NOTE TO USERS

ENTERPRISE APPLICATION INTEGRATION WITH QUALITATIVE QOS CONSIDERATIONS

Hassan Issa

A thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Applied Science

Concordia University

Montréal, Québec, Canada

September 2006

Your file Votre référence
ISBN: 978-0-494-20750-5
Our file Notre référence
ISBN: 978-0-494-20750-5

Abstract

Enterprise Application Integration with Qualitative QoS Considerations

Hassan Issa

The discussion in thesis involves two significant information science topics. These being Enterprise Application Integration along with Quality of Service. The quantity of information is enormous and humans have lost the ability to manage, interpret, and analyze the data. From this perspective, we target the concept of the "Digital Cockpit" which is a data/service enterprise integration built on the very concepts of loosely coupled asynchronous communication, Message Oriented Middleware, and real time integration and monitoring with interactive display. This is then followed by a new Quality of Service model viewing this issue from the perspective of information quality and accuracy as opposed to other existing similar systems. Our contributions are the following: First we present a literature review covering enterprise application integration and web services quality of service. Second we continue the discussion and propose a novel integration middle called "Digital Cockpit" where we highlight significant issues related to the specifications, design and implementation of that system. Moreover, We extend the discussion to include a novel approach concerning quality of service for web services composition. This is achieved by utilizing the new WS-Notification standard along with the use state charts. Fourth, we propose a mathematical framework which specifies and manages the quality from the information accuracy perspective. Therefore, each chapter of this thesis adds its own contribution and goes deeper into the discussion providing more technical and theoretical insights concerning the topic of discussion.

Acknowledgments

I would like to thank God for giving me the will, passion, attitude, energy and persistence that piloted me all along. Great appreciation also flows towards my mother, father, brother and sister, who continuously support and encourage me to peruse high goals and achievements. If anyone deserves a statue, then I would carve a statue having my mom's image because she is truly a great and wonderful mother.

Ample gratitude goes to Dr. Chadi Assi who is an amazing person. He continuously guided me through my studies and showed great support and help. I learned great academic and non academic experiences from him. Thank you for all your time, assistance, guidance and continuous advice. At the same time, Dr. Mourab Debbabi is a great professor who for the past two years taught me a great deal science. Moreover his everlasting excitement to knowledge and attention to details provided me with thirst to excel.

I am an extremely lucky person to have Dr. Jihad Boulos teach me three computer science courses during my undergraduate studies at the American University of Beirut. His words in class were: "I am not teaching you a lesson; I am teaching you a methodology of thinking." Excellent education, hard work, extending one's own boundaries and critical thinking are among the wonderful lessons I learned from him.

Hussien Rammal, is the best mathematics teacher and I will never forget your word: "Determination". One day Miss Hala Saab, my former high school English teacher whom I haven't seen for six years, asked to see me when she learned that I was leaving to Canada to continue my education. I can still hear her words as she said: "Hassan, always remember that we are our parents success." Her words still act like vitamin whenever I remember them. Moreover, special thanks go to Darlene Hnatchuk. You are a wonderful person who is full of energy. Each student's success is your success and I thank you for this.

The CIISE atmosphere has an interesting fine taste filled with a rich blend of wonderful people and experiences. Thank you all for creating this great atmosphere and special thanks to Sujoy Ray.

Contents

List of Figures

Chapter 1

Motivation

Information Systems (IS) have become the cornerstone that shapes the present and future of our economy and society. Information Systems have penetrated almost every aspect of our lives. This omnipresence is stemming from the significant advancements in computer systems, software middleware and applications, telecommunication infrastructure etc. Enterprises and organizations use a large variety of networked computer systems to collect, process and produce large volumes of information. The ability to transform these data into deeper insights may provide the institution with a competitive advantage in mission-critical situations. Decision makers, top management and leaders need robust and efficient automatic and/or systematic tools to sense and respond to real-time changes. However, geographically distant computer systems are disparate in terms of hardware, platforms, operating systems and software applications. Accordingly, there is an existing desideratum that consists in providing software platforms capable of:

1. Providing real time information integration.
2. Keeping databases coherent.
3. Producing analysis reports.
4. Drilling into more information details.

5. Managing security.

6. Providing a genuine graphical user interface.

7. Delivering optimal quality of service.

Materializing these crucial requirements in mind, we came to the production of a complex system for that very purpose. The system is entitled "Digital Cockpit" (DCP). The Digital Cockpit system is capable of achieving a synergistic integration of various information systems. Moreover, the Digital Cockpit displays 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. The Digital Cockpit will also use cutting-edge visualization technologies that will highlight and analyze complex relationships, patterns and trends.

Along with information integration, many aspects should be thoroughly considered. In addition to the depicted DCP, we also target the important issue related to Quality of service (QoS) of web services composition. The use of web services as a means for machine to machine communication is witnessing a significant surge nowadays. One of their benefits is that they can participate in a web services composition process. This is supported by the use of interoperable standards, and continuously new and refined specifications. The present infrastructure for service integration implements a one-time execution of a multi-task business process. Most business process languages compose remote services in different ways and the end-result corresponds to a static instance of information received during the time of retrieval. This technology exhibits a crucial drawback if Decision Support Systems (DSS) is to be implemented on top, since there is no guarantee on the freshness of data. The importance of fresh data can be leveraged in a rescue mission scenario, for example, which heavily depends on accurate data that is not always available. This is true since current technologies associated with web services do not tackle the significant issues of monitoring and re-execution of web services beyond the notion of their integration.

As a result of web services enormous potential, acceptance and utilization among organizations, an end-to-end Quality of Service infrastructure should be established. Various work has been done towards quantitatively evaluating QoS for web services composition, namely dealing with user experience and satisfaction measured in terms of service response time, delay, cost, availability, etc. Surprisingly, this does not encapsulate all QoS issues that might be of significant interest to the end user. To our knowledge, no or little work has been done covering the considerable qualitative QoS issue, i.e. guaranteeing the quality of data measured in terms of accuracy. Accordingly, a significant user dissatisfaction will be incurred if the user identifies that he was treated with inaccurate data along any segment of the composition process.

To overcome this profound problem, we are proposing the usage of the Web service-Notification specification as a base medium capable of sensing and routing any information change at the web service level to registered subscribers using a publish-subscribe mechanism. We then propose an algorithm called Region Switching Algorithm, that is capable of identifying the point of information change within the context of multiple web services composition scenario. This is then followed with an appropriate re-computation of a subset of the pre-established, global service execution plan. By utilizing these two factors (WS-Notification and Regions Switching Algorithm), the system ensures that the end result delivered continuously provides a higher end-to-end QoS based on information quality, and not just on quantifiable metrics. Our contributions to the field of web service QoS are the following: first we highlight the importance of qualifiable QoS aspect related to the issue of web services composition and monitoring. Second we describe an algorithm capable of capturing and reflecting the state of web services involved in the integration process, and finally we illustrate the usage of WS-Notification to aid in building such system.

The structure of this thesis is as follows. We will first illustrate various techniques and methodologies used for information systems integration. This is then followed with a literature review focusing on QoS for web services. We also shed the light on a new specification called WS-Notification that has recently emerged to provide messaging capability for web services. Thereafter, the Digital

Cockpit concept is depicted and details concerning its design and implementation are discussed. Subsequent chapters dive through qualitative web service integration delivering a novel way to guarantee freshness of data within a web service composition context, moreover, this chapter presents a mathematical model which further enhances the quality of service issue. Finally a conclusion chapter is presented followed by appendixes A,B, and C that provide a more detailed look on the implementation of the digital cockpit by providing more insight covering various use cases and class diagrams, and a list of acronyms respectively.

Chapter 2

Related Work: Integration Methodologies, Techniques, Web Service QoS and WS-Notification

2.1 Introduction

We are living in an information era where significant payoffs can be yielded if business companies implement data/service sharing and integration. The main problem that arises is interoperability. Data and services reside on different platforms and each communicates according to its specific communication's protocol. Before the emergence of Service Oriented Architecture (SOA), various integration technologies came to existence. Electronic Data Interchange (EDI) is an example which depends on formatted messages that perform various business transactions among business partners. Later on, with the advance in the communication infrastructure, computing power, and middleware software, various new technologies evolved to provide various capabilities to integrate data and services belonging to the same application but whose components reside on a distributed computing

environment. Common Object Request Broker (CORBA) [S.V97, Sol05] from Object Management Group (OMG), Distributed Component Object Model (DCOM) [Mic05, HK97, Raj98] from Microsoft and Remote Method Invocation (RMI) [Sun05, Spe99] from IBM and Sun Microsystems are the offsprings of the technological advances. Despite these breakthroughs, these technologies still lack the main essence they were created to solve. Interoperability among different system components residing on different platforms is very weak and difficult to achieve. For example, CORBA and DCOM can not communicate unless a bridge is placed between them. Still each of the newly emerged technologies has its own communication protocol. For example, CORBA uses Internet Inter-ORB protocol (IIOP), DCOM utilizes Object Remote Procedure Call (ORPC) protocol and RMI depends on Java Remote Message Protocol (JRMP). Understanding the limitations of existing solutions and to cope with the explosion of the web and e-commerce, SOA came to existence.

2.2 Enterprise Information System

An Enterprise Information System (EIS) is a core requirement for establishing an enterprise because the sole existence of all enterprises relies on their information infrastructure. EIS components correspond to business processes, Information Technology (IT) infrastructure and data working together to deliver various services. An Enterprise Information System provides the information backbone for enterprises. Its main intention is to expose services to its users. Depending on the EIS technology, services can be exposed at various levels of abstraction - business object level, data level, system level, function level, etc.

Enterprise Information Systems can exist in many forms depending on the technology used to implement them. They can be classified into four groups:

1. Legacy databases existing due to many reasons, (company merge for exapmle) host data that is critical to business processes.

2. Transaction programs residing on a main frame transaction processing system.

3. Application belonging to an Enterprise Resource Planning (ERP) set of applications.

4. Custom made enterprise applications developed using various programming languages (COBOL, C, C++, Java, etc).

It is also very common that enterprises host various EISs within their domain. Each EIS would then be dedicated for a certain class of services. Moreover, it is possible that different EISs communicate together via the various techniques that will be illustrated further on.

2.3 Enterprise Application Integration

Enterprise Application Integration (EAI) is the process of integrating applications belonging to the same EIS as well as to multiple EIS in a way that data and business processes/services can be easily and transparently shared (Figure 2.1). It is wise to note that a typical organization has multiple EISs each behaving as a island of information by itself. The power of EAI originates from the use of a predefined set of semantics that guide the process of application/data/service integration. Since there is a well defined methodology for business process(es)/service(s) communication and information integration, adding and removing a component (resource) should not pose any problems. Thus by utilizing standards, the newly integrated system possess an interesting functionality of being safely pluggable. More interestingly, the integration process requires no need to understand the inner workings of involved applications. The required knowledge is to know the set of services provided by each application, and how to interface with them via their Application Programming Interfaces (API). Moreover, with a standard methodology in mind, heterogenous environments no longer form any problem. Since "standard(s)" is the keyword here, we are currently witnessing a rush towards web-driven integration [Jud02]. Nonetheless, since organizations retain huge amounts of data, and the web allows fast and easy access to information, why not combine the two? By exposing services on the web, security considerations come to existence. All access should be conducted in a way where at any time the security policy of the organization remains intact.

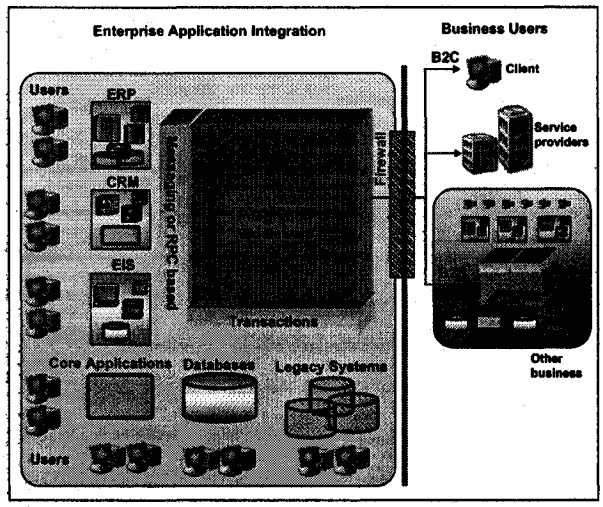

Figure 2.1: Complex EAI structure.

$$\forall t \in T, SEC(processI(t, user), user) \in POLICY$$

During any time t within the time interval T when the system was accessible, the execution of a specific process related to a specific user is guarded by the SEC function. This guarantees that process execution is conducted only to authenticated users whose set of privileges does not violate the organizational security policy.

2.4 Integration Techniques

Various integration tools and techniques have been devised and are widely used. Each of these approaches has its own complexity. To start with, below is a list of some means used for Enterprise Integration.

1. two-tier client/server.

2. Synchronous/Asynchronous adapters.

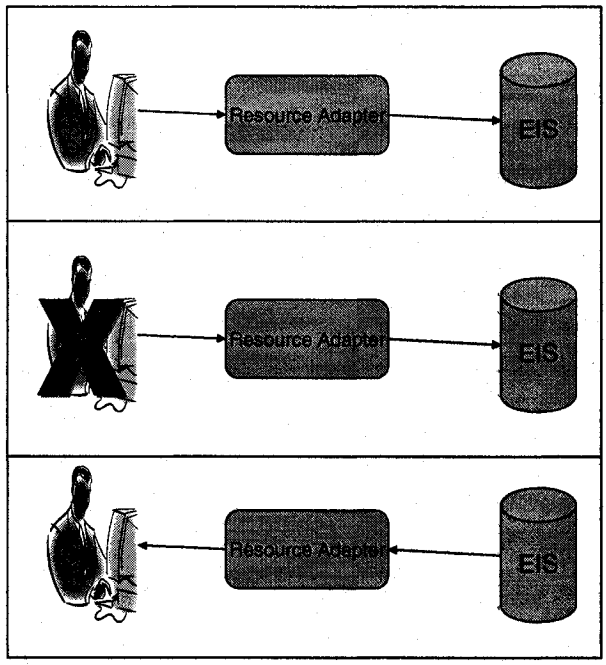

Figure 2.2: Synchronous communication with application blocking.

3. Message based communication.

4. Application Server.

2.4.1 Two-Tier Client/Server [RS01]

This integration approach is being less and less used with the advancement of the web, and web technologies. A client application communicates with the EIS server. Services of the EIS are exposed via an adapter which is an API used by the client to access the EIS. With this approach in mind, the complexity of the system rises exponentially because the developer has to manage transactions, security, scalability, and other life cycle management options since these are not taken care of, or even considered by the EIS. Knowing this, many holes can be left uncovered by the developer leaving the entire EIS susceptible for numerous security breaches and performance degrading.

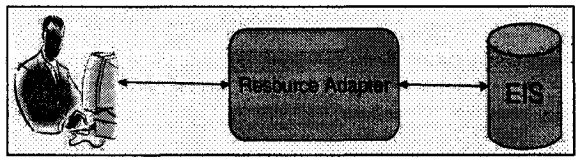

Figure 2.3: Asynchronous communication with application blocking.

2.4.2 Synchronous/Asynchronous Adapters [RS01]

Synchronous adapters provide a synchronous communication API exposing the services of an EIS to the client. Functions in this API block the client's application until a response to an earlier issued request is received by the requesting application (figure 2.2). This kind of design places the entire environment of the user in a blocked state whenever a request is issued. As a result, less work can be done within a given time frame. On the other side of the coin, asynchronous adapters do not operate in a request/reply fashion (figure 2.3). As a result, a client's application never enters into a blocked state. Whenever a request is issued, the application program continues executing regardless whether or not the response was received. With this design in mind, and since applications are rarely blocked, more processing can be accomplished compared to using a synchronous adapter for accessing an EIS.

2.4.3 Message Based Communication

Message based communication is a new paradigm for asynchronous communication between applications. Utilizing this method of communication, interacting components are said to be loosely coupled components which in turn significantly enhances the scalability of the overall system. Since this technique is widely useful within the process of EAI, we illustrate the various means in which message based communication can be accomplished.

1. Queue message based communication.

2. Publish/Subscribe message based communication.

2.4.4 Queue Message Based Communication [RS01]

A queue message based communication corresponds to a point to point access between an application client and an EIS. A queue behaves as a buffer which is independent of both application and EIS. Messages are stored in this queue for immediate or later retrieval by the client's application. Since messages can be saved for later delivery, the client's application need not be running when a message is generated. This option provides a more flexible and convenient access to information, knowing that an application can be communicated with old messages as soon as it starts running.

2.4.5 Topic Message Based Communication [RS01]

A topic message based communication corresponds to one-to-many/many-to-many message communication. Applications register their interest in a particular topic(s). With time, Enterprise Information Systems populate the topic with messages. These messages correspond in most of the cases to up-to-date information reflecting various interesting aspects of information that is of interest to the user. When messages are ready, the EIS places them in the topic and then all messages, at the same time, are forwarded to all online applications who have previously indicated interest in such topic(s). This method of communication is referred to as publish/subscribe. It should be noted that both client applications and EISs can behave as publisher and/or subscribers.

2.4.6 Integration Via Message Broker [RS01]

More serious integration can be conducted with the use of a message broker. With a message broker incorporated in the design and implementation of an EAI, more business logic can be implemented to manage the messages. Topics and queues can reside within the message broker's sight. The message broker can then intercept incoming messages and transform them from one format to another in order to make them compatible with other systems, thus making the system more interoperable. In addition to that, more than one message can be combined to produce a more comprehensive message for delivery. Thus the importance of a message broker is clearly significant. This is well

demonstrated since it behaves as an orchestrator, i.e., understanding all kinds of messages and applications as well. In simple words, a broker simply understands the big picture of the entire integration process (figure 2.4).

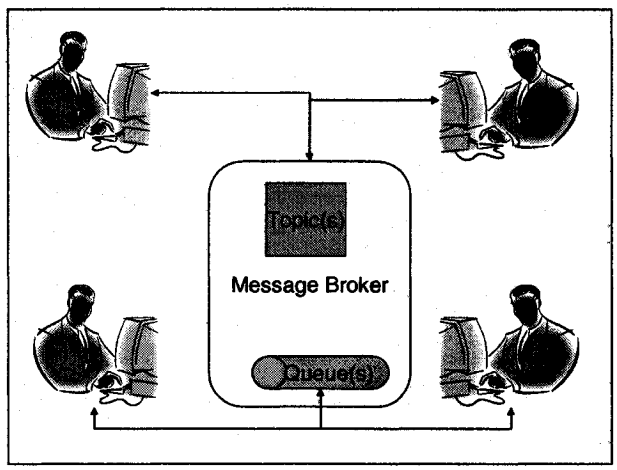

Figure 2.4: Broker based message communication.

