Architectural Approaches, Concepts and Methodologies of

Service Oriented Architecture

Master Thesis

submitted in partial satisfaction of the requirements for the degree of
Master of Science
in Information and Media Technologies

by

Selda Güner

Submitted to:
Prof. Dr. Joachim W. Schmidt

Supervised by:
Dip. Inf. Rainer Marrone

Software System Institute
Technical University Hamburg Harburg

Hamburg, Germany
4 August 2005

Declaration

I hereby declare that I am the author of this thesis, and all literally or content related quotations from other sources are clearly pointed out and no other sources rather than the ones declared are used.

Selda Güner

Hamburg, Germany
4 August 2005

[Signature]

Acknowledgement

I would like to thank Prof. Dr. Joachim W. Schmidt of Software System Institute (STS) for giving the opportunity to work on this Master Thesis. As well, I would like to thank Dipl. Inf. Rainer Marrone for supervising this thesis and his kind, essential advice, encouragement and guidance throughout this work.

I would particularly like to thank Dipl. Inf. Patrick Hupe of STS for his kindness and providing advice during the project. His guidance was great for me, and I am glad that I have found the opportunity to have thoughtful conversations with him regarding my project implementation.

I have special thanks to my parents for all their continuous support throughout my educational studies. They have been always kind, caring and patient with me, which gave me the strength to successfully finish my studies.

Table of Content

Abstract

Service Oriented Architecture (SOA) is an architectural style which allows interaction of diverse applications regardless of their platform, implementation languages and locations by utilizing generic and reliable services that can used as application building block. SOA includes methodologies and strategies to follow in order to develop sophisticated applications and information systems.

SOA is different from the traditional architectures as it has its own unique architectural characteristics and regulations, which needs to be analyzed and clarified so as to apply the information that should be included in the architectural model of SOA correctly to service based application development.

This thesis aims to describe SOA in detail with considering all the approaches, concepts and methodologies that surrounds architectural model of SOA. Service based application development, service oriented integration approaches, technologies for SOA development, frameworks and other related requirements are discussed in this study in order to have a complete and accurate figure of SOA and be competent in utilizing service orientation concepts in enterprise application development.

List of Figures

List of Tables

1. Introduction

From the period of the earliest computing units development to the present times, which we call as Information Age, software architectures evolve rapidly to achieve building of sophisticated application structures capable of not only fulfilling basic functionalities expected from each computing systems, but also effecting humans life by providing corporate agility, operational efficiency and innovative improvements that result in utilization of universally shared application functionalities and services. Service Oriented Architecture (SOA) provides this vision to cope with technical complexities faced with enterprise application development and integration, as well as aligning business needs and providing coarse grained business functionalities.

Service Oriented Architecture (SOA) is an architectural style and a combination of methodologies that aims to achieve interoperability of remotely or locally located homogeneous and heterogeneous applications by utilizing reusable service logic. Service orientation shows variation in adopting technology for implementation, rather than focusing on the technology itself, as SOA considers the description of the problem domain before concentrating on the usage of a specific execution environment.

Although SOA does not a direct implication of a certain technology and has been applied to software development before the invention of Web services, the capable architectures that realize the SOA vision in a more applicable way are built with Web service technologies. Driven by these competent technologies, the enterprise is practicing open standards for communication over network, messaging and description of service interfaces. SOA with Web services are at the present gaining momentum, as with Web services there is fundamental improvement in SOA based application development.

It is required to follow new approaches and particular methodologies when building service based application structure, rather than tracking the traditional approaches to software development. SOA needs unique development strategies, which replace the conventional approaches to building software architectures and promise the development of plug-and-play application structures and building modules capable of expressing definite business functionalities and problem domains.

SOA provides a strong architectural discipline and focus area centered on service creation, modeling and development, formation of process information, and service oriented integration approaches and strategies. Services are the building blocks of SOA and new applications can be constructed through consuming these services and orchestrating them within a business process. Services are reusable units for articulating common business and technology functionalities.

To implement a successful SOA in enterprise requires consideration of various concepts and implementation strategies, which formulate the essential characteristics of service oriented enterprise. A complete SOA implementation reflects on not only the deployment of services, but also the possibility of using them to integrate diverse application logics, and building of composite applications.

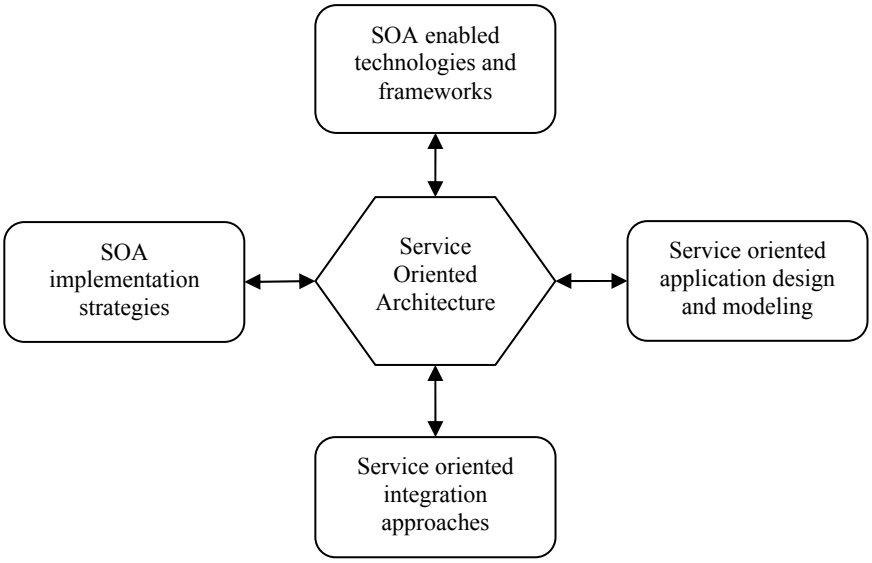


Figure 1.1: Service Oriented Architecture in Enterprise

Upon a successful implementation of SOA, the enterprise gain benefit by reducing development time, utilizing flexible and responsive application structure, and allowing dynamic connectivity of application logics between business partners.

1.1 Motivation

SOA provides an evolutionary approach to software development, however, it introduce many distinct concepts and methodologies that needs to be defined and explained in order to understand the SOA offerings in an accurate way and build a competent architecture that satisfy the SOA vision. The main issue is to analyze and assess the differences of SOA from past architectural styles, investigate the improvement that SOA has brought to computing environment, and apply this knowledge to service based application development so as to have a satisfactorily SOA.

Primarily, it is required to have a clear understanding of what SOA is and the fundamental characteristics of service orientation. Since SOA is a concept independent from any certain technology and focus on the definition of architectural aspects of the application including service design, processes and modeling, it is likely to observe different variety of SOA implementation in enterprise. Especially, nowadays SOA implementation with Web services is diverge from the initial SOA implementations done with CORBA, COM/DCOM and RMI technologies.

Traditional programming logic is mainly centered on user interface development, processing a database or execution of a single transaction, however, SOA offers an extensive operability area including process flow and service oriented integration, which finally reaches to the development of a unified and single application logic, which may include various services and applications within the enterprise and aims to solve a specific business problem domain and serve widely focused united functionalities.

It is obvious that service based application development is more challenging and have specific requirements than the development of a traditional application. To comprehend SOA in an effective way, it is needed to describe what it is, investigate which information must be included in the architectural model of Service Oriented Architecture, and look into possible implementation scenarios.

1.2 Objectives

Principally, this study will introduce the fundamentals of SOA considering all other required properties of service orientation including architectural principles, service design and modeling, and strategies for building an enterprise application by combining individual services to form SOA infrastructure.

This thesis will approach to SOA from the following perspectives:

- Service oriented applications have a different approach to its layered architectural model. It is needed to describe and clarify what SOA is, the relationships of SOA with other architectures and development approaches, and the requirements of SOA in order to demonstrate how the architecture can be applied to software application development.

- Service design and development is a significant concept of service orientation. To investigate basic service design considerations, model-driven service development, and functional and operational aspects of services is important to have successful SOA implementation.

- Another perspective of looking at SOA is to determine and evaluate available frameworks with the aim of finding out their capabilities in building effective service based architecture.

- SOA implementation in enterprise is another fundamental concept which includes service oriented integration approaches and other considerations required for a competent infrastructure.

- It is needed to demonstrate the basic principles and concepts of service based application development by designing and implementing a prototypical case study.

1.3 Structure of Thesis

The subsequent chapter is an introduction to Service Oriented Architecture. Different software development models and architectural styles will be discussed and compared to service based development. SOA will be explained in detail in this part with considering its entities, characteristics and layered architecture.

The third chapter of this study will focus on the technologies of Service Oriented Architecture. Each technology will be described and compared in order to find out the capabilities for developing and implementing service based applications.

The fourth chapter is for clarifying the principles of service orientation in software design and development. Characteristics of service oriented design, SOA Meta-Model, modeling approaches for service based development, and implementation models for services will be discussed and analyzed.

In the fifth chapter, different frameworks will be introduced and compared in order to demonstrate how they approach to Service Oriented Architecture implementation.

Chapter six will concentrate on the principles of service based application development by demonstrating design and implementation of a prototypical case study.

The seventh chapter is for describing strategies and concepts of service orientation in enterprise. In this chapter, different enterprise integration approaches will be introduced and discussed in term of how they realize SOA in their infrastructures. Additionally, other required considerations of SOA implementation will be presented.

The conclusion statement of this study will be the basis of fifth chapter. A summary of the Thesis will be provided in this chapter with possible further studies.

2. Introduction to Service Oriented Architecture

Software development turns out to be more challenging as the needs and desires grows to have complex infrastructures capable of solving real-world problems. Similarly, technological improvements through many tendencies and alternatives grounds to build compound architectures for developing software systems.

The architecture of software explores the software system infrastructure by describing its components and high level interactions between each of them. These components are abstract modules built as a "unit" with other components. The high level interactions between components are called "connectors". The configuration of components and connectors describes the way a system is structured and behaves [1], shown in figure 2.1.

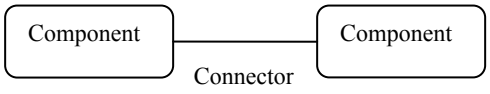


Figure 2.1: Abstract Definition of Software Architecture

The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationship among them [2]. To simplify the complexity of the architecture, conventionally, the system is built with modules, which involves functions, objects, components and services.

However, from the early days of the computing to the nowadays, the era of Information Technology, software development has passed through certain development stages, which broaden the scope of building applications from small departments to the enterprise and finally to the Internet. These stages shape the architecture of software systems and result in more powerful, capable and effective software which aims to meet the sophisticated expectations from computing systems.

Service Oriented Architecture (SOA) is a particular type of software architecture which has distinguished features and characteristics. The concept of SOA emerged in the early 1980s and become a significant architectural style especially after invention of Web services. Before examining the architecture in detail, it is important to evaluate the existing software development concepts and related technologies to discover the revolution of SOA so as to not to develop SOA from scratch.

2.1 Evolution of Computing Systems

The improvements in hardware technologies and specially the invention of networking enforce the software developers to gain more benefit from these enhancements and build composite software systems which endorse these technologies.

Networking has evolved from a few unified machines to enormous interconnected computing resources on the Internet. Accordingly, the complexity, size and power of the computing systems have advanced from *monolithic programs* to *distributed computing* infrastructures. Based on these emerging trends in computing technologies, developers and architects ought to renew their visions to replace the old approaches of application development to the expansion of new logical models for current enabling technologies.

The early approaches of computing have started with closed, monolithic mainframe systems. Monolithic applications were consequence of the evolution of single-processor systems in which the processing and management of data is completely centralized. ***Procedural development*** is the initial way of application development which comprises the process executed on a single machine and manipulates the data through direct access operations, as shown in figure 2.2. This computing method has many potential dependencies between the algorithms of the program and does not allow doing modification easily. If the data representation is modified, there can be substantial impacts on the program in multiple places [3].

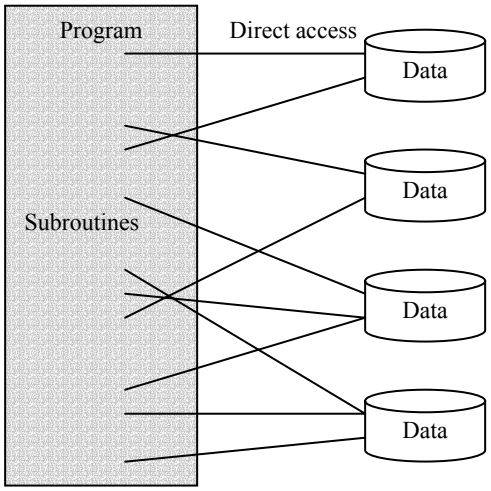


Figure 2.2: Procedural Software Development

FORTRAN, COBOL, C, Pascal and BASIC are the languages for procedural software development. To develop an application with these languages requires complex knowledge of physical storage of the data. Another difficulty is that although the developers have implemented shared libraries of code and decomposed the application structure into modules, still the program has many dependencies inside and changing was difficult to manage.

The difficulties faced with procedural software development give rise to design a new approach which involves decomposing larger processes into smaller ones [4], as illustrated in the following figure. ***Structured design and development*** reduces complexity of the program with designing small modules which can be reusable within the application. In this way, the data aspect of the program and the behavior part of it is signified separately.

Structured design allowed to development of more flexible and complex software structures than procedural software development, however, still the dependencies between individual modules and the data are high to build an effective software system.

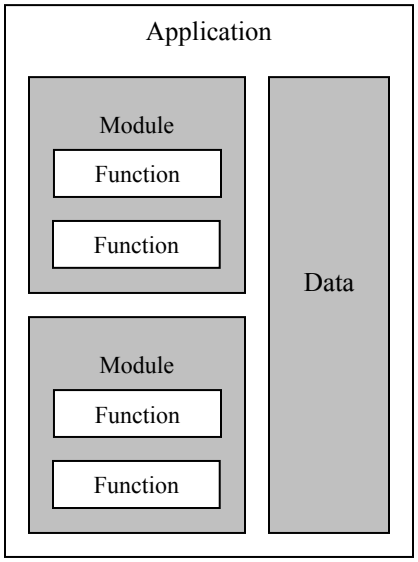
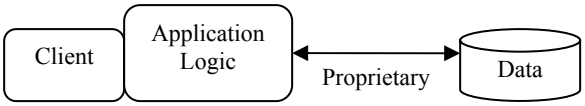


Figure 2.3: Structured Software development

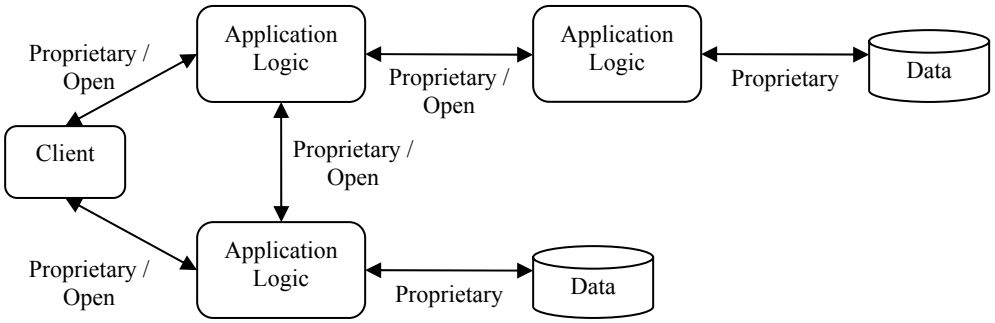
Client-server computing is the evolution of software development in which the emphasis is given to building of individual application systems. The design involves separation of the client and business logic from the keeper of the data, which is server, in both logical and physical way. It is an important factor in the software development that client-server technologies allows user to process data over network connection. The technology resulted in the evolution of *file-sharing system*, which is still used to access the available global file systems over supporting protocol such as HTTP, and *database-server* technologies. These developments are closely related with the evolution of distributed computing technologies.

