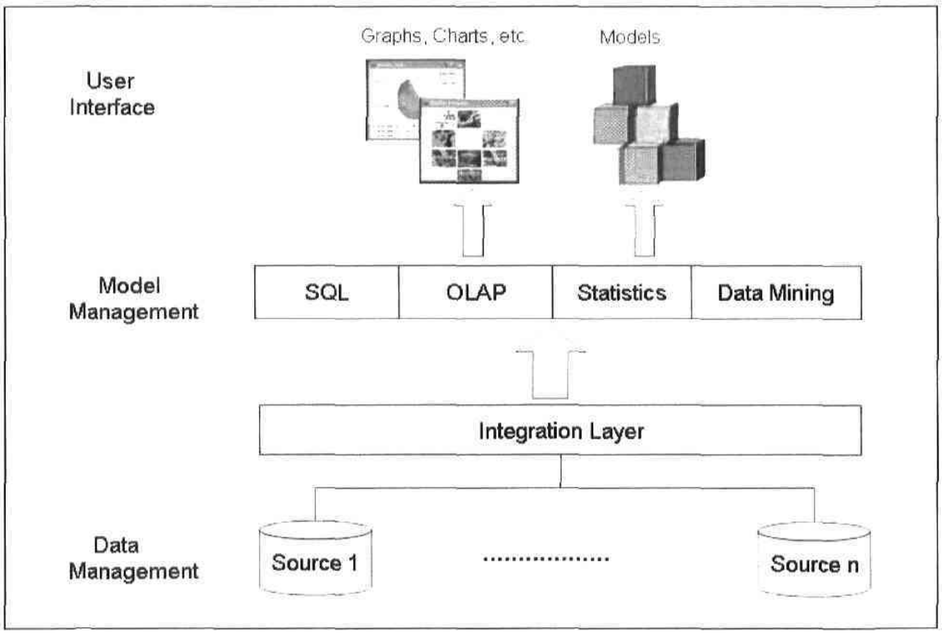


Figure 2.10: Generic DSS Architecture

- **Data Management:** Data management component stores and maintains information required for DSS system. This component is responsible for storage and retrieval of needed data in related DBMS and by other DSS components respectively. Also, in case of use of many DBMSs, it is responsible for ensuring communication and integration of different involved data sources together. As discussed before, this layer corresponds to information sources integration and messaging service middleware to retrieve data and services, and provide them as input to other components.

- **Model Management:** This component is responsible for the model management system that executes the business logic. As the DSS deals with multiple possibilities of solutions for a user request, it uses and runs different techniques through this component. These optimization techniques includes analytical processing like OLAP, statistical analysis like time-series, mission planning etc.

- **User-Interface Management:** This module is responsible for the presentation of client's answers. Traditional user interfaces with piles of data and tables are frequently used in the industries although several disadvantages associated to it. However, recently new trends in user-interface emerge to reflect real-time business events.

Since DSSs are special category of information systems, their architecture follows the conventional computing paradigm: central, client/server or distributed architecture

[73]. First, centralized computing [70] characterized the seventies (70's) and the beginning of the eighties (80's). This paradigm has been marked by the centralization of all program intelligence (business logic) and data into a single computer called central system or mainframe which is accessed by 'passive' terminals. The architecture of centralized systems is generally a star topology. Regarding to DSSs, the databases used by the DSS are accessed at a single location; therefore the decision is always based on up-to-date data. Also, the hardware architecture of large systems [30] allows the execution of complex DSS, such as large calculations, prediction tasks, simulations and expert systems with thousands rules. Moreover, adding new users is relatively easier since the main configuration lies in the central system. Finally, the security is another issue that can benefit from the centralized architecture since data and programs are in a single location, so building powerful security mechanisms is much appropriate. However, within this architecture, the single-point-of-failure characteristic may affect the availability of the system, and interactivity is less suited. Secondly, the client/server paradigm returned to mid eighties (80's). The monolithic applications centralized in a single system have been split into two halves: database in the server side and programs in the client side. The server provides access to shared resources (typically the database) and the client (typically a personal computer) hosts the business logic and is responsible for the graphical user interface [70]. This architecture gives better performance since the applications are consuming the client machine's CPU and memory. The client/server option provides more freedom to choose DSS tools according to the client operating system, and software tools are generally, more available and cost-effective for microcomputers than servers. On the other hand, dealing with the middleware and the network make DSSs development within this paradigm more complex than in a single computer. In addition, this architecture requires more skills in different domains (specific OS systems, DBMS, networking, etc.), and sometimes puts software compatibility problems between the client and the server. Lastly, within distributed computing, the notion of hierarchy (server/ client or terminal) has almost disappeared. The majority of the network nodes are in the same level (peer-to-peer architecture). Again, with distributed computing, the centralized database in the server side in client server approach has been splitted into many pieces of code located in different machines. Distributed applications are built over distributed systems infrastructures. Among these important infrastructures we have Common Object Request Broker Architecture (CORBA) from OMG group [60], Distributed-Component Object Model (DCOM) from Microsoft [50], etc. Nowadays, the main concurrent technologies to develop distributed applications are: Microsoft .Net from Microsoft Corporation and Java 2 Enterprise Edition (J2EE) from Sun Microsystems Inc., many other main software vendors (IBM, BEA, Oracle, etc.), and Open Source organizations.

Like other software application development over distributed platforms, there is no exception with DSS applications development. The multiple data sources queried (used by DSS may be hosted on different machines, in different locations, and heterogeneous in terms of data models, schemas and used protocols) [6]. Also, the DSS applications and access tools may be different from a client to another depending on the accorded privileges, profiles and type of users (light, fat, mobile, etc.). Within this architecture, many challenges have to be addressed such as performance, security, complexity, etc. On the other side, this approach allows a widely and universal presence with the ubiquitous computing paradigm.

2.8.3 Digital Cockpits

Mostly used to refer to the pilot's compartment, the digital cockpit [94] word appeared first in 1914. The digital cockpit contains read outs from instrumentation and controls used by the pilot to fly the aircraft. It affords him a clear and unobstraucted view above, below and around the aircraft.

By analogy to aircrafts, the information overload phenomenon has been the driving force behind the creation of digital cockpits or dashboards in businesses and organizations. A digital cockpit [75, 95] points to the new wave of software applications that allow real-time displaying of key information gathered from several sources on a computer screen, in a format tailored to the needs and wants of an individual knowledge worker. In other words, as a picture is worth a 1000 words, a digital cockpit is a graphical depiction of real-time business performance from far-flung operations. The digital cockpit provides a continuous stream of information, and time is its most important element. This viewing of what is happening in the business effectively lets managers make decisions clearly and instantly, and not in delayed mode. Here after, we provide the features of this new generation of real-time oriented software.

- Digital cockpits offer sophisticated user interfaces that provide the user with information without asking for it, once he has subscribed to. This happens with events notification mechanisms.
- Generally, the information displayed on the screen is sent by the back end application, using some sophisticated middlewares (Java Message Service (JMS) in our proposed middleware).
- As a result, use of these sophisticated technologies leverages many key features such as: guarantee of delivery of information; multi casting of information; and

the loosely coupled communication between the digital cockpit client and back end application.

- Digital cockpit is intended to provide the decision maker with analysis capabilities, which leverages the decision making process.
- From a user stand point, the digital cockpit is a multiple applications at the same time [75]: since that all displayed graphical components are key features representing a crucial business activity.

Thus, the digital cockpit paradigm allows connecting the most enterprize information sources, providing a better visibility, and setting metrics to monitor key performance indicators (KPI) to respond in real-time to business events. This reaction to events enables organizations to make well educated decisions based on key performance data.

As for each new approach or technology, the digital cockpit is still a new field in information technology. It has been successfully implemented in some large corporations such as General Electric [19]. However, technology behind it is still in a research phase, especially for the real-time notification of business events.

Implementing Digital Cockpit Systems

A successful design and implementation of a digital cockpit should focus on the principle organization's goals, because understanding the goals and processes is more important than the selection of the technology. Each key performance indicator (KPI) should appear on the digital cockpit interface.

Once what to display is well defined, there should be a very good understanding of data sources in the organization, because the graphics and visualizations are based on data coming from different sources. Since these data generally come from multiple and different sources, a data integration stage is mandatory to prepare displaying graphics and charts. Moreover, digital cockpits are meant to reflect real-time changes in data sources, therefore redrawing graphics and refreshing displays is another issue to consider, because mechanisms that feed with data should not send the entire data once again, but just the changes. This gives more performance and less traffic to the network, and smoothly changes in the graphical displays. In addition, graphical screens have to be well presented, provide a big picture on what is happening in the organization, and enable drilling into details.

Digital Cockpits: Technology Stack

Decision support systems are tools that help users in their decision making process. They enable to show graphically the enterprize KPI and allows taking informed decisions based on accurate and up-to-date data. Building a DSS is a multi-phase development, from the lowest level which is data management module to the highest one which is user-interface module, and between them model management module.

Data Management

As we have already seen, data management module aims to connection sources of information within and across the organization for information sharing purposes. Different approaches and related technologies are depicted in Chapter 2 (JDBC, JCA, JMS, etc.). In the next Chapter, we detail this phase in our proposed approach.

Model Management

For model management module, the domain of Business Intelligence (BI) plays a key role in this scope. Most of big corporations have their own analytical tools that help them to make sense over the massive amounts of data contained in their operational systems. These tools allow users to generate reports that analyze slices of data, or make complex queries to measure business performance. However, a major drawback of these solutions is that they require users to run reports and interpret results. That is why graphical methods for visualizing results of analysis are required. As defined above, the primary task of these analytical tools is to accept data, fed from different data sources including data warehouses, legacy systems and/or external sources and finally to allow synthesized display of information in a meaningful way to the end user. In this way, they are key components for the success of the organization since they allow intuitive insights into negative trends, positive developments, and emerging opportunities.

Also, the notion of BI is comprised of applications and technologies for extracting data from operational databases and organizing it into virtual databases. This data is then used as input to reporting, statistical analysis, data mining and On-line Analytical Processing (OLAP) tools. The role of OLAP tools is to show trends and generate relevant information that is directly used in decision-making process. Using BI tools like OLAP, very complex analysis and synthesis capabilities can be incorporated that clearly outweigh traditional query and generation of reports. Many related OLAP approaches are coined by software vendors to support multidimensional characteristics. For example Multi-dimensional OLAP (MOLAP) has been introduced to organize data in a non-relational database and Relational OLAP (ROLAP) is released to simulate the data cube used in MOLAP servers. As an example of such MOLAP databases, one can refer to Hyperion Essbase, Oracle Express and MS OLAP Server [12]. Vendors, who deliver ROLAP solutions, include Microstrategy and Informixs Red Brick [63]. A Hybrid OLAP (HOLAP) is a combination of the MOLAP and ROLAP approaches in which all data are stored in a relational database. Business Objects, Brios, Cognos, and Seagate all offer OLAP clients that can access a wide range of OLAP servers as well as traditional RDBMS servers [58]. The present evolution for OLAP is marked by its widespread distribution and accessibility in the web. Presently, OLAP clients are capable of supporting a multi-tier architecture where a middletier application server is used to access data from many sorts of databases [62]. Thus, OLAP is a flexible way to view, analyze and display data in the form of reports. However, there are certain limitations to OLAP technologies. For instance, the quality of the extracted information depends on the users interpretation of the results and is thus subject to error.

User-Interface

For the third component which is User-Interface; the added value by digital cockpits systems is mainly seen through data visualization tools. To take 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. Among the most prominent software solutions in this field, we cite Quadbase EsspressChart [38], JCLASS from Quest Software [78], and JFREECHART [16].

- As of Quadbase EsspressChart, it is a Java-based charting toolkit that allows users to build charts easily, interactively and programmatically. The required data for plotting may come from databases, text data files, XML files and Ms Excel spreadsheets. The API makes it easy to deploy charts with applets, servlets, JSPs and Java applications.

- Concerning JCLASS, it is a suite of Java components under J2EE and J2SE. These components help to build quickly and easily applications 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).

- On the other hand, there are some general purposes visualization tools that are directly available in the market. Examples of these tools are: ADVIZOR [15], OpenViz [83], NetChart Reporting Suite [54] and Xcelsius [65]. The ADVIZOR toolkit provides a presentation of business data along with a query capability.

OpenViz is a suite of software components that enables the development and deployment of interactive 2D and 3D visual presentations. NetCharts suite allows embedding charts and graphs into web-based applications. Xcelsius suite is a windows application that allows users to create real-time, interactive reports based on Excel spreadsheets.

2.9 Conclusion

The main objective of application integration is to enhance shareability of data and application services. EAI include several approaches: data integration, business process integration, portal oriented integration, and service integration. In the scope of this thesis, we adopt federated architecture to connect different systems together, and Java message service as a mean to push the fetched data to different remote clients. This choice leverages real-time integration and notification of events.

Digital cockpits represent a new paradigm for information systems integration and decision making that provides visibility of real-time business events and key performance indicators (KPI). They help to remove the greatest source of latency in the exchange of information and therefore provide support to the new event-driven economy.

They are intuitive and interactive, so managers can easily drill in/out into details and therefore can quickly get to the root cause of a business situation. Furthermore, contrary to other approaches of application integration, digital cockpits allow enterprises to address the information overload in a real-time way. This empowers companies to react quickly to business events and rapidly enhance their strategies to meet the customer's needs.

In the next Chapter, we present our approach to build a digital cockpit system.

Chapter 3

Digital Cockpit Architecture

In this chapter, we introduce a loosely-coupled generic approach based on a message-oriented middleware. This approach allows an asynchronous exchange of information, which solves a long-lasting organizational need: decouple the client application from the back end systems, and reflect the changes occurring in the information sources without any recursive requests. In this chapter, we also introduce a performance scenario to compare between the two technologies: MOM based middleware and RPC-based one. Finally, we close this chapter by presenting some challenges and issues related to this work.