2.4.7 Integration Via Application Server

Application servers are a suitable medium for web driven integration. This is true because application servers provide an environment for developing, deploying, and managing applications and components. In general, multi-tier applications rely on application servers. In a three-tier architecture, the EIS behaves as a base tier being the main source of integration, the middle-tier is where all the business logic is implemented and where a message broker is installed. The client-tier in this architecture can be a web based HTML client or an ordinary application. The utilization of an application server reduces the complexity of the system and boosts its power because it is designed in a way to manage:

1. Distributed communication.

2. Database access.

3. Security.

4. Transaction support.

5. Distributed communication.

6. Asynchronous messaging.

2.5 Towards Service Oriented Architecture

Service Oriented Architecture was developed to address the vast integration issues that were prominent in earlier technologies. The new wave of integration that SOA was built on, focuses on the independence of platform, languages, data, and communication protocols. So, on what concepts was SOA built? and how did it achieve them? SOA concepts are based on service description, publication, and binding. Enterprises describe their services and publish them in accessible repositories that other services/applications can find and utilize. All of this is achieved in a transparent way hiding the complexity from the end user. Most definitions of SOA point out to the use of web services in their implementation, which can be achieved using any service-based technology [enc05]. As for the implementation of SOA via web services, e-Speak [Mye02], a software product developed by HP in 1999, was the first web service implementation of SOA. Consequently, companies started developing frameworks for web service integration and the result was J2EE from Java, WebSphere from IBM, and .Net from Microsoft. This led to the evolution of web services, which are self described, self-contained, and modular units that are build on top of standard based protocols and exist to achieve service integration over multiple platforms. Therefore, web services are gaining huge momentum since they overcome the stated limitations of previous technologies. The architecture of web services is straightforward; service providers describe their services using Web Service Definition Language (WSDL) and place the corresponding description in a repository. A service requestor uses Universal Discovery, Description, and Integration (UDDI) [OAS05] to find the appropriate web

service which fits the profile of the requestor. When a match is identified, the service provider and the service requestor bind together and all further communication is achieved via Simple Object Access Protocol (SOAP) (figure 2.5 [?]). This usage of standard based protocols made the future of web services so prominent. However, using web services in such ways does not leverage the business world with the real expected value. Therefore, providing a link between web services and business workflow will grant the business a precious added value. Among recent initiatives in this direction, we find Business Process Execution Language for Web Services (BPEL4WS) [Jur05] and Web Service Choreography Interface (WSCI) [W3C05]. BPEL4WS is a business process and choreography specification standard designed by IBM and Microsoft. It provides a language for the formal specification of business processes and business interaction protocols, extending web services interaction, and providing more support to business transactions. Regarding WSCI, it took the first step towards standardization, thus ensuring the adoption of collaborative business applications. Actually, these languages define an interoperable integration model that should facilitate the integration of intra organizational processes and between businesses and organizations.

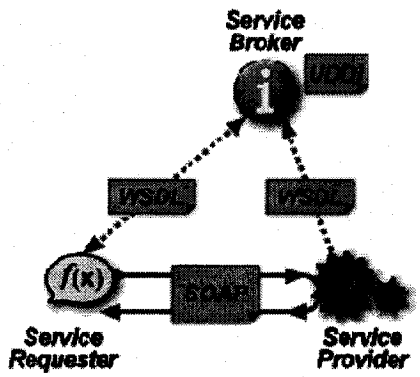

Figure 2.5: Web service communication structure.

The following sections highlight some interesting papers dealing with web services composition and their QoS.

2.6 Web service QoS

Most, if not all, web service QoS aware middleware depend on a broker to collect, select, compose, and monitor web services during the complex process of web services composition [YL04a, YL05, YL04b, TGRS04, HK04]. These brokers deal with various issues of efficiently selecting "possibly good" web services from different pool classes, then composing and monitoring them. The act of monitoring used by these brokers, however, has been limited to the narrow field of identifying service availability/failure according to some aspect(s) of quantifiable and functional QoS metrics, and trying to recover from it. Surprisingly, no broker provides the possibility of providing a higher level of monitoring, i.e., investigating web service resources, and reflecting changes to the user as they occur. This is true because until recently, existing technologies did not provide any support for such actions and requirements. Basically, most brokers depend on standardized QoS specifications reflecting various possible concerns to the end user; some specifications are HP's Open View Internet Services, IBM's Web Service Level Agreement framework (WSLA), among others. As a result, some QoS brokers depend on Service Level Agreement (SLA) to utilize and compose services, and use heuristic to present an optimized QoS experience to the user [Men04]. As a general rule, brokers depend on maximizing their utility function to deliver better rates to the service user [ZBN+04, YL04b]. In [YL04b] the authors focused on designing efficient service selection algorithms. The main objective of the defined algorithms is to maximize the utility function to meet certain static and dynamic QoS requirements. In [YL04b], the problem was modeled as a Multiple Choice Knapsack Problem (MCKP) and a QoS aware broker was developed to maximize the utility function with a service selection algorithm based on Pisinger's dynamic programming technique. The elements of the utility function were based on the service capacity, cost, and some weights and standard deviations related to cost and benefit. The authors of [YL05] proposed a web service composition middleware based on a broker capable of building an execution plan dynamically with QoS in mind. The broker maintained a database to track various QoS aspects of many web services. Information found in the database provided some statistical means that formed as input to the utility function to help in constantly

delivering an optimized solution. Another use of optimization was reported in [ZBN+04], where Integer Programming was used to deliver an optimized QoS experience to the user by choosing the best solution among all possible execution paths. AgFlow [YL04a] defines a middleware capable of delivering quality-driven composition of web services. AgFlow depends on user requirements quality constraints and its selector algorithm works to meet these requirements. Furthermore, It performs negotiations to guarantee the user's required QoS constraints. AgFlow is among the few systems that target some non functional QoS requirements such as service reputation. In [ZBN+04], the authors proposed the notion of modeling a web service composition action as a statechart and then proceeded to place each statechart node in a specific region to provide the possibility of recovering from failures. In order to make web services more active in the overall QoS issue, web services need to initiate interaction in the first place. This interaction can be achieved via the WS-Notification infrastructure technology [TN, Vin04, IFT05] which is part of the Web Service Resource Framework (WSRF) [Glo]. WS-Notification is emerging as a new standard for message basis notification within a Service Oriented Architecture (SOA) initiated by IBM, HP, SAP and many other information technology leaders. Although it is still in the process of standardization by OASIS [TN], some implementations exist that support different levels of this new emerging technology [M.H05]. Some of which are: GT4-Java, GT4-C, pyGrodWare, and WSRF.NET. Since WS-Notification is still in its infancy stages, its advantages for improving the QoS of web services have not yet been exposed to the best of our knowledge.

2.7 QoS Issues in Web Services [Men02]

Quality of Service measurements are first experienced by web services and later by end users in a web services composition process. As a result, to deliver a satisfying experience to end users, web services delivering this experience should maintain high QoS standards. QoS standards can be enforced by defining Service Level Agreements (SLA). These form a legal binding between a web service provider and its utilizer. SLAs agree upon various, well defined QoS criterias that should not

go below a certain threshold otherwise involved parties may face charges. Defining thresholds in an SLA is an interesting and yet a crucial job. As an example, to identify a threshold on throughput, the Forced Flow Law [Gru] can be very useful. The Forced Flow Law can be used to specify an upper bound on throughput for main services belonging to a web services composition plan.

The identified throughput of main services takes into consideration all throughputs of other involved ones. This would guarantee that requests will never experience any delays due to service overload. Therefore thresholds can be explicitly specified in the SLA to guarantee various QoS criterias. More interestingly, in [Men04], the authors discussed the possibility of composing services and providing an end to end QoS based on SLA negotiations among all involved components of the system. As a result, from a set of SLAs for each service the most optimal would be utilized, leading to an optimal global QoS experience. This paper approached the solution using heuristics and the documented results demonstrate that the end solution was truly an optimized one.

2.8 QoS-Aware Middleware for Web Services Composition [ZBN+04]

In this paper, The authors presented a middleware capable of composing web services in a particular way to maximize user satisfaction. This is achieved by utilizing the concept of utility functions over QoS attributes. The middleware builds on top of the AgFlow system. Moreover, it is based on state charts which models the overall process of web services composition. This paper presents many interesting concepts that favor QoS. To start with, execution paths and execution plans are defined for the corresponding state chart. An execution path extracts all nodes/tasks that would provide a rout from an initial task to a final one. Execution plans takes this one step further. An execution plan assigns a web service to execute each task since more than one web service can belong to the same class, i.e. providing similar services. Among the various QoS factors there are to consider, the paper focused on: execution price, execution duration, service reputation, successful execution rate,

and availability. Each of these important variants is captured within a specific function.

2.8.1 Service Selection for Web Services Composition

In this paper, two methods were illustrated to perform service selection and composition with QoS in mind. While the first being service selection based on local optimum, the second focuses on service selection, based on global optimum. Web services selection with local optimum in mind selects the best web service within a class of services to perform a certain task. This selection is based on the concepts of "scaling" and "weighting" of web services. While Scaling identified the importance of one QoS attribute among web services of the same class with respect to the best web service available, weighting provides the summation of all scaled QoS attributes for each web service. As a result, at each node belonging to the execution path, the best web services performing the task is selected. It is widely know that utilizing this technique guarantees a local optimum but not a global one. As a result, Integer Programming was proposed to lead to a better global optimum. Variables, objective functions and a set of constraints were used forming the environment of operation. Then this environment operates to provide the most optimal global plan possible.

2.9 A Broker-Based Framework for QoS -Aware Web Service Composition [YL05]

Quality of Service is a main factor to differentiate between web services providing similar operations. In this paper, the authors propose a broker-based framework for QoS aware web service composition supporting dynamic composition. This is performed with end-to-end QoS in mind. It should be noted that the QoS criteria's tackled in this paper are similar to the ones presented in the previous paper. The broker is where all the intelligence of the system resides. Therefore, it is wise to pay a closer look at the architecture of the QoS broker and it modes of operations. The main components of the broker are the following:

1. Service Information Manager.

2. Selection Manager.

3. Composition Manager.

4. Adaptation Manager.

The Service Information Manager behaves as a repository for available web services from which a selection can be performed. This manager retains detailed information about each and every service such as its ID, URL, service class, operations, etc. In addition to that, it also keeps a statistical table about the services. This table collects valuable QoS statistics about web services residing within the repository. This will later be used during the service selection process. The Composition Manager defines an execution plan which is identical to the one discussed in the previous paper. At this point, for each task, there exists a set of web services, from which one can be picked to accomplish the task. The duty of the Selection Manager is to select web services from each class to satisfy the overall request. It defines an objective function and tries to maximize it. The solution domain belongs to the famous Multiple Choice Knapsack Problem (MCKP). After a service Selection Manager terminates, the global plan starts its execution. At this point, the Adaptation Manager commences to monitor the system and measures whether the QoS constrains are violated or not. If a violation is detected, the violating service is identified and an alternative path is chosen which optimally complement the composition and execution process.

What is interesting is that many research concerning QoS related to web services composition revolves around the notions of brokers, utility functions and service selection algorithms. More interestingly, most of the proposed service selection algorithms work to maximize the output of the utility function by implementing a certain variant of dynamic programming. Another similar example is also demonstrated in the paper entitled "Service Selection Algorithms for Web Services with End-to-End QoS Constraints".

2.10 An Analysis of notification Related Specifications for Web/Grid Applications [PF05]

In this paper, the authors compare and contrast two classes of systems that perform message notification for web services. This is an import aspect to comment on because message based notification has a significant positive impact on the way web services communication, and moreover, can provide additional functionalities that were unavailable before. The two notification specifications under investigation are WS-Notification, part of the Web Service Resource Framework (WSRF), and WS-Eventing. WS-Notification is composed of three components, these being:

1. WS-BaseNotification
2. WS-BrokeredNotification
3. WS-Topics

WS-Eventing is self-contained and does not branch any further. Messages traveling between services can be one of many different types. WS-Notification provides the opportunity of NOTIFY messages as well as RAW messages. Each of these types of messages has a special structure providing information in different forms for different kinds of applications. WS-Eventing on the other hand delivers only one kind of messages and this being RAW messages. Both specifications support delegated management through which event subscription is performed via a subscription interface. XPath queries are compatible with both specifications and can be used to extract XML based information. WS-Notification also presents a more complex subscription mechanism. It provides the ability to subscribe, pause and resume. Interestingly, it does not provide a mechanism to unsubscribe. This issue is managed by an expiry time that is created when subscription first takes place. On the other hand, WS-Eventing provides functions to subscribe, renew, unsubscribe, and subscription end. Therefore, it becomes quiet clear that with functions provided via WS-Notification, it can be used in an intelligent and novice way to provide a higher class of QoS which will be illustrated within the coming chapters.

2.11 Conclusion

In this chapter we have laid the foundation of EIS integration. We demonstrated the various ways, tools and techniques to integrate applications and shed light on what seems even more promising, that being Service Oriented Architecture. Integration has become one of the top most priorities for enterprises if they want to remain on a competitive edge. With Integration, more value added services can be exposed to the organization as well as to the customers. With this in mind, many factors come into consideration. One important factor to consider is the quality of service experienced by users upon requesting a service involving the integration from more than one information source. This very concept will be tackled in later chapters and we propose an efficient algorithm that delivers high quality of service results.

Chapter 3

The Digital Cockpit Concept

The Digital Cockpit concept developed in this chapter presents an original contribution to this thesis. The Digital Cockpit is a data/service Enterprise Integration (EI) built on the very concepts of loosely coupled asynchronous communication, Message Oriented Middleware (MoM), and real time integration and monitoring with interactive display. This chapter highlights concepts related to the architecture of the DCP. It will further layout the advantages of a MoM, and outlines an extension leading to Service Integration (SI).

3.1 Introduction

The DCP under investigation presents an interesting analogy with the cockpit referred to by pilots. Its main intention is to provide people across the organization with real time information concerning the present situation of their enterprise in the same way a pilot's cockpit constantly delivers accurate information, allowing pilots to engage in more accurate decision making scenarios. The main challenge remains to integrate dissimilar back end information sources/services, monitor them, and instantly reflect changes to the user with minimal interaction from client's end. Achieving this would lay the foundation for a new breed of software systems behaving as an infrastructure for Decision Support Systems (DSS). The primary technological concepts that assemble the essence of the DCP

are asynchronous MoM operating over loosely coupled components.

Current integration middlewares establish a tightly coupled communication channel among involved components. This has a significant negative effect in terms of performance and network overload due to the following simple reasons. When constant refreshing is needed to view up-to-date information, numerous resources are constantly being wasted in terms of continuously establishing communication channels, issuing the re-execution of previously executed commands, re-compute the same business logic, and retransmitting a huge set of information each and every time. To avoid this costly operation, we are proposing an event driven notification mechanism based on Java Messaging Service (JMS) technology. This, in contrast with Remote Procedural Calls (RPC), eliminates the significant burden associates with previous technologies. This allows the creation of a MoM, which based on messages, creates an innovative environment for information/service(s) systems integration. Moreover, a robust MoM solution should be designed to avoid pitfalls of previous technologies, and should posses the following features:

1. Retrieval/storage and delivery of information: Using messaging concept, information regarding a request is fetched and stored in appropriate repositories (topics or queues), and then routed to appropriate subscribers.

2. Update notification: Upon the detection of any particular change (in the information set of data sources) that is of interest to at least one active subscriber, this change is reflected with minimal user intervention.

3. Performance: Performance is enhanced by reducing the number of sent messages by a factor of n-1 where n is the number of active participants. Moreover, network utilization is reduced since in contrast with the concept of refreshing, in this case only the change is communicated to the client application.

4. Scalability: With an integration concept in mind, the system should possess ports allowing the ease of adding/removing information/service(s) sources.

5. Autonomy of information: Since information/service(s) sources may belong to different organizations, the system should still be able to request and deliver information without violating organizational security policies.

6. Administrative and technological restrictions: Administrators need to understand the system in order to manage it.

7. Level of technological support: Different EISs support different type of technologies and might not be compatible with standards. Therefore a non trivial technique should be used to accomplish the integration.

3.2 Architecture

After identifying the motivation behind the DCP, and laying out its features, this chapter proceeds in detailing the architecture of the system. We will first proceed by illustrating the various components that need to exist and communicate in harmony to make the production of a feasible integration paradigm. Figure 3.1 identifies the required components:

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

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

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

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

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

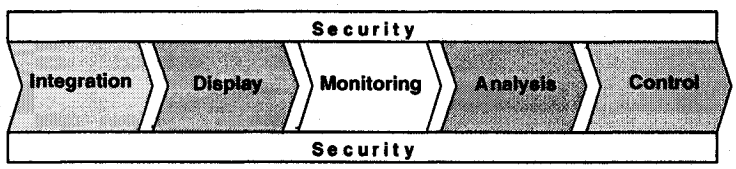

Figure 3.1: Digital Cockpit, a 5 phase paradigm

Figure 3.2 unfolds the hybrid architecture of the DCP where asynchronous Point to point, client/server communication over messages is the initial means of communication. Furthermore, all conducted transmissions respect the autonomicity of client/data/service sources whether they were inter/intra organizational.

Zooming into the details of the architecture, each department is composed of a number of client machines running the DCP application client, other machines having both the application server and integrator module setup with JMS running on them, and finally the data sources. The steps for data retrieval propagate as follows: The client application issues an inter/intra departmental data integration request. If the request source and the data sources reside within the same department, the corresponding application server and integration module will carry the request. Otherwise, the application servers and integration modules of all involved departments communicate together to deliver the required information set. Initially a queue is established as a repository for the first bulk of information transfer. All later changes to data sources will be communicated to all participants after first being stored in a topic. This is both feasible and sound since when a client identifies an interest in particular information, this information will then be constantly monitored, and the client

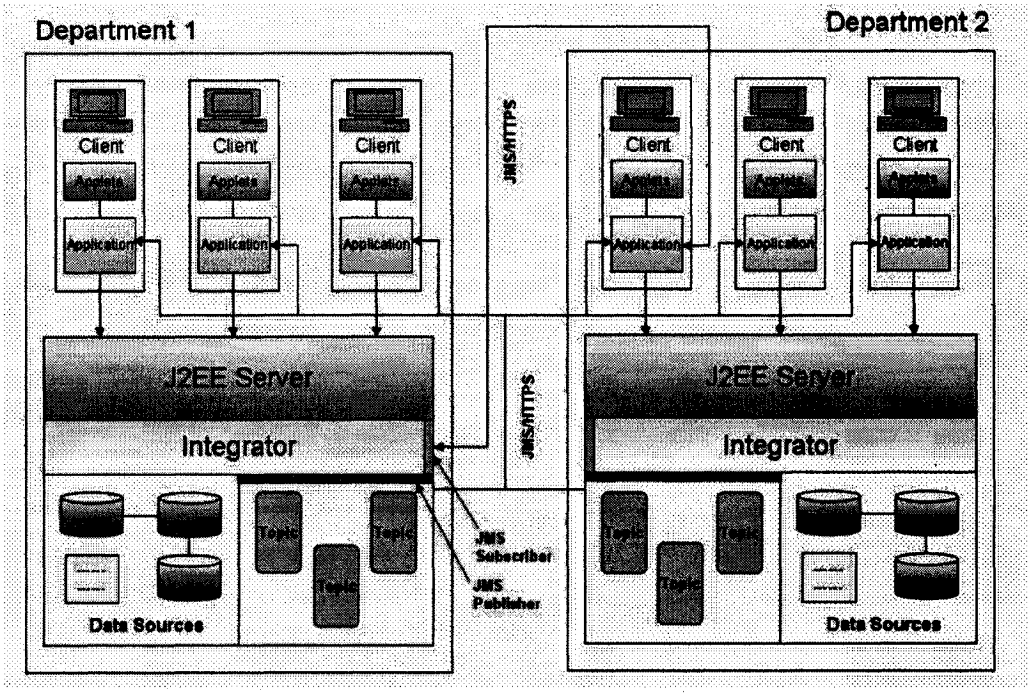

Figure 3.2: Inter/Inra departmental communication via DCP

will constantly receive a small amount of data reflecting changes as they happen. Finally, at the client side, the data retrieval process terminates by presenting a graphical display of the information set returned.

3.3 Multi-tier Architecture

After presenting the Integration Hierarchy within the architecture of the DCP, it is time to take a step backwards and view a more concrete representation of the overall hierarchy of the complex process of integration. Figure 3.3 identifies a multi-tier pattern capable of delivering a synergetic, asynchronous based, information integration middleware. The pattern is composed of four platforms identified as: Resource Tier, Integration Tier, Acceleration Tier, and Presentation Tier.