In this period, *transaction-processing monitors* have grown to enable the consistent and reliable maintenance of data integrity across distributed systems. Another development between the late 1980s and early 1990s is the *groupware* technologies which allow email and higher forms of interactive applications, such as chat rooms and videoconferencing.

Starting from early 1990s, the necessity to support clean separation of data and application logic layer from the presentation layer caused to replace client-server technologies with ***N-tier*** component-oriented solutions. The additional layers reduced the coupling between the modules and allowed more clients to access and converse with the business logic tier, as shown in figure 2.4.



Client-Server Computing



N-tier Software Development

Figure 2.4: Client-Server and N-tier Software Developments

The first approach to software development is mainly from private vendors, which control the construction of software through each step of product releases. These software are called *Proprietary Software* and mostly dependent on the vendor which produces it. Today's systems are based on proprietary software to varying degrees. The capabilities and quality of proprietary software can be high because of continuous debugging and supporting features consistent with the expectations of the end-user from the owner vendor of the software.

The other major category of commercial software is *Open Software*, which is an approach to software development in which multiple vendors collaborate to build specifications of the technology independent from proprietary software. The main benefit of the open software is that it provides a uniform terminology of its software structure, which is the foundation for building standard technology appropriate for many end-users. The additional benefit is the interoperability that it may provide between different software applications.

With the introduction of the ***World Wide Web***, a new era for software development is in progress. The Internet initially began as a way to publish scientific papers and evolved into dynamic HTML (Hypertext Markup Language) and eventually to XML (Extensible

Markup Language), which is a meta-language and a fundamental, standard, and enabling technology for data exchange between different platforms.

XML describes the data in an application-independent manner, and Web services use this technology to enable the sharing of distributed processes between heterogeneous computing environments. Today's application development involves creating independent services accessible through the firewall and support one discrete function. Clients interact with services, which are assembled to build the application infrastructure.

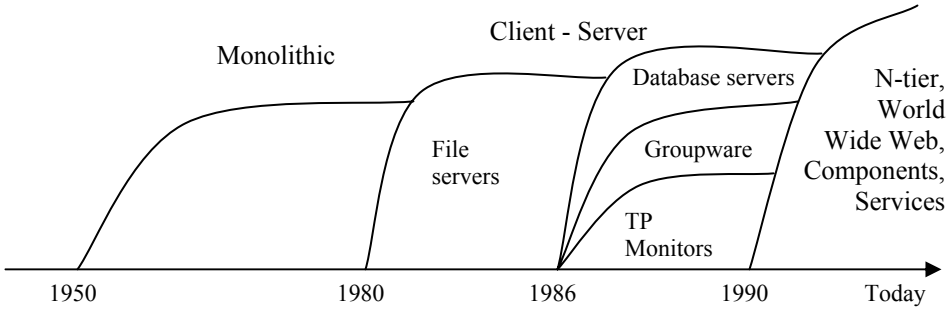


Figure 2.5: Physical Evolution of Computing Systems

2.2 Object Oriented Development

Object Oriented Development supports the development of software with encapsulating both data and behavior into abstract data types, called **classes** [5]. Instances of classes are formed into small modules, called **objects**, as shown in figure 2.6. Any changes in data representation only affect the immediate object that encapsulates the data. Classes can live everlastingly, however objects have a limited lifetime.

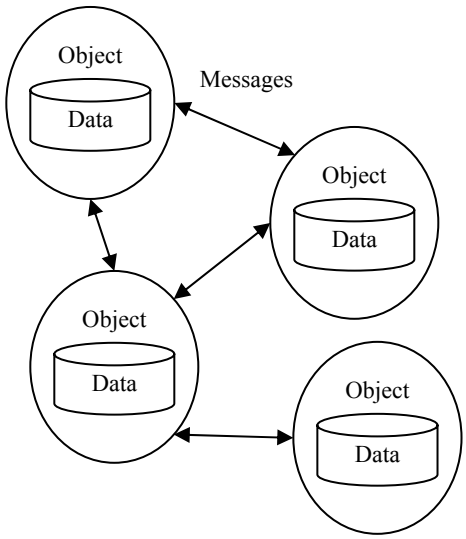


Figure 2.6: Object Oriented Development

Objects communicate each other through messaging. Object based development advances software design by providing more support for hiding behavior and data through objects and classes. There is almost no dependency between objects, however a large number of interconnected objects create dependencies that can be difficult to manage.

In Object Oriented development, everything is an object. The primary characteristics of Object Oriented development are as follows:

- **Encapsulation:** An object contains both the physical properties, called data, and the functionality of this data, described as behavior, to form a distinct software module, which is called as *package*.
- **Information Hiding:** An object keeps its internal mechanism and does not reveal object-specific information to the outside from its well-structured architecture and interfaces.
- **Associations and Inheritance:** Classes and objects can associate to each other. Inheritance is a special form of association which states is-a-relationship between objects and classes and forms a hierarchical structure by allowing classes to be extended into subclasses.
- **Polymorphism:** Object oriented development allows different implementations of the same message through two or more separate classes.

Object oriented technology is a set of techniques used for the development of software systems. ***Object oriented analysis*** extends the capabilities of information technology for modeling real-world business processes. This is an evolution compared to procedural technologies, which require modeling the business environments in terms of control flow and data representation. Object oriented analysis provides a mechanism for modeling reality that is relatively easy to communicate with end-users.

Object oriented design is another major software phase that has been successful commercially in the software process market. Object oriented design involves the planning of software structure, improving software quality by finding out the deficiencies of the structure, and rapid prototyping of software systems.

The other major category of object technology focuses on implementation. ***Object oriented programming***, which can be done with the languages such as Smalltalk, C++, Java, and C#, are one promising choice for implementation of application software. The major quality of object orientation is that the implementation allows the development of properly encapsulated applications.

The benefit of object orientation is that the software structures more easily map to real-world entities. Today, object oriented technology is widely used and a dominant paradigm for developing application software. This technology, when combined with component infrastructures, can enable interoperability between different software environments.

2.3 Component Oriented Development

Components are more sophisticated software modules than objects and require fundamental changes in systems thinking, software processes, and technology utilization. Component-based development allows developers to create more complex, high quality systems, because it provides better means of managing complexities and dependencies within an application.

A ***software component*** is defined as a unit of composition with contractually specified interfaces and explicit context dependencies. A software component can be deployed independently and is subject to composition by third parties [6]. It is a group of objects with has a specified interface, working together to provide an application function, as can be seen from following figure.

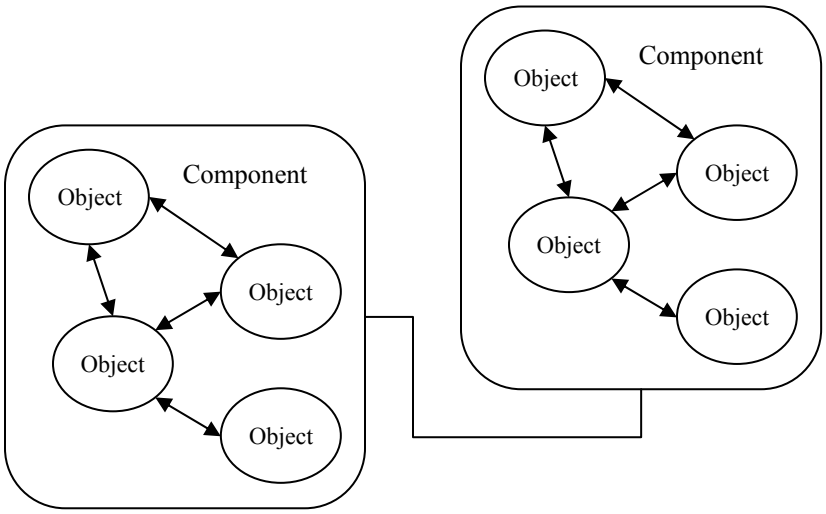


Figure 2.7: Component Oriented Development

Component may refer to many different software constructs, from single application logic to an entire functional system. In all cases, a component is a software package with one or more well defined interfaces. A component is executed on a component execution environment provided by an application server, such as J2EE container, which provides the required functions, such as transaction management and database connection pooling.

Components overlap the properties of object orientation, such as encapsulation and polymorphism, except it reduces the property of inheritance. In component thinking, inheritance is tightly coupled and unsuitable for most forms of packaging and reuse. Instead, components reuse the functionality by invoking other objects and components rather than inheriting from them. In component terminology, these invocations are called *delegations* [3].

Components have specifications to describe the component encapsulation, which means its public interfaces to other components. The reuse of component specifications is a form of polymorphism. Preferably, component specifications are local or global standards that are widely reused throughout a system, an enterprise, or an industry.

Components may be integrated to create a larger entity, which could be a new component, a component framework, or an entire system. This is called *composition*. The combined component acquires shared specifications from the constituent components. This is often called *plug-and-play integration*.

Reusable components are good reflection of effective software design. The architecture provides the design context in which the components are built and reused. The other important aspect for components is that the development of software architecture based on component specifications support parallel and independent building of the system parts. These computational boundaries that define an individual system part are testable subsystems and can be divided to one or more distributed project teams.

Many platform vendors have already produced software infrastructures which support component oriented technology. These component infrastructures, such as Microsoft .NET, Sun Java Enterprise JavaBeans, are central for building component-oriented enterprise application development. With support of XML, Web services and other standards, these technologies can interoperate for building sophisticated software applications.

2.4 Distributed Computing

Although Service Oriented Architecture (SOA) is not a direct implication of distributed computing, it has to incorporate existing middleware technologies and distributed computing concepts. A successful SOA should overcome the difficulties faced with existing middleware technologies by supporting an effective approach to application development and upcoming technologies with consideration of obtainable concepts and technologies.

The early approach to distributed computing is to set up a communication between two distributed programs directly on the raw physical network protocol [7]. Higher level protocols such as SNA, TCP/IP and IPX provided APIs that helped reduce the implementation efforts and technology dependencies. As the next evolutionary step, a communication middleware framework enables to access a remote application without knowledge of technical details such as operating systems, lower-level information of the network protocol, and the physical network address.

As distributed computing technologies evolve, it becomes increasingly necessary to provide multiple network implementations to satisfy various quality-of-service requirements. These requirements may include timeliness of message delivery, performance, throughput, reliability, security and other nonfunctional requirements.

Proper object-oriented distributed computing infrastructures provide access transparency and give developers the freedom to select the appropriate protocol stacks to meet the application quality-of-service requirements. Consequently, the evolution of techniques for the distribution of enterprise software components resulted in the promise of universal application interoperability.

Applications communicate with each other basically in two communication modes: *synchronous* and *asynchronous* mechanism. However, in reality, there are usually numerous variants of these basic modes of communication.

In ***synchronous messaging***, the exchange of messages requires simultaneous engagement of both endpoints. This is often called as *request-response* messaging.

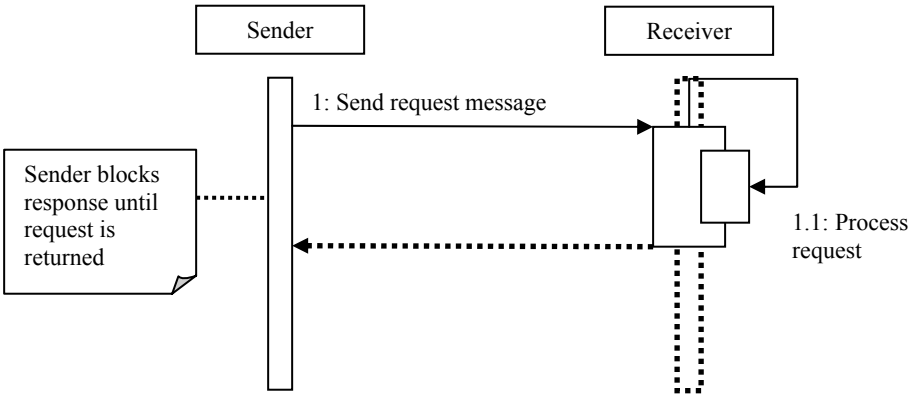


Figure 2.8: Synchronous Messaging with Response

In ***asynchronous messaging***, the client sends a message and continues processing without waiting for a return. Based on some a priori agreement, such as a predefined and agreed-upon endpoint, the receiver sends a response to the sender.

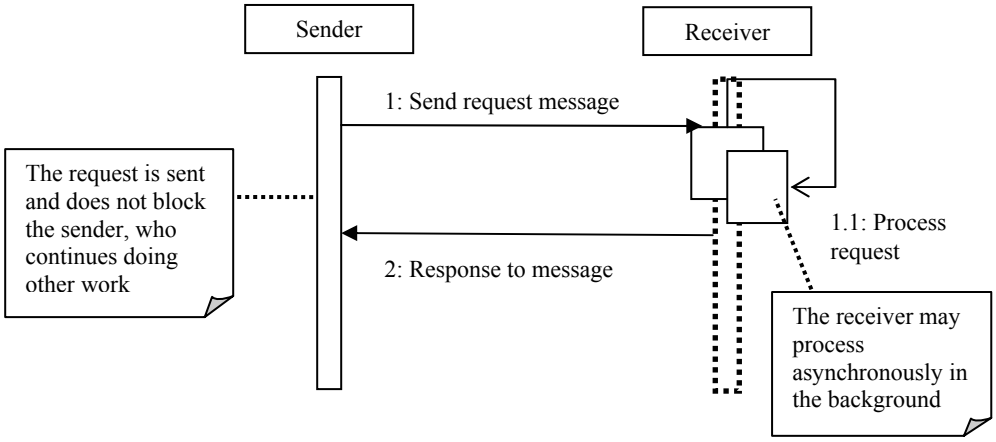


Figure 2.9: Asynchronous Messaging with Response

2.4.1 Remote Procedure Call (RPC)

The development of the Remote Procedure Call (RPC) concept was driven by Sun Microsystems in the mid of 1980s and is specified as RFC protocols 1050, 1057, and 1831. A communication infrastructure with these characteristics is called RPC-style, even if its implementation is not based on the appropriate RFCs.