3.1 Introduction

Monolithic and tightly-coupled request-response middlewares suffer from restrictions of performing asynchronous operations on the application level. Therefore, a middleware solution on top of frequently changing data in different sources requires periodic requests from the application and manual re-configuration of programs to receive present status of information. The system, as a result, suffers from limited performance and carries a huge unnecessary network load. This scenario can be explained with a simple Java servlet of JSP web-based application. Technically, a user submits a request through a web-based form or application running in a web-browser's virtual machine. It directly places a command-based process call as a payload over communication protocol. A request triggers a local process in the remote server end, that collects information or perform specific operations mostly in a synchronous way and responds to the request by refreshing the application with static instance of information. The connection during

the whole process is managed by a request-response broker (such as CORBA). Limitations of such process call are well-known and mentioned before.
Within this research, we propose an approach to build a message-based middleware capable to execute asynchronous operations. Opposite to other available products, the proposed middleware does not work in a request/response mode. Therefore, there is no bounding between the application and the resource tier. As a result, clients will get up-to-date data without continuous asking for it or by periodic refreshing.
As we have already stated above, this approach is an event-based mechanism and implemented through Java Messaging Service, which provides much better performance comparing to RPC-based mechanisms. Such an approach has the following features:

- **Retrieval of Information:** The MOM model provides ways to retrieve and fetch data by setting up certain mechanisms that allow subscription to data and services.
- **Updates Notification:** Such a MOM-based approach reflects the changes occurring in the data source tier, in a real-time or near-real-time manner, and sends them to subscribed users.
- **Performance:** One of the most important features that characterize an integration approach. Therefore, this MOM-based approach should ensure optimal use of resources either in client side or in the data source tier.
- **Scalability:** The model should provide easy extensibility, by adding new data or service sources; without modifying the main architecture and keeping an acceptable level of quality of service.
- **Autonomy of Information:** As the data and service sources are the properties of certain organizations/ departments with their own policy of information distribution, the approach should be open enough to allow enterprises to apply their rules and regulations.

Consequently, the proposed model of middleware is expected to be generic and having the features above. However, its implementation can be done in many phases. In what follows we detail these phases, and we refer to the resulting architecture by "Digital Cockpit".

To sum up, the main contribution here is the proposition of a new paradigm for real-time integration and display of heterogenous information, based on a MOM that overcome the limitations related to the conventional RPC middlewares. Therefore, we include in this chapter the methodology including different phases to design and implement such a solution. Also, we provide a mathematical model that shows the added value of this approach compared to other ones that are based on RPC mechanisms.

3.2 General Description

Digital cockpit project accomplishes an efficient and useful decision support system mainly based on the information and service integration. Digital cockpit system relies on the following modules: integrator, subscriber, display manager, monitor, analyzer, and controller.

The integrator is in charge of gathering the required information by interacting with the different data sources as required by the user. It retrieves data from local sources directly as well as uses Java messaging technology in the case of remote sources. There is a subscriber module, proposed at the client's end that is used to express interest in the needed information and therefore open a new connection depending on user privileges. Using JMS point to point subscription mechanism, the subscriber is also able to provide the end user with the capability of subscribing to the information of interest directly from the remote data sources. On the other hand, if the digital cockpit client updates any information in the data sources, the integrator interface associated to the sources on server side will trigger a publish mechanism that uses a common JMS topic to publish the necessary changes. When such information is produced in the topic, the messaging service notifies the relevant subscribers and supplies fresh information to them. The monitor module is in charge of tracking the organization's assumptions with this change of information and responds actively to dashboard thresholds without any direct user-intervention. The client module also contains some local services to analyze and optimize the different scenarios comprised with the aggregated data. The analyzer module is responsible for performing the needed simulation, pattern and trend analysis, while the controller is the module that optimizes the different scenarios and processes as required by the user. Finally, at the end-user, the display manager takes care of the graphical layout and the production of structured and navigational "big picture" of the system.

3.3 Layer-based Integration Approach

The presented message oriented middleware follows a multi-tier paradigm (Figure 3.1) that can be achieved with three distinguished phases. The first one provides a unified framework over the heterogenous data of information and services. The second, relies on messaging service to get the fetched data and push them to the client side. Finally, the third phase explains the client side of this middleware. It provides users with subscription capabilities to information from different sources, and is also responsible of display and analysis of this information.

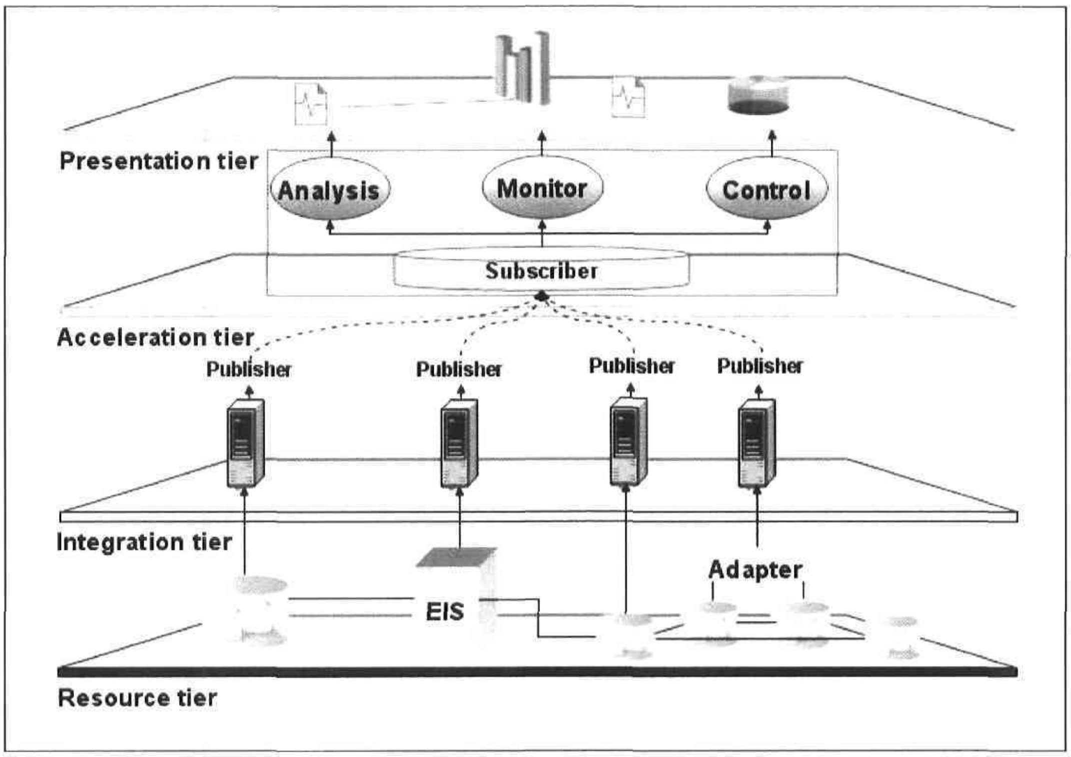


Figure 3.1: Layer-based Approach

3.3.1 Phase 1: Unified Framework

In this phase we build a unified 'wrapper' on top of the resource tier that provides a common programming abstraction in order to access information. The resource tier is responsible for the persistent storage of heterogenous data sources such as relational databases, spreadsheets, legacy applications, etc. and service sources with various XML formats. However, the connection of all these systems is very complex since they follow their platform-specific semantics.

Data integration is related with the retrieval of data from different sources. To achieve this, we adopt a federated architecture that keeps local autonomy of data sources. Hence, we establish a software mediator or wrapper on top of different data sources. This will create a virtual repository that allows an on-demand access to needed data. The implementation of the wrapper is resource-specific. Once a wrapper, on top of a data source, receives an object-message, it converts it into the source semantic. Similarly, it also transforms the returned response from the data source into a standard format useable by the service call.

For example, connecting to an Oracle or Sybase database, we create a JDBC connection to retrieve information. On request from a remote client, the wrapper converts the object-message request into a standard SQL query and applies it on the application running JDBC connection. Similarly, on receiving a Resultset, it is once again converted into an object-message and sent to the middleware that carries it out to the requestor.

We implemented this approach with pure J2EE technology. This includes JDBC and JCA in order to invoke information from the structured and semi-structured/unstructured data sources respectively.

3.3.2 Phase 2: Messaging Service

Being the core of the proposed middleware, the messaging service allows synchronous and asynchronous binding of information and services. RPC-based middlewares directly invoke the communication protocol, such as HTTP, which is a request-response model. We approach the proposed middleware by adding a new layer over the communication protocols, called messaging service. More explicitly, the proposed model uses a modified request-response broker that allows to keep the connection details and receive multiple responses generated from a single request via messaging technology. Such a middleware requires both one-to-one (P2P) communication and one-to-many (publish/subscribe) communication simultaneously. At this level, we choose Java Message Service API, provided by Java Community Process (JCP) as a specification to implement standard messaging, which provides support for both types of connection Queue and Topic, and allows both synchronous and asynchronous communications. The core of the JMS setup is a Lightweight Directory Access Protocol (LDAP) server offered by a JMS service provider. JMS uses Java Naming and Directory Interface (JNDI) in order to retrieve the implementation-specific settings kept in the LDAP server. This JNDI/LDAP creates the Connection Factory.

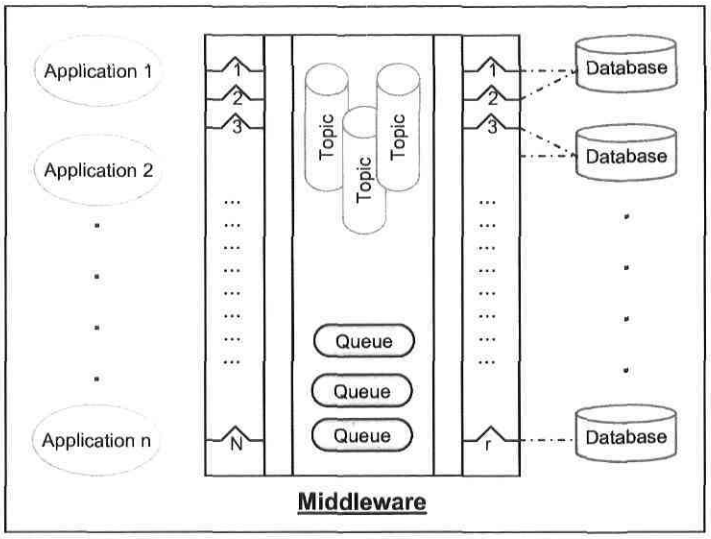


Figure 3.2: Integration Model

In our proposed solution (Figure), a client system in the beginning requires an initial step to start a component, which consists of a direct communication between the client and the server. Messaging service provides a direct P2P communication mechanism (e.g. JMS-Queue) between the requested client and related server/s. Once the client system starts displaying retrieved information, it only requires to gather real-time notification of changes. With the update of data, the underlying framework will automatically trigger a publisher module to publish information in a JMS- Topic that follows publish-subscribe mechanism. A client-end software solution simply invokes a listening process in the background and listens to the changes published in the required topic. As the changes are found, listeners receive the information as a message and update the visualization of the information in client's side.
This mechanism significantly improves the middleware with the following features:

- **Asynchronous and Loosely Coupled Communication:** A user application may perform other tasks without waiting for a response.
- **Real-Time Notification:** Applications receive real-time notifications from remote sources in real-time and without recursive requests.
- **Cross-Vendor Communication:** Processes running in different virtual machines(VM), may communicate through messages without any remote process

call as implemented in RPB-based mechanims such as Java RMI. This eventually solves the cross-VM data access problem of using applets as discussed before.

- **Quality of Service:** This approach provides more control on messages communication. The guarantee of delivery, messages priority, security, easy extensibility, and scalability give more reliability to the solution.

Moreover, the use of JMS in this approach allows temporary storage of information that may be helpful to serve a big number of clients quickly without repeating the process of requesting data or service sources. On the other hand, any major change of information can be explicitly notified to all the interested users.

3.3.3 Phase 3: Client Application

In order to receive asynchronous message objects coming from the lowest layers through JMS, as shown in Figure 3.1, the digital cockpit client requires a subscribing component.
However, there are two major issues to resolve here: First, the client application requires sending request to the JMS server and registering to the request-response broker in an assigned namespace within the JMS server. Second, the application in the client-side should invoke listener/s to keep itself updated with the published changes and therefore reflects these changes in the presentation tier. Accordingly, we approach the problem using two basic modules in the client system: A subscriber module and a monitor module. Figure [illegible] includes the subscriber module in the acceleration layer, at the client side. Once an application user expresses his interest in some information, the subscriber sends the request to the JMS server. Afterward, message service layer intercepts the request and creates a message object that may be sent asynchronously. In the other side, the subscriber component invokes JMS listeners. Listeners receive the changes using publish/subscribe mechanism if a data source is updated and redraw the visualization of information in client's presentation layer (refer to Fig. [illegible]). Java Messaging Service provides built-in `TopicRequestor/QueueRequestor` methods. On retrieval of message, JMS listener uses `onMessage()` method. When a new message arrives in the middleware, JMS server publishes it into a 'topic', it informs the subscribers about the new information and the client subscribes to it. This model could be achieved synchronously, asynchronously and even as durable subscriber with the JMS destination, depending on the user's need.

In the user-side, each displayed set of information is interactive and provides straight forward means to drill in/out of information, rotate, zoom in/out, import/export and change to other views, where applicable. This architecture is much similar to Model View Control (MVC) and Observer design pattern. In the proposed middleware architecture, the presentation tier acts as "view" while the acceleration tier requires a "controller" element in the form of 'Monitor' module. Monitor module is implemented as an observer that locally controls the threads in the background of digital cockpit user application. It selects the active JMS listener according to client's choice, sends request to subscriber component and finally observes, updates and filters the real time notifications coming from the subscriber.

3.4 Digital Cockpit Artchitecture

Figure 3.3 depicts the architecture of inter/intra organizational integration of information sources with some relevant technologies that were used to support our architecture.