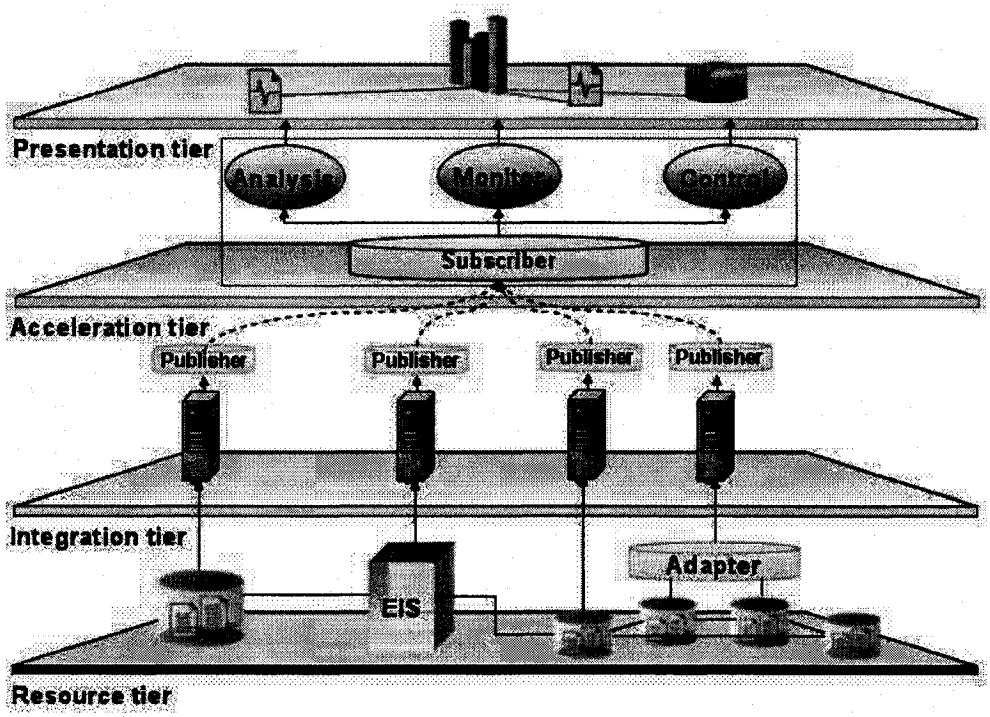

Figure 3.3: A Multi-tier approach for enterprise integration

3.3.1 Resource Tier

The Resource Tier is an environment of dissimilar data sources that forms the back end of the system, and is the source of all information. Data within this environment can be structured, semi structured, or no structured at all. In addition to that, since the DCP is a MOM, the message format should be universal. Moreover, Java Database Connectivity (JDBC) and Java Connector Architecture (JCA) are chosen as API's for accessing data of various structures because Java was adopted as the development programming langauge.

3.3.2 Integration Tier

The Integration Tier corresponds to the central intelligence of the DCP as a whole. It is the beating heart of the MOM and is the place where all integration, message reformatting, storage, routing, and notification takes place. The Integration Tier accepts requests from the user, generates appropriate queues for initial delivery, and then generates appropriate topics for efficient multiparty delivery of updates. As a result, we can identify the following advantages related to the integration tier:

1. Asynchronous and loosely coupled clients: Since this is an asynchronous message based request-response paradigm, client applications are never blocked.

2. Real time notification: This fact eliminates the need to constantly refreshing the client's display to deliver updates. This allows the user to spend more time utilizing tools regarding data analysis and decision support.

3. Quality control: The use of MoM allows the utilization of standard QoS features implemented within a messaging API, such as reliability, guaranteed delivery, priority, etc.

4. Ease of extensibility: The most complex part to extend would be introduced by adding a new resource tier. Other than that, a reformat of the message structure introduces ease of extensibility, due to the XML nature of a message.

5. Message storage: With message storage capability, multiple users can be served at once, and

more durable subscriptions can be managed. As it will be illustrated later, this functionality can be used to leverage QoS.

6. Interoperability: Since messages are based on standards, this makes the system even more interoperable and able to communicate across different platforms.

3.3.3 Acceleration and Presentation Tier

The Acceleration Tier along with the monitor module joins forces to compose the client application. The monitor module is the window through which the client application sends and receives messages from the integration tier. Upon the reception of a message, the data is processed, and delivered as an interactable chart on the screen. Furthermore, whenever a new notification message is received, the chart is automatically refreshed without any user intervention. At this point, the acceleration tier joins to provide decision support aid to the users. Since charts are fully intractable, the acceleration module allows the possibility of performing statistical and simulation scenarios that would better help managers understand the big picture of the organization.

In the following section we present a powerful extension to the DCP allowing service integration.

3.4 Service Integration

Enterprise integration is not limited to data integration. Service Integration (SI) is gaining huge momentum and is witnessing significant acceptance due to the standard nature of web services i.e. XML, WSDL, UDDI, etc... As a result, incorporating service(s) integration within the DCP system seams to complete the puzzle. Figure 3.4 illustrates a model capable of achieving service integration. This is accomplished with the combined effort of an Integration Container, Execution Engine, and a Notification Manager.

Now we propose an enhanced version of the digital cockpit by expanding its integration tier to include a novel service integration container. This additional feature raises due to the limitation of the digital cockpit to integrate only heterogenous data sources and not web services. As a result,

adding a new container to the present architecture will complement it by providing means for service integration and monitoring. This container will be the core, providing the ability to detect and reflect real-time changes without any user intervention. Within this container there resides an execution engine whose duty is to create and manage a business process plan dealing with global service planning, execution, and re-execution. In addition to the execution engine, there exists a notification manager which receives updates from corresponding services and feeds these updates to the execution engine. This by itself will trigger the re-execution of the preestablished global plan.

What follows details the inner workings of the components involved in service integration:

1. Integration Container: The need for an integration container raises 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 interest in some services. After this point, the user is guaranteed to be informed of all instant changes as they occur. This will provide analytical methods of the decision support system with accurate information to work upon. In addition to this, as it will be illustrated along this thesis, the Integration container will not need to re-execute the entire global process plan. Instead, it will use the theory explained in later chapters to detect the point of change and re-execute services that are affected by the change.

2. Service Execution Engine: The Service Execution Engine within the integration container is where all the intelligence of this framework resides. Once the integration container receives a request from the user, it will pass it to the Service Execution Engine. At this point, the engine initiates a global planing event which decomposes the users request into smaller ones and builds up business process plan which models the sequence of events that need to be executed. The system will proceed in executing the plan i.e services, and upon completion, the end result

will be sent to the user and displayed in a fancy graphical way. At any time, if the system detects a change, the process will re-execute the pre-established execution plan, without any user intervention. This procedure will by itself guarantee the freshness of information delivered to the end user. It should be noted that the detection of changes is recognized via constant polling.

3. Notification Manager: Once a user identifies an interest in some service(s), the integration container registers its notification manager to all participating services identified during the service planing phase. After receiving a notification via polling, 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.

Figure 3.4 represents the overall architecture of the integration container as a stand alone system. It exposes the entire process of integration, monitoring and re-execution with a weather web service as an example. First, the client sends a request to the integration container indicating an interest with specific web service(s). This request is captured by the binding components and redirected to the service execution engine. Binding components are mediums that forward messages to the appropriate receiver. The execution engine will in turn compose the global service composition plan and will initiate the execution process. moreover, the notification manager will register itself with the corresponding web services. On the other hand, the weather web service has schedulers that inform the notification manager whenever a change takes place in their corresponding data sets. The notification manager will then inform the service execution engine that a change took place. This will re-initiate the process of re-execution. It should be noted that inner/outer interaction between system components and services is achieved asynchronously using messages and publish/subscribe mechanism.

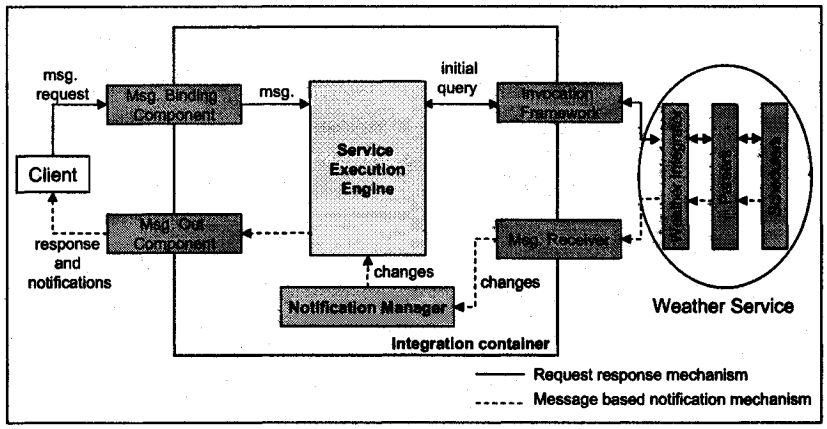

Figure 3.4: Illustrating various components required for service integration

3.5 Case Study: MoM

In this section, we explore the inner workings of the system in terms of response and execution time. It is considered that N applications can be served by the DCP middleware, and that "r" requests can be served by the middleware to the resource tier at any time (Figure 3.5). As a result, the total time of information retrieval converges to:

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

3.5.1 Case Study I: Single thread performance

Since we are working with asynchronous communication in mind, clients can send requests and continue execution without being blocked waiting for a response. In this case study, we assume that a single thread at the integration tier can service $C(C \geq 1)$ simultaneous requests from connected applications. As a result, the total time needed to serve R requests is calculated as follows:

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

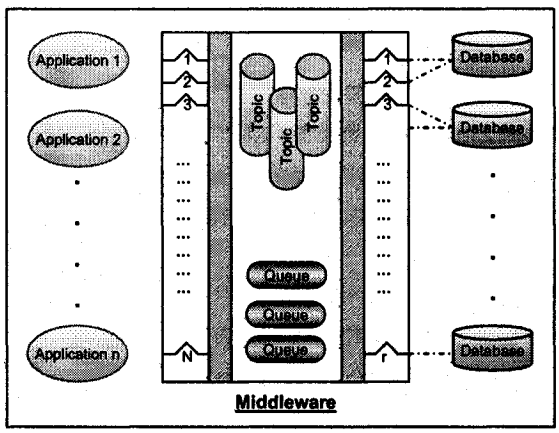

Figure 3.5: Serving N applications via r requests

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

3.5.2 Case Study II: Groups performance

It is believed that a number of clients will issue similar requests to the system. Since these requests might fall within a common time range, then the system could be implemented with an optimized feature computed, i.e. information is stored for later retrieving by other subscribers. This would reduce waiting time to clients and would result in better performance. This can be achieved by properly configuring JMS topics and queues. Topics can be used as common repositories for similar requests. Therefore, the system will not execute every query, its job is only to forward the contents of the topic (if available)to clients. It should be noted that clients will always be receiving up-to-date information because the integration middleware will constantly push up-to-date information as they happen. Stating this in a more formal way, consider there exists g number of groups among R requests with q requests in each group on average. As a result, the total number of requests issued by the integration tier corresponds to:

(R - (g x(q-1)))

3.5.3 Case Study III: Total time over hierarchy

In this case study, we evaluate the total time required by events to travel up the DCP hierarchy. Real time events are generated and triggered by the resource tier. Following that, events travel to the integration tier which places them in appropriate repositories. After storage, events are communicated to clients. This stated in a formal way, consider that the DCP middleware requires T_p time to route information to appropriate repositories after receiving a trigger from the resource tier. Moreover, let T_i be the time needed to communicate message i between the DCP middleware and client application. Let t_p be the time required to process multiple events within a resource, and t_i be the time needed for message i to travel from the resource to the integration tier. As a result, the overall time required to receive x events from the resource tier to the client's application is

$$\mathsf{T}_{MOM} = t_p + Max_{i=1}^{x}(t_i) + T_p + Max_{i=1}^{x}(T_i);$$

3.6 Case Study: RPC

By definition, MOM compared to RPC is expected to have a better performance due to its asynchronous nature of communication. RPC applications place the program in a waiting mode until the procedure terminates its execution. With MoM, blocking in never an option. In what follows we provide a comprehensive evaluation of RPC in relationship to the case studies illustrated in the previous section.

3.6.1 Scenario I: Information Retrieval

With RPC in mind, program execution blocking can not be avoided. As a result, the time needed to serve N threads by an RPC-based application is $rt + w$. Moreover, the time needed to serve R users equals:

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

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

Comparing this result to Case Study I illustrated earlier, it is clearly visibale that MoM delivers a much better performance.

3.6.2 Scenario II: Managing Common Requests

RPC based applications not only bock processes but are also point-to-point and tightly coupled systems. As a result, RPC based applications are not designed in a way to manage common requests. Therefore, if K similar requests are issued, then the RPC system will proceed to execute all K requests and block all K applications. Moreover, while all overhead and query executions will be executed K times in RPC, the same requests will be executed only once with the utilization of a MoM.

3.6.3 Scenario III: Delivery of Real Time Notifications

Continuous polling is the only solution that can be utilized in order to build RPC based applications capable of delivering and reflecting real time changes. This is true due to the tightly coupled point-to-point nature of RPC based applications. Thus if such applications want to stay connected to a specific destination, then the application will enter into a blocked state. As a result, the only way to reflect updates is achieved via continuous polling. Therefore, polling requires at least twice the amount of time identified in case study III. In a more formal way, let $t_p^{'}$ be the time required to retrieve information issued by $x^{'}$ requests. Let $T_p^{'}$ represent the average processing time for a request within the middleware. Combining all these variables together, the total polling time would be:

$$\mathsf{T}_{RPC} = 2 \times \sum_{k=1}^{x}(t_k) + 2 \times T_p^{'} + 2 \times \sum_{k=1}^{x^{'}}(T_k) + t_p^{'}$$

3.7 Conclusion

This chapter delivered fruitful insight concerning the requirements of a MoM, an architecture for the "Digital Cockpit" along with an extension providing SOA integration. Finally, it presented a request/response time analysis in a MoM and compared it to that of an RPC. The end result proves that MoM are more efficient than its rival RPC.

Chapter 4

Design And Implementation

4.1 Introduction

This chapter highlights the complex requirements of building a Digital Cockpit, materialized from the architecture presented in the previous chapter. What follows is a realization of the DCP project, that can be customized to satisfy the needs of any organization. The contributions of this chapter are five folds. First we illustrate the set of requirements identified in the Software Requirements Specifications (SRS) document. Second we target design issues that would lead to sound implementation. Afterwards, we proceed by identifying implementation issues and challenges. Following that, we will provide details on the technologies used to build the system. Fifth and finally, we provide some snapshots of the system while it is up and running.

4.2 System Requirements and House of Quality

This section delivers condensed material extracted from the actual SRS document. The requirements were compiled after gathering numerous concerns that identify vast functionalities related to the DCP. As an infrastructure to our work, the Quality Functional Deployment (QFD) diagram (following six sigma approaches) was used to identify various aspects of consideration that should be

noted before commencing with the design and implementation phase. What then follows is an illustration of the domain model, use case model, and sequence diagrams related to the overall Digital Cockpit.

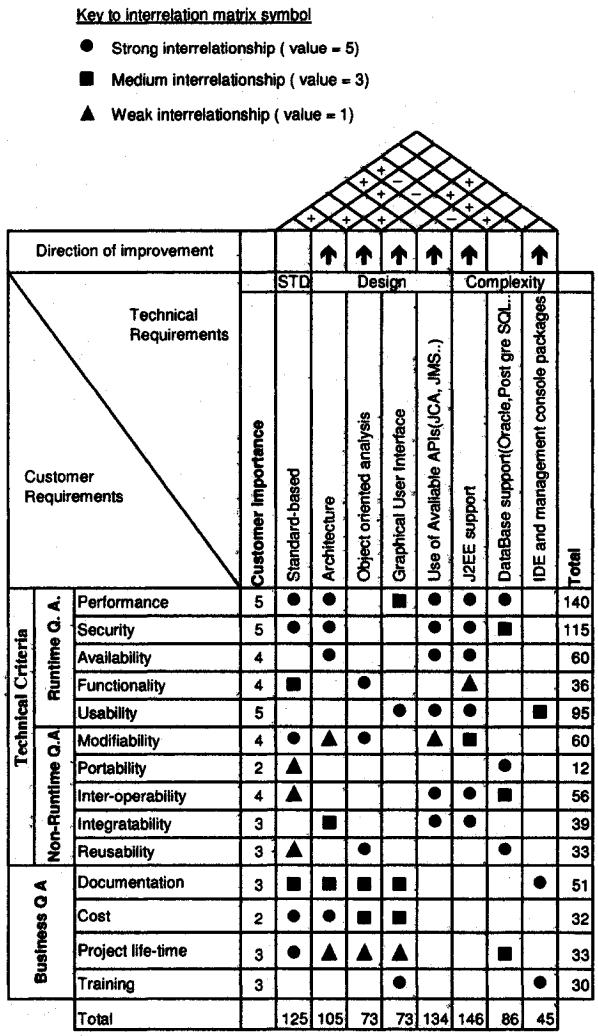

Figure 4.1: Quality Functional Deployment Diagram

The house of quality renders critical-to-quality concerns into a weighted set of goals, alternatives, and improvement opportunities. This would aid in the production of better products yielding high standards and qualities. This is achieved by determining the interrelation between technical and customer requirements combined together. The domain of the customer requirements is divided in

two sets identified as: 1- Set of Business Criteria, 2- Set of Technical Criteria. (Figure 4.1).

The set of Technical Criteria is composed of 10 components: Performance, Security, Availability, Functionality, Usability, Modifiability, Portability, interoperability, Integrity and Reusability. The set of Business Criteria is composed of 4 components: Documentation, Cost, Project Life cycle, and Training. It should be noted that all criteria's are assigned weights reflecting their relevant importance to the customer.

On the other side of the coin, the set of criteria considered by the development team is composed of three classes being: Standards, Design, and complexity.

4.3 Domain Model

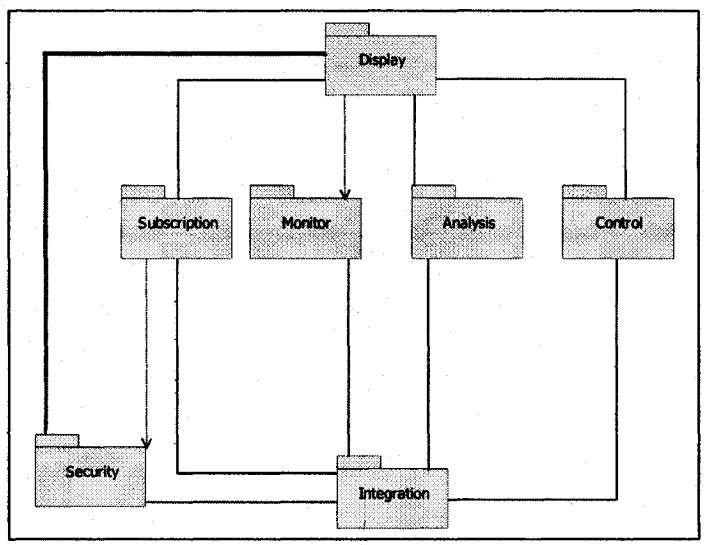

Figure 4.2: Digital Cockpit domain model

The domain model (figure 4.2) captures the relationship among business entities, thus defining the relationship among them, allowing better understanding of the system. In the DCP project, the domain model is composed of seven entities/packages identified as: Display, Subscription, Monitor, Analysis, Control, Security, and Integration. Following is a brief overview of the functionalities

associated with the modules of the system.

1. Display Module (client side): The Display Module runs in the client application and is responsible on delivering an intractable display to the user allowing viewing information in a variety of available charts. Furthermore, it allows users to drill more into the details of the charts providing more insight to decision makers.

2. Analysis Module (client side): The Analysis Module is used to provide the user with analysis capabilities to better understand data at hand. It delivers the ability to perform trend analysis, cause and effect, what if scenarios, and probability operations on historical data.

3. Control Module (client side): The Control Module delivers the ability to perform optimizations, compare and contrast scenarios, and suggests means to help decision makers better manage their organizations.

4. Monitor Module (Client side): The Monitor Module provides the DCP with the ability to capture, reflect, and route real time changes to the Display Module. This action would eliminate the user from the heavy burden of constantly refreshing the system, which as explained earlier, consumes a lot of computing resources and human time. The power of the monitor module is inherited from the utilization of the observer-observable design pattern.

5. Subscription Module (client/server side): The Subscription Module is responsible on managing all un/subscribing events related to data sources and services.

6. Security Module: The Security Module is responsible on providing means for users to log on/off. In addition to that, it provides authorization, authentication, and secure communication.