RPC involves the execution of a function or procedure of a distributed application which encapsulates the code and makes it reusable for remote access. The remote call is routed through the network to another application, where it is executed, and the result is then returned to the caller. Most RPC implementations are based on a synchronous, request-reply protocol, which involves blocking the client until the server replies to a request.

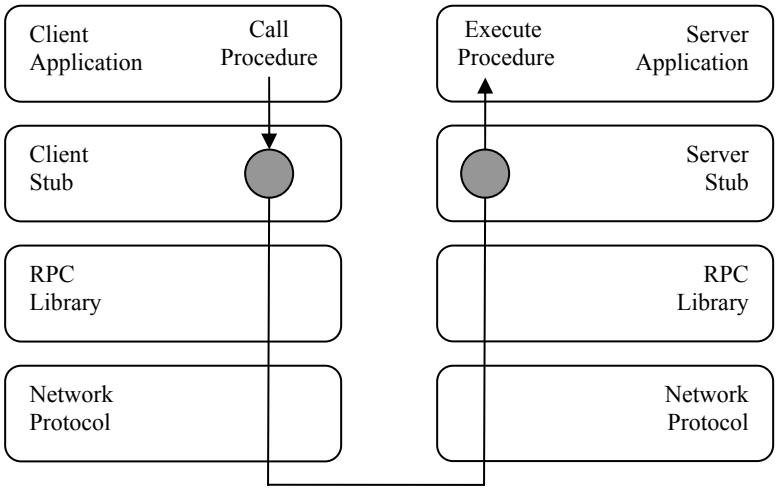


Figure 2.10: Remote Procedure Call Conceptual Model

RPC model has several components. The *network protocol* provides switching, routing, packet sequencing, addressing, and forwarding between virtual circuits to transmit data from node to node. The *RPC library* is specific to certain technology and provides transferring data between hosts. It is also responsible for end-to-end error recovery and flow control. The *Client and Server stubs* establish, coordinate, and terminate connections, exchanges, and dialogs between applications. Finally, the *application* layer contains the actual application and end-user processes, where the business functionality is executed.

The main aim of the development of RPC style protocols is the need to provide platform independent applications. At the end of the 1980s, *DCE (Distributed Computing Environment)* emerged as an initiative to standardize the various competing remote procedure call technologies. However, DCE did not gain a widespread industry support. Other technologies, which include CORBA, COM/DCOM and Java Remote Method Invocation (RMI), are used in practice today and provide robust RPC-based distributed computing platforms.

2.4.2 Distributed Objects

The concept of Distributed Objects, emerged in the early 1990s, is the evolution of object-oriented programming which provides remote procedure or function calls as the replacement for the traditional modular programming styles.

Typically, distributed objects are supported by an Object Request Broker (ORB), which manages the communication and data exchange with potential remote objects. It provides an object oriented distribution platform, location transparency and enable objects to hide their implementation details from clients.

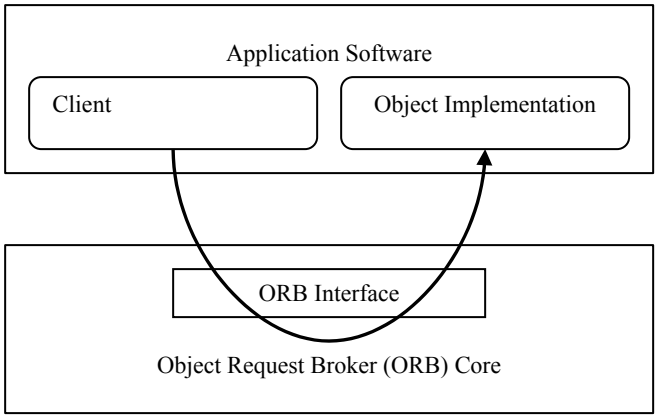


Figure 2.11: Object Request Broker

CORBA is the most common RPC-based ORB implementation technology. It does not pay specific attention to data or program execution services, as its main aim is to provide an implementation of a proper distributed object framework.

2.4.3 Message Oriented Middleware (MOM)

Message-oriented Middleware (MOM) is typically a piece of software that locates between communicating parties and provides a mechanism for connecting various applications. It is responsible for handling different dependencies between them, such as operating systems, hardware, and communication protocols. A MOM exposes its facilities using an API that defines how distributed applications should use the underlying MOM to communicate with each other through messages. Messages relate to a specific function that should be executed upon receiving the message.

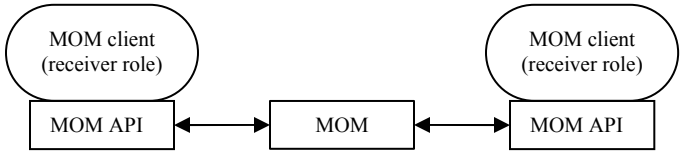


Figure 2.12: Message Oriented Middleware

MOM based messaging solutions can be architected in various topologies, such as *centralized MOM*, which is designed as a central hub-and-spoke topology where the MOM acts as a message bus between application components; or *hybrid MOM* topology, which has a central MOM that acts as a router between communication parties that use their own local MOMs.

MOM plays a significant role by providing an environment that enables two applications to set up a conversation and exchange data. The features of a communication middleware can be described as follows:

- It does not require the sender and receiver to be simultaneously connected
- It ensure strong delivery guarantees on the request and response between participants
- It adds functionality in some cases by translating and formatting messages route between participants

MOM encourages loose coupling between message consumers and message producers, and enables dynamic, reliable, and high-performance systems to be built. However, it ought not to be underestimated that the underlying complexity of MOM-based systems makes it difficult to ensure a proper and efficiently working system development.

2.5 Definition of Service Oriented Architecture (SOA)

Service Oriented Architecture (SOA) is an architectural style which utilizes methods and technologies that provides for enterprises to dynamically connect and communicate software applications between different business partners and platforms by offering generic and reliable services that can be used as application building blocks. In this way it is possible to develop richer and more advanced applications and information systems.

Although SOA is not a new concept, especially after the invention of Web Services, the new developments in this area bring about a new way of constructing software application architectures, a new approach to rebuild available software infrastructures and possibility of communicating with other enterprises according to the available services.

However, building SOA is still challenging for the following reasons:

- The way SOA approaches to software resources is different from traditional architectures,
- SOA needs a level of architectural regulation,
- SOA implementation needs an environment capable of being accessed by different enterprises.
- Definition and composition of services into new ones in a secured and managed environment is another aspect that needs a particular concentration.

2.5.1 SOA Entities and Characteristics

Service Oriented Architecture is an architectural style that defines an interaction model between three main functional units, in which the consumer of the service interacts with the service provider to find out a service that matches its requirements through searching registry. A meta-model describing this interaction is shown in following figure.

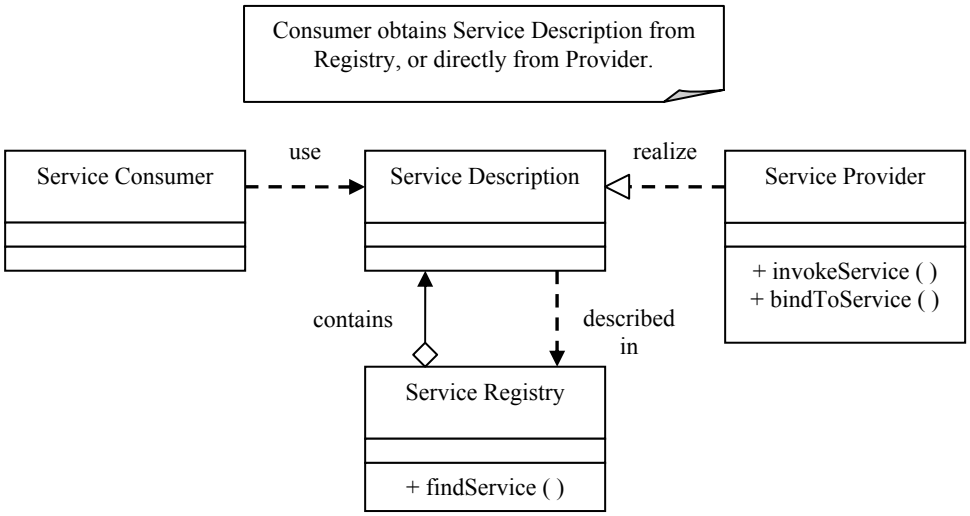


Figure 2.13: Service Oriented Architecture Conceptual Model

SOA contains 6 entities in its conceptual model, described as follows [1]:

Service Consumer: It is the entity in SOA that looks for a service to execute a required function. The consumer can be an application, another service, or some other type of software module that needs the service. The location of the service is discovered either by looking up the registry, or if it is known, the consumer may directly interact with the service provider.

Service Provider: It is the network addressable entity that accepts and executes requests from consumers. It provides the definite service description and the implementation of the service. The service provider can be a component, or other type of software system that fulfills the service consumer's requirements.

Service Registry: It is a directory which can be accessible through network and contains available services. Its main function is to store and publish service descriptions from providers and deliver these descriptions to interested service consumers.

Service Contract: A service contract is the description that clarifies the way of interaction between the service consumer and provider. It contains information about request-response message format, the conditions in which the service should be executed, and quality aspects of the service.

Service Proxy: It assists the interaction between service provider and service consumer by providing an API written in the local language of the consumer. It is supplied by the service provider and a convenience entity for the consumer. As well as it can enhance performance and provides caching facilities. Service proxy is an optional entity in SOA.

Service Lease: It specifies the amount of time that a service contract is valid. It is managed by registry and determines the executive well-defined timeframes of binding to the services. Usage of service lease supports loose coupling between service provider and consumer and maintenance of state information for the service.

Service oriented architecture reflects specific principles and characteristics that need to be applied when building service-oriented application infrastructures [8], which are described as follows:

- *Services are discoverable and dynamically bound*

Services need to be discoverable dynamically at run-time. The consumer of the service finds the required service through searching of registry and gets all the information necessary to execute the service. There is no compile-time dependency to bind to the service, apart from the service contract that the registry provides.

- *Services are self contained and modular*

A service supports unified and functional interfaces aggregated to perform specific and concrete business logic. These interfaces are related to each other in the context of a module and contain sufficient information to be authentic without any dependency to other software modules or applications.

- *Services are interoperable*

Services express the ability to communicate with each other without any platform and language dependencies. Because each software module might have proprietary and tightly coupled structures, service based architecture utilizes interoperable technologies that support the protocols and data formats of the service's current and potential consumers.

- *Services are loosely coupled*

Coupling describes the level of dependencies between software modules. Loosely coupled modules are flexible and have well-defined dependencies, on the contrary, tight coupled software systems are difficult to configure because of unknown requirements of modules within the software structure.

Service oriented architecture stress the development of loosely coupled services as the software construction unit. Loosely coupled is achieved by usage of service registries. The service requires no other system specific information to be executed autonomously. However, tight coupling cannot be avoided at the interface definition level or binding to a specific protocol.

- *Services have a network-addressable interface*

A service should publish its interface on the network to conform service oriented architecture design principles, so that the consumer is able to invoke the service in a distributed manner over the network. In this way it is probable to use the service at any time with different consumers. A service can also be accessed through local interface without usage of network. However, one of the main aims in building SOA is to allow the consuming of services in a location-independent manner.

- *Services have coarse-grained interfaces*

The concept of granularity is related with the way of implementation of interfaces within software system. If the interface supports all the functions necessary for complete business logic, it is a coarse-grained interface. In contrast, if the interface implements only a part of certain functionality, it is considered fine-grained. Granularity can be applied to the entire service implementation, and also to the individual methods of the interface execution. A service oriented software system supports coarse-grained interface design by nature with different granularity levels. The objects which build the service may still be fine-grained, however these objects is kept inside the physical structure of the service.

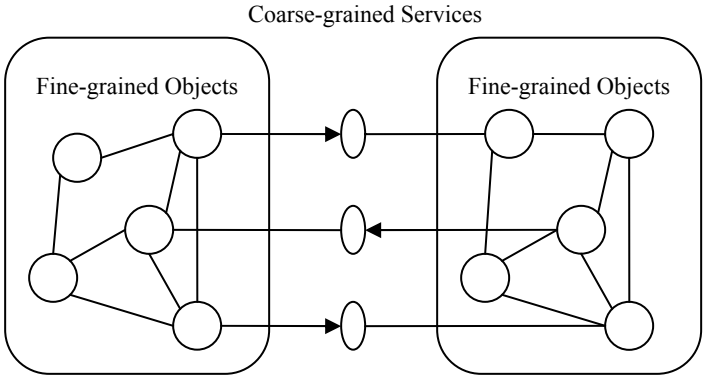


Figure 2.14: Granularity

- *Services are location transparent*

SOA promotes location transparency, which means the consumer of the service does not have pre-knowledge about the position of the service until to execute it at run-time through registry. The only dependency between consumer and provider is the service contract, which can shift from one location to another one without affecting consumer.

- *Services can be composed into new applications*

Another key characteristic of SOA is to enable building new applications composed from existing services. Composition is an effective design which mainly focuses on service modularity and reuse of service functionality without having pre-knowledge of which applications will use the service in the future.

- *SOA supports self-healing*

Self-healing is described as the ability of a system to recover itself in case an error occurs during its execution. Service oriented systems should give more importance to support self-healing than other architecture styles as the services are combined to execute a business function at run-time. The consumer should have the ability to find different services on the network that supports the same functionality for the aim of a proper and regular execution of the software system.

2.5.2 Service Oriented Development

Services are the evolution of components in which multiple component interfaces form into a single interface to perform a specific function. A *service* is an abstract resource with the capability of performing a task [9]. Services have the potential reflection of business functions as well as technical task definitions.

Service based development advances component based development in a way that services involves development of distributed, cooperating components realized in different programming languages, but also service orientation supports the features such as platform independence, dynamic discovery of services and improved level of reusable application modules.

Service orientation is as well evolved into nowadays' service-based application development in which services is the natural fruition of Internet networking and World Wide Web technologies. These technologies alter the application structure to allow more dynamic and open software construction and solve interoperability problems by applying standard developments that is acceptable from many software organizations and vendors.