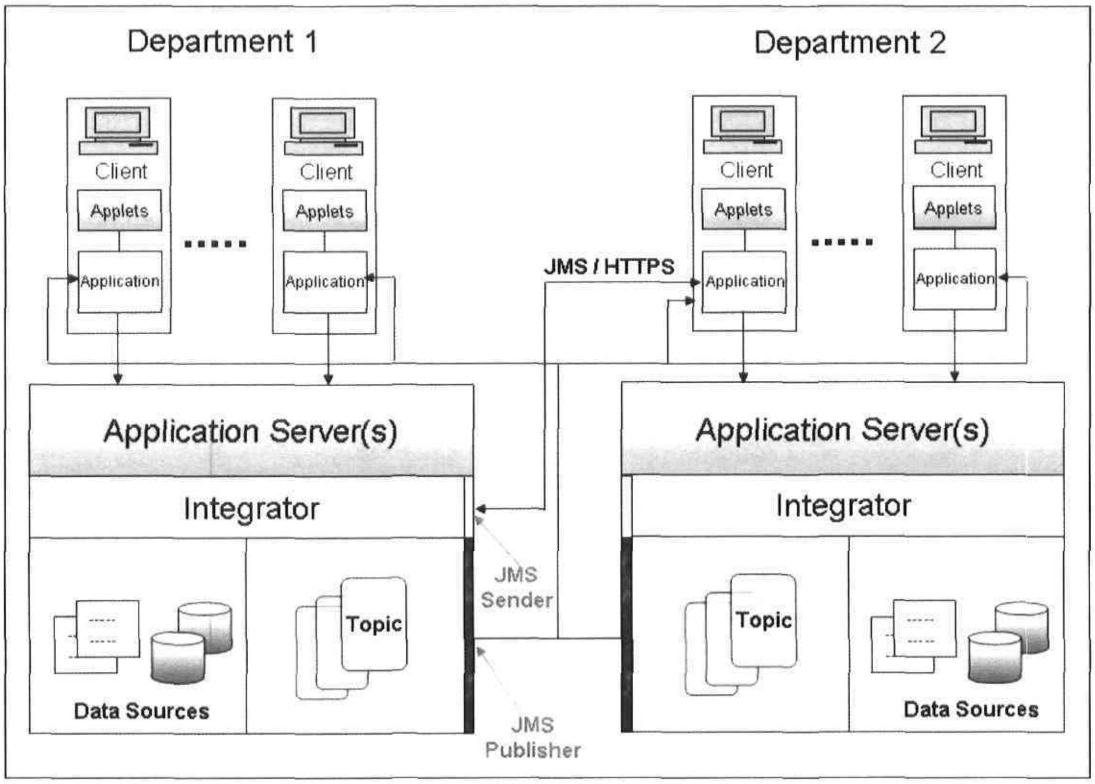


Figure 3.3: Detailed Architecture

The figure clearly illustrates the means of communication between various departments. We propose a message based point to point asynchronous inter-organizational communication (JMS Queue) as supported by JMS [51], followed by a tightly coupled

inter-organizational information retrieval using JCA and JDBC. Initially, when the client first requests to view some data, the middleware sets up a P2P communication between the client and various information sources. So when the initial set of data is ready, the MOM places it into a queue from which the client will directly consume the data. Then the middleware discards the queue and creates a topic which serves as a repository that holds updates whenever previously requested data changes.
The benefit of using such a technique (Topic) lies in the fact that since many clients can issue interest in the similar set of information, groups of these clients can subscribe to a corresponding topic and as a result, changes will only be calculated once, and communicated to many since the system implements the observer pattern and their subject is the corresponding topic. This will significantly reduce the bandwidth consumption on the network. Once the observers consume the updates in the topics, the corresponding graphical visualization of the affected components will directly update itself to reflect in real time any changes that took place in the resource tier.

3.5 Extending Digital Cockpit Architecture: Service Integration

The integration of services is another important part that completes the integration phase of the digital cockpit. In this section, we provide a mechanism to enhance the digital cockpit capabilities to include a service integration container, on top of integrated web services. To illustrate the service integration component, we considered a remote "weather service". Figure represents the detailed architecture. In what follows, we present the integration components: the integration container, the notification manager, and the execution engine.

- *Integration Container:* The need of an integration container rises due to the limitations of current technologies used to compose web services. Since web services are composed and executed in a one time fashion, the user will not acknowledge any change that took place after he last submitted a request unless he constantly keeps on interacting with the system requesting it to display its updated results. With the introduction of an integration container, the user interacts only once with the system indicating an interest in some services. After this point the user is guaranteed to be informed of all instant changes as they occur. This will provide the DSS analytical methods better aid managers since it will be acting on enough accurate and up-to-date information. In addition to this, the container will not need to re-execute the entire global process plan.

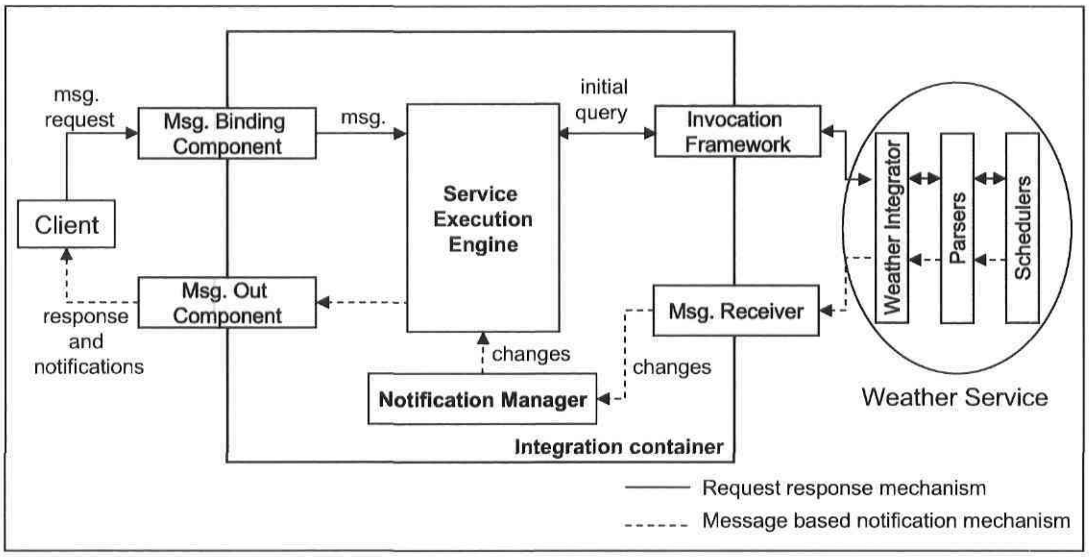


Figure 3.4: Architecture of Service Integration

- *Notification Manager:* Once a user requests (a) service(s), the integration container registers its notification manager to all participating services identified during the service planing phase. Here we are considering that services exhibit a mechanism which informs the notification manager that a change occurred in its information set. After receiving this message, the notification manager will inject this information to the execution manager which in turn will initiate some processes to reflect this update to the user. The importance of this notification manager is that it will free the user from the burden of continuously checking if data has changed since the last request was issued. Moreover, it will enhance the DSS process as a whole because the analytical capabilities of the DSS will process accurate data every time.

- *Execution Engine:* The execution engine within the integration container is where all the intelligence of this framework reside. Once the integration container receives a request from the user, it will pass it to the execution engine. At this point, the engine initiates a global planing event which decomposes the user's request into smaller ones.

3.6 JMS/RPC-based Middleware Comparison

To explain the difference between JMS based middleware and RPC-based one, we provide the following.

Suppose a middleware having N threads and invoking **r** requests on average to the resource tier in a single pass. Now, if **w** be the time for pre-request works of **N** threads and **t** be the time taken by each request to retrieve information. The total time to retrieve information, given that the requests being asynchronous, there is no waiting for response from the source tier, is approximately (only the network time is considered as it is most significant) [46]:

$$\delta = Max_{k=1}^{r}(t_k) \tag{3.1}$$

Scenario 1: Information Retrieval

We assume that a single thread is servicing **c** (**c** $\geq$ 1) simultaneous requests each time from users. Therefore, the time required to serve **R** requests is [46]:

$$\mathsf{time}_{MOM} = \mathsf{if} \begin{cases} ((R\ div\ c \times N) + 1) \times \delta,\ R\ not\ divisible\ by\ N \\ (R\ div\ c \times N) \times \delta,\ R\ divisible\ by\ N \end{cases} \tag{3.2}$$

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

In case of RPC middleware, since each request waits for the preceding one, an RPC-based application manages N simultaneous threads on average (r.t + w) seconds. So, the time taken by the system to serve **R** requests is:

$$\mathsf{time}_{RPC} = \mathsf{if} \begin{cases} ((R\ div\ N) + 1) \times (r.t + w), R\ not\ divisible\ by\ N \\ (R\ div\ N) \times (r.t + w), R\ divisible\ by\ N \end{cases} \tag{3.3}$$

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

From 3.2 and 3.3, it is clear that waiting time for response in case of messaging (time_{MOM}) is less than RPC-based middleware (time_{RPC}). Figure [illegible] illustrates this scenario.

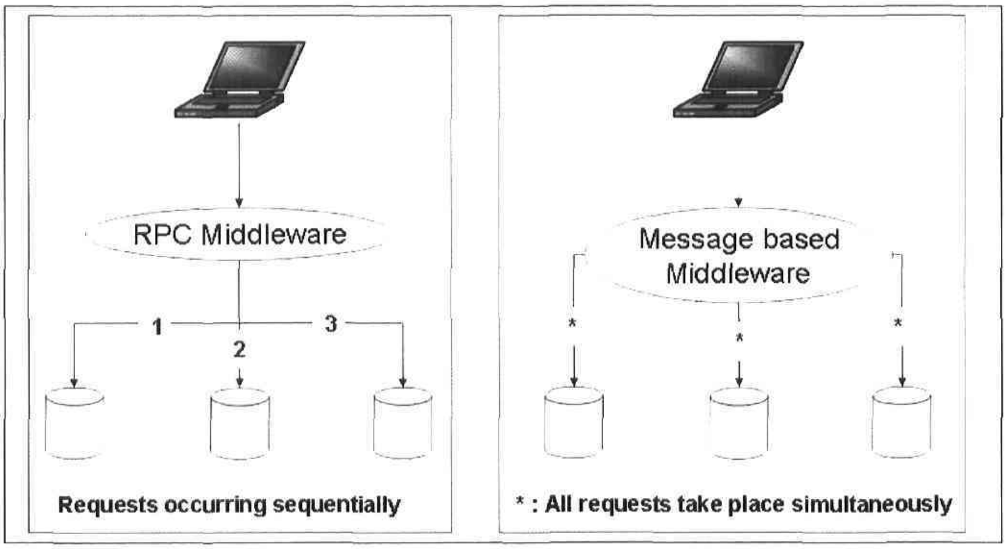

RPC Middleware
1
2
3
Requests occurring sequentially
Message based Middleware
*
* : All requests take place simultaneously

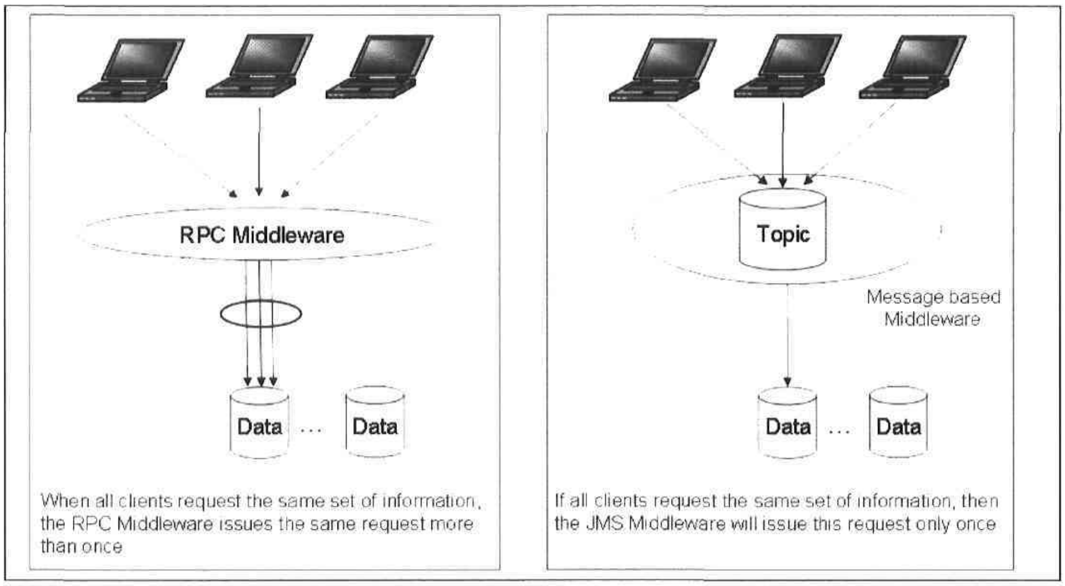
Figure 3.5: Retrieval of information for Single Client Application

Scenario 2: Common Requests Processing

A multi-tier middleware solution allows a number of client-applications to invoke requests to the resource layer. If there exists **X** requests that are using an RPC-based middleware, the server will repeat the Scenario 1 for each of these **X** requests irrespective of the nature of the requests. Now, at the point of implementation, it is obvious that a group of clients issues identical requests in nearly same duration as they have almost similar interfaces in front of them.
The above-mentioned Message Oriented Middleware (MOM) system stores the information temporarily through JMS-connection components: queue and topic, that can be reused to provide client support if implemented in an optimized way. The notion leverages significant improvements over performance as a whole as the MOM saves the duration of information retrieval from the sources. Mathematically, if there exists **g** number of groups, each having **r** requests on average. The total number of requests issued by the middleware to the resource tier is reduced from X in case of an RPC-based middleware to:

$$No.\ of\ Requests = g + (X - (g \times r)) \tag{3.4}$$

in case of a MOM. This scenario is shown in (Figure [illegible]).



When all clients request the same set of information, the RPC Middleware issues the same request more than once

If all clients request the same set of information, then the JMS Middleware will issue this request only once

Figure 3.6: Multiple Clients Requesting the Middleware

Scenario 3: Real-Time Changes Notification

The proposed approach leverages notification mechanism, a significant feature of asynchronous messaging using connection components.
If a remote user subscribes to a set of information, the proposed solution offers notification to a remote client application with the changes of dynamic information. The RPC model has no scope without repetitive polling of the requests to realize the changes. It not only requires more time to submit a request and process it through each layer of the mutli-tier but also creates huge network load unnecessarily. Figure 3.7 illustrates this scenario.