7. Integration module: This module constitutes the "brain" of the system. It establishes the link between information/service(s) sources and client applications. It manages user requests, and queries various heterogeneous information sources using appropriate adapters. It is also responsible for detecting changes, re-computing the new data set, identifying the difference

between the old and new data sets, and communicating the difference to the Display Monitor. This would in turn reduce bandwidth utilization, and increases efficiency.

4.4 System Use Case Model

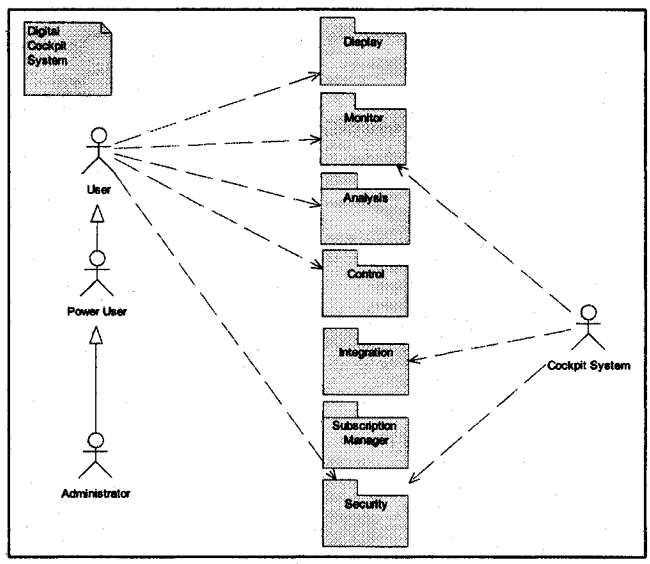

Figure 4.3: System use case model.

The System Use Case Module (figure 4.3) illustrates the desired functionalities of the system. This module represents abstract/actual means of interaction between a user (machine/software/human) and the system. Figure 4.3 highlights three users of the DCP:

1. Normal User: The normal user's use of the system is derived according to the set of privileges associated to his profile. This is solely limited to viewing a predefined set of information, performing various monitoring, analysis and control over data returned.

2. Power User: The power user's use of the system inherits the usability of the normal user adding to it the capability of tracking user operations according to the security policy specified within the department.

3. Administrator User: the administrator user's use of the system inherits the usability of the power user but also has the capability of modifying the access rights of all other users according to the organizational security policy. Furthermore, the administrator has the ability to modify the extent of the integrator module by adding/removing data/service(s) sources residing at the resource tier.

4.5 Software Design

In this section, we depict the software's multiple design issues related to the Digital cockpit project. This would cover the assumptions, methodology, and class diagrams.

4.5.1 Policy and Assumptions

No sound design can exist without taking into consideration multiple policies and assumptions. This step would help in better understanding the system and mold it to work according to predefined rules. The following assumptions are widespread and it is wise to consider them at early stages of design and implementation.

1. Sources: It is assumed the existence of various kinds of information/service sources. These can be Oracle, Sybase, MsExcell, spreadsheets databases and other types of sources(s).

2. Analysis: This is limited to statistical operations, time based analysis and simulations.

3. Scope of Implementation: This factor is limited to known sources of information. It should be noted that dynamic integration of servers and data sources is beyond the scope of this elaboration.

4. Client application: After performing excessive research and testing concerning the mode of implementation of the client side application. It was decided that the implementation should proceed as a Java application and not as a Java applet. More light is shed on this topic in the "challenges" section to follow.

5. Standards: Design takes into consideration the standards available for technologies and API's.

4.6 Display Module and Graphical User Interface

The Display Module, connected to the Graphical User Interface (GUI) from one end and to the Business Logic from another end delivers what is know as "Digital Cockpit View" which is very much similar to what every pilot sees on the plane's dashboard. The User Interface (UI) is what makes the entire system alive, and animates the entire business of the organization. Therefore, instead of developing a traditional GUI we indented to develop an Interactive Graphical User Interface (IGUI). This would provide means to better understand information according to the taste of each and every DCP user (figure4.4-4.10).

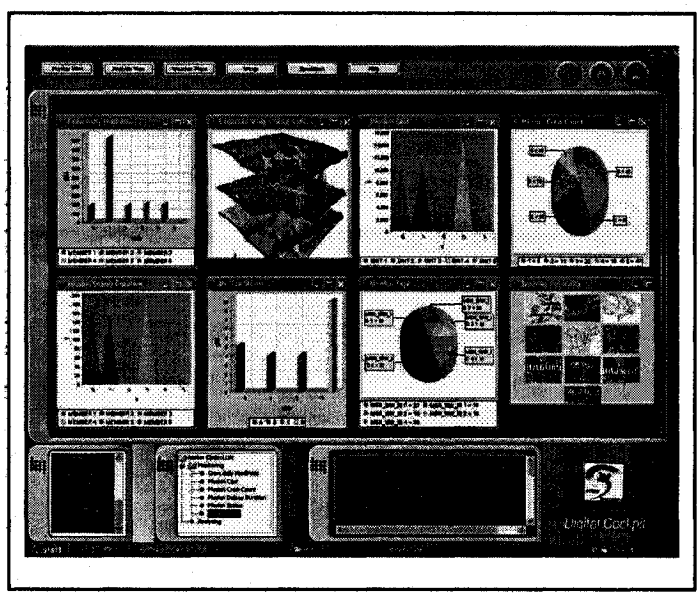

Figure 4.4: DCP client subscribed to 8 different components.

The IGUI delivers data, reflects real time changes, and reduces the time of interaction between the user and the system to one interaction per visible component. Beyond that, as long as components are alive, automatic component refreshing is conducted whenever the observable (information source) of the observer (display component) is modified at the remote end. This IGUI provides the capability

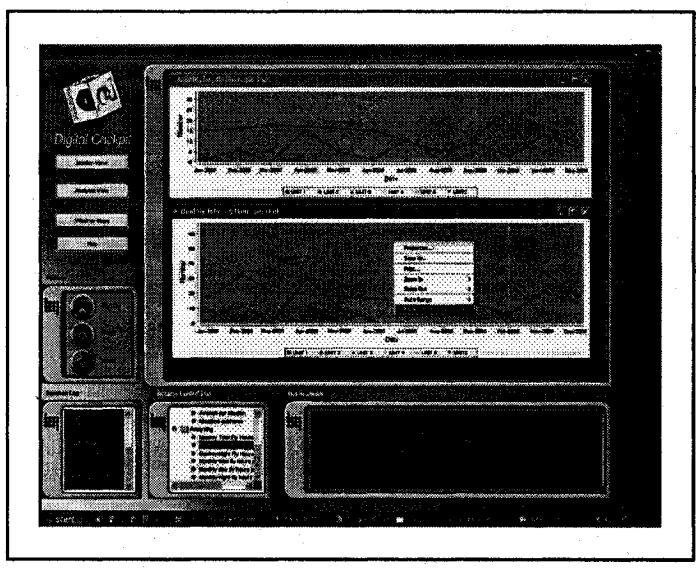

Figure 4.5: DCP client providing an analysis capability to a specific component.

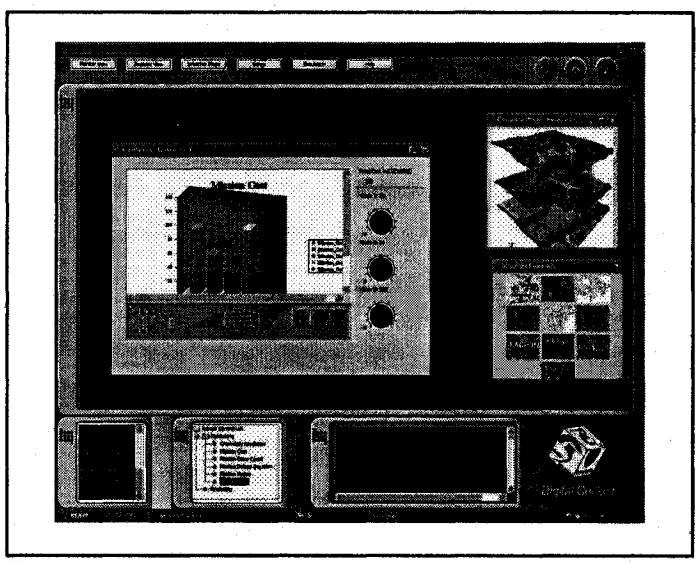

Figure 4.6: DCP client providing simulation options and what-if-scenario's

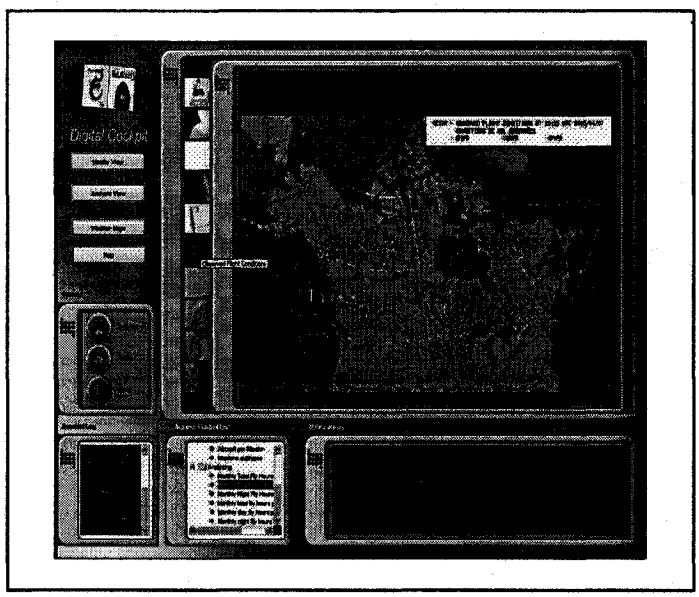

Figure 4.7: DCP client providing a detailed weather component.

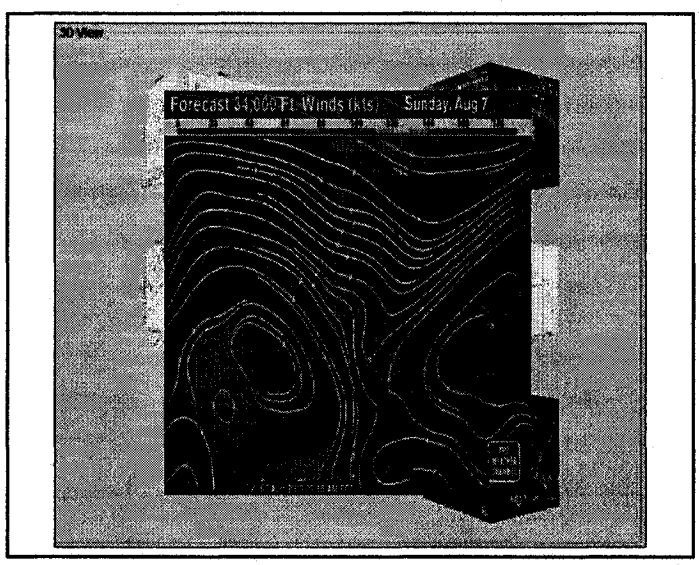

Figure 4.8: DCP client providing a weather component based on service integration.

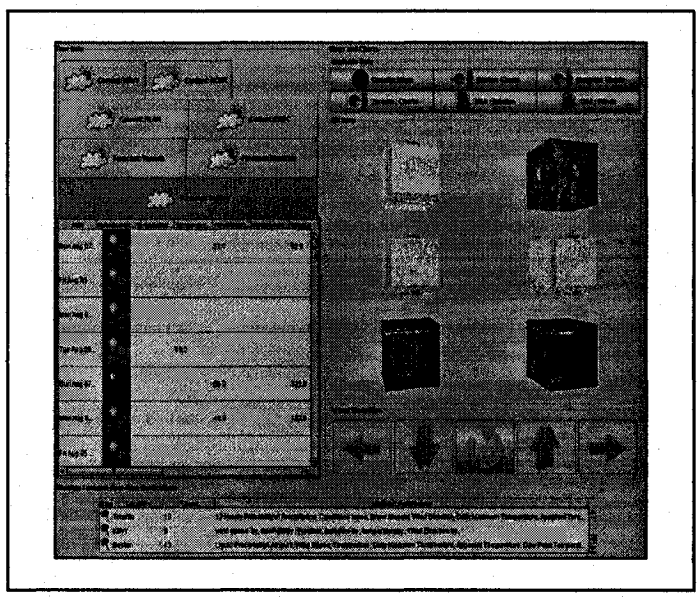

Figure 4.9: DCP client showing 6 weather components related to service integration.

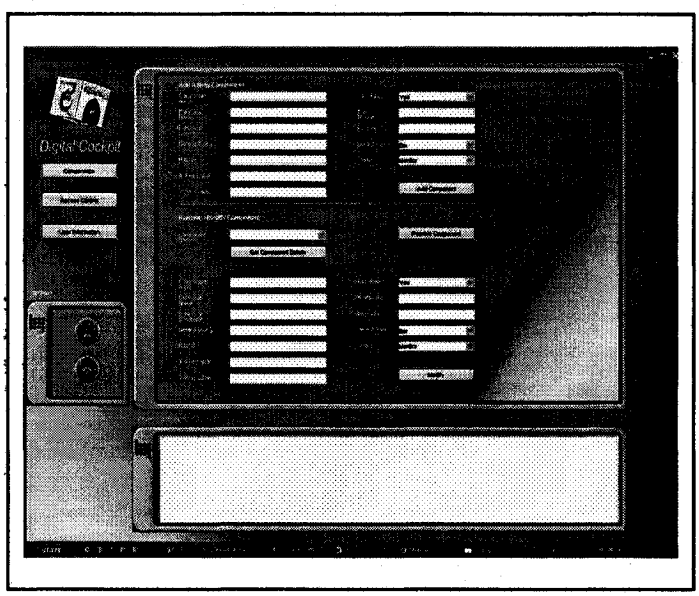

Figure 4.10: DCP administrator console: System configuration/priveledges modification.

of displaying charts, curves, knobs, histograms, reports, animated maps, Geographic Information Systems (GIS) etc. Many challenges had to be defeated during the development of the IGUI. Some of these are: managing listeners, delivery of remote events, managing threads, animating maps, and rendering objects based on remote events. The intractable feature allows users to change the graphical view of their components. In addition to that, it provides means for saving, zooming, navigating, reporting, and drilling. Drilling is an interesting option since it allows viewing data in a graphical way (chart) either as summary or at a more detailed level.

4.7 Implementation and Technologies

Excessive implementation was conducted to develop the modules of the system. A significant number of challenges were faces. This is true because the full capabilities of existing technologies and API's are still unknown. The implementation concluded with tens of classes, working together to produce a comprehensive DCP system.

4.7.1 Challenges

Initially, the DCP was intended to run as a web based application. The main challenge was inherited from the specifications regarding HTML technology. HTML is an all or nothing technology. If you want to refresh a small section of your page then this can be achieved only if you refresh the entire page. This would result in requesting the entire page content again. Since this is not feasible (utilizes huge computing resources), the HTML concept was dropped. The second best option was to develop the Digital Cockpit as an applet running within a browser. This at first looked promising, but it was then discovered to be impossible due to security limitations imposed on applets by the Java Virtual Machine (JVM). Even after signing the applet, providing it with a policy file to access remote databases, and modifying the global policy file of the JVM, applets were still not allowed to connect to servers other that the ones they were downloaded from. Since our main concept is information systems integration from multiple sources, applets posed a significant limitation. Facing

all challenges, the only path left to follow was to proceed with the system as a Java application.

4.8 Technology

The data/service integration project presented is built over the knowledge of numerous existing concepts. These core concepts being Messaging, Business Process Language, and Remoting. Baring these approaches in mind, many overwhelmingly competing products are present. Therefore identifying and evaluating the technologies available for each concept is challenging by itself. To start with, we will present a set of existing technologies within each concept and will proceed to illustrate some interesting ones. Figure 4.11 provides a well formed detailed summary which clearly identifies some of the best technologies within each domain of interest along with some useful remarks.

4.8.1 Messaging

Messaging is simply a new way for inter application communication relying on the concept of asynchronous communication. Message Oriented Middleware relies on storage, routing and transformation of information. Storage is for asynchronous delivery, routing propagates messages along the middleware components and it can also perform multicast/broadcast and transformation of messages. eXtensible Messaging and Presence Protocol (XMPP) promoted by the Internet Engineering Taskforce (IETF), Java Messaging Service (JMS) promoted by Sun Microsystems, and Microsoft Message Queuing (MSMQ) is a small list of Message Oriented Middleware systems and technologies. Although the list can grow, JMS has become the wide spread utilized MoM in the industry. Therefore, what follows is a comprehensive list of open source and proprietary systems for JMS providers.

Open Source JMS implementations:

1. ActiveMQ from Logic Blaze.

2. JBossMQ from JBoss Inc.

3. Joram by Joram Comunity

4. MantaRay from Coridan

5. OpenJMS from The OpenJMS Group.

proprietary JMS implementations:

1. BEA WebLogic Server JMS from BEA Systems.

2. Websphere MQ from IBM.

3. IONA JMS from IONA Technologies.

From this exhaustive list, we chose ActiveMQ as a solid provider for JMS technology. It is a complex open source messaging provider with numerous hot features. ActiveMQ can be configured to work as a broker within Apache Geronimo, a broker embedded within Spring framework, or as an ActiveMQ client that can use JNDI to look up topics and queues.

4.8.2 Application Server

The main components of the system should be housed and run within an application server. Our choice revolved around numerous J2EE application servers including JonAS, Jboss Application Server, Sun J2EE Reference Implementation, and Apache Geronimo. Among these, Jboss is one of the easiest open source J2EE application servers built purely in Java. Furthermore, it has excellent documentation and tremendous community support. Unfortunately, it conflicts with applications that directly use Apache AXIS SAOP. This is true since deploying a WAR file that utilizes AXIS SAOP API directly conflicts with it. AXIS is used by Spring Framework to generate web service client stubs. So avoiding using Spring was not an issue here. J2EE RI is good but unfortunately is accompanied with a restrictive license and poor community support. On the other hand, Apache Geronimo is an open source project backed with great community support. Moreover, it is integrated with ServiceMix JBI implementation, and embeds one of the best JMS providers "ActiveMQ". What is more, Apache Geronimo has strong community support but lacks good documentation.

4.8.3 Java Business Integration

Java Business Integration (JBI) is a process leading to the creation of Service Oriented Architecture. Java Business Integration is still a Java Specification Request 208 (JSR) built over the concept of web services producing a pluggable container hosting services acting as producers/consumers. This container is communicated through Binding Components (BC) if services are external to the JBI container. On the other hand, services can reside within the Service Engine (SE) within the JBI container. Service Engines can also embed business logic within them. Moreover Java Management Extensions (JMX) is used to manage the life cycle of BC and SE. JBI is intended to work with Enterprise Service Bus (ESB) but this fact has not been accepted by major J2EE vendors. Nonetheless, many open source JBI implementations based on ESB are available. ObjectWeb Petals, Apache ServiceMix, and Mule from Symphony Soft are some examples. We settled on choosing ServiceMix as a JBI container since our choices were limited due to the limited number of available open source JBI's. In addition to that, ServiceMix has ActiveMQ as a JMS provider embedded within it, so this factor also contributed on choosing ServiceMix as a JBI container. JBI documentation is available in JSR 208. This JSR provides the necessary knowledge to understand what can be done with JBI. In addition to that, ServiceMix's website has many articles related to JBI. JSR 208 describes the two kinds of JBI components identified above. These being BC and SE. BC are classes that are used to communicate on ESB. So in short, the SE will do the processing and the BC acts as a bridge between the JBI container and other external Legacy Systems.