Services are designed and developed to support the following characteristics:

- Each service defines a specific business function and can match to real-life activities
- A service may have various procedures and operations
- Services interact with other services and system components in a loosely coupled, message-oriented environment to accomplish business goals
- Services has clearly defined interfaces and can be used by many different other services and applications
- Services do not need to be in a distributed environment

The following figure illustrates service based development in the context of components and objects.

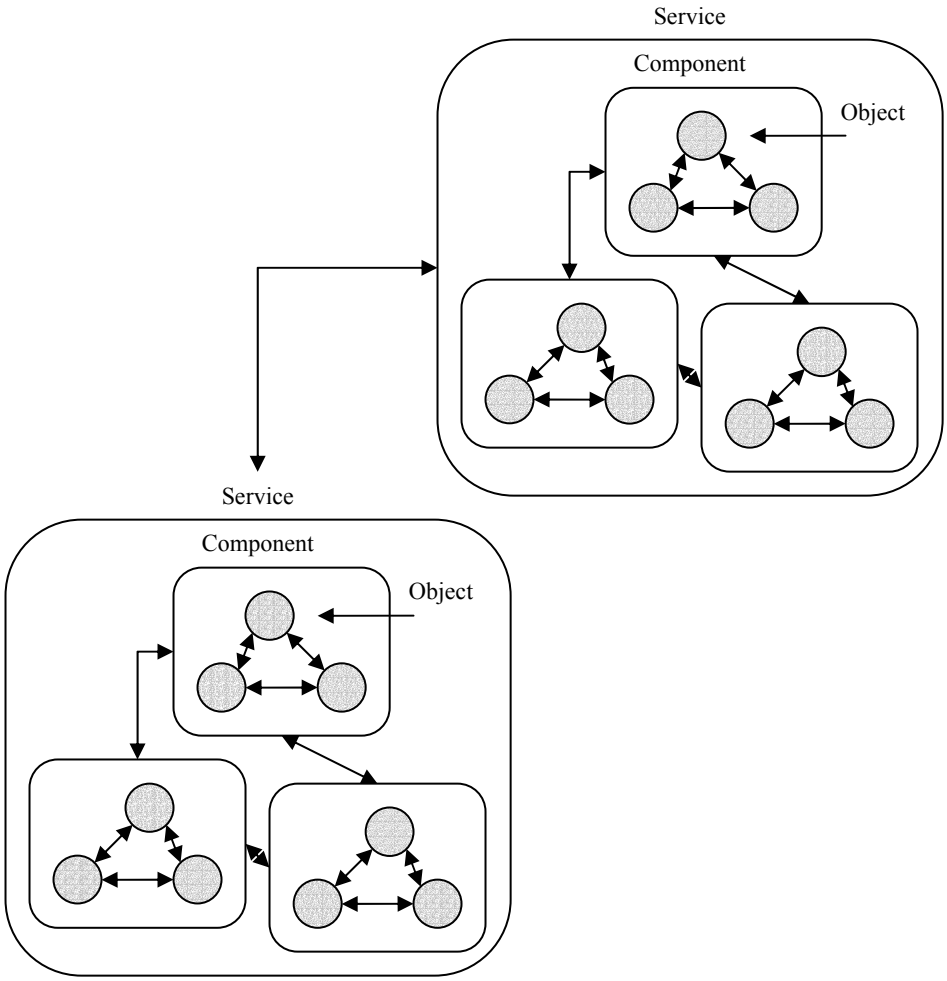


Figure 2.15: Service Based Development

Software development has passed through certain stages and architectural models until to enable the development of platform independent, reusable services. Each architectural model provides a better means of dealing with software difficulties than the previous approaches in order to accomplish enhanced software structures that fit real-world requirements.

The early architecture of software is based on structured design, which has rigid rules for the development of software constructs and limited support to enable robust and sophisticated application development. Object oriented technologies result in flexible software development that supports encapsulation of business logic through more coarser-grained functions and classes; however, the tangible benefits of robust application development are gained through the progression of components and services.

The following table discusses the characteristics and features of each software architectural models.

Structured Development	Object Oriented Development	Component Based Development	Service Based Development
Very fine system structuring Low reusability Tight coupling Have compile time dependencies Intra-application communication scope	Small – grain system structuring Low reusability Tight coupling Have compile time dependencies Building blocks are individual classes Encapsulation, Inheritance, Polymorphism Functionality is described by class declarations Dynamic but large number of connected object	Medium – grain system structuring Medium reusability Loose coupling Have compile time dependencies Building blocks consist of several classes (components) Interactivity, connectivity, and exchangeability of components Functionality is described by interface declarations	Coarse – grain system structuring High reusability Loose coupling Have only run time dependencies Building blocks consist of components Published interface definition Dynamically discoverable distributed services Functionality is described by network addressable component interface declarations Inter-enterprise communication scope

Table 2.1: Comparison of Architectural Development Models

2.5.3 SOA Layered Architecture

The early systems are structured as ***two-tier*** layers in which the client have direct access to database and network APIs without any logical model in between. This approach can still be used in software systems where the development is small-sized and prototypical. As well, some modules in advanced systems may apply this approach to provide certain functionalities. However, two-tier application development is limited to short life-cycled systems and does not support flexible APIs. It does not provide sufficient implementation code isolation from the client which makes the architecture rigid and not scalable to many concurrent users.

Currently the most frequently used application development model is based on ***three-tier*** architectural structure, which supports an additional layer between client and data storage tiers. The additional layer, called as business logic layer, provides code isolation from client and sharing of the application logic between various client implementations. It is a competent approach to software development for flexible managing of data and usage of system resources.

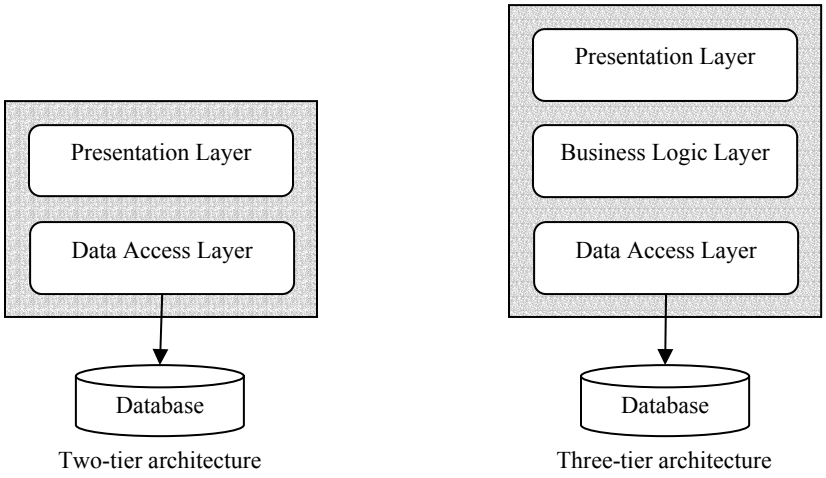


Figure 2.16: Two-tier and Three-tier Architectural Models

SOA is based on ***n-tier*** application development in which services are layered on top of components that are responsible for providing certain functionalities and maintaining quality of service requirements for services [10], as shown in figure 2.17.

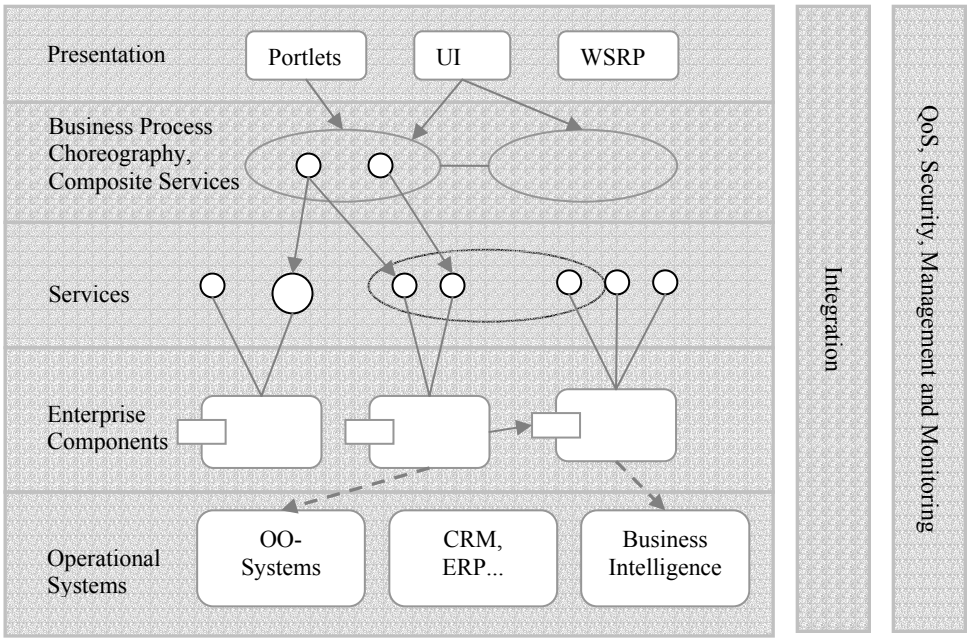


Figure 2.17: The Layers of Service Oriented Architecture

Each layer in SOA has specific architectural characteristics, as described below:

Operational Systems Layer: This layer contains existing applications including CRM and ERP packaged applications, object-oriented systems, and business intelligence applications. These applications provide the background for services and each of them has its own proprietary structures, databases and other system resource access.

Enterprise Components Layer: These are specialized components to provide certain functions and requirements for services. They are the business assets for service implementations, and other system necessities such as management, availability and load balancing of services.

Services Layer: This layer contains the actual services which can be discoverable and invoked by other applications to provide a specific business function for enterprises. Services realize and deploy enterprise component interfaces as service descriptions and are published on the network.

Business Process Composition Layer: The services can be composed into a single application through service *orchestration* or *choreography*, which supports specific use cases and business processes.

Presentation Layer: It provides user interfaces for services and composite applications. Although presentation layer is not a straight concern for SOA, because of the objective of using services as user interface bring about the development of standards such as portlets and Web Services for Remote Portlet (WSRP) specifications.

Other concern for SOA is the integration of services and composite applications within the enterprise by supporting the features such as reliability, proper routing and coordination of services, and managing other technical details including protocol and integrating party agreements. QoS (quality of service) requirements, security, management and monitoring of services are also other requirements that need to be clarified when designing service based application architectures.

3. Technologies for Service Oriented Architecture

The initial service-oriented technology was introduced in the late 1990s from Sun Microsystems, which is called as *Jini Network Technology* [11]. Jini is a lightweight environment for dynamically discovering and using services on a network. Its main aim is to allow devices such as printers to dynamically connect to the network and register their available services.

Consequently, the interest to use services as a software construction unit lead to development of more set of technologies for implementing service oriented architecture. However, the challenges of building SOA especially related with interoperability, integration complexities and conveying to industry standards, influence today's approach to service based application development.

3.1 J2EE

The Java 2 Platform Enterprise Edition (J2EE) is a container-centric technology which supports the design, development, and deployment of component-based distributed applications [12]. J2EE uses multi-tiered application architecture in which the application logic is divided into functionally different components and each component may run on different machines.

J2EE components are written in Java language and assembled into a J2EE application. Java applications access external systems through infrastructure services provided by the J2EE-compliant application server. The J2EE specification defines the following J2EE components:

- *Client components:* Application clients and applets
- *Web components:* Java servlets and Java Server Pages
- *Business components:* Enterprise JavaBeans

The J2EE specification proposes several services and each of these services has different functionalities which can be used in building service oriented application development. Clients access these services by provided APIs. Some of the basic services are defined as follows:

- ***JavaBeans:*** JavaBeans are ordinary Java classes that conform to coding standards and conventions.
- ***Enterprise JavaBeans (EJBs):*** EJBs are distributable, server-side components that depend upon an infrastructure implemented by an application server. EJBs support the development of application business logic.

- ***Java Naming and Directory Interface (JNDI):*** JNDI provides access to directory and naming services. Java clients use JNDI as a step in accessing EJBs, databases, and other resources.
- ***Java Database Connectivity (JDBC):*** Provides access to relational databases.
- ***Java Message Service (JMS):*** JSM is an architecture and API for using messaging from Java classes.
- ***J2EE Connector Architecture (JCA):*** JCA is a framework and API that defines a standard interface through which Java programs can access a variety of legacy and other non-Java systems.
- ***Java Transaction Service (JTS):*** JTS is a Java implementation of the Object Transaction Service (OTS), an industry standard for distributed transactions.
- ***Java Mail:*** Provides an API for sending email from Java classes.

3.1.1 Java Message Service (JMS)

Messaging systems at enterprise level provides reliable transport of information between different applications across a variety of heterogeneous computer networks and systems. These systems have the characteristics of allowing interacting programs to run at different times and with hiding network complexities. *Java Message Service (JMS)* is a Java enabled development interface which supports standardized asynchronous message interaction and offers the capability of simulating synchronous request/response communication mode.

JMS provides Java applications with a standard and consistent interface to the messaging services of a MOM provider or a messaging server [1]. JMS based applications provide integrating capabilities for legacy systems and distributed applications in heterogeneous environments. The architectural structure allows queuing of messages with guaranteed, timeliness delivery and ensuring offline applications to process messages later when they are capable of receiving them.

A *message* is a collection of data that one application seeks to send to another, typically on another machine [13]. Messaging systems provide a mechanism to store incoming messages on a *queue*, allow to the receiver to process messages and return back the response at some later point in time. The complexity in this mechanism is to have a reliable transportation of messages between sender and receiver.

JMS has a rigid set of rules to manage message communication for reliable and stable message transportation with supporting features of message persistence, message acknowledgement, and administration of the message consumer's performance, which may disconnect and reconnect to JMS provider.

The following figure is the illustration of the process of communication between message sender, JMS provider and the consumer of messages.

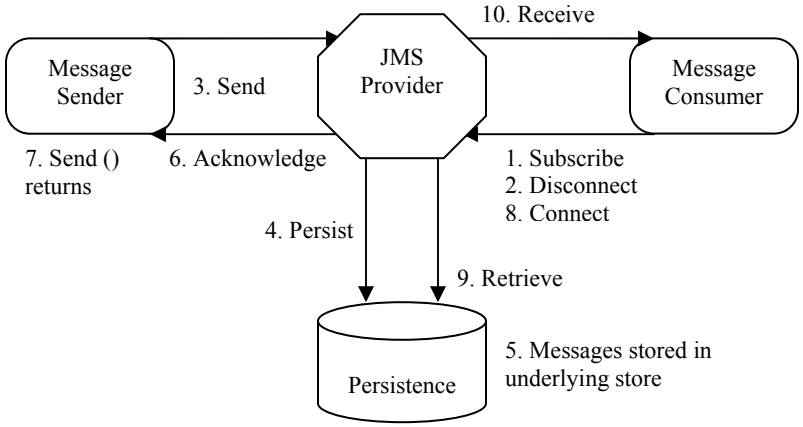


Figure 3.1: Conceptual Model for Java Message Service

JMS supports a variety of message types, such as byte-message, object-message, and text-message. JSM can also be used for XML messaging by replacing a text-message with an XML document.