3.7 Scope and Limitations

The architecture of the proposed middleware is sound and powerful; however, implementation utilizing available technologies faces significant challenges.
Designing a system affiliated with JMS requires a JMS provider, i.e. JNDI server which manages and contains all topics and queues. As in Figure [illegible], the subscriber module resides on the client side. Since it is impractical to install a JMS provider (server) on each client machine, we resolved this issue of posting JMS request within a Java Bean and sending it over HTTP. However it is possible in JMS to listen to the remote

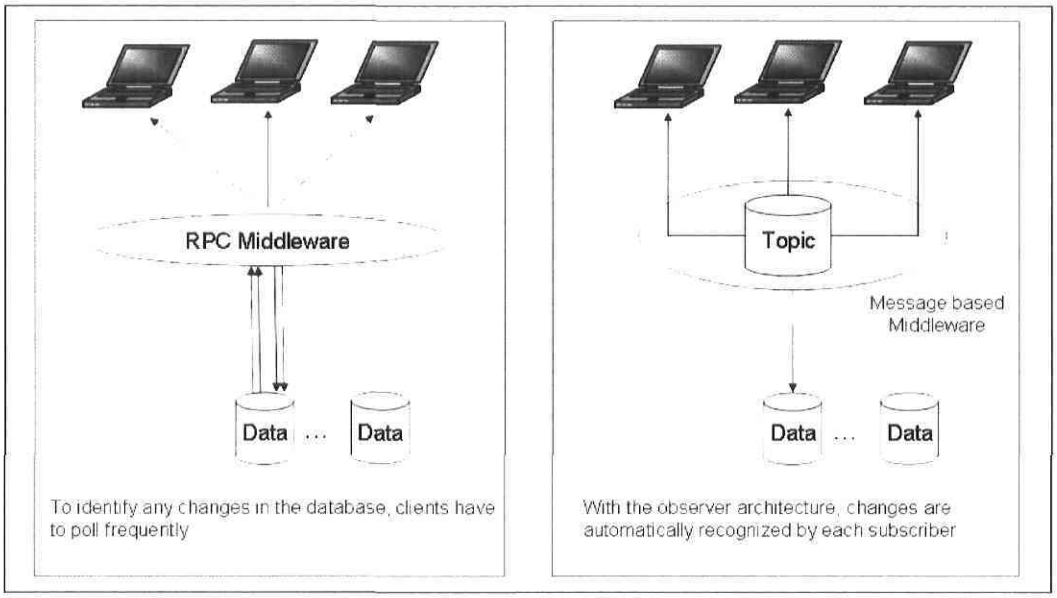

RPC Middleware
Data
...
Data
To identify any changes in the database, clients have to poll frequently
Topic
Message based Middleware
Data
...
Data
With the observer architecture, changes are automatically recognized by each subscriber

Figure 3.7: Notification Mechanism: Comparison

listeners from the client's end.
Furthermore, the approach can be extended to a web based application running within a browser. The "Digital Cockpit" system can then behave as a Java applet executed within the JVM of the browser. However, various issues should be considered if this approach is to be implemented. Applets introduce a big load on the network. First of all, it requires a significant compromise with the heavy features of the system in order to make it portable. Furthermore, if the applet requires listening to a JMS-listener inside the client's system, modifications to the Java policy files may be needed. Last but not least, it should be mentioned that applets should also be configured to accept PKI supported certificates before loading.
It should be noted that our implementation does not tackle the issue of service integration precisely, although it is possible. With the presented architecture, we can encapsulate a SOAP message/request on top of JMS which in turn will trigger the execution of the service. There is no doubt about this, since JMS allows SOAP requests over it.
Finally, another problem that we faced is related to serialization. JMS only accepts serializable objects to be encapsulated within its message. Unfortunately, not all objects are serializable, for example: ResultSet object from Java. As a result, we had to do some kind of transformation which in turn leads to loosing some attributes found in the initial unserialized object.

Chapter 4

Digital Cockpit Implementation

4.1 Introduction

In this chapter, we highlight the implementation of a our proposed middlware called Digital Cockpit. Digital cockpit is a message oriented middleware that integrates in an asynchronous way data and services from different sources and present them as a big picture that allows drilling into details. First, we lighten the software requirements. Second, we provide the domain model related to this project. After, we detail the software design phase, highlighting the approach, architecture and some of the most important class diagrams. Also, within this chapter, we leverage some snapshots of digital cockpit implementation.

4.2 Software Requirements

This section "Software Requirements" demonstrates the concerns and specification of the digital cockpit application corresponding to the needs of the organization customer. Actually, these software requirements represent the result of comprehensive gathering of customer inquiries and profound analysis that identify the numerous functionalities and aspects of the digital cockpit. To achieve this, we followed a Six Sigma approach to analyze them into Quality Function Deployment [39] as shown in Figure . Afterwards, we identified the domain model, usecase models, and sequence diagram.

The intention behind the digital cockpit we developed to our customer is to manage all aspects of real time monitoring and display of various kinds of data from diverse remote information sources. By hosting this digital cockpit on the customers systems, the user will be able to access heterogeneous databases seamlessly, monitor data in a real time mode and perform various data analysis and some simulations. As a result, better decisions could be made that would yield great results. The main features of the digital cockpit are the following: Integration of data sources, Subscription to data, Display, Analysis, Monitor, and Control. Each one of these features corresponds to a phase in the project development.

4.2.1 House of Quality

Software requirements gathering aims to get the functionalities and aspects of the digital cockpit. To this end, we followed an approach that is particularly fitting for the Six-Sigma house of quality [39]. Using this tools, customer requirements and technical parameters are put together, in order to determine interrelations between the customer and technical requirements, as well as between technical requirements.

The house of quality is presented in Figure . We identified the following customer requirements and technical requirements.

The customer requirements set itself include technical criteria and business criteria. The technical criteria list consists of: Performance, security, availability, functionality, usability, modifiability, portability, inter-operability, integrability, and reusability. The business criteria we considered in our project are: documentation, cost, project life-time, and training. Every quality attribute among the previous requirements, either is technical or business possesses a weigh that reflects its importance for the customer.

On the other hand, the technical requirements represent the concerns of the technical team. In our case, they are classified into three classes: standard, design, and complexity.

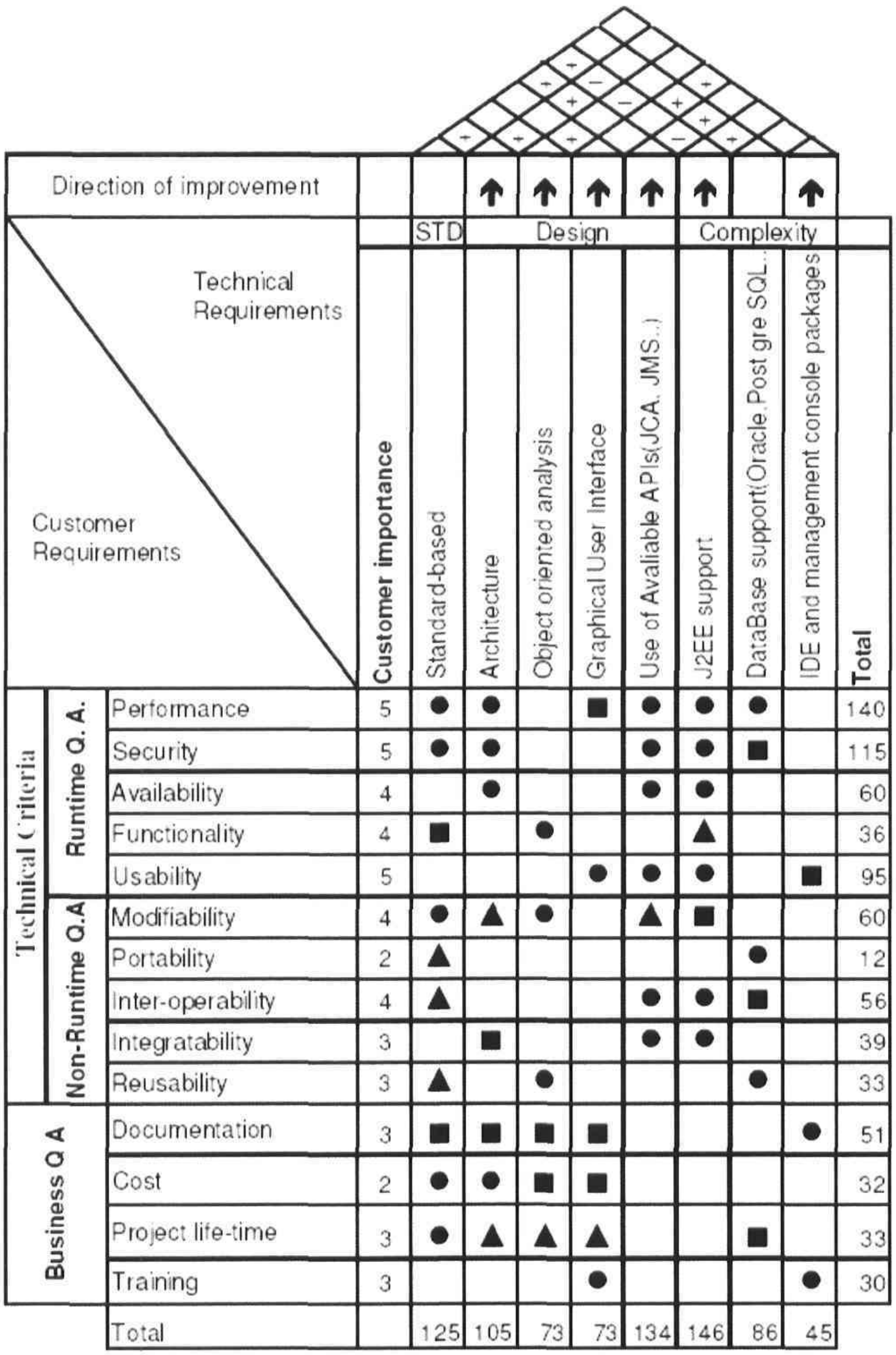
Figure summarize the technical requirements, customer requirements, and defines the interrelations between them. It also shows the importance of each quality attribute from customer requirements and how it affects each criterion in the technical requirements side. The application of such an exercise provided the results shown in Figure . The requirements are therefore classified according to their importance based on a well proven and established approach, from quality systems engineering [39].

Key to interrelation matrix symbol

● Strong interrelationship (value = 5)

■ Medium interrelationship (value = 3)

▲ Weak interrelationship (value = 1)



		Direction of improvement			↑	↑	↑	↑	↑		↑
				STD	Design				Complexity		
		Technical Requirements / Customer Requirements	Customer importance	Standard-based	Architecture	Object oriented analysis	Graphical User Interface	Use of Avaliable APIs(JCA, JMS..)	J2EE support	DataBase support(Oracle,Post gre SQL..	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 4.1: House of Quality: Digital Cockpit Project

4.2.2 Domain Model

Domain model represents functionalities of the system through distinguished packages. Initially, the domain model is broken down into seven components: integration, display, analysis, monitor, control, security, and subscription.

The integration module used to provide a uniform access to local and remote data as needed by the rest of the system. It is in charge of maintaining the local repository up-to-date with the remote databases and informing the display module of new updates that need to be signaled to the user. These modules together form the digital cockpit system providing all the features that complete the specified needs. The display module is used for all visualizations within the system and provides the ability to export current data or import past data. For the analysis module is used to analyze real time data to find trends, do What-if scenarios, cause and effect and even probabilistic operations to compare it to past data. The Monitor module is used to help analyze current usage of the system broken down into two (02) categories: by hook and by user. It also allows for the analysis of previous event logs broken down into three categories: by hook, by user and by time span. The Control module is used to optimize then compare and contrast scenarios to find the best outcome. For the Security module is used for logon/log off of the users as well as user authentication and authorization. This package also handles all the data logging and access restrictions. Lastly, the Subscription module is used for all subscription management events. It will handle the subscription to and un-subscription from data and services.

These modules together form the digital cockpit system providing all the features that complete the specified needs. Figure [illegible] represents inter-relationship among packages.

4.2.3 Use Case Model

The system consists of different use cases manageable by the users of the digital cockpit system. We differentiate between three types of system users. The first type of users is the normal users who use the system as per their privileges, analyze and control different views generated from the available data and services of his/ her domain. Users in the second category called "Power Users", a part the privileges of normal users they have, these users are able to track the operations of various users as per the organization requirements and their own privileges. Finally there is an administrator local to the system who is capable to exploit all the facilities of the digital cockpit including the

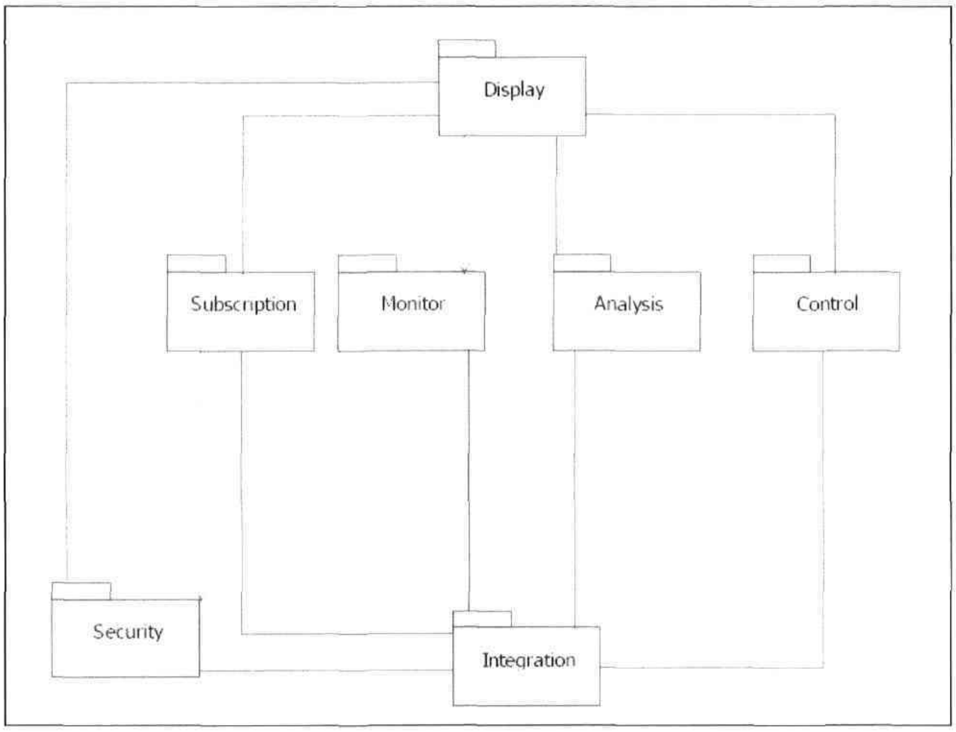


Figure 4.2: Domain Model: Digital Cockpit Project

creation of connection to a new set of data and service sources. Figure represents the overall use case model that illustrates the capability of the system from the users perspective.

After a clear description of the domain and use case model, we focus on the functionalities of each module through initial use cases. The list of all considered use cases is provided in the appendix.

Figure is an example of a sequence diagram that shows the interactions among digital cockpit components. It corresponds to initialization of display components in the digital cockpit after verification and confirmation of the user privileges and subscription to the services. The system establishes a session with each service/data provider and starts interacting with it. Afterwards, the system starts real-time monitoring and performs business logic operations on the retrieved data. Finally, the system presents information corresponding to a service through its component on the digital cockpit.