4.8.4 Business Process Execution Language

A Business Process Execution Language is a means aiming to animate the business functionality via a series of well defined steps that each accomplishes a part of the entire global vision of the business. Many Business process workflow languages have been devised, and a small set is:

1. Business Process Modeling Notation.

2. WSCL : Web Services Conversation Language.

3. WS-CDL: Web Services Choreography Description Language Version 1.0.

4. WSFL : Web Services Flow Language.

5. XPDL : XML Process Definition Language.

6. YAWL : Yet Another Workflow Language.

These languages are workflow specification languages. They then need to be transformed into something understandable and executable by service engines. This is where the Business Process Execution Language (BPEL) kicks in along with its specific engines. Following is a small list Some of the promising BPEL engines:

1. ActiveBPEL Engine: A comprehensive BPEL runtime environment that is completely written in Java.

2. IBM WebSphere Process Server: A comprehensive BPEL engine running over WebSphere Application Server J2EE platform.

3. Oracle BPEL Process Manager: A BPEL engine running on top of the Oracle Application Server.

4. Twister: The first open source implementation of the WS-BPEL standard.

4.8.5 Graphics

After executing the BPEL process, numerous data is available that should not be presented in a tabular fashion but rather in a fancy graphical means. This is achieved via the JfreeChart charting engine. JfreeChart is an open source library having the capability of delivering cutting edge 2D and 3D charts. In addition to JfreeChart, Java 3D was used to manipulate 3D scenes. Furthermore, a huge set of online tutorials are available to guide through the details of creating and manipulating 3D scenes. Also, GeoTools is an open source GIS API that was used. It renders and displays maps and allows plotting various weather information retrieved by the Jweather API. Moreover, MapObject

for Java is another GIS API from ESRI that was used during the development. MapObject delivers high quality interactable GIS maps and supports rapid application development via reusable beans.

4.8.6 Weather API

Jweather is an open source API used to acquire and parse METAR and TAF data. METAR stands for METeorological Aerodrome Report and TAF stands for Terminal Aerodrome Forecast. This API requests weather information that is returned in an XML format, it then parses the huge data file, extract relevant weather data, and forwards it to appropriate functions for further manipulation and finally display.

4.9 Conclusion

In the light of the above, it becomes clear that the Digital Cockpit offers sophisticated user interfaces that keeps on providing the user with up-to-date information without continuous user intervention, using event notification mechanisms. Generally, information displayed on the screen is sent by the back end application, using some sophisticated middleware technologies (JMS- in our proposed middleware). The use of these sophisticated technologies leverages many key features such as: guaranteed information delivery; multi casting of information; and loosely coupled communication between the digital cockpit client and back end application. Furthermore, the Digital cockpit is intended to provide the decision maker with analysis capabilities, which leverages the decision making process. The Digital cockpit can behave like multiple applications at the same time, because each displayed graphical component is a key feature representing a crucial business activity. All of these factors identify that the Digital Cockpit is an important middleware with numerous advantages.

Feature	*Chosen component*	*Remarks*
Build Environment	Maven2	Avoids boilerplate code for setting up builds.
XML Parsing and Processing	Java API for XML Processing (JAXP)	Integrated into Java 5.
Java Business Integration (JBI) Engine	ServiceMix	Provides Spring integration, as well as POJO[M] programming for JBI components.
J2EE Application Server	Apache Geronimo	Strong architecture and integrates the best freely-available J2EE components, including ServiceMix.
BPEL Engine	Fivesight PXE	Integrated into ServiceMix
Inversion Of Control Container	Spring	Provides many components to ease POJO programming.
SOAP Implementation	AXIS, XFire and ActiveSOAP	All of these implementations are complementary
Charting Engine	JFreeChart	The most popular free charting engine.
3D Engine	Java3D	High-level compared to OpenGL Java bindings.
METAR and TAF Parser	jWeather	
Java Messaging System (JMS) Broker	ActiveMQ	Most-efficient Open Source Messaging Oriented Middleware, integrated into Geronimo.

Figure 4.11: A list of all API's used

Chapter 5

Qualitative Service Integration

One of the benefits of web services is their ability to participate in a web services composition process. Therefore, an end-to-end QoS infrastructure should be established. Work conducted in this domain has mainly focused on functional QoS requirements such as service response time, delay, cost, etc. In this chapter, we target QoS from the perspective of data freshness and accuracy. Therefore, we propose the usage of the WS-Notification specification as a base medium capable of sensing and routing any information change at the level of web services using a publish-subscribe mechanism. We then detail the Region Switching algorithm that is capable of identifying the point of information change within the context of multiple web services composition scenario. This is then followed with an appropriate re-computation of a subset of the pre-established, global service execution plan that would deliver up to date information to the user. Thereafter, a mathematical model capable of deciding if a notification message is worth any re-computation or not is presented and evaluated. This is worth to consider since some services may take a significant amount of time to produce results where a slight change to the information set might not alter the outcome of the service. Evaluated factors, computed thresholds, and a sound design can significantly work to our favor and leverage a higher QoS sensation. Following is our contributions: first we highlight the importance of qualifyable QoS aspect related to the issue of web services composition and monitoring, second we

describe an algorithm capable of capturing and reflecting the state of web services involved in the integration process, then we illustrate the usage of WS-Notification to aid in building such systems. Thereafter, we enhance our system with powerful extension followed by experimental results.

5.1 Introduction

Web services popularity is increasing due to the numerous advantages they provide to organizations upon their utilization. This is due to the fact that web services allow interoperable communications, and are based on standards with strong foundations [CDK+02] such as XML, SOAP, UDDI, WSDL. This fact leads to the notion of web services composition, delivering a broad spectrum of compiled information to the end user [BDFR03, 56401, MBE03]. Therefore, web services composition corresponds to the act of assembling generic services, put together to produce both control-flow and data-flow harmony between services and the information they deliver. As with any technology/system, good QoS should exist; otherwise the technology will become unusable. Most work related to QoS does not target this issue at the application level but rather at lower levels such as network QoS. As stated in [TGRS04], transport technologies like DiffServ, IPQoS, UMTS, and ATM are QoS aware technologies. Moreover, QoS at the server level is achieved by taking into consideration HTTP request differentiation, load balancing and many other factors. Unfortunately, all QoS related work deals with quantifiable metrics and not qualifyable ones. More specifically, they deal with response time, cost, delay, throughput, security, etc [YL04b, Men02, YL05] and drop out the notion of identifying/delivering contemporary data. Hence the aspects delivered by these brokers solely belong to the class of QoS functional requirements without dealing with the class of nonfunctional QoS needs. Therefore, a huge dissatisfaction will result if a user detects that a service she used did not provide the latest set of information and accordingly, the user will shift interest to another service provider. As an example, if Alice spends 10 minutes using an online web service to check the fare of a train ticket that is leaving in 135 min. Assume during the time Alice spends thinking whether or not she should buy the ticket, the train company announces a 50 % discount on

selected train seats. Alice, unaware of this sudden change of prices proceeds to buy a non refundable ticket at the ordinary price. After completing the purchase, she notices the new rate and is frustrated because she could have bought the ticket at a lower price if she was delegated with correct information before proceeding with her actions. As a result, although this service provider might offer excellent quantifiable QoS metrics, Alice might choose not to utilize its services in the future, because the qualifyable QoS essence was missing. To resolve this profound problem, we propose an algorithm called"Region Switching" (RS) algorithm that is capable of managing an entire multi web services composition process. RS algorithm models the composition process into a Finite State Machine (FSM) form built on top of the WS-Notification infrastructure, which provides means for message delivery. When a web services composition request is initiated, the system registers interest with all participating services via the WS-Notification standard. This will trigger notifications to be routed to all involved/registered clients whenever a change of particular interest occurs within the web service. Upon receiving a notification message, the RS algorithm identifies the web service affected by this notification, and evaluates how this will affect other web service(s). It will then issue a request to re-compute a potential subset of the entire process plan to reflect this simple, yet significant update to the user. As a result of utilizing both the RS algorithm and WS-Notification, the user is guaranteed to receive better service, leading to more accurate decision making possibilities, and keeping the satisfaction level high.

Section 5.2 starts by providing an overview of the related work, delivers insights and explores the details of the new WS-Notification standard, indicating how this will be used to accomplish our objectives. Moreover, this section presents and interesting research paper dealing with the traditional QoS functional requirements. Section 5.3 illustrates the algorithm to be used to capture the state of web services and reflect real-time information to the end user without any intervention. This is then followed with a case study evaluating the strength and correctness of the system along with some implementation consideration. The work is then extended to provide a mathematical model placing rules to manage the importance of a notification message. This would help in deciding if a

notification is worth any re-computation. Experimental results then follow to prove the effectiveness and importance of the system.

5.2 Related work

To our knowledge, little or no work has been done to provide a qualifyable end-to-end QoS to the end user. Most, if not all, web service QoS aware middleware depend on a broker to collect, select, compose, and monitor web services during the complex process of web services composition [YL04a, YL05, YL04b, TGRS04, HK04]. These brokers deal with the various issues of efficiently selecting "possibly good" web services from different pool classes, then composing and monitoring them. The act of monitoring used by these brokers, however, has been limited to the narrow field of identifying service availability, failure, and performance measurements according to some aspect(s) of the various functional QoS metrics, and trying to recover from it. Surprisingly, no broker provides the possibility of providing a higher level of monitoring, i.e., investigating web service resources, and reflecting changes to the user as they occur. This is true because until recently, existing technologies did not provide any support for such actions and requirements. Basically, most brokers depend on standardized QoS specifications reflecting various possible concerns to the end user; some specifications are HP's Open View Internet Services, IBM's Web Service Level Agreement framework (WSLA), among others. As a result, some QoS brokers depend on Service Level Agreement (SLA) to utilize and compose services, and use heuristics to present optimized QoS experience to the user [Men04]. As a general rule, brokers depend on maximizing a utility function to deliver better rates to the service user [ZBN+04, YL04b]. In [YL04b] the authors focused on designing efficient service selection algorithms. The main objective of the defined algorithms is to maximize the utility function to meet certain static and dynamic QoS requirements. In [YL04b], the problem was modeled as a Multiple Choice Knapsack Problem (MCKP) and a QoS aware broker was developed to maximize the utility function with a service selection algorithm based on Pisinger's dynamic programming technique. The elements of the utility function were based on the

service capacity, cost, and some weights and standard deviations related to cost and benefit. The authors of [YL05] proposed a web service composition middleware based on a broker capable of building an execution plan dynamically with QoS in mind. The broker maintained a database to track various QoS aspects of many web service. Information found in the database provided some statistical measurements evaluated be a utility function to help in constantly delivering an optimized solution. Another use of optimization was reported in [ZBN+04], where Integer Programming was used to deliver an optimized QoS experience to the user by choosing the best solution among all possible execution paths. AgFlow [YL04a] defines a middleware capable of delivering quality-driven composition of web services. AgFlow depends on user requirements quality constraints and the selector algorithm works to meet these requirements. It also does negotiations to guarantee the user's required QoS constraints. AgFlow is among the few systems that targets some non functional QoS requirements such as service reputation. In [ZBN+04], the authors proposed the notion of modeling a web service composition action as a state chart and then proceeded to place each state chart node in a specific region to provide the possibility of recovering from failures. In order to make web services more active in the overall QoS issue, web services need to initiate interaction in the first place. This interaction can be achieved via the WS-Notification infrastructure technology which is part of the Web Service Resource Framework (WSRF). WS-Notification is emerging as a new standard for message based notification within a Service Oriented Architecture (SOA) initiated by IBM, HP, SAP and many other information technology leaders. Although it is still in the process of standardization by OASIS [TN], many implementations exist to support different levels of this new emerging technology [M.H05]. Some of which are: GT4-Java, GT4-C, pyGrodWare, and WSRF.NET. Since WS-Notification is still novelle, its advantages for improving the QoS of web services have not yet been exposed to the best of our knowledge.

5.2.1 Web Services Notification

In this section, we provide an overview related to the WS-Notification infrastructure. WS-Notification allows the notion of a message based middleware to be interestingly materialized from the perspective of web services. This specification adapts well to the backbone of web services since it is built on top of XML, XML Schema, WSDL, and SOAP. From this stand point, a topic based publish-subscribe mechanism can be implemented between web services, where some services behave as message producers, and other behave as message consumers. Moreover, a message is generated and delivered based on a situation of a particular interest to registered subscriber(s). The WS-Notification specification family is based on three standards: WS-BaseNotification, WS-BrokeredNotification and WS-Topics. WS-BaseNotification represents the minimum requirements needed to be implemented if a publish-subscribe pattern is to be deployed over web services. NotificationProducer, NotificationConsumer, and SubscriptionManager are some interfaces that provide techniques for describing message exchange formats. The NotificationConsumer interfaces provide means to subscribe to particular producers and specify the type of message they can support. The NotificationProducer interfaces provides a formal description and modes to create and send messages to consumers. The NotificationProducer should also support exchange messages presented in the WSRF. The WS-BrokeredNotification specification states how to decouple service providers from consumers. It builds on the fact that non of the services need to know about the existence of the others, but should know about the existence of the broker. WS-BrokeredNotification defines an interface for creating a broker as an intermediator between services. Moreover, it allows messages to be initiated from non service providers as well. WS-BrokeredNotification existence depends on the usage of WS-Topics and builds on top of WS-BaseNotification. This means that it accepts/redirects messages that conform to interfaces defined within WS-BaseNotification. Both producers and consumers should subscribe to the broker before a notification message can travel along via the broker. The WS-Topics specifications specifies methods to tailor and deposit each situation of interest in specific repositories. Therefore, it creates a collection space, where updates/notifications can be stored before being forwarded to

registered notification utilizers. It allows the creation of a hierarchy of topics connected together leading to the placement of messages in more accurate containers. Moreover, it should be noted that all interfaces of the WS-Notification are XML based. Although WS-Notification was not built with QoS in mind, few aspects were identified that help in strengthening the overall quality of message delivery as follows:

- WS-Notification allows the possibility to issue "pause" and "resume" commands with the possibility of retrieving all lost messages after a "pause" command took place.
- Utilizing topics in a WS-Notification system increase scalability by allowing message filtering to take place at the level of a producer, consumer, or both.
- Filtering notification messages can be achieved using XPath, preconditions, and selector expressions. This will in turn reduce the amount of messages transmitted over the Internet and therefore improving the network bandwidth efficiency.

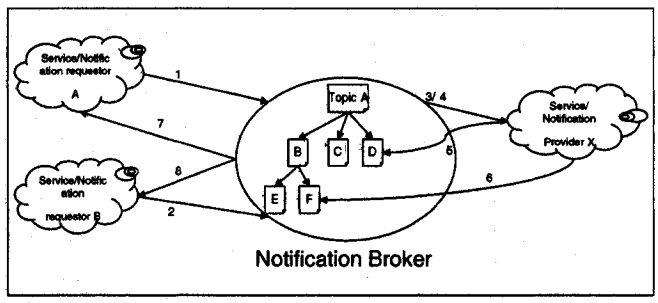

Figure 5.1: Broker based publish-subscribe pattern with topic hierarchy

In figure 5.1, we show an illustrative example, both services 'A' and 'B' request subscription for specific, yet different notifications from the same service/notification provider 'X' via the broker. Both services will implement the NotificationConsumer interface, the broker implements the WS-BrokeredNotification and WS-Topics specifications and service 'X' implements the NotificationProducer interface. The appropriate hierarchy of topics is created to reflect various domains of interest

according to different notification requestors. As notifications become ready, they are routed to their appropriate topics based on XPath, precondition, and selector expressions. Finally, the broker forwards the notifications to the services requiring them.

5.2.2 Functional QoS Requirements

In this section we tackle the issue of QoS related to web services composition from the perspective of functional requirements. Our main focus revolves around [ZBN+04]. The authors addressed the issue of QoS aware middleware over web services by utilizing the concept of Integer Programming (IP). Therefore, utility functions were devised based on functional QoS attributes. As with any optimization technique, IP tends to maximize the benefit related to the utility function. The services composition process was first modeled as a state chart. Nodes within the state chart refer to tasks rather than to actual web services and tasks indicate the type of job to be performed. Therefore, it becomes clear that there exists many web services capable of achieving the same task. The only difference between these services is the QoS level they provide. Accordingly, a mechanism should be devised to select the optimal web service to complete the task. This selection can be based on local optimum (considering the best web service within each task) or on global optimum (considering the QoS level of multiple web services across multiple tasks). Modeling the composition process as a state chart produces two new concepts. Execution paths and Execution Plans. An execution path represents the required set of tasks needed to complete a request. It also specifies the order of task execution. On the other hand, the execution plan takes the path one step further and assigns specific web services to complete each task. Thereafter, the service selection mechanism is based on the following criteria: execution price, execution duration, service reputation, service success rate, and availability.

Figure 5.2 denotes the aggregated functions utilized to compute the QoS of execution plans.

Criteria	Aggregation function
Price	$q_{pr}(p) = \sum_{i=1}^{N} q_{pr}(s_i, op(t_i))$
Duration	$q_{du}(p) = CPA(p, q_{du})$
Reputation	$q_{rep}(p) = \frac{1}{N} \sum_{i=1}^{N} q_{rep}(s_i)$
Success rate	$q_{rat}(p) = \Pi_{i=1}^{N} (q_{rat}(s_i)^{z_i})$
Availability	$q_{av}(p) = \Pi_{i=1}^{N} (q_{av}(s_i)^{z_i})$

Figure 5.2: Aggregation Functions for computing QoS of Execution Plans

Figure 5.2 depicts aggregated functions on which the service selection mechanism is based. The execution price aggregate function provides the sum of the execution prices dealing with the execution of particular web services provided by the execution plan. The execution duration if hereafter calculated via the Critical Path Algorithm [Pin01] which calculates the longest total sum of weights located on the nodes. The reputation function calculates the reputation of the execution plan by summing the individual reputation of all involved services and dividing it by number of services available in the plan. The successful execution rate function is the product of the execution rates of involved services. Here there are two type of services to consider, critical and non critical ones. The authors describe that the failure of a non critical service will lead to include a new service to execute the task without affecting the successful execution rate. On the other hand, the failure of a critical service will definitely have an effect on the successful execution rate. As a result, services are denoted as critical or not by raising them to a power of z_i, where z_i equals to 1 or 0 depending whether the service was critical or not respectively. The function modeling the availability is similar to the notion illustrated in the successful execution rate function.

The main intent of the service selection engine is to assign a service to each and every task. This can be achieved via local optimum and/or global optimum. The local optimum service selection chooses the best service to execute each task regardless how this service will affect others. Service choice is achieved by applying the Multiple Criteria Decision Making technique [HLa81]. This depends on two concepts: Scaling services and Weighting them. While scaling evaluates the quality of

each web service located for a certain task, weighting assigns a QoS value for each web service taking into consideration all its QoS criteria. Interestingly, service selection based on local optimum does not guarantee global optimum. As a result, the authors proposed the use of Integer Programming as a means to achieve a result with a higher QoS. For the IP approach, three inputs were taken into consideration. These being: Variables, Objective function, and a set of constraints. Constraints were defined over the duration, price and reputation criteria, thus providing an upper limit that should not be exceeded. Moreover, once the global plan starts execution, the state of the system is then monitored. Nodes in the state chart are assigned to regions according to their status (i.e. ideal, terminated, executing). This would then help in performing a replanning whenever a certain QoS criteria is violated. With this in mind, the replanning will affect tasks that are currently being executed. Therefore, the concept of dividing the state chart into regions provides an efficient means to perform appropriate replanning.

After presenting how QoS for web services is addressed, we tend to propose a different variant to this notion. This variant will be depicted in the following section.

5.3 Region Switching Algorithm

The following section details our intention on providing a higher QoS approach for web services composition based on the accuracy of information delivered to the end user.