The JMS specification [14] only defines a programming interface; however it leaves the implementation details to message provider, which is located at each J2EE-compliant application server. Messaging in general achieves loose coupling as the sender and consumer interact through the messaging transport which requires separate transactions. The provider guarantees the delivering messages to its destination; however the processing of the message is independent from sender.

JSM API supports two messaging models:

Point-to-point: In this model, the message created by sender, which is then put on a *queue*, addressed to a single targeted recipient. The consumer reads the incoming messages from queue for processing.

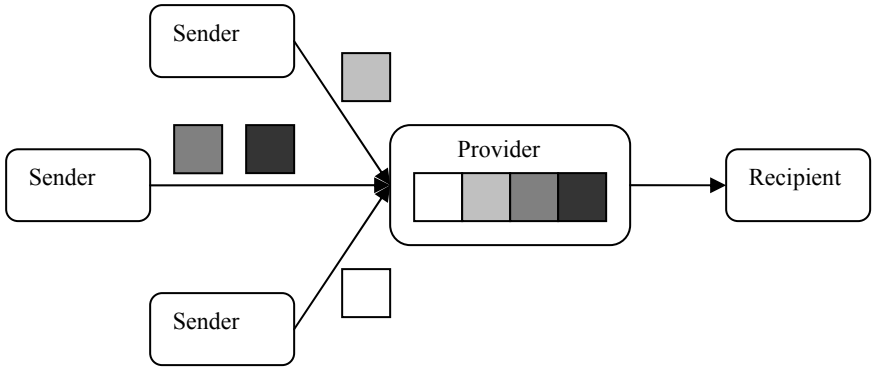


Figure 3.2: Point-to-Point Messaging

Publish-subscribe: In this model, the senders publish messages, which are then put on a *topic*. Multiple recipients can subscribe to the topic and deliver a copy of the particular message.

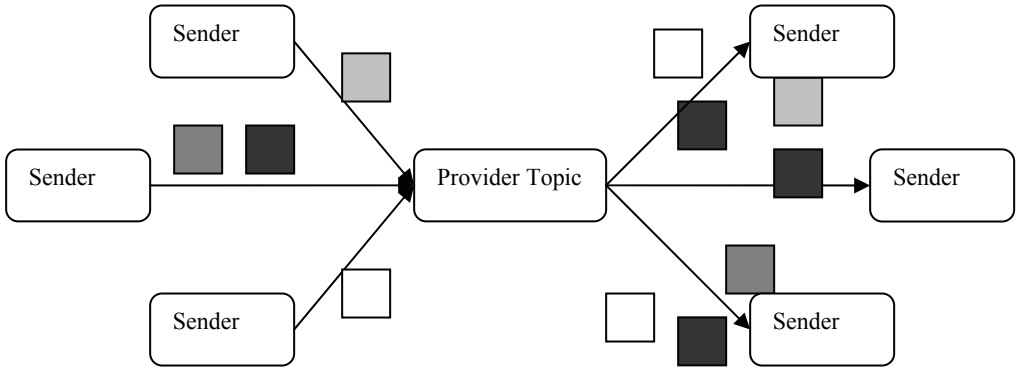


Figure 3.3: Publish-Subscribe Messaging

These models use the same fundamental concepts, which indicate that they share a set of base interfaces. Each base interface has separated sub-interfaces which supports for individual model type.

JMS often supports communication across firewall boundaries. Also many vendors extend their provider implementation to support SOAP messaging over HTTP. In this way, HTTP is used as transportation protocol under JMS API and passes SOAP messages as text-message. This is often called as *SOAP over JMS*.

3.1.2 Remote Method Invocation (RMI)

Remote Method Invocation (RMI) is the action of invoking a method of a remote interface on a remote object. Most importantly, a method invocation on a remote object has the same syntax as a method invocation on a local object [15]. RMI allows the invocation of methods across a distributed network of clients and servers with providing management of objects and the ability to overtake them between each machine.

RMI is a lightweight due to its easy-to-use native Java model. It supports small-sized distributed object-based application development. A *remote object* has methods which can be invoked by another application located potentially on a different host. The interfaces which state the methods of the remote object are called *remote interfaces*.

RMI distributed object application have two major separate programs:

- A *server* which creates a number of remote objects, makes references to those remote objects as to make them accessible, and waits for clients to invoke methods on those remote objects,

- A *client* which gets a remote reference to one or more remote objects in the server and then invokes methods on them.

The references of remote objects can be programmed either using RMI's naming facility, called as *RMI Registry*, or the application can pass and return remote object references as part of its normal operation. RMI framework provides the necessary mechanism for loading objects and transformation of the required data. It utilizes web servers to load complied classes written in the Java programming language using any URL protocol such as HTTP or FTP.

The following figure is the architectural diagram of RMI application model.

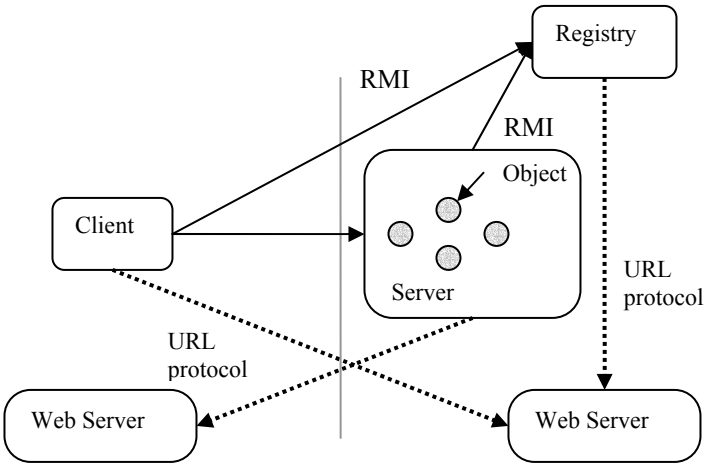


Figure 3.4: RMI Distributed Application Model

RMI is based on remote procedure call (RPC) implementation. A client stub is used as a local representative or proxy for the remote object. Each remote object may have an equivalent server skeleton which is created from a common interface object. The client stub is used to connect RMI Registry, whereas the server skeleton transmits the remote method invocation to the actual remote object implementation.

RMI provides a mechanism, called as *Object Activation*, in which the management of the executed object implementation and persistent references to those objects are provided. An instantiated remote object is called as *active*, while if the object is not instantiated, it is *passive* object. Transforming a passive object into an active object is the process of *activation*. RMI uses lazy activation, which means the object is only activated when the client first invocate the object's method.

The core RMI API provides a default network protocol called as *Java Remote Method Protocol* (JRMP). RMI-over-JRMP (RMI/JRMP) is a proprietary, lightweight protocol and only permits to communicate with other Java RMI objects. Its main features are ease-of-use and usage of distributed garbage collection [13].

The limitation of RMI/JRMP can be achieved by using the communication protocol Internet Inter-ORB Protocol (IIOP) over RMI. RMI-over-IIOP (RMI/IIOP) does not require knowing a separate language, because it uses Java as native language to define the service interface, and it provides flexibility and performance enhancements. RMI/IIOP can interoperate with the applications written other than Java.

3.2 COM / DCOM

Component Object Model (COM) is a part of Microsoft Windows family of Operating Systems and used by developers to create re-usable software, link components together to build applications, and take advantage of Windows services [16]. COM is the combination of other Microsoft technologies including COM+, Distributed COM (DCOM), and ActiveX Controls.

COM is a component-based software architecture which allows components from variety of applications to be combined and built into a new higher-level software application. COM defines a standard for component interoperability. It is independent from particular programming language, available on multiple platforms, and extensible [17].

A COM object is defined as some piece of compiled code used for providing certain service. These objects is different than the objects defined in object-oriented programming, such that one object fulfills a comprehensive requirement in the system. COM defines these objects as *component object* or simply an *object*.

Applications communicate with each other through collection of functions called as *interfaces*. Interfaces provides the application logic used between components. A *pointer* figures out the interface that component objects implements during the interaction of different components. It is only used to call the function without modifying system state. Interfaces are strongly typed and a component object can implement multiple interfaces.

Figure 3.5 displays the relationships among component objects; interfaces and pointers within the context of COM based application.

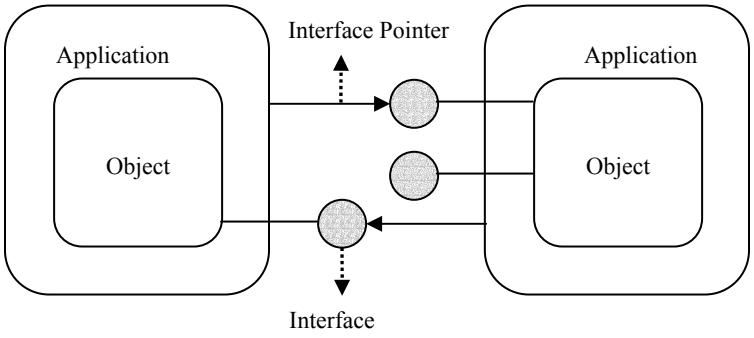


Figure 3.5: COM Architectural Model

Distributed Component Object Model (DCOM) extends COM so as to support communication among objects on different computers on a LAN, a WAN, and the Internet [18]. It allows the components to be used at distributed environment by handling the low-level details of network protocols.

In COM, the interaction between different components is handled by usage of some form of inter-process communication provided by the operating system, since the component cannot call directly the other component that it wants to communicate. When these components are located in different machines, DCOM restore local inter-process communication with a network protocol and hides the communication details from client components.

The following figure is the illustration of DCOM architecture. The model consists of COM runtime, DCE RPC and security provider, which all together provides a background for DCOM wire-protocol implementation.

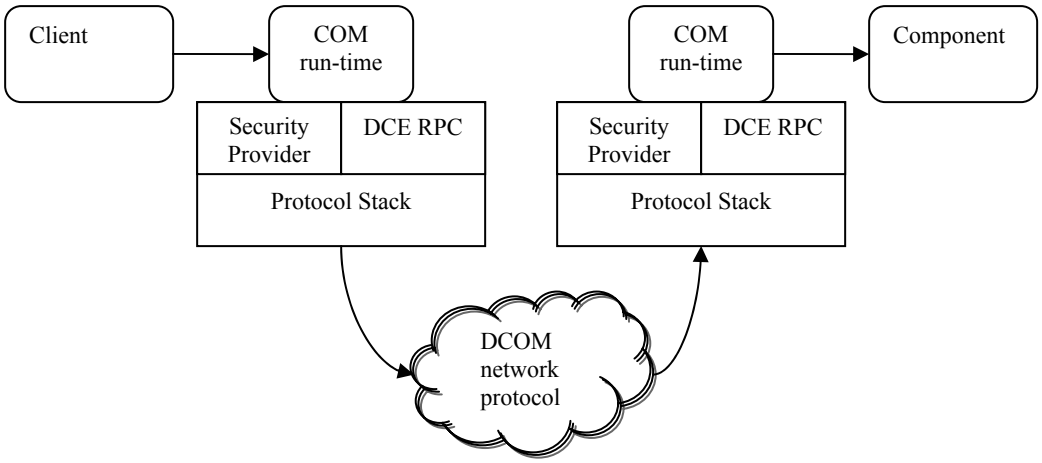


Figure 3.6: DCOM Architectural Model

Architecturally DCOM provides cross-platform development. It allows the integration of platform-neutral development frameworks and virtual machine environments to build up a single distributed application.

3.3 CORBA

Common Object Request Broker Architecture (CORBA) is an object system which provides a framework where objects can communicate with each other in a distributed manner without platform and language dependencies. An object system is a collection of objects that isolates the requestors of services from the providers of services by a well-defined encapsulating interface [19]. In particular, clients are isolated from the implementation of services as data representations and executable code.

CORBA is a specification from Object Management Group. It defines *Interface Definition Language (IDL)*, a core API which classifies a communication infrastructure based on *Object Request Broker (ORB)* for distributed applications, and a TCP/IP based communication protocol called as *Internet Inter-ORB Protocol (IIOP)*.

IDL is a language for describing distributed object attributes and operations which combines client applications to the server for usage of CORBA specific services. Clients and server converse with each other which is based on a specified *contract* and as long as the contract is valid, the implementations of the client applications are independent from this contract.

The main component in CORBA's architecture is the Object Request Broker (ORB), which is a software component for insuring a proper communication of objects across heterogeneous environments. It connects all application components, interfaces and services to be able to achieve interoperability and provide certain functions for distributed applications.

Figure 3.7 is the illustration of the components in architectural structure of CORBA.

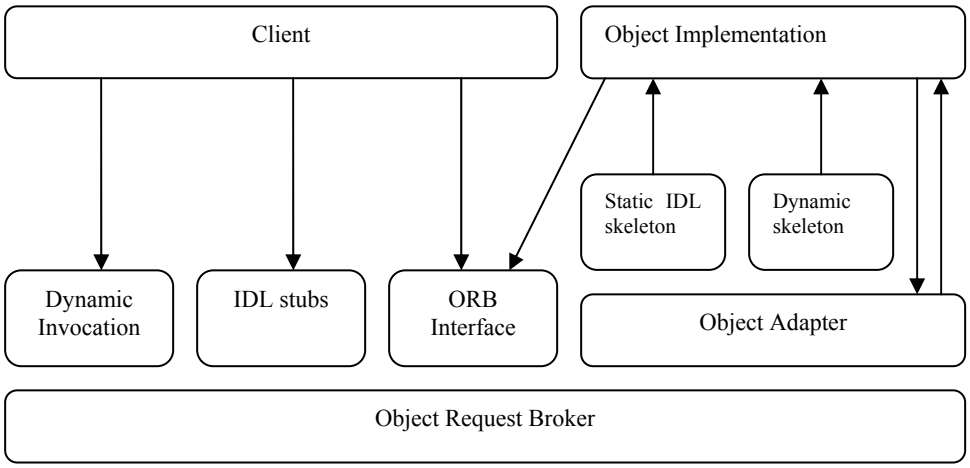


Figure 3.7: The Structure of CORBA Object Request Interfaces

The architecture of CORBA enables the client to be aware of server objects and invoke them through dynamic invocation. Clients can also use IDL stubs which define the way for clients to invoke server objects. ORB Interfaces are the APIs for CORBA specific services. The skeletons provide environment for object implementation. Lastly, the object adapter handles communication between an object and the ORB.

CORBA defines variety of services designed for the enablement of sophisticated communication between different parties with allowing implementation of Quality of Service (QoS) requirements.