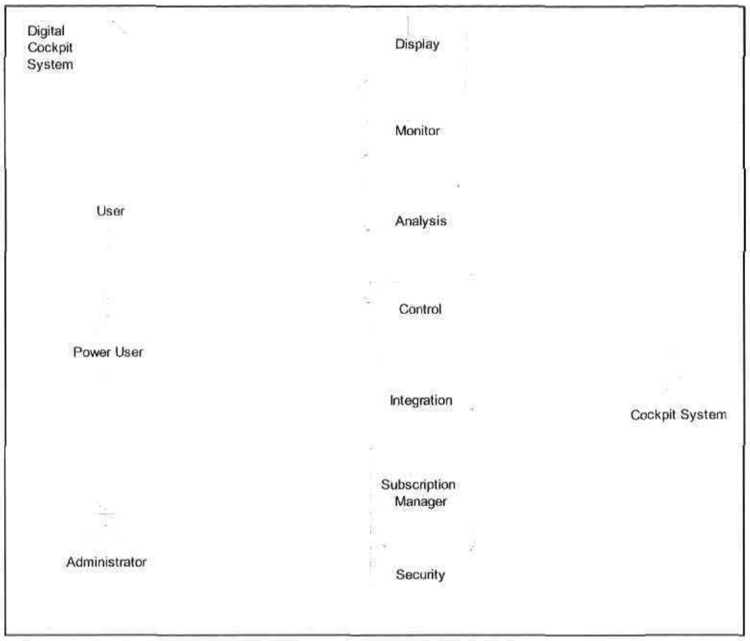


Figure 4.3: Use Case Model: Digital Cockpit Project

4.3 Software design

The software design specification demonstrates the concerns and specification of the digital cockpit software solution corresponding to the customer needs. In what follows, we present the design approach assumptions, the followed methodology, some of the most important class diagrams, and the software implementation illustrated by some screen snapshots of the achieved product.

4.3.1 Assumptions and Policies

The proposed middleware represents a distributed structure on the top of existing client-server architectures as available within intra-departmental networks. The idea is adopted to favor the inter-departmental information sharing for Canadian Air Force Command and Control system considering their existing intra departmental client server architecture of the data-access. We limit our scope according to the clients requirements such as:

- The software is proposed assuming the existence of many kinds of different data

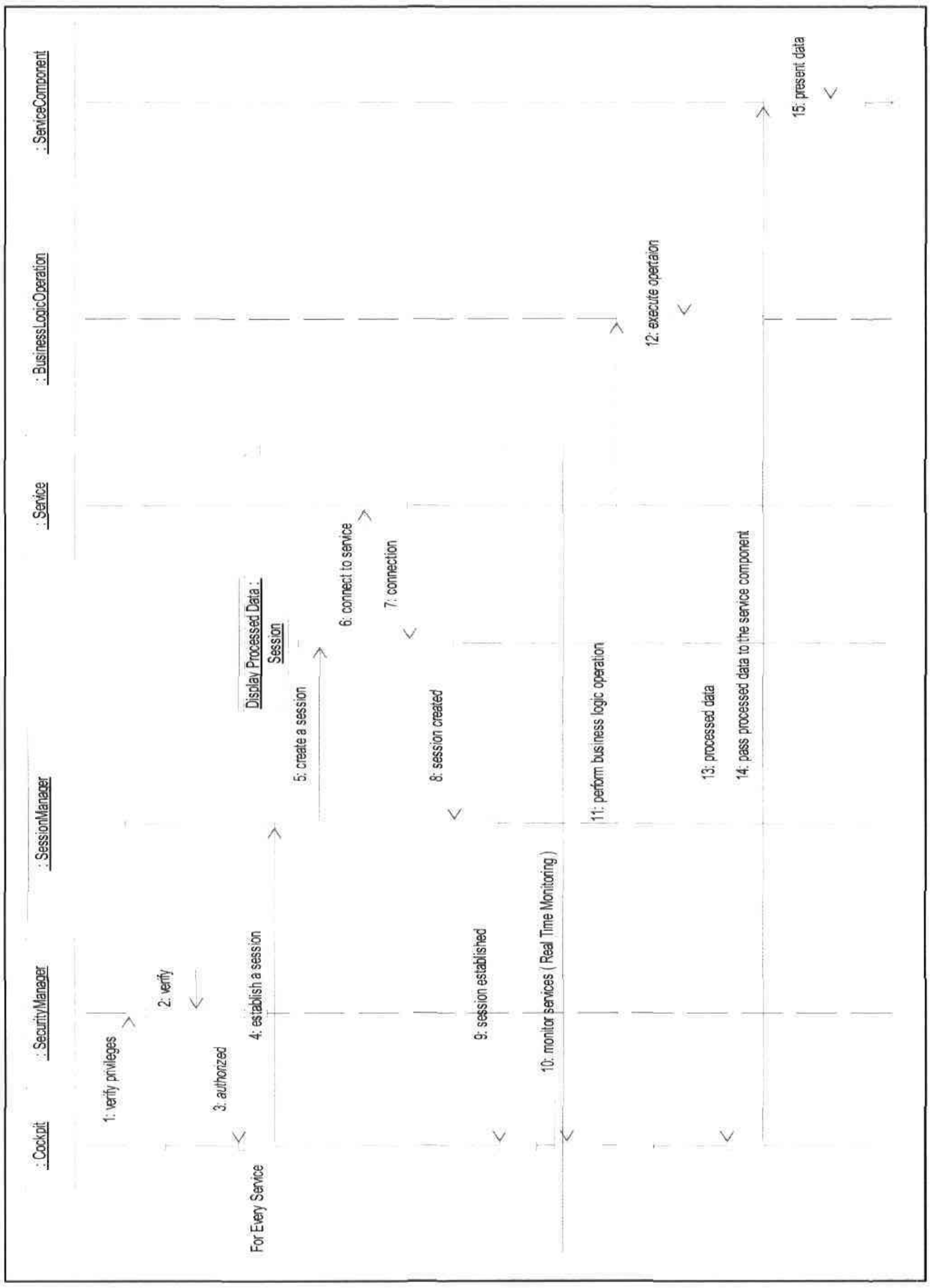


Figure 4.4: Sequence Diagram of Display Module: Digital Cockpit Project

sources, kept through Oracle and Sybase database, MsExcel spread sheets, and other specific applications. Service integration is implemented by integrating a commercially available free weather service information kept in DWML format.

- According to the requirements, Digital Cockpit prototype supports mathematical analysis such as statistical operations, time-based analysis and simulations.
- The design and implementation is provided considering the known source of information. Dynamic integration of data sources and services are beyond the scope of implementation.
- The digital cockpit user interface is quite different from the traditional graphical user interface. It shows most of the required information from the single screen as well as allows user to navigate in to the details by clicking on a particular component. It communicates and represents the changes of information under current inspection, without any user intervention.
- We propose to run an application behind simultaneously with the browser-based user interface. The intent of such proposal is to overcome the limitations of the applet within the browsers JVM. This Java application can better handle the retrieval of information as well as saving the files within the system.
- The design of the system aims the implementation using the standard based APIs that are proven in terms of technology, performance and other software attributes.

The intention behind digital cockpit paradigm is to manage all aspects of real time monitoring and the display of various kinds of data from multiple heterogenous information sources that enables performing various data analysis and simulations which provides better decision making.

4.3.2 Methodology

The choice of a proper middleware may significantly speed up the retrieval of data or services, reduce integration complexities and increase the performance as well. Architecturally, the proposed middleware represents a distributed structure on the top of existing client-server architectures as available within intra-departmental networks. The idea is adopted to favor the inter-departmental information sharing since almost all the organizations have the same structure: many departments communicating between them.

This section focuses on the high level methodology of Digital Cockpit project, that realizes a five phase paradigm as follows:

- Integration: to connect all the information and service sources within and across the organization, for an information sharing purpose.
- 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 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: 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: to optimize procedures, events and scenarios that will enhance the used processes, methods and strategies.

Security, is not an exact phase of this paradigm, however considered as a vital quality attribute that must be taken care of at each phase of this paradigm. Figure illustrades the ideology more clearly.

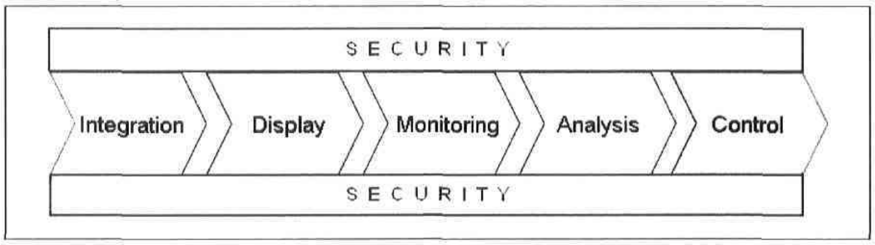


Figure 4.5: Digital Cockpit 5-Phases Paradigm

The advantages of this five phase paradigm is mentioned as follows:

4.3.3 Important Class Diagrams

Digital cockpit system allows users to get the desired information in a graphical and well presented way. It addresses different concerns as we have already seen in the

Cost-effective	The proposed solution is mostly based on free technological components.
Secure	The proposed solution is designed with security in mind.
Standards-Based	The proposed solution is based on published, well accepted and widely adopted standards.
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 very efficient since it uses performance proven technological components.
Proven	The proposed solution is proven since it has already been widely deployed in major, leading, large cap corporations.

Table 4.1: Advantages of Digital Cockpit Solution

methodology: integration, display, monitor, analyze, and control. Achieving all these functionalities, we classified them into several packages. Figure B.3 illustrates the display class diagram package. In the appendix associated to this thesis, we also provide some other class diagrams.

4.4 Implementation

The conceptual model of the above mentioned research is partly validated through a software implementation called "Digital Cockpit". Digital cockpit offers users an unprecedented level of information hierarchy through a layer-based approach while it allows access to real-time information by readily accepting the desired changes in the remote information and service sources. Moreover, it encourages users to navigate to desired information by clicking on respective components. The digital cockpit addresses two major concerns of service integration. First, it achieves a synergistic integration of the various service sources in an asynchronous loosely-coupled architecture. Second, the digital cockpit platform allows a notification mechanism from the integrated services that brings the effect of changes in service information directly to the client machine.

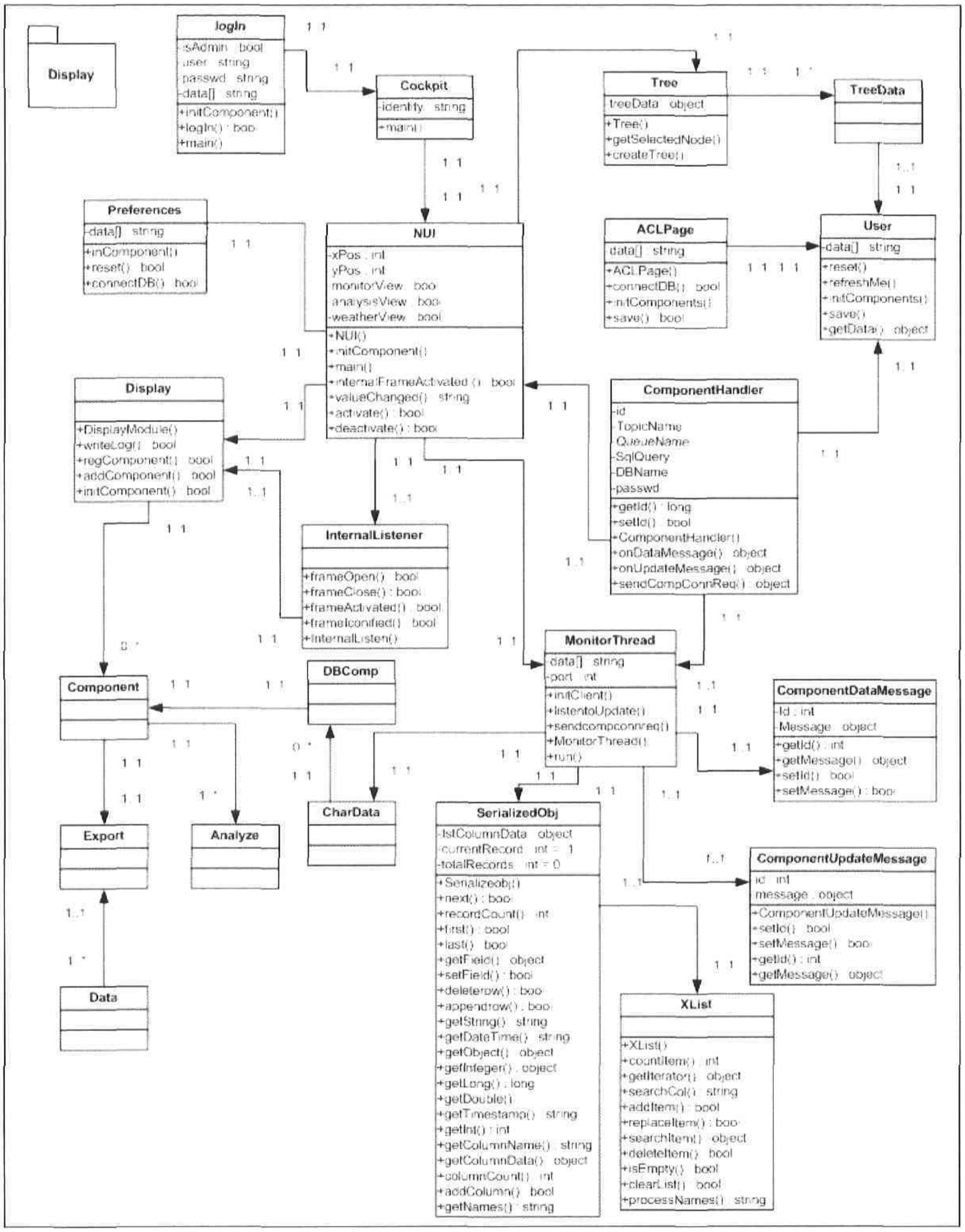


Figure 4.6: Display Class Diagram

4.4.1 Technology

Middleware based integration is already achieved using Remote Procedure Calls in a synchronous architecture. The digital cockpit proposes a middleware paradigm that implements notifications on top of the asynchronous Java message service. As of the implementation procedure, the choice of technology is particularly crucial for a decision system that will constantly evolve with ever-changing organizational business rules.

Display and Monitoring

Display in digital cockpit was achieved through a set of Java based frameworks and APIs. First, JFreeChart, JFreeReport (from JFreeChart.org) and their multiple packages were the most used technologies in the display and analysis modules. Second, Espresschart (from Quadbase Inc.) was mainly used to achieve simulation of scenarios.

Framework

Local communication between classes is achieved by interface contracts, and wiring is achieved thanks to an Inversion Of Control (IoC) [10] container called Spring Framework [4]. IoC is an emerging paradigm that greatly improves the reusability, flexibility, maintainability and unit-testability of components. The localization and instantiation of classes instances is transparently achieved through the Singleton and Factory Design Patterns [47] while removing the need for the developer to maintain these Locator classes. This allows for full decoupling of 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, Spring Framework provides helper classes that eases J2EE development. Furthermore, Spring integrates nicely with Java Connector Architecture (JCA), Java DataBase Connectivity (JDBC) and other persistence frameworks to provide declarative local or distributed transaction demarcation.