5.3.1 Concept

The notion of service composition deals with the issue of creating a value added service capable of delegating/integrating multiple basic web services. Therefore, making valuable, compiled, and tailored information easier to access and obtain [BDFR03, 56401, MBE03]. The RS algorithm not only builds on top of WS-Notification, but further projects the web services composition process into a FSM representation. FSM are behavioral diagrams consisting of nodes and transitions. They

capture and model the state of the system based on various stimulating events. Nodes represent the configuration of the system at a precise time, and a transition leads to state alteration based on input events. The projected FSM includes three types of nodes, as shown in figure 5.3. The first being nodes representing initial and final states, the second type of nodes represent Web Service Nodes (WSN) representing services whose actions are needed to produce information during the composition process. Finally there exist the Integration and Distribution Nodes (IDN) representing local services that compose and tailor information obtained from WSN. These nodes also have the possibility of forwarding their outcome to one or more WSN or a termination node. As it will be illustrated, the RS algorithm delivers a stateful-aware system capable of monitoring, updating, and reflecting the current situation of any service in concern without any user intervention. The

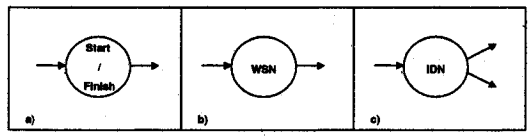

Figure 5.3: Node definition: a)Start / Finish Node b)Web Service Node c)Integration and distribution node.

system proceeds as follows: a global plan (execution path) is generated; it specifies all services to be utilized along with the appropriate sequence of execution. Then this global plan is projected to an FSM form, where each node corresponds to a service. When the global plan starts its execution, different nodes/services will belong to different regions according to their execution status. This is true because while some nodes/services are executing, other will be waiting to have all their input ready. Moreover, since these services implement a publish-subscribe pattern, then a notification message will be generated if a change in the service status/information set took place. When the system captures this message, it will assess how this change affects the service that generated the message and how this will have a propagation effect on services that are beyond the point of change. All of this is computed according to regions to which each node belongs. Moreover, such change might alter the status of other services and this is captured, reflected, and corrected by reassigning

the nodes to their appropriate new regions. This will allow the system to identify which service(s) to re-compute and which service(s) remain intact. Utilizing such marking schema introduces the possibility of re-executing a partial subset of the entire global plan, thus delivering a higher QoS to the user.

5.3.2 Notations

Here we identify the notations used by the RS algorithm. We first define the FSM quintuple, then we illustrate the various regions to which nodes belong, and finally we identify further functions that should be applied. The FSM is a quintuple $< \zeta, S, S_0, \delta, \tau >$, where:

- ζ: Represents the condition to transition from N_i to N_{i+1}. Transitions take place iff $FIN(N_i) = true$, where $FIN(N_i)$ is a boolean function indicating whether service N_i terminated its execution or not.
- S: Represents the set of all nodes in the FSM.
- S_0: Represents the initial service to be executed.
- δ: Represents the state transition function : ζ x $S \longrightarrow S$.
- τ: Represents the final service executed which delivers the end result to the user.

Each node in the FSM should belong to one of the following regions:

- R_α: Set representing nodes/services that have completed execution.
- R_β : Set representing nodes/services that are being executed.
- R_σ: Set representing nodes/services that have not started their execution.
- R_ω: Set representing junction nodes (Integration and Distribution Node) that received part but not all of the required information to start executing; therefore such nodes enters into a waiting phase.

Finally, we define the meaning of variables and functions used as follows:

- N_i: Represents a service node in the FSM.
- μ: Represents a notification message received at service node N_i.
- $P = path(N_p, N_r, N_i)$: Function *path* will identify the set of all nodes belonging to the path between two junction points containing service node N_i; N_p and N_r are junction nodes on the left and right of service node N_i respectively.
- N_i^{+1}: Corresponds to the set representing N_i's first successor node(s).
- N_i^{-1}: Corresponds to the set representing N_i's first predecessor node(s).
- $R(N_i) = R_\alpha | R_\beta | R_\sigma | R_\omega$: This function returns to which region a node belongs.

5.3.3 Rules for Re-Computation

In this section we identify the rules used by the RS algorithm to evaluate and update the region to which each node belongs to. The rules are specified in the form of first order predicate logic.

Rule 1: Change in the status of a node receiving notification from WS-Notification System.

- 1.1 ($(N_i \in R\alpha) \wedge \mu) \longrightarrow R(N_i) \in R_\beta$
- 1.2 ($(N_i \in R_\beta) \wedge \mu) \longrightarrow (N_i \in R_{\beta'})$
- 1.3 (($N_i \in R_\sigma) \wedge \mu) \longrightarrow (N_i \in R_\sigma)$
- Rule 1.1 indicates that if a service node that has terminated its execution (i.e., belongs to the R_α region) receives a notification, then this node will transition to the R_β region, and starts executing again.

- Rule 1.2 indicates that if a service node is being executed and receives a notification, it will stay in its region but will re-initiate its execution.
- Rule 1.3 indicates that if a node did not start executing, then any change received will be disregarded and will not cause the node to change its state. Since at the current moment, this service belongs to the exterior of the execution boundary.

At this point, it should be noted that only one of these rules will be true and executed at one time since a node can only belong to one region at any single time.

Rule 2 : Change in the status of nodes residing between two junction points

- 2.1 $P = path(N_p, N_r, N_i)$
- 2.2 $\forall((N_a \in P) \wedge (N_a \leq N_i^{-1})) \longrightarrow R(N_a) = R(N_a)$
- 2.3 $\forall((N_h \in P) \wedge (N_p^{+1} \leq N_h \leq N_r^{-1})) \longrightarrow R(N_h) = R_\sigma$
- Rule 2.2 indicates that all nodes preceding N_i will remain in their current region.
- Rule 2.3 indicates that all service nodes after N_i and before the next junction point will belong to the new region R_σ, that is they are not yet executed.

It should be noted that the algorithm should sequentially execute all three rules(2.1, 2.2, 2.3), since this will ensure that all nodes on the affected path will be reassigned to belong to the correct regions. This will in turn identify the propagation effect of the notification message on more than one service.

Rule 3 : Change in the status of a junction node.

- 3.1 $R(N_r) = R_\alpha \longrightarrow R(N_r) = R_\omega \wedge (\forall N_a \in \forall path(N_r^{+1}, N_{terminal}), R(N_a) = R_\sigma)$

- 3.2

  $i\ \exists N_b \in N_r^{-1} \wedge (R(N_b) = \alpha) \wedge (|N_b| < |N_r^{-1}|) \longrightarrow R(N_r) = R_\omega$

  $ii \forall N_b \in N_p^{-1} \wedge (R(N_b) = \alpha) \longrightarrow R(N_r) = R_\beta$

- 3.3 $R(N_r) = R_\omega \wedge \exists N_m.((N_m \in N_r^{-1}) \wedge (R(N_m) = R_\alpha) \longrightarrow R(N_r) = R_\omega$

- 3.4 $R(N_r) = R_\omega \wedge \exists N_m.((N_m \in N_r^{-1}) \wedge (R(N_m) \neq R_\alpha) \longrightarrow R(N_r) = R_\sigma$

- 3.5 $R(N_r) = R_{sigma} \longrightarrow R(N_r) = R_\sigma$

- Rule 3.1 indicates that if a junction node terminated it execution and a service notification occurred before it, then all nodes after this junction node up to the termination node should transition to the R_σ region.

- Rule 3.2-*i* indicates that if some but not all N_b nodes have completed their execution, then the junction node will enter the R_ω region, thus it will keep on waiting until all it predecessors complete their execution. On the other hand, Rule 3.2-*ii* indicates that all predecessor nodes to N_r terminated their execution, then this junction node will start execution and will therefore enter the R_β region.

- Rule 3.3 indicates that if a junction node N_r belongs to R_ω and there exist at least one node N_r^{-1} that has terminated, then N_r stays in the waiting region R_ω.

- Rule 3.4 indicates that if a junction node N_r belongs to R_ω, and there doesn't exist any node N_r^{-1} that has terminated, then N_i moves to the region R_σ.

- Rule 3.5 indicates that if the junction node N_r was initially in the not yet region,R_σ when the notification took place, then it will remain in this region.

It should be noted that only one rule from Rule 3 should be executed at any time. Compiling all that have been presented, we would like to illustrate that achieving a stateful-aware system

is accomplished by modeling the web services composition process as a finite state machine and assigning each node to a set of four different regions. The capability of active monitoring is achieved by implementing a publish-subscribe pattern materialized via WS-Notification infrastructure. This makes web services more active during the entire composition process allowing them to signal any change that is of interest to subscribers. Since a message delivery corresponds to change in status, then the process of updating the status of the FSM nodes is achieved via Rules 1,2, and 3 of the RS algorithm. This in turn will effectively re-compute affected web services and reflect the new updates to the end user. All of this is computed in a seamless manner from the end user whose sole duty is to initiate the composition process.

5.3.4 Case Study

A detailed case study is illustrated demonstrating the power and effectiveness of the RS algorithm. Along the scenario of services composition, consider that the services composition global plan is generated, the corresponding FSM is computed, registration to the services notification system is completed and execution commenced. As a result, the FSM will look like in figure 5.4, showing the region to which each node belongs.

The FSM of figure 5.4 is defined as follows:

- ζ: Corresponds to the $FIN(N_i)$ function.
- S: $\{Start, A, B, C, D, E, F, G, H, I, Finish\}$.
- S_0: $\{Start\}$.
- δ: ζ x $S \longrightarrow S$.
- τ: $\{Finish\}$

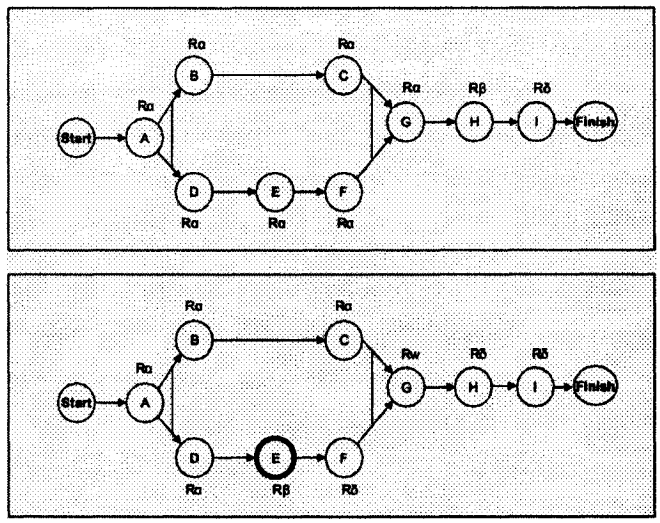

Figure 5.4: Case study: a) Indicating the initial global FSM created by the system b) Indicates the newly updates regions in the FSM after receiving a notification at node 'E'.

Figure 5.4 shows an entire FSM as the result of the global planning performed to produce a web services execution process. Figure 5.4-a, shows in details the region to which each and every node belongs (taking into consideration that execution already was initiated). Considering the present state of each node in Figure 5.4-a, a change in the data set of service E triggers this service to send a notification message to its subscribers. At this point, the WS-Notification System of our framework will capture this message and forward it to the execution engine which will redistribute the nodes to their new region. As a result, Figure 5.4-b will represent the new status of the FSM, thus identifying that a change at node/service "E" will have a propagation effect on nodes/services $\{E, F, G, H\}$.

The re-computation is as follows:

- Node E $\in R_\beta$ by rule 1.1.
- $P = \{A, D, E, F, G\}$ computed by rule 2.1.
- Node A $\in R_\alpha$ by rule 2.2.
- Node D $\in R_\alpha$ by rule 2.2.

- Node F $\in R_\sigma$ by rule 2.3.

- $R(G, H, I) \in R_\sigma$ by rule 3.1.

As a result, only service E,F,G and H will be re-computed while services A,B,C,D and I will remain intact. Therefore a system implementing the RS algorithm along with WS-Notification will yield the following advantages: First, it will always deliver accurate and contemporary information to the end user. Second, it does not require any user intervention. Third, it re-computes a minimal set of affected services and thus it will not overload servers with unneeded service executions, and will not overload the bandwidth with redundant information that has no added value. Fourth, it can be considered as a trust worthy source of information delivered as input to various systems that function based on real time data.

5.3.5 Implementation

Despite the fact that WS-Notification is still under standardization by OASIS, some implementations do exist. Our implementation is based on the open source ServiceMix WS-Notification implementation [Ser]. This ServiceMix implementation is based on an open source implementation Java Message Service (JMS) called ActiveMQ. The MessagibgActiveMQNotificationBroker interface is used to establish connections to an ActiveMQ broker. In addition to that, the service composition and execution is performed using the Business Process Execution Language for Web Service Composition (BPEL4WS). The only overhead is that a different BPEL4WS file should be created starting from each web service involved. This would alow the re-execution of a subset of the entire plan. The performance of the system is related entirely to the underlying network. The duty of our proposed system is to identify the point of change and initiate the execution of a new BPEL4WS specifications. Publishing and subscribing is conducted as shown below:

Publish:

$$EndPointReferenceTypeconsumer = createEPR(ReceiverComponenr.SERVICE,$$

$ReceiverComponent.ENDPOINT$);

$wsnBroker.subscribe(consumer, "myTopic", null, true)$;

$Elementbody = parse(" < update > newMsg < /update > ")$;

$wsnBroker.notify("myTopic", body)$;

Subscribe:

$PullPointpullPoint = wsnCreatePullPoint.createPullPoint()$;

$Subscriptionsubscription = wsnBroker.subscribe(pullPoint.getEndpoint(), "myTopic", null)$;

In the coming sections, we further present an extension to the system using a mathematical model. This model delivers and added QoS layer complementing the work of the RS algorithm.

5.4 The Model: An Extended QoS Layer

Consider a complex web services composition scenario that involves services that requires a significant amount of time to deliver results. In addition to that, consider that the environment of operation is constantly changing, i.e. information changes frequently and as a result, more notifications will be issued. This would lead to frequent re-computation. The question that comes to mind is: Are these re-computation necessary? Would they introduce any change? Can we limit re-computation according to some predefined factors?

What follows will provide answers to these posed questions. Figure 5.5 is an example showing a complex web services composition plan. This plan exists within the framework explained earlier. There exists a message based broker which routs notifications to appropriate services. As indicated in the figure, node i is where a notification message is received. As a result of applying the *RSalgorithm*, the highlighted region will be recomputed. This means that 61% of the global plan needs to be re-evaluated. Consider that three of these nodes that need re-computation require a lot of time to deliver results. Also consider that the final result after the re-computations terminates

turns out to be the same/similar to the previous result. Baring this in mind, it is acknowledged that a huge amount of time was wasted without any significant change or added value.

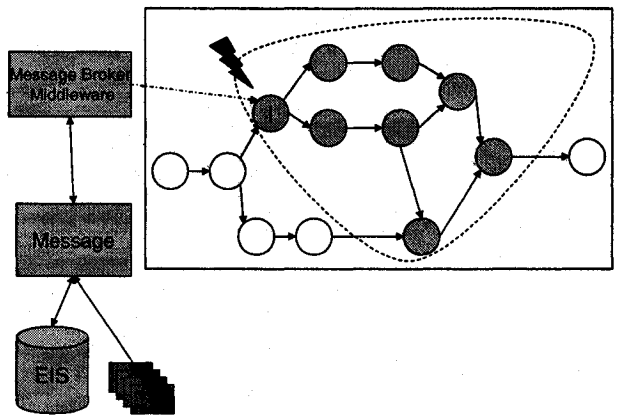

Figure 5.5: Highlighted are affected nodes due to a notification message.

Therefore, our objective is to perform re-computing only when necessary. We intend to devise a model which would decide when to re-compute. Three important factors are identified that play a major role in affecting the outcome of computations. These factors are:

1. Size : Size of notification message with respect to the old one.
2. Range: Change in the range of information.
3. Type : Change in the type of information.

If a received notification message has a significant change in size with respect to the old message, then this would imply that the information has changed significantly and as a result the end result can also be altered. A change in information range may also alter the end result; for example, a change in the wind speed of a hurricane might or might not change the category to which this hurricane belongs. Finally, the type of information delivered is also important. If the type has changed from one kind of message to another, then there might be a need for a new re-computation.

Presenting it in a more formal way, we provide the following functions:

$$\mathsf{f(S,S}_i\mathsf{)} = \mathrm{a} \times \mathrm{S} + imp(S_i) \begin{cases} 1;\ if\ \mathsf{f(S)}\ \geq\ Th_s \\ 0;\ if\ \mathsf{f(S)}\ <\ Th_s \end{cases}$$

$$\mathsf{f(R,S}_i\mathsf{)} = \mathrm{b} \times \mathrm{R} + imp(S_i) \begin{cases} 1;\ if\ \mathsf{f(R)}\ \geq\ Th_r \\ 0;\ if\ \mathsf{f(R)}\ <\ Th_r \end{cases}$$

$$\mathsf{f(T,S}_i\mathsf{)} = \mathrm{c} \times \mathrm{T} + imp(S_i) \begin{cases} 1;\ if\ \mathsf{f(R)}\ \geq\ Th_r \\ 0;\ if\ \mathsf{f(R)}\ <\ Th_r \end{cases}$$

where S, R, and T are coefficients assigned according to the importance of each factor. It is important to mention that these values are context specific. Each organization would provide coefficients that are well suitable to capture their business process. Moreover, thresholds for all factors should be defined. If the result of any function exceeds the predefined threshold, then it is worthwhile to re-compute. Function $imp(S_i)$ provides the weighted average of each service within the composition process. More precisely, it identifies the importance/involvement of each node within the entire process. As a result, the three defined functions take into consideration the following factors:

1. Size, Range, Type of message and information.
2. Coefficient factors devised be the organization.
3. Thresholds also devised by the organization.
4. The involvement and importance of each node within the global plan.

These functions provide more insight to whether a notification message is of importance or not. We now define the function *Decision* which gives the order to re-computer or not.

$$Decision = f_{SIZE} \vee f_{RANGE} \vee f_{TYPE}$$

This new function takes into consideration all factors of change combined together. As a result, if $Decision = 0$, then a re-computation will not necessarily introduce a significant change to the end result. On the other hand, if $Decision = 1$, then a re-computation request should be triggered because the notification message identified that a change in information exceeded a predefined threshold of safety and a re-computation is needed to ensure that the end result is as accurate as possible.

5.4.1 An efficient Messaging Design

The total efficiency of the system depends on the structure of a notification message. With some educated designs, the overall efficiency can be significantly enhanced. There are two types of notification messages. Simple message and complete message.

5.4.2 Simple Message Notification Structure

A notification message with a simple structure is a message with very simple and primitive set of information. The information encapsulated within this message is limited to indicate that a change took place. It provides no additional details. Thus upon receiving this kind of message, the system would have to perform the following steps:

1. Time needed to prepare and send request for service re-execution, denoted by T_{prep} and T_{send} respectively.

2. Time needed to wait for service re-execution and for result delivery, denoted by T_{wait}.

3. Time needed to process result then apply the above stated functions to decide whether a re-computation is required, denoted by $T_{process}$ and T_{decide} respectively.

The end result might or might not recompute any subset of the global plan. Therefore the total time required to decide if a re-computation is required is:

$$T_{total} = T_{simple} = T_{prep} + T_{send} + T_{wait} + T_{process} + T_{decide}$$

If the decision was to re-compute a designated subset of the global plan, then the total time required to re-evaluate the entire result is:

$$T_{total} = T_{simple} + \sum_{i=0}^{n} T(s_i)$$

If the "Decision" function decides that there is not need to re-compute since no threshold has be exceeded, then the total time required to re-evaluate the entire result converges to:

$$T_{total} = T_{simple}.$$

5.4.3 Complex Message Notification Structure

A better approach can be implemented in order to reduce the time overhead introduced by the simple message notification structure. Instead of sending a notification message with little information within, it would be much better to encapsulate more data within a message that would help the remote end system decide in a more efficient manner. The kind of information required is to indicate whether the change was of type Size, Range, or Type. Along with that, the appropriate values should be encapsulated as well. As a result, upon receiving the message, the system will perform the following steps:

1. Time needed to process message, denoted by $T_{process}$.
2. Time needed to decide, denoted by T_{decide}.

Therefore the total time required to decide if a re-computation is required is:

$$T_{total} = T_{complex} = T_{process} + T_{decide}$$

If the decision was to re-compute a subset of the global plan, then the total time required to re-evaluate the entire result is:

$$T_{total} = T_{complex} + \sum_{i=0}^{n} T(s_i)$$

If the "Decision" function decides that there is no need to re-compute since no threshold has be exceeded, then the total time required to re-evaluate the entire result is:

$$T_{total} = T_{complex}.$$

5.4.4 Evaluation

After illustrating the two methods of message structure, it becomes clear that $T_{simple} >>> T_{complex}$ since it encapsulated a lot of overhead that is not necessary. In the simple messages design, we always need three message transmission back and forth. Therefore, less time overhead is noticed. With complex messaging system, things are different. The number of messages send and received by the system depends on the outcome of the *Decision* function. If a re-computation is needed, then the number of messages exchanged is three. But when no re-computation is required, then only one message is sent to the node. As a result, if no re-computation is required, then traffic on the network is reduced by 33% . So it all depends on the probability that a re-computation request will be initiated. Regards of that, it is clear now that utilizing a complex message structure would result in a higher QoS.