Some of these services are defined as follows:

- **Event Service:** provides the proper allocation of events to the related components which register or un-register their interest to certain messages or events.
- **Life Cycle Service:** provides life-consistency of CORBA distributed objects.
- **Naming Service:** allows for distributed components to discover each other by searching one another by its name.
- **Persistence Service:** provides an interface used for storing components on a variety of database management systems.
- **Security Service:** provides a security framework for distributed objects.
- **Transaction Service:** provides two-phase transaction standards among recoverable components.

Interoperable Object Reference (IOR) is the network-addressable reference given to an object in CORBA. IOR encodes the hostname and port to which the messages are being sent and provides a mechanism to distinguish different objects.

3.4 Web Services

Web services are a set of XML-based technologies which aim to provide a standard way for communication of different applications and interoperability of heterogeneous computing environments. Web services use standard Internet technologies for messaging and data exchange, which makes them suitable for development of an application accessible in a platform-independent environment.

A formal definition of Web services are as follows [9]:

"A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine process-able format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards".

The implementation logic of a Web service can be written in any language and in any platform. The technologies of Web service makes this implementation logic accessible using standard Web technologies, such as HTTP and Web browser, and result in a faster and more dynamic communication for connected applications.

Web services architecture is service-oriented in which a consumer searches for an available provider of a service to execute a certain function. A service registry contains published services and when there is a request for a service, it returns back the information which allows requestor to locate and invoke the service.

Figure 3.8 illustrates the roles in Web services architecture.

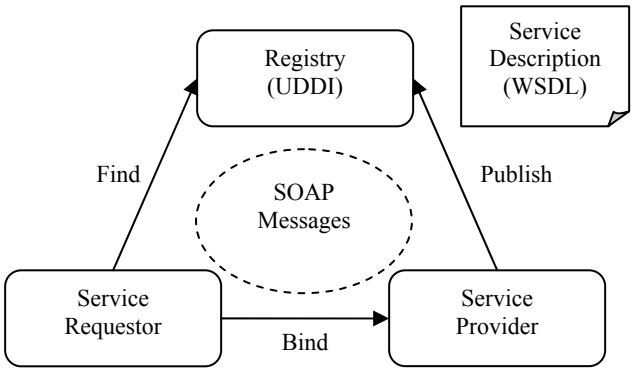


Figure 3.8: Web Services Architecture

Web services architectural model is based on a layered family of technologies. Each layer is interrelated with each other, and provides a level of abstraction and functionality for Web service based application development.

The following figure illustrates some of these base technologies.

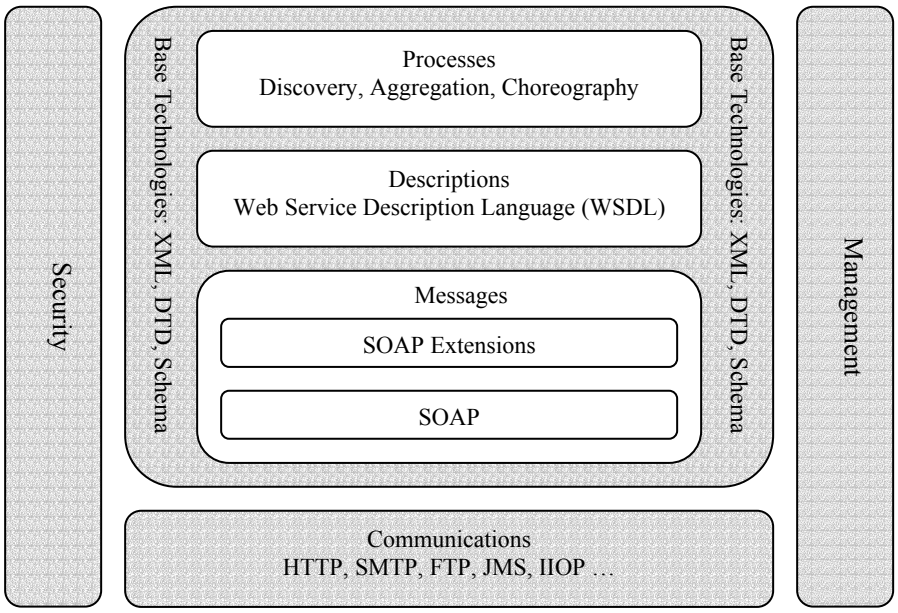


Figure 3.9: Web Services Technology Stack

XML (Extensible Markup Language) [20] is the basic foundation of Web services and a base language for defining and processing of data. XML support dynamic content creation and management. It expresses the data as self-describing elements, which then may have child elements. It is a family of technologies used for creation, definition, and transformation of XML documents.

Some of XML-based technologies include:

- *XML schema*: an XML document that defines data types, structure and allowed elements in an associated XML document,
- *XML namespaces*: the uniquely qualified names for XML document elements and applications,
- *Extensible Style-sheet Language Transformations (XSLT)*: used for transformation of XML documents into other XML document format.

Simple Object Access Protocol (SOAP) [21] is a specification for describing how to read and format XML messages between service consumer and service provider in Web services architecture. It provides a messaging framework independent from operating system, programming language, or distributed computing platform. A SOAP message has three parts:

- The **SOAP envelope:** it is the top element of the XML document and specifies that the message is a SOAP message.
- The **SOAP header:** it contains optional attributes, application context information and directives used in processing of the message. It specifies the position of the message which can be either an intermediary point or an eventual end point.
- The **SOAP body:** it contains the actual application data being sent between consumer and producer.

SOAP messages support one of the two types of message exchange patterns: *document-oriented format* and *RPC style format*. Document-oriented message exchange allows service consumer and producers to exchange XML documents as data representation, whereas in RPC style the data is passed as arguments and contains sufficient information to invoke a method and fulfill a function offered from producer. SOAP is fundamentally one-way communication model; however it is possible to implement a request/response model.

Web Service Description Language (WSDL) [22] is an XML-based language for describing Web services interfaces and specifying a service contract. It provides a standard way to describe the data types passed in messages, the operations to be performed on the messages, and the mapping of these operations to the transport protocol.

WSDL document is structured into logical sections to create a final Web service description, which is then used from service consumer to interact with the service. The description contains the following major elements:

- The **definitions** element is the root element of the XML document and declares the namespaces and the service name used by the service.
- The **types** element contains data type definitions used in the conversation of service producer and consumer.

- The **message** element represents the input and output parameters passing between the service requester and the producer, and describes various messages the service exchanges.
- The **operations** element represents a particular interaction in the conversation.
- The **portType** element describes the set of operations that the service supports and provides information including operation name, input and output parameters.
- The **binding** element represents a particular binding of the operations to a specific implementation protocol, such as SOAP.
- The **service** element is a collection of ports. A *port* describes a network location where the service can be invoked.

Service consumers use WSDL to interact with the service using the following scenario shown in the following figure [1].

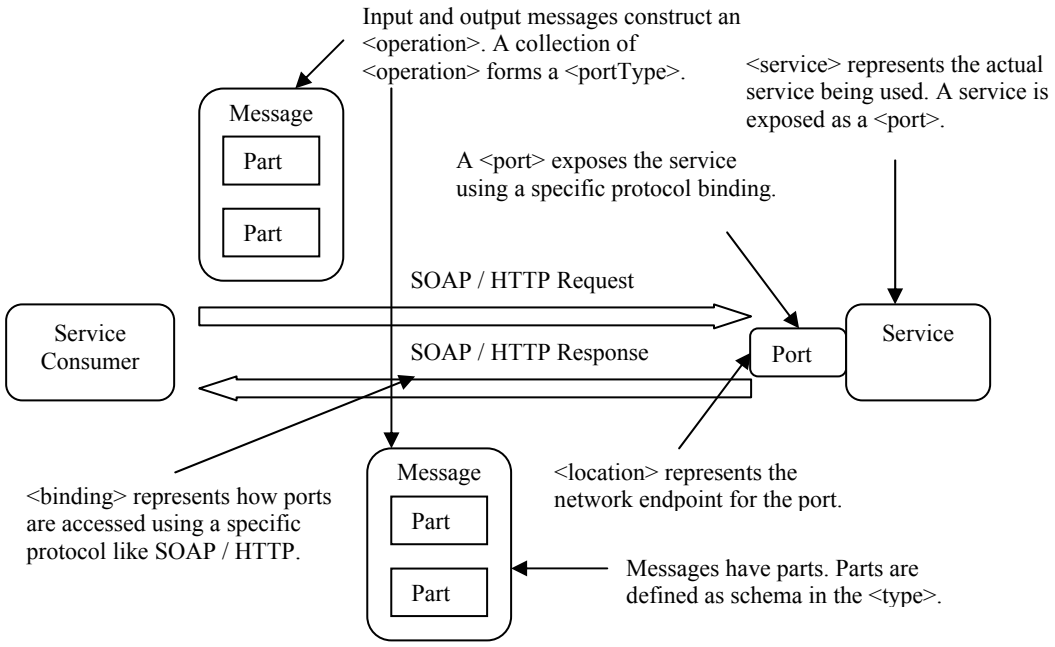


Figure 3.10: The Interaction of Service with Its Consumer

The other main technology of Web Service is the ***Universal Description, Discovery and Integration (UDDI)*** [23] specification which defines a framework for businesses to publish their services, discover the available services and construct a global registry. A registry provides the available services to consumers, and allows service producers to be known their services by consumers.

There are three types of registries:

- **White pages:** allows consumers to search for an address, contact information, and known identifiers.

- **Yellow pages:** includes classification of information based on standard taxonomies.
- **Green pages:** indicates the services offered and reference it to a specified business process.

UDDI includes four main parts in its architecture, which are businessEntity, businessService, bindingTemplate, and tModel. The businessEntity contains descriptive information about a particular business organization and provides references to the offered services. The businessService indicates the actual service which has a unique identifier. The information necessary for a client to invoke a service is provided by bindingTemplate. The tModel (technical model) is primarily used to point to the external specification of the service being provided.

The following figure represents the usage scenario of these primary core data structure types [1].

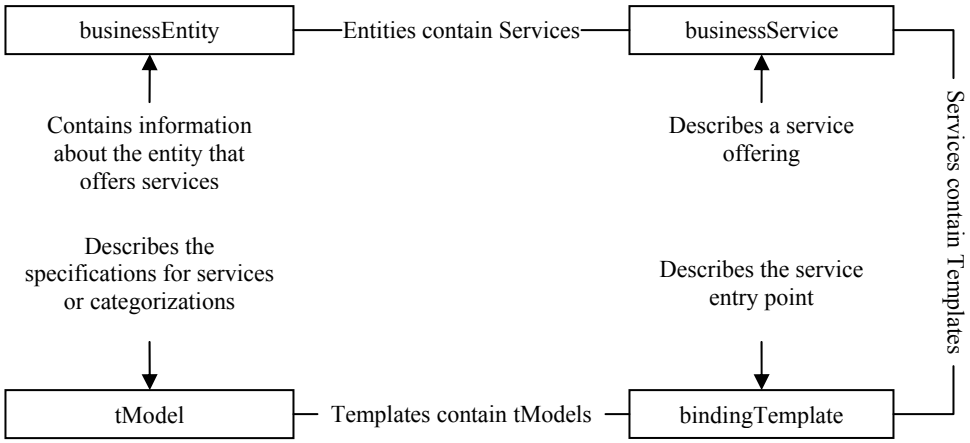


Figure 3.11: UDDI Informational Structural Model

Web services have broad usage areas including accessing back-end software systems, such as databases, integrating diverse applications to achieve *business-to-business (B2B)* interaction and *enterprise application integration (EAI)*, and building Internet-based applications which provide user to access the system resources.

3.5 Feature Based Comparison of Technologies

The technologies that have been introduced in this section can be used to implement Service Oriented Architecture. However, the capabilities of each technology determine the complexities that it aims to solve when building a SOA. Although Web services are new and revolutionary approach to service-based software development, it is critical to conclude a certain technology as a universal remedy.

The following part will set up various significant SOA characteristics and elaborate these technologies to discover the competencies of conforming SOA requirements.

Development Model

Although SOA involves the development of services in its structure, especially the approaches before Web services for building service oriented applications are based on distributed object communication, such as JMS, which focuses on principally messaging, and RMI. A key difference between CORBA and COM/DCOM with Web services is that CORBA and COM/DCOM provides object oriented component architecture, however Web services supports service based messaging structure.

Interface Definition Language

Each technology introduced its own interface definition language used to describe the service in public. Respectively, COM/DCOM uses Microsoft Interface Definition; CORBA uses CORBA Interface Definition Language (IDL), and J2EE uses Java programming language. Web Service Definition Language (WSDL) is proposed within the Web services technology stack and universally accepted language. It utilizes XML and provides the ability to bind to multiple transports.

Coupling

A significant feature of SOA is to enable a loosely-coupled architectural model development in which the services are recombined and repackaged in order to build a new and compound application structure. A Tightly-coupled application has predetermined relationships within its application structure and it is difficult change this structure.

JMS is based on messaging structure. As the producer and consumer of the message are not dependent on each other, JMS achieves loose-coupling by some means. Web services also target loosely-coupled application development; however there can be tight coupling at the interface definition level and protocol binding. J2EE, COM/DCOM and CORBA require rigid interface development and do not allow loosely-coupled interface definitions.

Platform Independence

Architecturally services might be combined from different heterogeneous applications and platforms which involve diverse component structures written in a variety of programming languages. A platform independent service oriented technology allows building a competent and sophisticated service-based application infrastructure by assembling these services to build a service oriented architecture.

CORBA promotes platform independent development environment in which the application can be written in any language, however, it is still needed for communicating applications to use the same Object Request Broker software from the same vendor for a full support of interoperability. Web services support platform independence by leveraging standard Internet technologies such as XML and HTTP. J2EE and COM/DCOM provides vendor-specific application development environments and the communicating parties should act upon the same standards.

Interoperability, Support for Open-Standards

The previous technologies before the introduction of Web services support their own protocols located over TCP/IP. The applications built with these technologies such as CORBA and DCOM are dependent on a single vendor's implementation. JMS and RMI address the usage of the identical environments on both the communication sides to interoperate. Web services standardize the communication protocol by using widely-accepted, open Internet technologies for building interoperable distributed applications. This advancement allows for applications easily access to and communicate with each other.

Web services requires no certain programming language, protocol or operating system between communicating parties, so it achieves a high-level of interoperability requirements and provides a means of universal solution when building a service oriented architecture in terms of interoperability.

Location Transparency

Location Transparency is one of the main characteristic of SOA. SOA requires that the consumer of a service should dynamically locate the service from registry without knowledge of the exact position of the service. Accordingly, if it is required to move the service from one place to another, the consumers of this service should not affect from this changing.