Integration Container

The Digital cockpit leverages a standard integration container standardized as Java Business Integration (JSR 208) [53]. The JBI container accomplishes the access of service in a transport independent manner.Therefore it takes full advantage of the asynchronous features built into BPEL4WS, compared to a classical SOAP over HTTP approach. The JBI standard being relatively new, there are still few Open Source JBI implementations available, mostly CodeHaus ServiceMix [2]and Sun's Reference Implementation. When implementation was started, Mule Enterprise Service Bus [2] did not support JBI. On the one hand, RI provides a more restrictive license than ServiceMix's Apache license, has few community support and provides a very limited set of Binding Components (BC). On the other hand, ServiceMix provides support for Spring as well as JSR 208 deployment unit, ships with a number of BC, embeds Fivesight's BPEL Process eXecution Engine, and is integrated with other CodeHaus projects such as ActiveMQ citecodehaus05 JMS, Jencks [3] Java Connector Architecture, etc. However, ServiceMix documentation is still in its infancy whereas RI's one is a little more complete. Since BPEL support was needed as well as a JMS BC, ServiceMix has been chosen. Luckily the developers are responsive and provide good support through Mailing Lists and forums.

Business Process

The execution engine runs business processes that are capable of composing multiple services from distant locations. The Business Process Execution Language for Web Services (BPEL4WS) builds on the Web Services standards and provides a high-level language to compose Web Services. This proposed Middleware uses BPEL4WS. There are several competing BPEL engines. The most popular Open Source alternatives include ActiveBPEL [1], Apache Twister [26], Fivesight PXE [84] and IBM BPWS4J [37]. BPWS4J has never been updated and its community support is nearly non-existing. 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 provides no other transport than SOAP over HTTP. As a consequence, ActiveBPEL cant be easily integrated inside a JBI container. Finally, Fivesight PXE has been chosen because of its integration inside ServiceMix. Thanks to its extensible architecture, it can be embedded as a Service Engine (SE) inside a JBI container, and integration is provided with ServiceMix.

JMS Binding Components

ServiceMix provides support to number of binding components (BC) including ActiveMQ JMS, Jencks JCA, etc. Implementation of asynchronous distributed communication is achieved thanks to ActiveMQs JMS implementation, usually considered as the most flexible Open Source JMS broker. Spring framework provides helper classes for JMS templates which avoids programming and maintaining the ever-needed initialization and cleanup code. Furthermore, ServiceMix supports JMS BC. 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 to asynchronously receive JMS messages.

Service Notification

The integration approach encourages the use of a "notification manager" inside the JBI container that asynchronously informs the business process instances of newly available information from the remote services. However, it should be mentioned that this approach assumes known locations of the remote services. An extension of this approach could be realized to find services dynamically by integrating with an outside XML registry, however not discussed within the scope of this thesis.

4.4.2 User Interface

This subsection contains various interfaces related to the digital cockpit system. All these interfaces are not solely related to user interface however discusses important internal interfaces between various modules of the system.

Figure [illegible] is a snapshot from the digital cockpit system. It shows real time monitoring for the entire country. Moreover, the map is shown in a zoom in mode, which leverages a better information view of incidents what is really happening.

Figure [illegible] illustrates analysis capabilities of digital cockpit system. It shows an analysis component in a zoom in mode and beneath it there exist numerous knobs. These knobs when moved can change the data values in the corresponding components. This is very useful tool to help in showing the result of some actions and can thus prevent unwanted events.

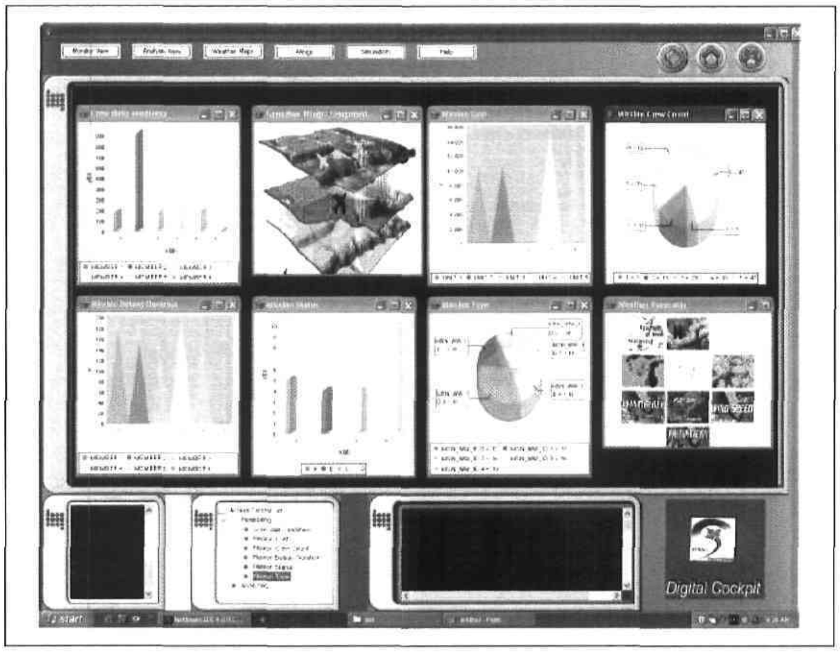


Figure 4.7: Display User Interface

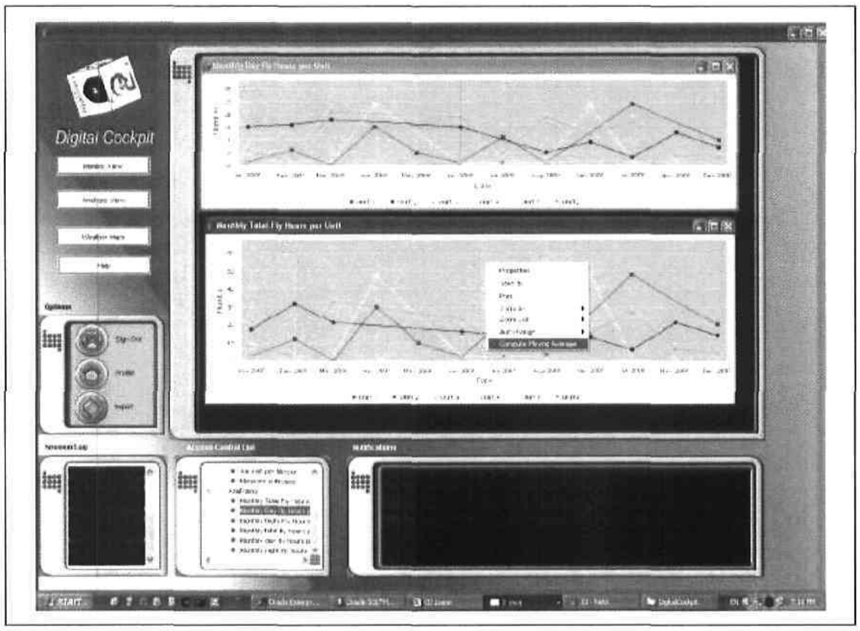


Figure 4.8: Digital Cockpit Analysis Example

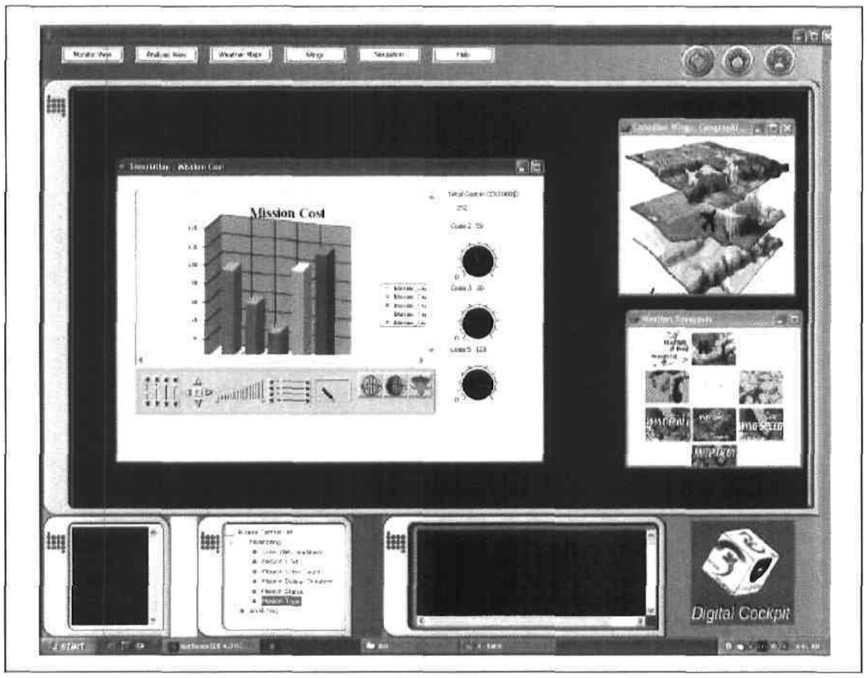


Figure 4.9: Digital Cockpit Simulation Example

Figure provides a visualization of the optimization capabilities and explores the data dependencies among more than one component. This yields the ability to clearly understand how the change of one component can affect another. It also uses some optimization technique to optimize the best possible scenario.

The focus on service integration is mainly based on specific requirements of the customer. In this project, we implemented an infrastructure for 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). This service receives changes in remote services data thanks to internal schedulers and notifies the client though asynchronous JMS notifications. It re-calculates the necessary part of the relevant business process if required and informs the user interface that instantaneously reflects the changes. Finally, the Java3D and JFreeChart libraries have been leveraged

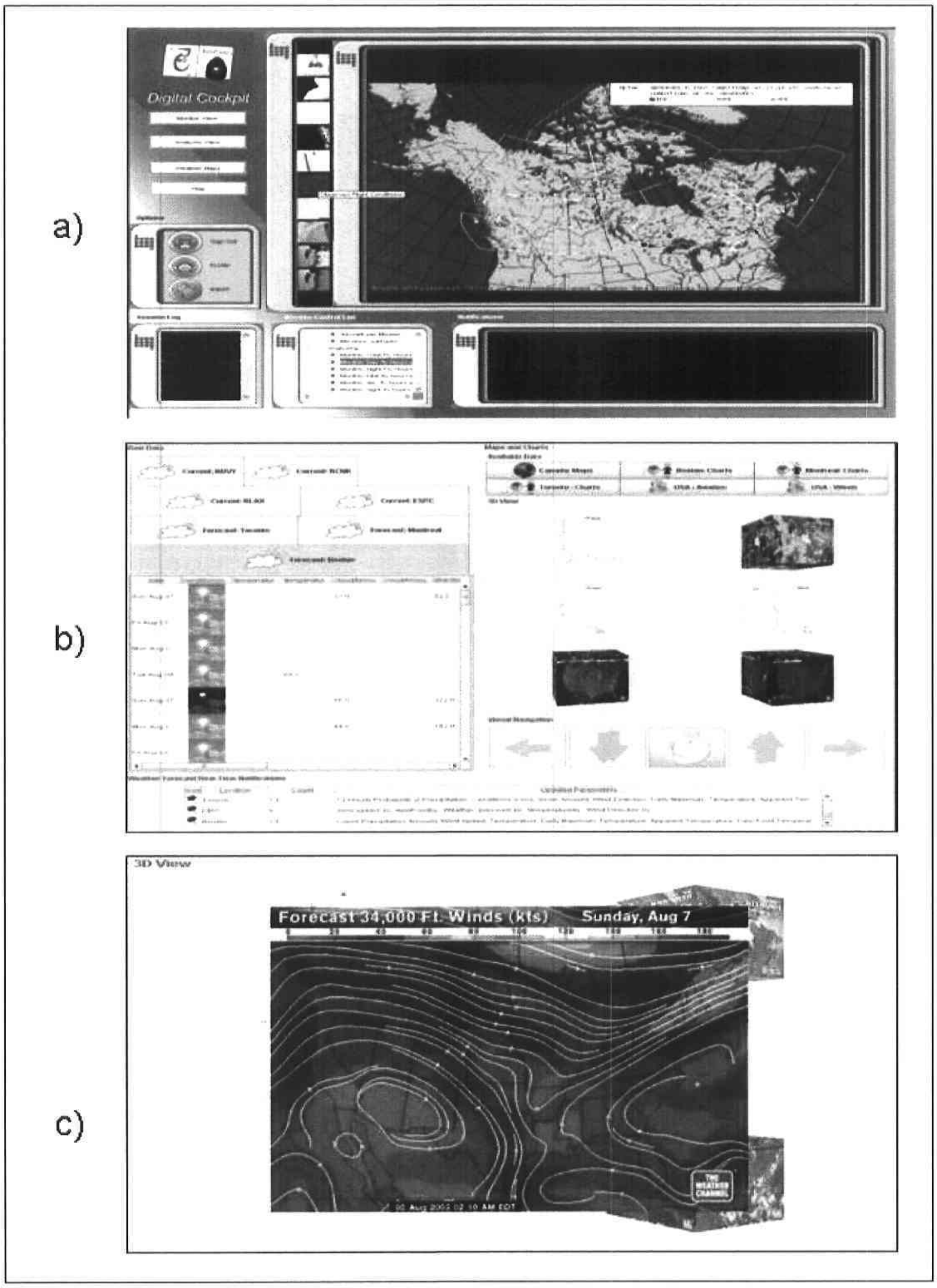


Figure 4.10: a)The overall view of Weather Scenario b)Interested user drills down in detailed information c) A specific weather component: Wind forecast

for visualizing weather information in a user-friendly way. Java3D provides a high-level programming interface for rendering three dimensional scenes whereas JFreeChart is the most widely used Open Source charting library. JFreeChart can generates graphics that are used as textures by Java3D.

The figure [illegible] represents a "drill down" scenario through the digital cockpit implementation. A successful integration of weather service in figure [illegible]a shows an overall weather map representing certain information. An interested user may focus on specific region of interest and request more information as shown in figure [illegible]b. Sufficient service integration successfully enables the platform to represent bulk of information from the remote service. Finally figure [illegible]c shows a specific wind condition, representing the lowest layer of drilled down information. It should be mentioned that all this representation is real-time and changes with the fluctuations at remote end.

4.5 Conclusion

In this chapter, we highlighted the implementation phase of digital cockpit project. Actually, it was a challenging phase in this project. We had to decide among a plethora of technologies. We have been faced to multiple heterogeneous data and service sources such as Oracle, Sybase, MsAccess DBMS, customer legacy systems, Ms Excel Spreadsheet, jWeather service, etc. We have also been faced to the choice of the right technology to implement different concepts such as integration layer (choice of application server and service execution language), display and monitoring (Spring lightweight container, Inversion of Control, JFreeChart, Espresschart Quadbase, Java3D, etc.), and analysis and control (JFREEChart data analysis library, Jakarta POI, etc.).