5.4.5 Simulation

The following case study will further evaluate the efficiency of the proposed model and will investigate its delivered added values.

We all have witnessed the recent devastating hurricane season that hit the American gulf coast lately. Since then, it has become clear that if there were accurate information at the right time, a huge segment of the disaster could have been avoided, hundreds of lives could have been saved. One of the main problems encountered during hurricane Katrina [Kat] was the lack of real time information concerning the hurricane and how it would have an impact on other resources. Taking

this into consideration, a hurricane center simulation was developed based on the model defined. This simulator would constantly receive messages about a change concerning a specific hurricane. Depending on the type and information content of the message, a decision to re-compute or not would be triggered. This decision would be based on predefine thresholds that will be illustrated hereafter.

The simulation isconducted using four separate components each providing a different service. Moreover, these services can be composed to provide a well defined structure that can help in the unfortunate event of a hurricane. The four services are the following: Service HurricaneInfo is the most important service as it implemets logic defined in our model. The service exposes a functionality being "analyzeChange" that takes as input two hurricane information and analysis the change between them. Moreover, this service keeps a flag recording the difference between the two hurricanes. This exposed service utilizes an inner method which evaluates the flag and captures whether or not thresholds are exceeded. Thresholds can exceed the three predefined attributes (Size, Type, and Range). The second web service is called Analysis. It exposes a method which checks the severity of a hurricane based on its category, current location and its direction. Web service EmergencyResponders exposes a method which identifies the number of available emergency troops in a given location. Finally, web service SituationRoom exposes a method, which based on the outcome of all previous services, would issue a final decision indication if this hurricane is of any danger or no.

In figure 5.6, the entire service integration process is made visible. For simplicity reasons, only the Range threshold was considered during the simulation. Two runs were made, one establishing the Range threshold to one (1), and the second establishing the Range threshold to two (2). The results of both simulations are depicted in figure 5.7 shows the results of the system when configured to two different thresholds. When the Range threshold was set to 1, the system responded to 25 complex message notifications by 25 re-computations (100% of input messages). Out of these 25 re-computations, 20 were redundant, i.e. 80% of the re-computations did not produce any added value (lines 4-6,8-13,15,17,18,20-27). Therefore many resources and time was wasted. Alternatively, when

the Range threshold was set to 2, out of the 25 input messages, only 9 re-computations were necessary. 2 of these 9 re-computations were redundant(lines 20 and 23), i.e 22.22% reproduced known results. Moreover, a third simulation was conducted which yielded some interesting results. While the input data remained the same, the system was modified to re-compute when the Range attribute is greater or equal to three (3). The following was documented. Out of the 25 input messages, only the first one was regarded leading the system to say "No Danger". All other messages showed changes, where the Range threshold was not affected to go beyond the newly assigned threshold. From the previous results, system status such as "Alert,Troops stay alert" and "Red Alert, request more troops" are not evaluated. As a result, this demonstrated that the choice of a threshold is a very critical procedure. This can be clearly demonstrated from chart 5.8 which shows the number of re-computations performed related to each newly assigned threshold and consequently, it shows the number of redundant re-computations. Therefore, by setting more accurate thresholds, the system can achieve more efficiency and can significantly enhance bandwidth utilization, reduce resource exhaustion, and time. Moreover, chart 5.9 shows how much the system is accurate in responding to notification messages. It becomes clear that not any value of thresholds can be assigned. There is a limit that can not be exceeded otherwise the system accuracy would suffer a lot. It should be noted that the first two simulations produce the same results but one has more redundancy than the other, requires more time, and leads to useless and excessive resource exhaustion. It should also be noted that there is an upper bound on threshold ranges. If this bound is exceeded, then the system will be resistant to change and will mask notification messages. As thresholds increase, little but significant changes will not be taken into consideration.

5.5 Conclusion

This chapter lays the foundation for exploring the qualitative aspect of web services composition, thus looking beyond other important quantitative metrics. This in turn lead to the production of a system capable of providing a complete end-to-end higher QoS to the user utilizing both the RS algorithm and WS-Notification. In this procedure, we raised the level of monitoring from being passive to active, i.e. monitor information content rather than simply checking web services availability as performed by other systems. This enhancement yields significant payoffs to other systems that heavily depend on real-time accurate data. Better Decision Support Systems (DSS) for example can be built on top of our system which can produce better analysis and optimization scenarios. Further research can be conducted illustrating the usage of WSRF to provide a higher layer of QoS since it allows the possibility to capture and manage the state of web services as well as their resources. Moreover, integrating this system with existing QoS aware brokers for web services composition and evaluating the over all QoS measured from both the qualitative and quantitative perspective is a matter of concern. Other research area can be conducted to see how this can integrate well and enhance grid computing. A additional level of QoS was also administered. The developed system proves the effectiveness of the proposed system by providing system evaluation metrics. The combination of the RS algorithm along with thresholds on re-compution, the over all performance and accuracy of the system was leveraged. By utilizing these methodologies, more trust can be obtained from the system. This would lead the foundation for an infrastructure for decision support systems based an up-to-date accurate information.

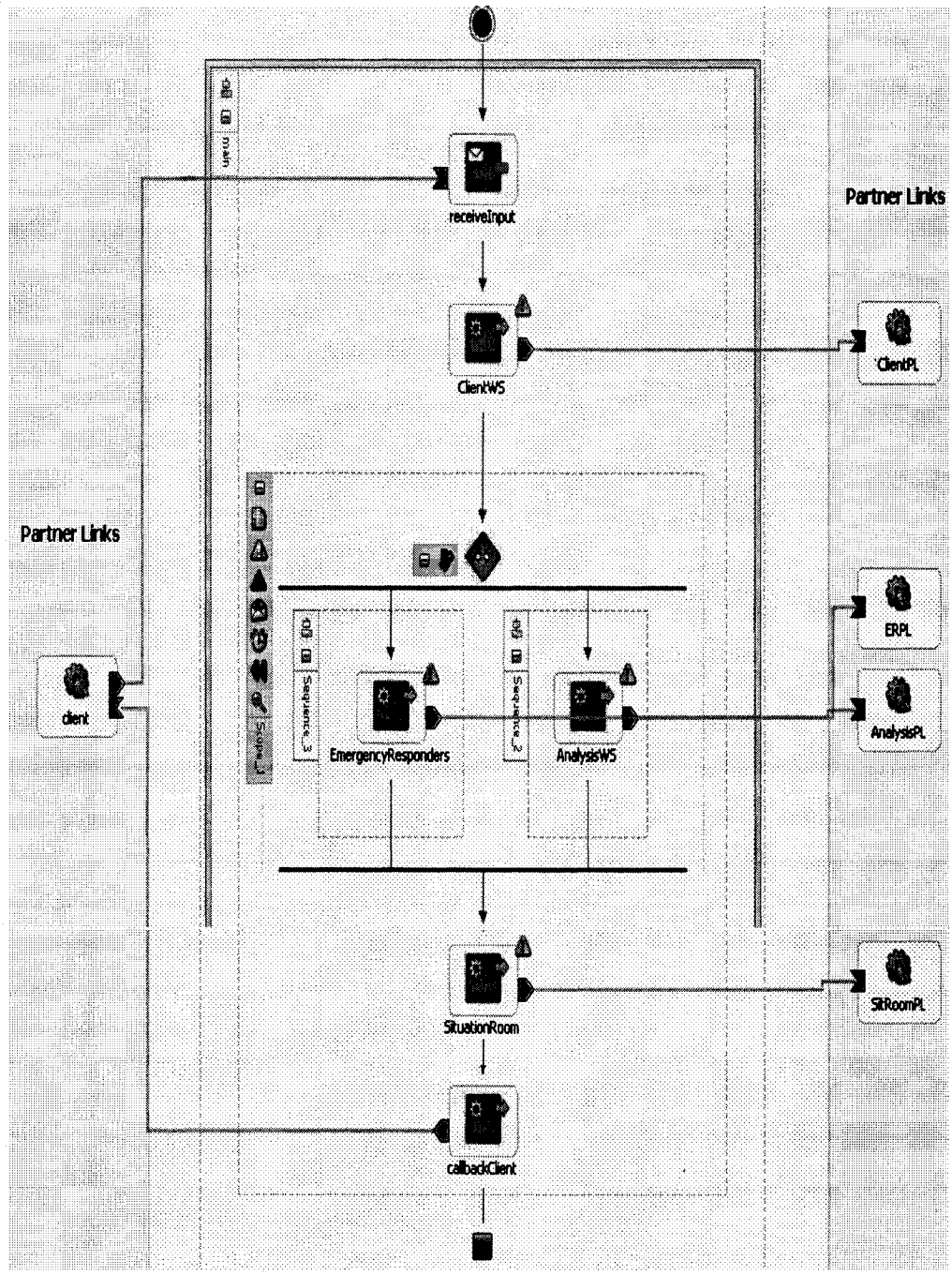

Figure 5.6: BPEL design showing multiple service integration.

```
1  //Range threshold =1
2
3  No danger
4  No danger
5  No danger
6  No danger
7  Alert,Troops stay alert
8  Alert,Troops stay alert
9  Alert,Troops stay alert
10 Alert,Troops stay alert
11 Alert,Troops stay alert
12 Alert,Troops stay alert
13 No danger
14 No danger
15 No danger
16 Alert,Troops stay alert
17 Alert,Troops stay alert
18 Alert,Troops stay alert
19 Red Alert, request more troops
20 Red Alert, request more troops
21 Red Alert, request more troops
22 Red Alert, request more troops
23 Red Alert, request more troops
24 Red Alert, request more troops
25 Red Alert, request more troops
26 Red Alert, request more troops
27 Red Alert, request more troops
```

```
1  //Range threshold = 2
2
3  No danger
4  Notification received, NO RECOMPUTATION NEEDED
5  Notification received, NO RECOMPUTATION NEEDED
6  No danger
7  Alert,Troops stay alert
8  Notification received, NO RECOMPUTATION NEEDED
9  Notification received, NO RECOMPUTATION NEEDED
10 Notification received, NO RECOMPUTATION NEEDED
11 Notification received, NO RECOMPUTATION NEEDED
12 Notification received, NO RECOMPUTATION NEEDED
13 No danger
14 Notification received, NO RECOMPUTATION NEEDED
15 Notification received, NO RECOMPUTATION NEEDED
16 Alert,Troops stay alert
17 Notification received, NO RECOMPUTATION NEEDED
18 Notification received, NO RECOMPUTATION NEEDED
19 Red Alert, request more troops
20 Red Alert, request more troops
21 Notification received, NO RECOMPUTATION NEEDED
22 Red Alert, request more troops
23 Red Alert, request more troops
24 Notification received, NO RECOMPUTATION NEEDED
25 Notification received, NO RECOMPUTATION NEEDED
26 Notification received, NO RECOMPUTATION NEEDED
27 Notification received, NO RECOMPUTATION NEEDED
```

Figure 5.7: Simulation results.

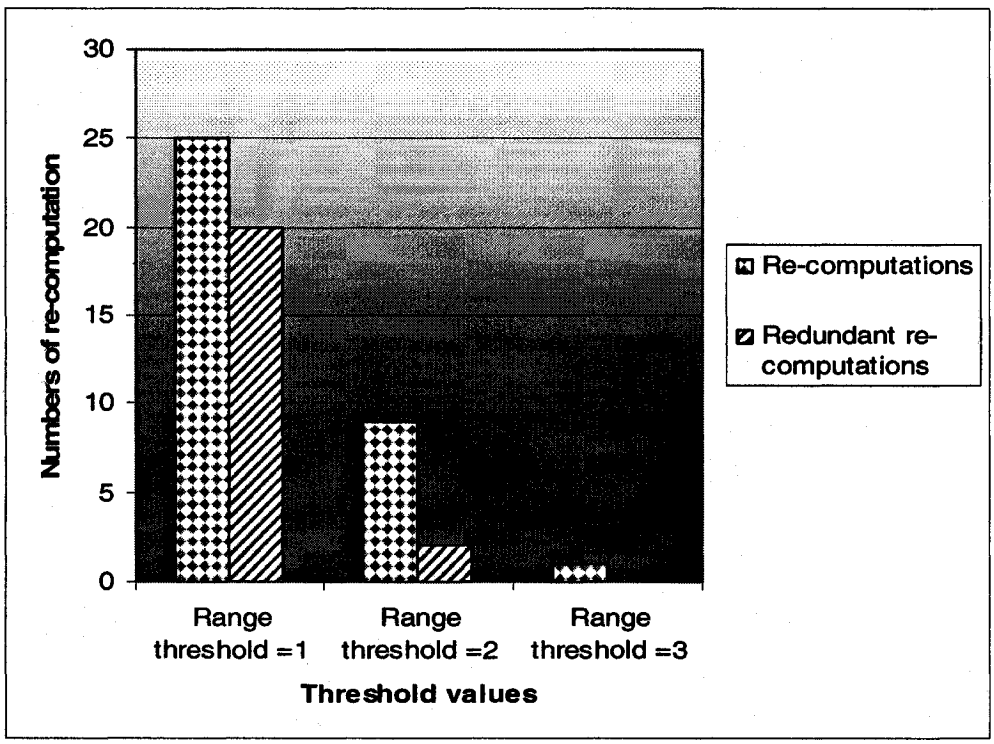

Figure 5.8: Chart showing the number of re-computations performed.

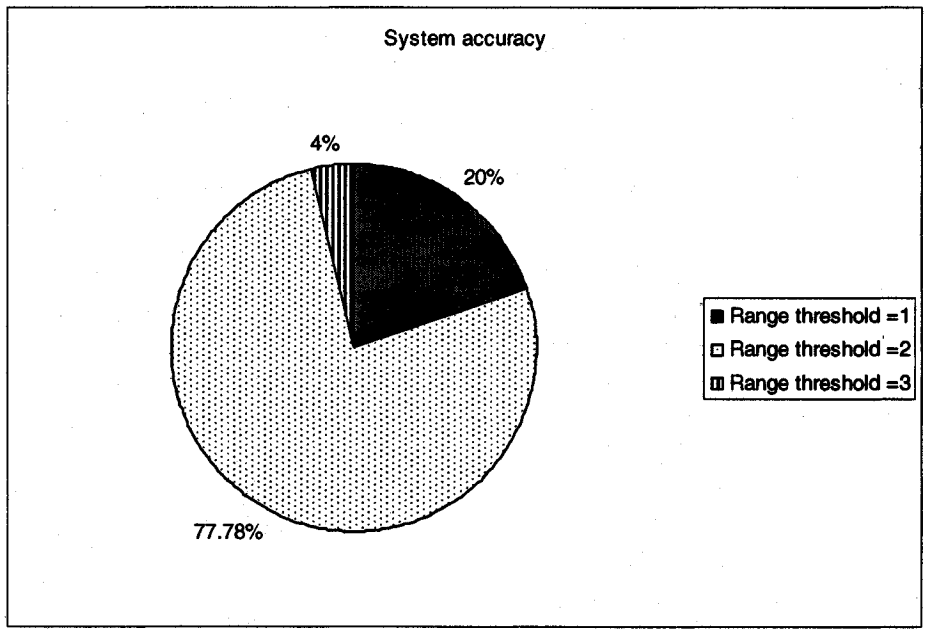

Figure 5.9: Chart showing the accuracy of the system.

Chapter 6

Conclusion

The tremendous research, work, coding and simulation lead to the development of the "Digital Cockpit, a message based middleware for realtime information systems integration". The bag of challenges were numerous and existed during the entire engineering phases of this system. To state some, surveying existing technologies along with their potentials and limitations, and understanding existing legacy databases and extracting useful displays as well as analysis scenarios served as big challenges. Moreover, identifying security requirements and constraints posed some significant alterations in the design and development of the system. Last but not least, the main and biggest challenge was the development of the entire graphical user interface of the "Digital Cockpit". This task went through numerous cycles of refinements that led to the final production of the system as it was presented in earlier chapters. Moreover, we targeted the issue of QoS from a fresh perspective dealing with the core concept of "How accurate is the information". This is achieved by identifying that the new WS-Notification standard can be utilized to deliver a better QoS experience to web services implementing it. In addition to that, we devised the "Region Switching Algorithm" as a base infrastructure needed to aid in the issue of web services composition and QoS.

Each chapter presented its unique set of contributions. As a whole, we went through the entire software engineering phases dealing with requirements, analysis, design and development of the

system. This was then followed by a detailed comparison between MoM and RPC based systems. Moreover, the work was extended to take into consideration the issue of web services integration within the "Digital Cockpit" system with various qualitative QoS considerations, which is a unique approach to deal with the issue of QoS. Simulation results of the QoS system along with the set of functions guarding thresholds prove the effectiveness of the system and displays its accuracy.

The work presented here delivers an interesting fusion of two important information science topics that are of primary interest of people in the academia and the industry. These being "Information Integration" and "Decision Support" systems. The digital cockpit delivers both and this is what makes an infrastructure on which more complex systems can be built on top of it.

Future work can span across many paths, to state a few. Some considerations to enhance the "Digital Cockpit" can revolve around the following topics. Security considerations when accessing local and remote information should be well managed. Access control list assignment is also of prime importance in such a middleware. Moreover, dynamic query building i.e XML based queries, and execution from the client side would also provide an added value. But it should be noted that this poses significant security considerations. More work can be conducted in terms of providing a collaborative decision support mechanism. One potential direction is the use of collaborative game theory since it provides the bases for such work. Moreover, game theory if witnessing a wide spread utilization among intrusion detection system so its utilization in this domain would significantly enhance the system. As for the work related to the qualitative QoS, some future work could focus on the following: Such systems can deliver an added value to grid computing so this should be taken into consideration. Moreover, the system can be significantly enhanced if a method is devised to help in the correct assignment of threshold value to items of considerations within the organization.

Appendix A

Use Cases of Digital Cockpit

Following is a comprehensive list of all use cases that should exist in a system like "Digital Cockpit". Each section illustrates possible use cases that should be considered when implementing similar modules.

A.1 Display Module Use Cases

Below is a list of possible use cases that should be accompanied with a Display Module.

1. Initialize the Display Module.
2. Change the properties of the Display Module.
3. Add/remove a component to the Digital Cockpit.
4. Alter visual representation of a graphical component.
5. Drill in/out component details.
6. Import/export component data.

A.2 Subscription Module Use Cases

Below is a list of possible use cases that should be accompanied with a Subscription Module.

1. Subscribe/unsubscribe to local/remote information sources.

2. Subscribe/unsubscribe to local/remote web services.

A.3 Integrator Module Use Cases

Below is a list of possible use cases that should be accompanied with a Integrator Module.

1. Access local/remote data.

2. Access local/remote services.

3. Publish/Subscribe data.

4. Retrieve information from distant requestors.

5. Add/remove data hook.

6. Add/remove service hook.

7. Save information to local sources.

A.4 Monitor Module Use Cases

Below is a list of possible use cases that should be accompanied with a Monitor Module.

1. Monitor for real time information change.

2. Reflect information change.

A.5 Analysis Module Use Cases

Below is a list of possible use cases that should be accompanied with a Analysis Module.

1. Retrieve data for analysis.
2. What-if scenario analysis.
3. Probability and statistical analysis.
4. Cause and effect analysis.

A.6 Control Module Use Case

Below is a list of possible use cases that should be accompanied with a Control Module.