CORBA Interoperable Object References (IORs) provide location transparency for objects and strongly typed. Web services use URLs to indicate theirs location, which are not strongly typed.

Communication Mode

Most distributed object based technologies support RPC-style synchronous communication, such as COM/DCOM and CORBA. JMS specifically supports asynchronous communication, however still it is possible to be able to build synchronous communication-based JMS applications. Web services can be used with both synchronous and asynchronous communication mode.

Service Discovery and Registry Support

The technology used in building SOA should provide a mechanism for consumers to dynamically discover the available services from registry, and for a provider to publish its services, which then will be available publicly.

CORBA provides standard services for service discovery and registry support, such as the Naming service, which maps a logical name to an object reference, and the Trading service, which allows finding a service based on specified service properties. Web services use UDDI, a general-purpose registry that can be used to query for available services.

Security

Security is the main concern especially when building large-scale, distributed application infrastructures. It involves supporting of features such as authentication, authorization, encryption, data-integrity and auditing.

JMS provides J2EE-compatible security standards mostly implemented by the JMS provider and it is vendor-specific. CORBA supports security by standard CORBA Security service. The main drawback for Web services is that there is still ongoing-work for Web services to support a sophisticated security framework. Web services can use HTTPS for security support at transport layer, or other Internet technologies to build security such as SSL or XML-signature. The standardization effort is mainly around WS-Security (see Appendix), which enables an interoperable end-to-end security model in the future.

Transactional Support

Certain applications require that the data should be persistent and transactional. Web services currently do not support transactions. WS-Coordination and WS-Transactions are ongoing standardization models for Web services to support loosely-coupled transactional models with compensation. JMS only supports transactions to the messaging queue entry point and the consumer of the message is responsible for transactional support after getting the message from the queue. CORBA provides a satisfactory transactional model by CORBA Object Transaction Service which offers both short lived database transactions and long lived business transactions that may involve integration of different autonomous business systems.

Table 3.1 displays a complete comparison of these technologies based on the discussed characteristics and features.

	JMS	RMI	COM/DCOM	CORBA	Web services
Development Model	Object Oriented Messaging Development	Object Oriented Development	Object Oriented Component Development	Object Oriented Component Development	Component Oriented Service Development
Interface Definition Language	Java Programming Language	Java Programming Language	Microsoft Interface Definition Language	CORBA Interface Definition Language (IDL)	Web Service Definition Language (WSDL)
Coupling	Loose-Coupling	Tight-Coupling	Tight-Coupling	Tight-Coupling	Loose-Coupling
Platform Independence	Targets Java Platform	Targets Java Platform	Targets Microsoft Windows Platform	Platform Independent	Platform Independent
Interoperability, Support for Open Standards	Not Interoperable, Java Standards	Not Interoperable, Java Standards	Not Interoperable, Windows Platform Standards	Not Interoperable, CORBA Standards	Interoperable, Open Internet Standards
Location Transparency	Location Transparent through Message Provider, Vendor-specific	Location Transparent through RMI Naming Facility	Location Transparent through Interface Pointers	Location Transparent, CORBA IORs	Location Transparent, URL Addresses
Communication Mode	Asynchronous Mode, Possible to Implement Synchronous Mode	Synchronous Mode, RPC-based Development	Synchronous Mode, RPC-based Development	Synchronous Mode, RPC-based Development	Supports both Synchronous and Asynchronous Mode
Service Discovery and Registry Support	Registry of Messages by Message Provider	RMI Registry	Service Discovery through Interface Pointers, No Registry Support	Standard CORBA Services, CORBA Naming and Trading Service	Supports through UDDI
Security	J2EE-compatible security standards	J2EE-compatible security standards	Microsoft Windows-compatible security standards	CORBA Security Service	Ongoing-work for Standardization
Transactional Support	Partial Support	Supports	Supports	CORBA Object Transaction Service	Ongoing-work for Standardization

Table 3.1: Feature Based Comparison of Service Oriented Technologies

As it can be seen from the table, Web services provide a significant improvement in the evolution of service-oriented computing environment. As Web services introduce platform independent, loosely coupled and open-standard based communication background, the new approach to application development bring in flexible interaction of business partners and ease of solving integration complexities between distributed applications.

However, Web services technologies are still evolving and there is continuing effort to support Web services with additional features and functionalities, such as security and transactional support. On the contrary, J2EE and CORBA are robust and mature computing environments with offering satisfactorily features support, reliability and scalability.

Other concern when building a SOA is that the technology should allow *rapid application development (RAD)* with providing a universal programming model, which Web services support. Although especially CORBA is widely used technology and provides a rich development environment, it requires learning a new programming model and does not support a straightforward and cost-saving application development.

Presently Web services are gaining momentum to be a dominant technology over its competitors. The future evolutions of Web services will shape its acceptability and competency, and determine its position in computing environment.

4. Service Orientation in Software Design and Development

Service based architecture utilizes software services to build an application structure in which the organization's requirements are satisfied through reusable and distinct functional modules. A service provides well-defined possibilities for application integration, distribution of business functions inside the organization and across organizational boundaries, and improvement of software quality by allowing adaptable interfaces suitable for accessing across a network.

To construct service oriented architecture requires a clear understanding of business functional domains and processes including intelligent decisions about how to implement business functions as separate services. Modeling and designing of services within an application structure needs more sophisticated approaches and considerations than traditional application development.

Developing a service based application is different from a standalone application in a sense that designing, developing, managing and maintaining of services are precious and require advanced development stages. The main benefit of SOA is gained over the long term of service implementations.

4.1 Service Oriented Design

Services are complex modules that need to be applicable to certain criteria and development considerations. Good service design result in effectively encapsulated business logic and data communication which correlates real-world processes with the technical computing environments. Services are designed in a way that they represent a precise problem domain modeled to solve, require minor maintenance while they are executing, and provide high usage alternatives in specified backgrounds.

Service based enterprise application architecture involves variety of services including business process services, technical services, and user interface (UI) services. These services can be structured independently, without requirement of the placement on a single computer. Such application infrastructure is obviously differ from a traditional architecture and requires unique design considerations which allow the application to utilize networked computational resources with offering several key advantages.

It is needed to take into consideration that services bring the main benefit when they are designed for integration [24]. Services are used to integrate disparate applications and software systems within different communicating partners and computing environments. The design of services also should support the upcoming potential applications not known at development phase of services.

The essential difficulties when developing enterprise service oriented architecture is to decide on which service is needed to support the business processes of the organization, clarified through ***service oriented analysis***, and how each service should be built, described in ***service oriented design***. The other concern is to simplify the technical infrastructure in which the services are deployed and managed.

Service oriented analysis involves the clarification of the problem domain and definition of the functional and nonfunctional requirements necessary for the entire system. Since service orientation requires specific decisions on the structure and modules of the application architecture, it is needed to consider some methodologies and design issues when building services in the design and analysis phase of the application development.

Service oriented analysis articulates some of the following key characteristics differ from traditional software analysis:

- Services support multiple applications that are dynamically constructed. Therefore, a clear explanation of the service descriptions and boundaries is required to maintain the needs of other applications.

- It is needed to portray the business domain of the organization and functional decomposition of the business into atomic subject areas and processes. The analysis of the subsystem consists of creating object models to represent the internal structures, which is then realized by services.

- A service is needed to be designed and structured in a way that it is agile to system changing, however it is supposed to be up to date.

- Nonfunctional requirements are needed to be defined to satisfy the quality aspects of the services. These aspects involves definition of reusability of services, performance and security levels, availability, maintainability and other issues that surrounds services.

The analysis of a service based system contains a feature list that the services are going to support, the use cases that describe potential ways for consumers to utilize these features and a detailed explanation of the components that will implement the services. The *business architecture* [25] helps the designer to classify the problem domains and elucidate possible usage of services in businesses. It describes potential use cases by illustrating the way of business procedures and is independent from any technology.

4.1.1 Service Design Considerations

Service design includes several fundamental issues and guidelines that need to be clarified to overcome the complexities of building service based application. During the design phase of a service based application the services are structured according to the specific design considerations, some of them will be described as in following.

Modularity

Modularity is an important concept in service construction. A service needs to be applicable for the following principles to support good design [26]:

- *Modular composability:* the service should allow to be integrated with other services to structure a new application. It is an indication of a good design if the service can be composable dynamically with other services.

Modular composability, also called as *bottom-up design*, starts from defining integration services based on the functionality of existing applications and the requirement for the integration. The promise of this approach is to set up interoperability standards between diverse applications.

- *Modular decomposability:* it refers to be able to separate the whole application into small modules, which state a discrete business function. These small modules are reusable in composition of services and building composite applications.

Modular decomposability is often referred as *top-down design*. This approach to SOA design starts with building a business model of the enterprise, defining both processes and semantics, and then mapping this model onto business services [27]. It allows better arrangement of business and application perspectives.

- *Modular understandability:* the service does not need any other service or application module to be able to articulate a business function. The boundaries of the service are the verification of understandable and functional business logic.

- *Modular Continuity:* it implies in case there is an implementation altering in one service; it should not require the other services and consumers to be effected. A service needs to hide its internal implementation structures from outer surface.

- *Modular Protection:* if there is an unusual condition caused by one service, the other services and consumers should not collide in order to support an expected execution of services in its regular environment.

Service Abstraction

Service abstraction is a design issue which guarantees that a service is independent of any specific application infrastructure and technology. The main aim is to focus on the functionality of the service rather than how the service implements this functionality. Abstraction requires that the internal working of the implementation is needed to be hidden from service consumer. In this way, it is possible to invoke the service by different existing applications.

The following figure illustrates different parts of the service including its description and implementation logic within the context of other services. The key feature of a service is that one description might have more than one service implementation associated with it [28].

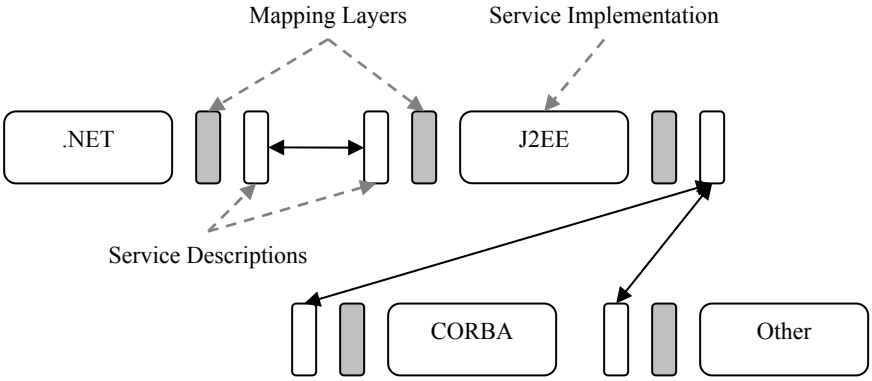


Figure 4.1: Service Abstraction

Service abstraction allows to access different service types including services that supports wrapped legacy applications, composite services and applications built with other services.

Service Classification

When the services are identified, service classification helps to determine the composition and the layering of services in a service hierarchy, as well as coordinates the individual and interdependent services in SOA layered architecture. Classification also facilitate to the realization and creation of new services which are based on small-grained services layered on the bottom.

Service Interfaces

Services communicate each other through messaging. When defining a service, it is needed to identify which messages being sent and received by the service with the proper sequence of these messages. The service interface reflects this message exchange to the other communicating parties.

RPC style communication is based on *interface semantic* in which the interaction between services is done with method invocation encoded in request/response calling. Message based communication handles the service interaction through messages that carry the data and other information. This is referred as *payload semantic*. Web services use XML as message transportation and provide *document-centric* messaging approach. Document centric messages are semantically rich messages where the operation name, its parameters, and the return type are self-descriptive.

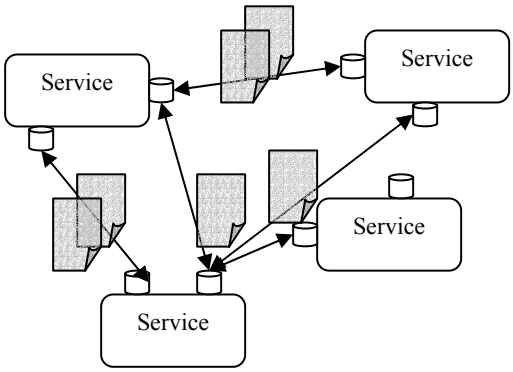


Figure 4.2: Services and Messages

Service interfaces describe the complete information needed for the messages to activate. This information involves the functionalities that the services provide including parameters and return values of the service methods. Designing a service interface is challenging in a sense that the interface should be sufficient descriptive to support multiple service consumers in an efficient way.

Aggregation and Composition

Aggregation and composition of services provides a better design of service oriented architecture in a way that fine-grained services can be used internally and coarse-grained services provides the application function to the communicating parties. Figure 4.3 illustrates that services can be delivered at a number of different granularities to suit different requirements [29]. Fine grained generalized services are composed into coarser grained services that are specialized to provide a specific function within the enterprise.

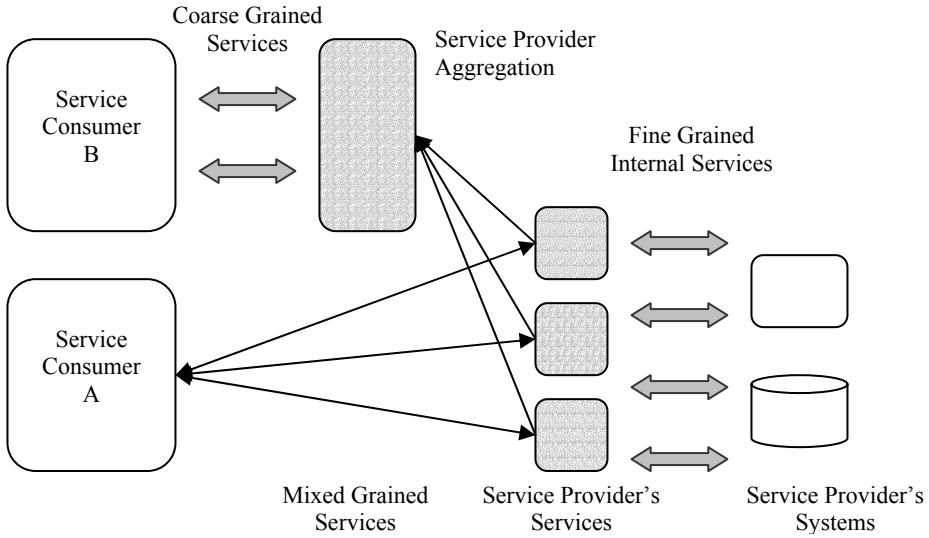


Figure 4.3: Aggregation and Composition of Varying Granularity of Services

Exception Handling

It is unavoidable that the system may have errors during its execution. To define the service exceptions is a good design issue which allows handling of many errors programmatically in advance.