Chapter 5

Conclusion

Within this thesis, we reveal the enormous scope lying in the cohesion of two parallel research areas: information systems integration and decision support systems. We provided a new approach for integrating information systems within and outside the organizational boundaries. We also illustrated an idea of a decision support system for taking consolidated, quick and correct decision on the available information and services using business intelligence. As a proof of concept, we have presented an architecture of our digital cockpit model. This thesis depicts an overview of the widespread notion of our continuing research work in order to relate the aforementioned research areas into a new paradigm.

Regarding the future work related to this research, many enhancements and researches can be done especially how to inject some "intelligent" decision support to the aforementioned middleware platform. Game theory is an interesting track that can add big value to this software platform. A more elaborated web services component can also enhance the digital cockpit productivity enabling the use of available services and their composition from multiple services providers.

Finally, because of the distributed nature of the middleware, security is another issue that accomplishes this work. Therefore, elaborating a global security policy for the entire platform, and local mechanisms for each phase will better protect the information and consequently having a secured digital cockpit. Given the pace of evolution, we expect that this solution will be an inseparable part of the present business needs for real-time information integration and decision making.

Appendix A

Digital Cockpit Use Cases List

- **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 Module:**
  - Use Case: Optimize data.

- **Monitor Module:**
  - 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 Module:**
  - 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.
- Monitor Module.
- Display Module.

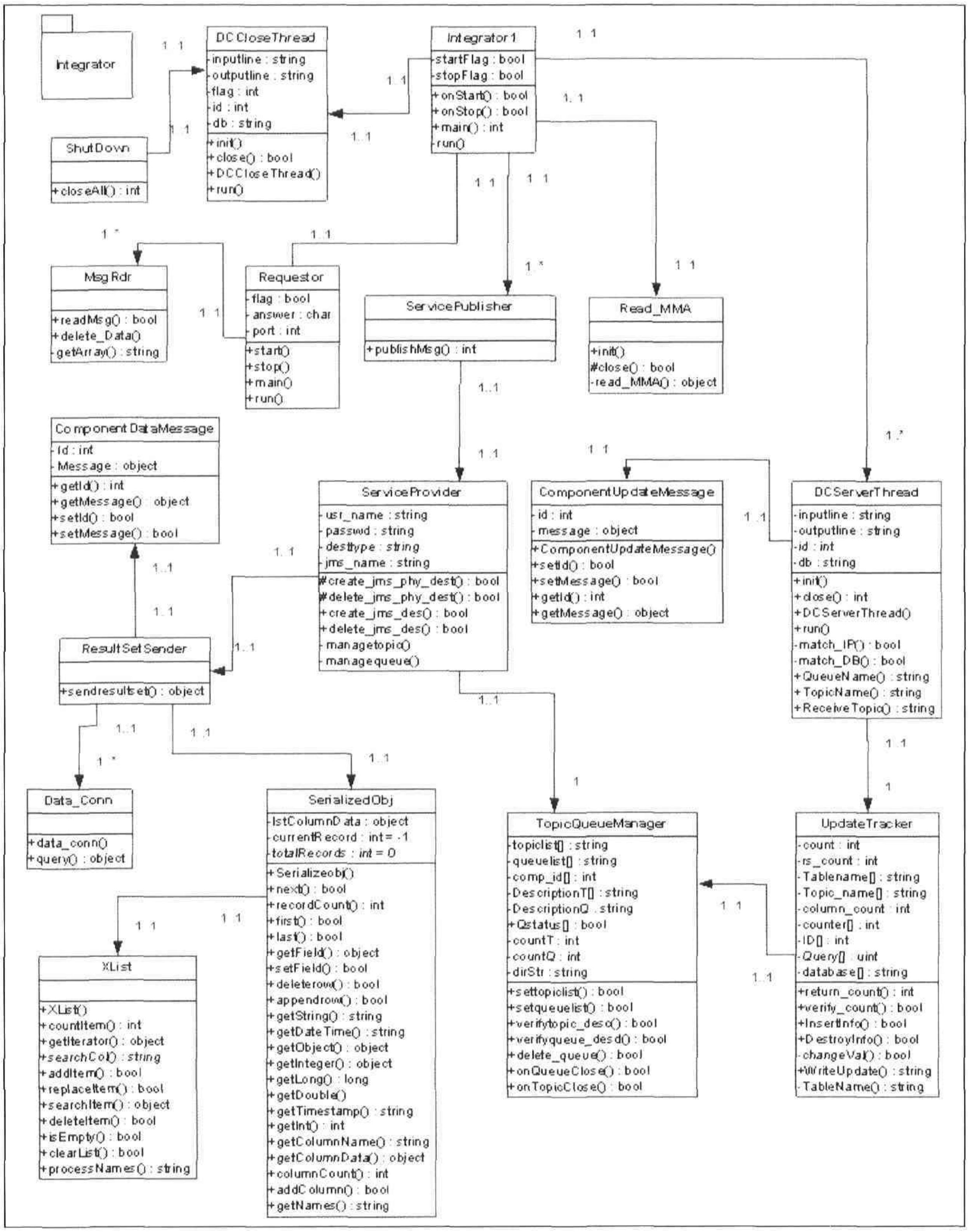


Figure B.1: Class Diagram: Integrator Module

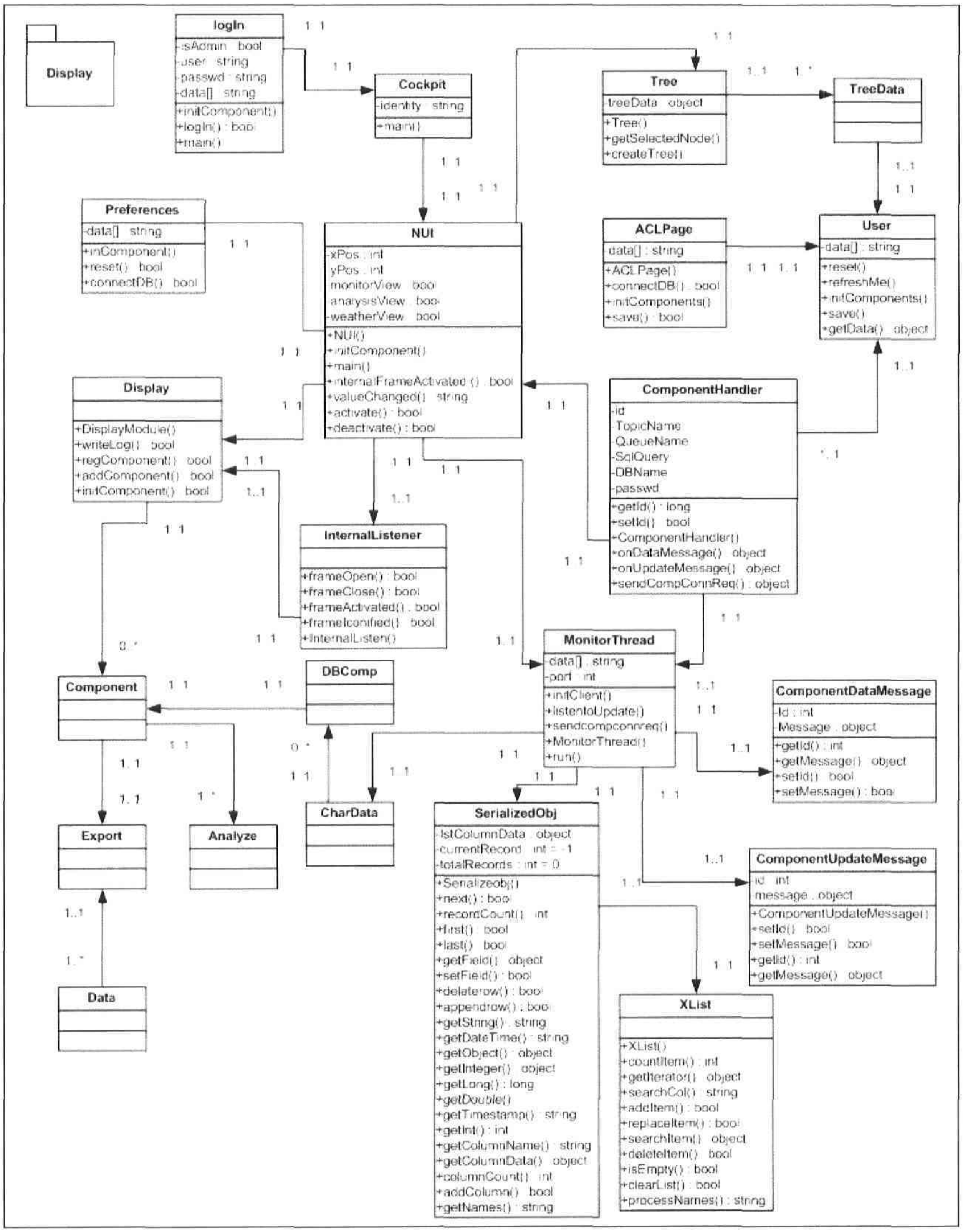


Figure B.2: Class Diagram: Display Module

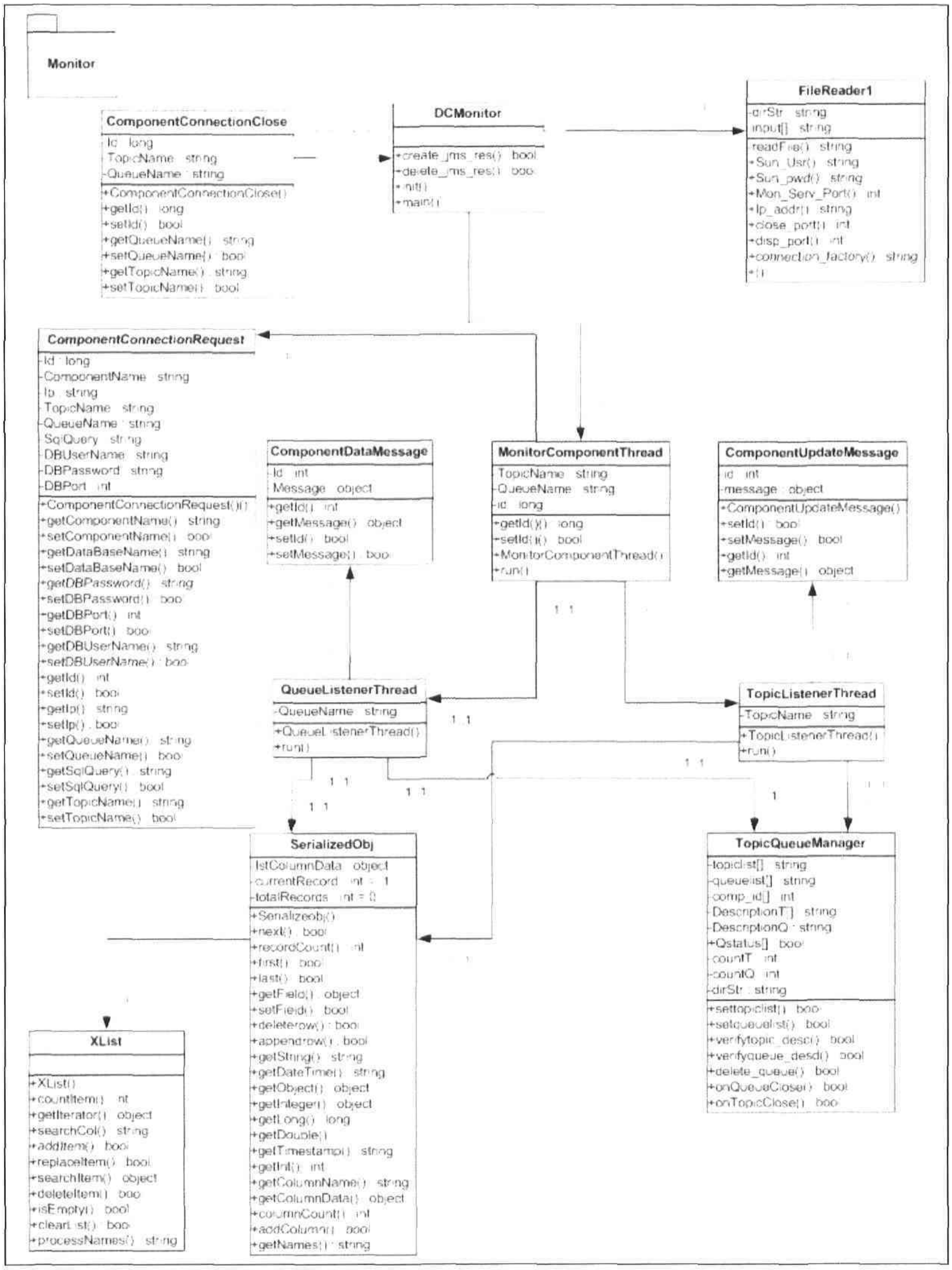


Figure B.3: Class Diagram: Monitor Module

Appendix C

Challenges and Technology Choice

Component	**Challenge**	**Technology**
Information and Service Sources	• Heterogeneity of sources. • Geographically in different locations. • No integrated platform.	• Data Integration. – Oracle database. – Sybase database. – Legacy System. • Service Integration. – WSDL description. – XML schema. ∗ Digital Weather Markup Language (DWML). ∗ METeorological Aerodrome Report (METAR) data.

Table C.1: Technology Stack of Digital Cockpit Implementation

Component	**Challenge**	**Technology**
Integration Phase	• Choice of application server. • Asynchronous communication.	• Application Server. – Sun J2EE. – Apache Gerenimo. • Data integration – JDBC API from Sun Microsystems. – JCA API from Sun Microsystems. – JMS API from Sun Microsystems. – ActiveMQ JMS.
Integration Phase		• Service Integration. – BPEL4WS implementation: Fivesight PXE. – JBI implementation for Integration Container. – ServiceMix implementation from Codehaus.

Table C.2: Technology Stack of Digital Cockpit Implementation

Component	**Challenge**	**Technology**
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. • Espresschart (Quadbase Inc.) • Java3D. • JMS API: Msg. communication.
Analysis and Control	• Understanding Scenarios. • Suitable analysis and optimization algorithms.	• JFREEChart data analysis library. • 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 C.3: Technology Stack of Digital Cockpit Implementation

Appendix D

List of Used Acronyms

API: Application Programming Interface.

B2B: Business To Business.

BI: Business Intelligence.

BPI: Business Process Integration.

BPIOAI: Business Process Integration Oriented Application Integration.

BPML: Business Process Modeling Language.

CIO: Chief Information Officer.

CORBA: Common Object Request Broker Architecture.

CRM: Customer Relationship Management.

DB: Data Base.

DBMS: Data Base Management System.

DCOM: Component Object Model.