1. Optimization of events.

A.7 Security Module Use Cases

Below is a list of possible use cases that should be accompanied with a Security Module.

1. Log-on/Authenticate user.
2. Log user/administration operations.
3. Log system events.
4. Privileged update of user information and privileges.
5. Retrieve logged events.

Appendix B

Inside look: Class Diagrams

This appendix highlights the inner working of classes involved that made the realization of the "Digital Cockpit" so real. What follows is an inside look at the structure of various class in some packages. The Display package corresponds to twenty two (22) classes all associated in a way to deliver a seamless monitoring capability. The Integrator package corresponds to seventeen (17) classes all in addition to other existing APIs communicating together in a way to integrate information in an efficient and transparent means. The Monitor package corresponds to twelve (12) classes communicating in harmony to deliver real and up to date information.

B.1 Class Diagrams and Packages

In this section, we present the class diagrams showing all packages/classes and class relationships needed to exist to deliver a comprehensive display for the digital cockpit system, the ability to integrate/access heterogenous information systems and provide a real time information monitoring and delivery. This is shown via the Display package, Integrator package, and Monitor package respectively.

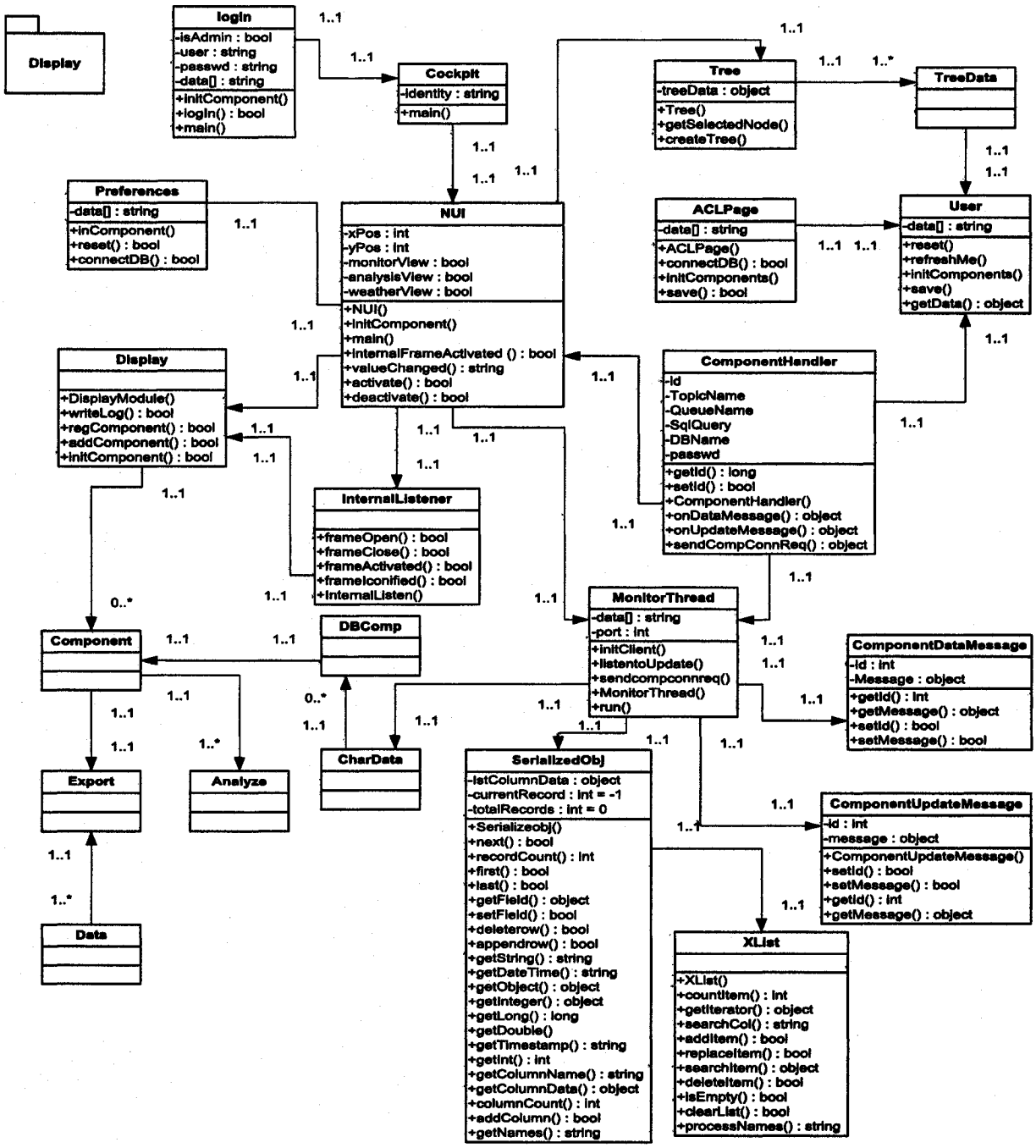

Figure B.1: Classes from the display package.

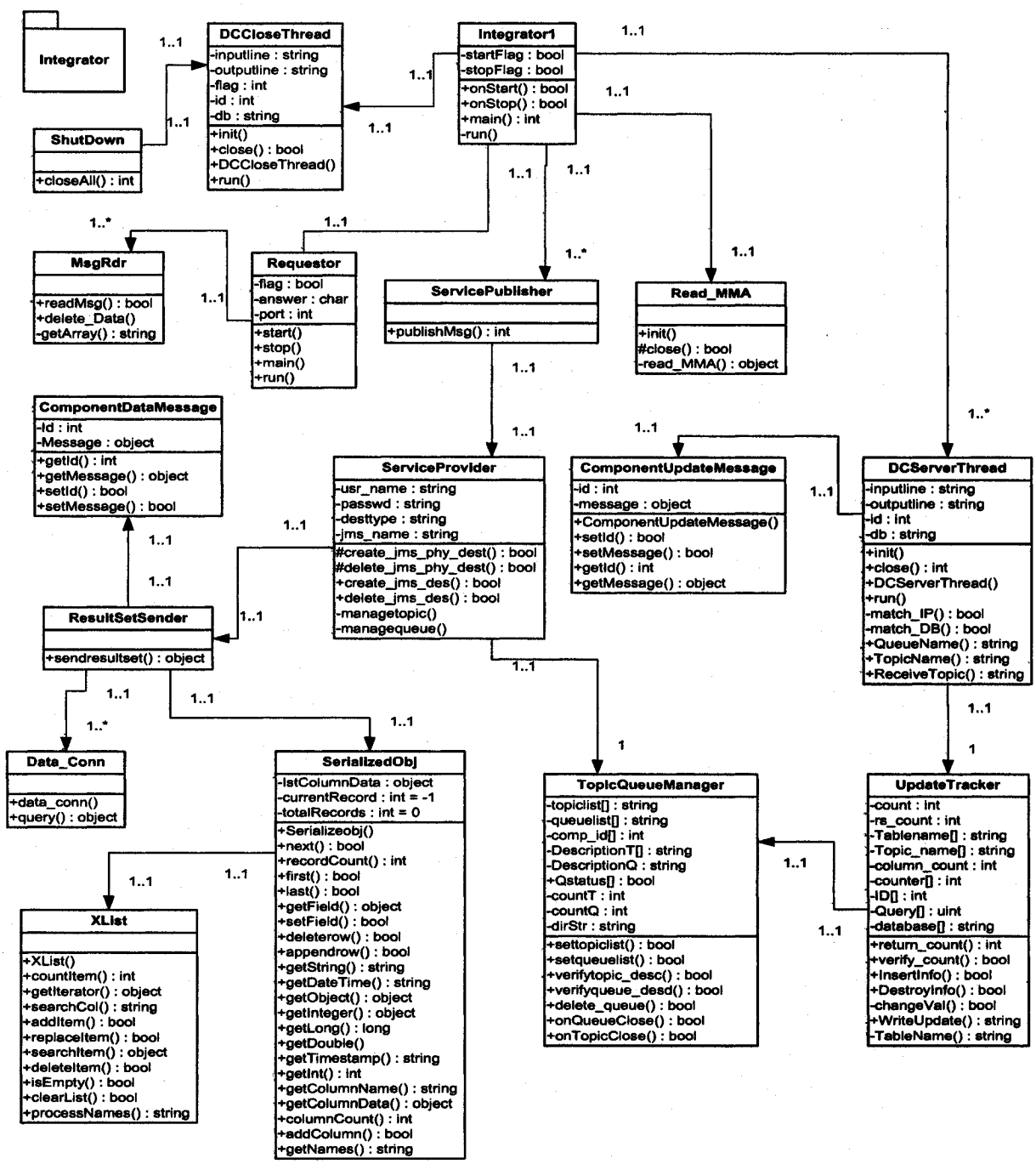

Figure B.2: Classes from the integrator package.

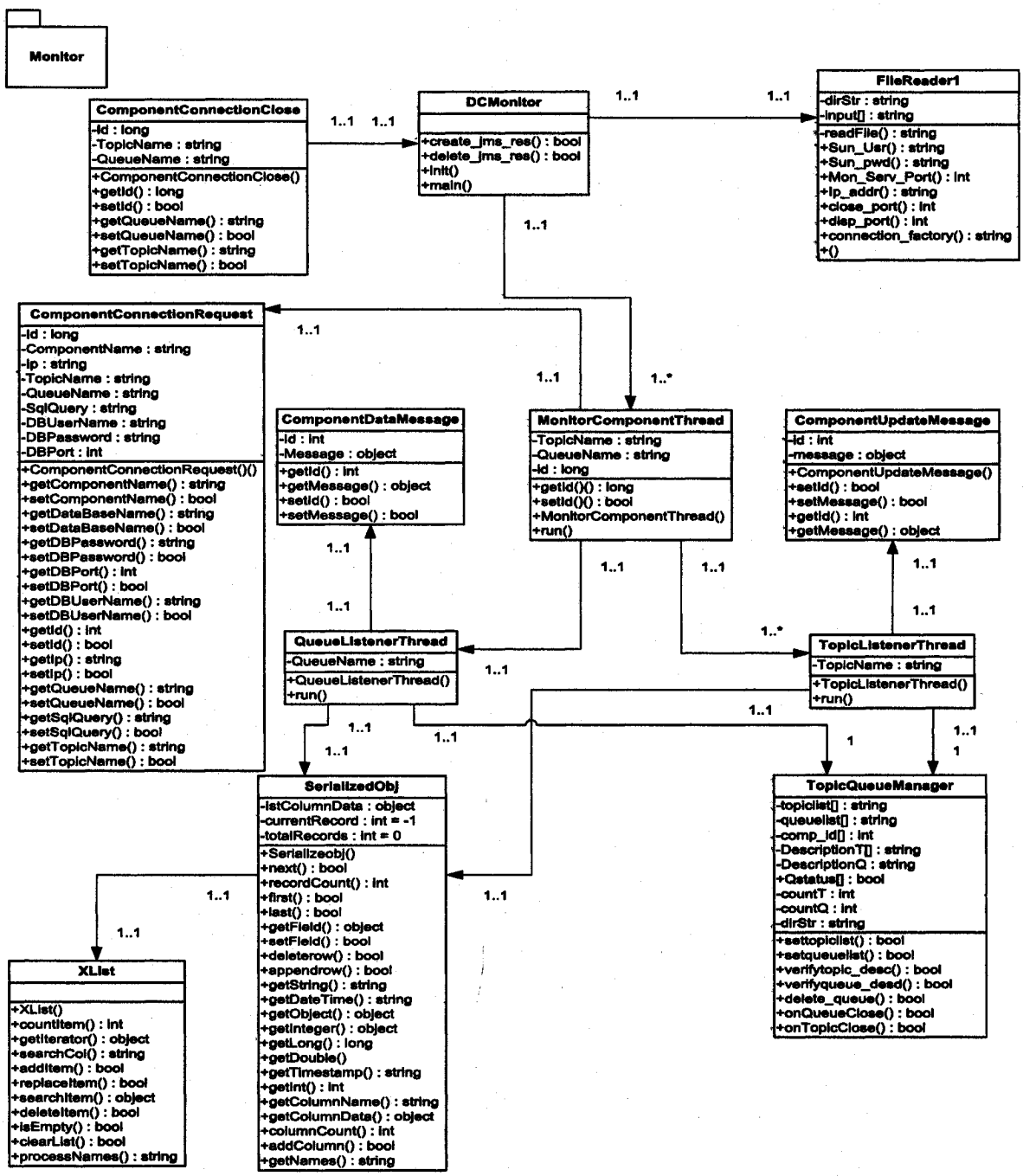

Figure B.3: Classes from the monitor package.

Appendix C

List of Acronyms

In the order of appearance.

IS: Information System

DCP:Digital Cockpit

QoS: Quality of Service

DSS: Decision Support System

SOA: Service Oriented Architecture

EDI: Electronic Data Interchange

CORBA: Common Object Request Broker

DCOM: Distributed Component Object Model

RMI: Remote Method Invocation

IIOP: Internet Inter-ORB Protocol

EIS: Enterprise Information System IT: Information Technology

API: Application Programming Interface

HTML: Hyper Text Markup Language

J2EE: Java 2 Enterprise Edition

WSLA: Web Service Level Agreement

SLA: Service Level Agreement

MCKP: Multiple Choice Knapsack Problem

EI: Enterprise Integration

MoM: Message Oriented Middleware

SI: Service Integration

RPC: Remote Procedural Call

JMS: Java Messaging Service

JDBC: Java Database Connectivity

JCA: Java Connector Architecture

WSDL: Web Service Descriptive Language

UDDI: Universal Description, Discovery and integration

SRS: Software Requirements Specifications

QFD: Quality Functional Deployment

GUI: Graphical User Interface

UI: User Interface

IGUI: Interactive Graphical User Interface

JBI: Java Business Integration

JSR: Java Specification Request

BC: Binding Component

SE: Service Engine

ESB: Enterpirse Service Bus

BPEL: Business Process Execution Language

RS: Region Switching

FSM: Finite State Machine

Bibliography

[56401] Dynamic and adaptive composition of e-services. *Inf. Syst.*, 26(3):143–163, 2001.

[BDFR03] Boualem Benatallah, Marlon Dumas, Marie-Christine Fauvet, and Fethi A. Rabhi. Towards patterns of web services composition. pages 265–296, 2003.

[CDK+02] Francisco Curbera, Matthew Duftler, Rania Khalaf, William Nagy, Nirmal Mukhi, and Sanjiva Weerawarana. Unraveling the web services web: An introduction to soap, wsdl, and uddi. *IEEE Internet Computing*, 6(2):86–93, 2002.

[enc05] Wikipedia encyclopedia. Service-oriented architecture, 2005. `http://en.wikipedia.org/wiki/Service-oriented_architecture`.

[Glo] Globus. The ws-resource framework. http://www.globus.org/wsrf/.

[Gru] Drik Grunwald. Computer performance modeling. http://moodle.cs.colorado.edu/file.php/5/operational-laws.pdf.

[HK97] Markus Horstmann and Mary Kirtland. Dcom architecture, July 1997. `hhttp://msdn.microsoft.com/default.aspx`.

[HK04] Randy Howard and Larry Kerschberg. Brokering semantic web services via intelligent middleware agents within a knowledge-based framework. In *IAT*, pages 513–516, 2004.

[HLa81] K.Yoon H.C.-L and. *Multiple Criteria Decision Making.* Springer-Verlag, 1981.

[IFT05] DONALD F. FERGUSON JEFFREY FREY STEVE GRAHAM TOM MAGUIRE DAVID SNELLING IAN FOSTER, KARL CZAJKOWSKI and STEVEN TUECKE. Modeling and managing state in distributed systems: The role of ogsi and wsrf. In *IEEE*, pages 604–612, 2005.

[Jud02] Peter Judge. .net vote rigging illustrates importance of web services. http://news.zdnet.co.uk/software/0,39020381,2102244,00.htm, January 2002.

[Jur05] M. Juric. Business process execution language for web services: A practical guide to orchestrating web services using bpel4ws, 2005. `www.ebpml.org/bpel4ws.htm`.

[Kat]

[MBE03] Brahim Medjahed, Athman Bouguettaya, and Ahmed K. Elmagarmid. Composing web services on the semantic web. *The VLDB Journal*, 12(4):333–351, 2003.

[Men02] Daniel A. Menasc. Qos issues in web services. *IEEE Internet Computing*, 6(6):72–75, 2002.

[Men04] Daniel A. Menascé. Mapping service-level agreements in distributed applications. *IEEE Internet Computing*, 8(5):100–102, 2004.

[M.H05] J.Gawor J.Bester S.Lang I.Fositer S.Meder S.Pickles M.McKeown K.Jackson J.Boverhof M.Rodriquez M.Humphrey, G.Wasson. State and events for web services:a comparison of five ws-resource framework and ws-notification implementations. In *The 14th IEEE International Symposium on High Performance Distributed Computing (HPDC-14)*, 2005.

[Mic05] Microsoft. COM: Component Object Model Technologies. `http://www.microsoft.com/com/default.mspx`, 2005.

[Mye02] Judith M. Myerson. Web service architectures. Technical report, Sun MicroSystems, 2002. http://www.webservicesarchitect.com/content/articles/webservicesarchitectures.pdf.

[OAS05] OASIS. Universal Description Discovery and Integration, 2005. http://UDDI.org.

[PF05] Shrideep Pallickara and Geoffrey Fox. An analysis of notification related specifications for web/grid applications. In *ITCC (2)*, pages 762–763, 2005.

[Pin01] M. Pinedof. *Scheduling: Theory, Algorithms, and Systems.* Prentice Hall, second edition, 2001.

[Raj98] Gopalan Suresh Raj. A detailed comparison of corba, dcom and java/rmi, 1998. http://my.execpc.com/~gopalan/misc/compare.html.

[RS01] Tony Ng Rahul Sharma, Beth Stearns. *J2EE Connector Architecture and Enterprise Application Integration.* Addison Wesley, first edition edition, December 2001.

[Ser] ServiceMix. http://www.servicemix.org/site/ws-notification.html.

[Sol05] Richard Mark Soley. Middleware that works, 2005. http://www.omg.org/corba-corner/corbalive.pdf.

[Spe99] S. Spence. Remote method invocation for persistent systems. In *In Proceedings of the International Symposium on Distributed Objects and Applications (DOA'99)*, number 2 in IEEE Press, Edinburgh, Scotland, February 1999. http://www.cs.wustl.edu/schmidt/PDF/vinoski.pdf.

[Sun05] Sun. Java Remote Method Invocation (Java RMI). http://java.sun.com/products/jdk/rmi/, 2005.

[S.V97] S.Vinoski. Corba: Integrating diverse applications within distributed heterogeneous environments. In *IEEE Communication Magazine*, number 2, February 1997. http://www.cs.wustl.edu/schmidt/PDF/vinoski.pdf.

[TGRS04] Min Tian, A. Gramm, Hartmut Ritter, and Jochen H. Schiller. Efficient selection and monitoring of qos-aware web services with the ws-qos framework. In *Web Intelligence*, pages 152–158, 2004.

[TN] OASIS Web Service Notification (WSN) TN. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn.

[Vin04] Steve Vinoski. More web services notifications. *IEEE Internet Computing*, 8(3):90–93, 2004.

[W3C05] W3C. `WebServiceChoreographyInterface`, 2005. www.w3.org/TR/wsci/.

[YL04a] Tao Yu and Kwei-Jay Lin. The design of qos broker algorithms for qos-capable web services. In *EEE '04: Proceedings of the 2004 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'04)*, pages 17–24, Washington, DC, USA, 2004. IEEE Computer Society.

[YL04b] Tao Yu and Kwei-Jay Lin. Service selection algorithms for web services with end-to-end qos constraints. In *CEC*, pages 129–136, 2004.

[YL05] Tao Yu and Kwei-Jay Lin. A broker-based framework for qos-aware web service composition. In *IEEE*, pages 22–29, 2005.

[ZBN+04] Liangzhao Zeng, Boualem Benatallah, Anne H.H. Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. Qos-aware middleware for web services composition. *IEEE Trans. Softw. Eng.*, 30(5):311–327, 2004.