DSS: Decision Support Systems.

DTD: Document Type Definition.

EAI: Enterprise Application Integration.

e-business: Electronic Business.

ebXML: electronic business Extensible Markup Language.

e-commerce: Electronic Commerce.

EDI: Electronic Data Interchange.

EIS: Enterprise Information System.

EIS: Enterprise Information Systems.

ERP: Enterprise Resource Planning.

ETL: Extract, Transform and Load.

GUI: Graphical Unit Interface HTML: Hyper Text Markup Language.

HTTP: Hyper Text Transfer Protocol.

IBM: International Business Machine.

IDE: Integrated Development Environment.

IIOP: Internet Inter-ORB Protocol.

IOAI: Information Oriented Application Integration.

IT: Information Technology.

J2EE: Java 2 Platform, Enterprise Edition.

JCP: Java Community Process.

JDBC: Java Database Connectivity.

JMS: Java Message Service.

JRMP: Java Remote Method Protocol.

JSR: Java Specification Request.

MOM: Messaging Oriented Middlewares.

OASIS: Organization for the Advancement of Structured Information Standards.

ODBC: Open Database Connectivity.

OLAP: Online Analytical Processing.

OMG: Object Management Group.

ORPC: Object Remote Procedure Call.

P2P: Point To Point.

POAI: Portal Oriented Application Integration.

RDBMS: Relational Database Management System.

RMI: Remote Method Invocation.

RMI: Remote Method Invocation.

RPC: Remote Procedure Call.

SCM: Supply Chain Management.

SOA: Service Oriented Architecture.

SOAP: Simple Object Access Protocol.

SQL: Structured Query Language.

TSIMMIS: The Stanford-IBM Manager of Multiple Information Sources.

UDDI: Universal Description, Discovery, and Integration.

W3C: World Wide Web Consortium WAN: Wide Area Network.

WSDL: Web Service Description Language.

XML: eXtensible Markup Language.

Bibliography

[1] ActiveBPEL 2.0 Roadmap, 2005. http://www.activebpel.org.

[2] Codehaus Project, 2005. http://www.codehaus.org.

[3] Jencks: a lightweight JCA container, 2005. http://jencks.org.

[4] Spring Framework, 2005. http://www.springframework.org.

[5] Bloor Research 2004. Data Integration Connect, Stream and Federate from Attunity, 2004. http://whitepapers.tmcnet.com/detail/RES/1107283074_317.html.

[6] A.Benssam, S.Ray, A.Boukhtouta, F.Guerroumi, C.Assi, and M.Debbabi. A new Paradigm for Information Systems Integration. In *Montreal Conference on eTechnologies (MCETECH'05)*, pages 63–72, Montreal, Canada, January 2005.

[7] A.Boukhtouta, 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.

[8] A.Dickman. Two-Tier Versus Three-Tier Apps. *InformationWeek Magazine*, pages 74–80, November 1995.

[9] A.Halevy. Data integration: A status report. In *Proceedings of the German Database Conference,BTW-03*, 2003.

[10] Apache Foundation. Inversion of Control, 2005. http://excalibur.apache.org/framework/guide-patterns-ioc.html.

[11] C.Davis. Distributed Objects and Components, May 2003. http://www.cs.ucl.ac.uk/staff/W.Emmerich/lectures/3C05-02-03/aswe19-essay.pdf.

[12] Hyperion Solutions Corporation. Managing OLAP and Relational Data with SQL Queries, 2005. http://dev.hyperion.com.

[13] Sonic Software Corporation. JMS Performance Comparison: Publish/Subscribe Messaging. SonicMQ vs. TIBCO Enterprise for JMS, 2003.

[14] Leo Crawford. JCEA Part 1: Messaging, January 2003. http://www.leocrawford.org.uk/work/jcea/part1/messaging.html.

[15] D.Adams. Data visualization: Interactive, multidimentional and animated data representation tools can help improve businesses processes and the bottom line. White paper- ADVIZOR Solutions Inc., 2003. http://www.visualinsights.com/Accenture

[16] D.Gilbert. JFreechart 0.9.20, 2004. http://www.jfree.org/jfreechart.

[17] D.Hudson and J.Johnson. Client-Server Goes Business Critical. *Dennis, MA: The Standish Group International*, 1994.

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

[19] D.Lindorff. General Electric and Real Time. Case study, Ziff Davis Media Inc, 28 East 28th Street New York, NY 10016, November 2002.

[20] Dmreview. Guide to Enterprise Information Integration (EII), August 2004. http://www.dmreview.com/whitepaper/WID1011596.pdf.

[21] Daniel Drasin. Get the message? *IBM developerWorks*, February 2002. http://www-128.ibm.com/developerworks/java/library/i-jms.

[22] E.Cerami. *Essentials of Web Services.* OReilly publications, 2002.

[23] Grace B. F. Training Users of a Decision Support System. Ibm research report, IBM Thomas J. Watson Research Laboratory, May 1976.

[24] Jim Farley. Java enterprise breakthroughs, part 1. *ONJava.com:O'REILLY*, May 2002.

[25] F.Coyle. *Xml, Web Services and the Data Revolution.* Addison-Wesley Information Technology Series: Addison-Wesley Professional, 2002.

[26] Apache Foundation. Twister and Business Process Management, 2005. http://www.smartcomps.org/twister.

[27] Jake Freivald. Integration 101: Understanding Web Services, EAI, and the Acronym Soup. Information Builders Summit 2004 User Conference, May 2004.

[28] M.Ginzberg G.Ariav. DSS Design: A Systemetic View of Decision Support. *Communications of the ACM*, 28(10):1045–1052, October 1985.

[29] G.Evans and J.Riha. Assessing DSS effectiveness using evaluation research methods. *Information Management*, 16(4):197–206, November 1989.

[30] Efrem G.Mallach. *Decision Support and Data Warehouse Systems*. McGraw Higher Education/Irwin, 2000.

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

[32] Crimson Consulting Group. High-performance jms messaging: A benchmark comparison of sun java system message queue and ibm websphere mq, 2003.

[33] Kim Haase. Java message service api tutorial, 2002. http://java.sun.com/products/jms/tutorial.

[34] R. Hackathorn. Convergence: The ethics of business process management. Business Integration Journal, March 2005. http://www.bijonline.com/Article.aspArticleID=1127&DepartmentId=7.

[35] H.Garcia-Molina, J.Hammer, K. Ireland, Y.Papakonstantinou, J.Ullman, and J.Widom. Integrating and Accessing Heterogeneous Information Sources in TSIMMIS. In *In Proceedings of the AAAI Symposium on Information Gathering*, pages 61–64, Stanford, California, 1995.

[36] H.Kreger. Web Services Conceptual Architecture (WSCA 1.0), May 2001. http://www-306.ibm.com/software/solutions/webservices/pdf/WSCA.pdf.

[37] IBM. BPWS4J: A platform for creating and executing BPEL4WS processes, 2005. http://www.alphaworks.ibm.com/tech/bpws4j.

[38] Quadbase Systems Inc. Esspresschart v4.2 API, 2004. http://www.quadbase.com/espresschart.

[39] iSixSigma. Quality Function Deployment (QFD) House of Quality, 2005. http://www.isixsigma.com/tt/qfd.

[40] J.Hanson, P.Nandi, and S.Kumaran. Conversation support for business process integration, September 2002. http://www.research.ibm.com/convsupport/papers/edoc02.pdf.

[41] Lev Kochubeevsky. Generic request response broker for jms. *JAVA DEVELOPER'S JOURNAL*, 10(4), April 2005.

[42] L.Haas, P.Schwarz, P.Kodali, E.Kotlar, J.Rice, and W.Swope. DiscoveryLink: A system for integrated access to life sciences data sources. *IBM Systems Journal*, 40(2):489–511, 2001.

[43] E.Lin L.Haas and M.Roth. Data Integration Through Database Federation. *IBM Systems Journal*, 41(4):578–596, 2002.

[44] MSDN Library. Integrating Layer: Portal Integration, May 2004. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/archprocessintegration.asp.

[45] David S. Linthicum. *Next Generation for Application Integration: from Simple Information to Web Services*. Addison Wesley publications, August 2003.

[46] Brian Maso. Jms: A solution in search of a problem? http://archive.devx.com/java/free/articles/MasoJMS02/Maso02-5.asp.

[47] M.Chitnis, P.Tiwari, and L.Ananthamurthy. Using Design Patterns in UML, 2005. http://www.springframework.org.

[48] M.Hicks. Survey: Biggest Databases Approach 30 Terabytes, November 2003. http://www.eweek.com/article2/0,1759,1377106,00.asp.

[49] Microsoft. Microsoft message queuing (msmq) center. White Paper, 2004.

[50] Microsoft. COM: Component Object Model Technologies. http://www.microsoft.com/com/default.mspx, 2005.

[51] SUN Microsystems. Java^{TM}message service [jsr 914], 2000. http://java.sun.com/products/jms/docs.html.

[52] SUN Microsystems. Javatm message service specification[jsr 914], 2000. http://java.sun.com/products/jms/docs.html.

[53] Sun Microsystems. Java Business Integration (JBI) 1.0, 2005. http://jcp.org.

[54] Visual mining Inc. NetChart Pro V 4.5. Java programmers dynamic graphing library. Product brochure, 2004. http://visualmining.com/documentation/NetChartProDataSheet-20040419.pdf.

[55] M.Mandviwalla, P.Gray, L.Olfman, and J.Satzinger. The Claremont GDSS Support Environment. *Information Science*, pages 600–607, 1991. http://ieeexplore.ieee.org/iel2/882/4724/00184192.pdf?arnumber=184192.

[56] M.Myerson. Web Services Architectures. Technical Teport, Tect., 2002. http://www.webservicesarchitect.com/content/articles/webservicesarchitectures.pdf.

[57] MODULUS. Modulus technologies. White Paper, 2003.

[58] ENT News. Choosing the right olap client, 1999. http://www.entmag.com/archives/article.asp?EditorialsID=4268.

[59] OASIS. Universal Description Discovery and Integration. http://UDDI.org.

[60] OMGGROUP. CORBA: Standards Based on Specs. http://www.corba.org/standards.htm, 2005.

[61] P.Irassar. WebSphere Business Components and Web services architectures. *DeveloperWorks: IBM's resource for developers*, October 2000.

[62] A. Primer. Business intelligence system scalabilitywhite paper- crystal decisions, 2002. http://support.businessobjects.com/communityCS/TechnicalPapers/cescalability.pdf.

[63] P.Russom. Data warehousing and business intelligence: The right architecture for e-business intelligence, 2000. Hurwitz BalancedView Report.

[64] E. Rahm and H. Hai Do. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull*, 23(4):3–13, 2000. DBLP, http://dblp.uni-trier.de.

[65] Xcelsius Press Release. Infommersion Animates Excel with XCelsius Pro 3.0, 2004. http://www.infommersion.com/pr 20040308.html.

[66] TechMetrixTM Research. Adoption of Web Services & Technology Choices, February 2003.

[67] R.Hull. Managing Semantic Heterogeneity in Databases: A Theoretical Perspective. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 51–61, Tucson, Arizona, May 1997. ACM Press.

[68] R.Hull and G.Zhou. A Framework for Supporting Data Integration Using the Materioalzed and Virtual Approaches. In *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, pages 481–492, Montreal, Quebec, Canada, 1996. ACM Press New York, NY, USA.

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

[70] R.Orfali, D.Harkey, and J.Edwards. *The Essential Distributed Object, Survival Guide*. John Wiley and sons, Inc, September 1995.

[71] R.Schreiber. Middleware Demystified. *Datamation. Cahners Publishing Company*, 41:41–45, September 1995.

[72] S.Alter. *Decision Support Systems: Current Practice and Continuing Challenges.* Pearson Addison Wesley, August 1979.

[73] S.Alter. A Work System View of DSS in its Fourth Decade. In *Proceedings of Americas Conference on Information Systems*, Dallas, TX, August 2002.

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

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

[76] Dirk Slama, Karl Banke, and Dirk Krafzig. *Enterprise SOA: Service-Oriented Architecture Best Practices.* Prentice Hall PTR, 2004.

[77] S.McClure. Oracle's Solution for Heterogeneous Data Integration. IDC: Analyze the Future, August 2003. http://www.oracle.com/technology/products/dataint/pdf/idc_integration_wp.pdf.

[78] Quest Software. Jclass: Java components for j2ee and j2se, 2004. http://www.quest.com/jclass.

[79] SonicMQ, Sonic Software Corporation. Getting Started with SonicMQ circledR V6.1. Documentation, September 2004.

[80] S.Steinke. Middleware Meets the Network. *LAN: The Network Solutions Magazine*, 13(10):56–61, December 1995.

[81] Java Sun. J2EE JDBC Technology, 2005. http://java.sun.com/products/jdbc.

[82] S.Vinoski. Corba: Integrating Diverse Applications within Distributed Heterogeneous Environments. In *IEEE Communications Magazine*, February 1997.

[83] Advanced Visual Systems. Creating a culture of better decisions, 2004. http://http://www.avs.com/software/soft b/openviz/pdfs/openviz_enterprise.pdf.

[84] FiveSight Technologies. Process eXecution Engine (PXE), 2005. http://www.fivesight.com/pxe.shtml.

[85] TIBCO. Tibco rendezvous. White Paper, 2005.

[86] T.Roth, M.Arya, L.Haas, M.Carey, W.Cody, R.Fagin, P.Schwarz, J.Thomas, and E.Wimmers. The Garlic Project. In *SIGMOD Conference*, page 557, Montreal, Quebec, Canada, June 1996.

[87] Liviu Tudor. Msmq: Architecture and simple implementation using vb. Documentation, March 2002. http://www.devarticles.com/c/a/Visual-Basic/MSMQ-Part-1-Architecture-and-Simple-Implementation-Using-VB.

[88] IBM UK. IBM WebSphere MQ Everyplace, Version 2.0. Whitepaper, 2002.

[89] UN-CEFACT. eXtensible Markup Language. http://www.ebxml.org.

[90] S. Venkataraman and T. Zhang. Heterogeneous Database Query Optimization in DB2 Universal DataJoiner. In *In Proc. of the Int. Conf. on Very Large Data Bases (VLDB)*, pages 685–689, New York, USA, August 1998.

[91] W3C. Extensible Markup Language: Xml Activity Statement, 2004. http://www.w3.org/XML.

[92] Webopedia. XML, 2004. http://www.webopedia.com/TERM/X/XML.html.

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

[94] Wikipedia. Cockpit, 2005. http://www.answers.com/cockpit&r=67.

[95] W.Mougayar. Optimize: Spanning The Globe In Real Time, September 2002. http://www.optimizemag.com/article/showArticle.jhtml?articleId=17700772.