4.2 SOA Meta Model

The W3C Services Architecture Group defines SOA model in terms of invocation message, implementation, owning organization, and metadata describing the service [9]. The *message-oriented model* defines a message in terms of its content, delivery transport, and creating and executing agents. The *resource-oriented model* defines resources in terms of URI, representation, and owning organization. The *policy model* defines policy in terms of its subjects, organization, and supporting policy.

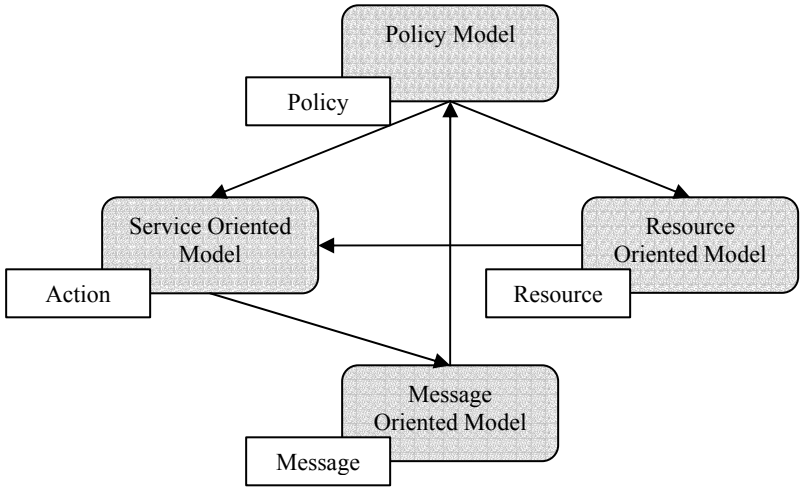


Figure 4.4: Meta Model of Service Oriented Architecture

The basic service oriented model consists of *services*, *processes*, and *organizations*. SOA models the enterprise as collection of business services accessible across the enterprise. The main features of the business services can be described as coarse grained, process centric, loosely coupled, distributed and have stateless invocation. Business services are orchestrated into business processes to provide certain required business functionality. Organizations manage the creation, execution, and maintenance of the services and processes.

Extended SOA model provides to the enterprise *semantic data model* and *documents* as an extension of SOA definition. The semantic data model defines the standard business data to effectively create ontology of the enterprise. Documents are legal entities that define the responsibilities of the enterprise and communicating parties. A SOA implementation should include documents and conserve the connection between enterprise transactional data and supporting documents.

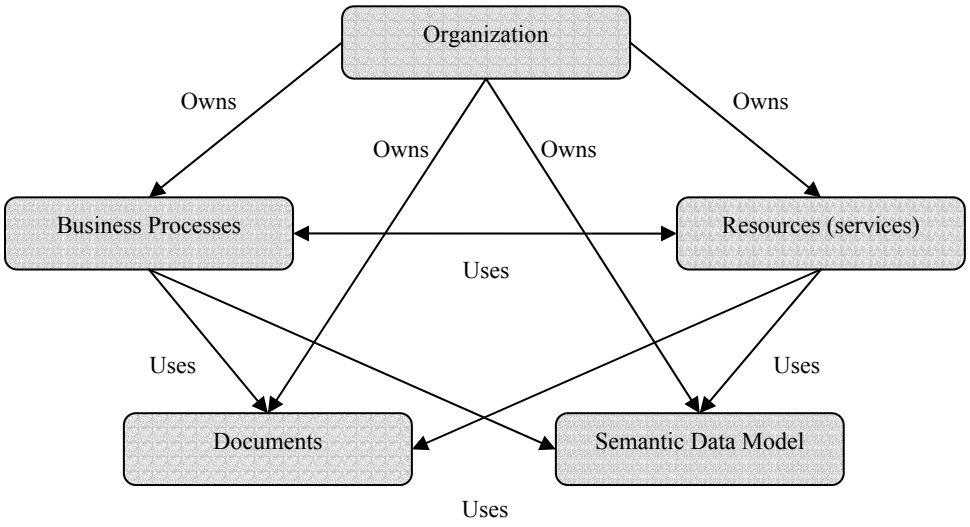


Figure 4.5: Extended SOA Model

The W3C Services Architecture Group defines Web services architecture as an instance of SOA in which the service is described as an abstract notion that must be implemented by a concrete agent. The agent is the concrete entity, meaning that a piece of software, that sends and receives messages, while the service is the abstract set of functionality that is provided.

4.3 Service Oriented Modeling

Modeling is a required activity all through the range of software development life cycle phases and involves design and measurements of necessities, definition of architectures and software systems, and clarification of modeling types appropriate for the needs of software modules. Modeling provides an effective way to handle the requirements and complexities of software development.

Service oriented modeling utilizes effectual approaches to identify the fundamentals of SOA by applying analysis and design techniques of service orientation. Modeling allows clear description of the elements in SOA layers and provides an environment in which these elements are visualized or textually described to be able to formulate significant architectural decisions for structuring the service-based application.

Basically the process of service oriented modeling and architecture consists of three general steps [10] which includes identification, specification and realization of services, components and service flows, also called as choreography of services.

Service identification is the clarification of services which is done through decomposition of business domains, analysis of existing system resources, and definition of required functionalities by applying top-down or bottom-up design approaches.

Service classification or categorization is a required activity in *service specification* which allows aggregating all services into a service hierarchy after they have been identified. This process should reflect the granularity, usage areas, composition and layering of services. The components that build the services are also specified in this phase.

Service realization involves recognizing of the services within the service oriented application. The services are possibly used for different purposes such as integration of diverse functionalities, business or data services, or to provide certain functions such as security and monitoring. The modules, components, and legacy applications that use these services need to be identified with the aim of an appropriate execution environment.

The activities described above are illustrated in the following figure.

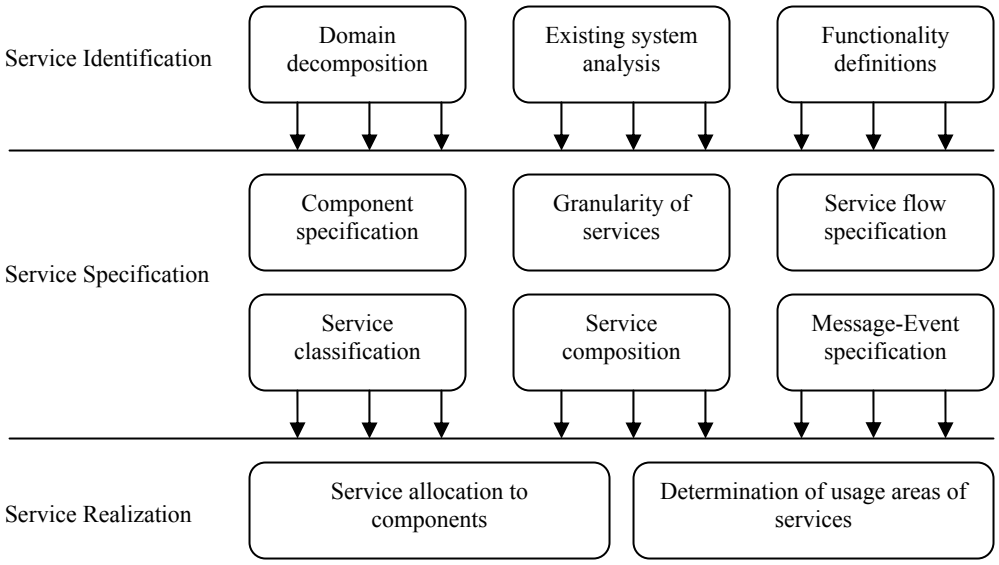


Figure 4.6: Service Oriented Modeling Method

Service oriented modeling method described above provides specific direction on the analysis and design activities for determining fundamental aspects of service oriented architecture. The modeling of services should reflect creation of business specific services and decisions about how they will be composed into applications using choreography.

4.3.1 Unified Modeling Language

Unified Modeling Language (UML) is proposed by Object Management Group (OMG) [30] and a language for modeling, visualizing, constructing and documenting all features of software systems and related artifacts. It offers a better support for *model-driven development* (MDD) by providing advanced abstraction and automation methods for modeling software.

UML standardizes the modeling of software by providing both graphical and textual syntax and semantic, so that it is possible to apply generic rules of UML to all stages of software development. It allows for developers to concentrate on the abstraction of software rather than the implementation details.

Primarily UML provides use case diagram, class diagram, behavior diagrams, and implementation diagrams. All these diagrams can be used to model different functionalities of application architecture. The UML specification [31] describes overall aspects of UML diagrams including version 2.0 features.

UML provides in version 2 a meta-model for software construction which can be applicable for transformation of models to interoperate with each other and generation of automated tools for compilers, testing and other features in software development stages. It is an extension version which provides a standard way to model specific areas such as web-based applications and service oriented architectures through *profiles* in which the specific problem domains and solutions are abstracted and modeled in an effective approach. A profile identifies the UML elements to be used, extension points, and the rules for gathering these elements. UML 2.0 improves modeling of large-scale software systems and grounds validation and clarification of various modeling concepts.

An example UML 2.0 profile is provided [32] which demonstrates modeling of services, service oriented architecture, and service-oriented solutions. The aim of this profile is to provide a common language for describing enterprise-wide service portfolio. The logical divisions contain service specifications which act as the contracts between the service clients and providers, and other related artifacts.

UML 2.0 Meta class	**Stereotypes**
Class	Message, Service Partition, Service Provider
Classifier	Service Consumer
Collaboration	Service Collaboration
Connector	Service Channel
Interface	Service Specification
Port	Service, Service Gateway
Property	Message Attachment

Table 4.1: UML 2.0 Meta Model Elements for SOA

UML 2.0 profile outlines individual stereotypes, the details of which specify its Meta class, properties, and any constraints which should be applied when using the profile. As an example, a stereotype *message* is represented as a class and provides semantics for representation of concepts defined in the Web Service Description Language specification. It has property named as “encoding”. A description for stereotype *service* can also be described as it extends port and provides semantic for an end-point implementation of service consumer-provider interaction.

Simplified illustration of the example profile is shown in following figure.

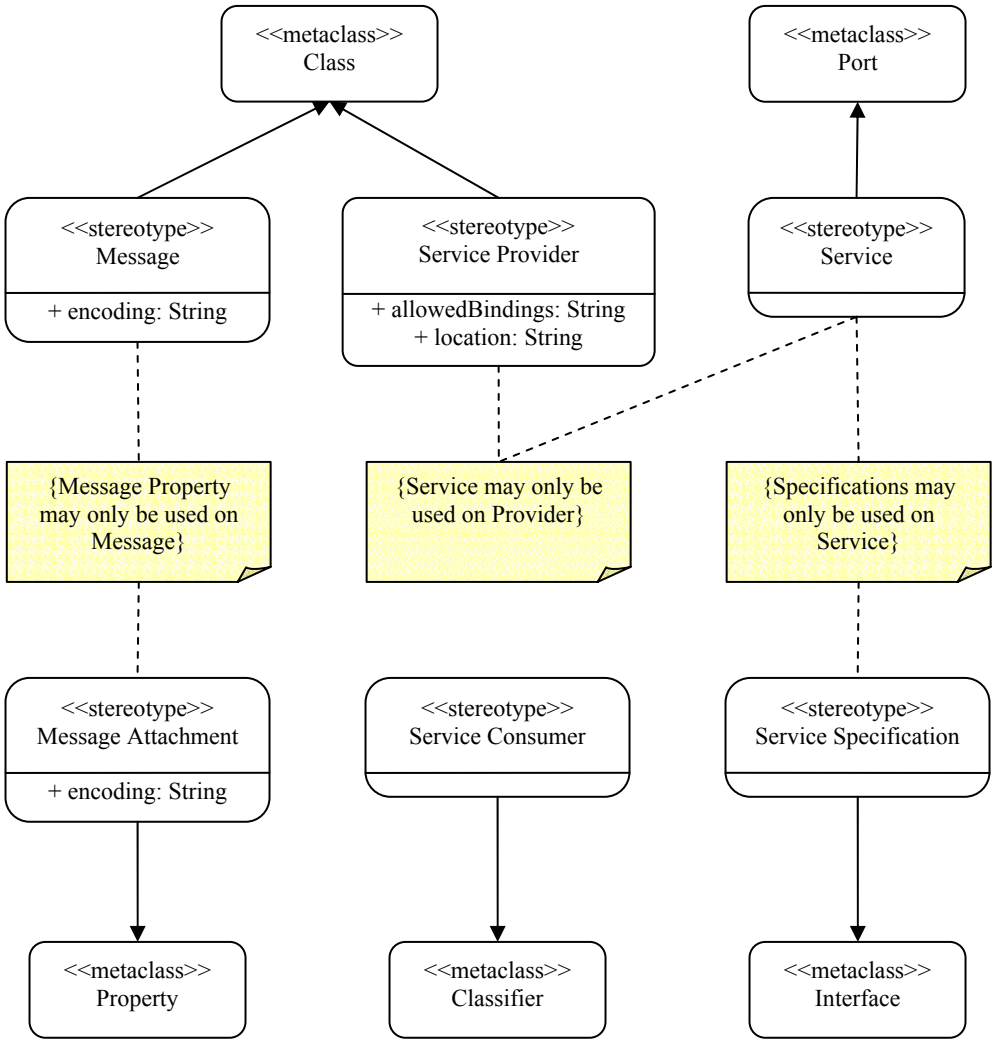


Figure 4.7: An Example UML Profile for Services

This example profile is mainly rested on stereotypes which categorize the elements of a model. Stereotypes are good for the readability of the model and guiding the behavior of a model translator tool. Profiles as well can be defined for a specific technology, such as CORBA, and particular data modeling technique.

4.3.2 Model Driven Architecture

Model Driven Architecture (MDA), proposed by Object Management Group (OMG), is a conceptual framework which allows modeling at distinct levels of abstraction and defines transformation between different model types, as well supports automated tools for the evolution of modeling and model-driven development (MDD) technologies [33].

MDA is based on several open OMG's modeling specifications:

- **UML (Unified Modeling Language):** used for identifying application development models and transformations
- **MOF (Meta-Object Facility):** used for defining *meta-models* which describe the models in a clear and definite way in order to have the ability to analyze, automate and transform them.
- **CWM (Common Warehouse Meta-model):** It covers the full life cycle of designing, building and managing data warehouse applications and supports management of the life cycle.
- **XMI (XML Metadata Interchange):** provides an XML tag set used for passing UML information between tools, databases, and applications.

MDA allows modeling of enterprise applications using these modeling specifications and transformation of the models into specific application development platforms including CORBA, J2EE, .NET and web-based platforms.

MDA is the evolution of model driven software development technologies in which the modeling is not characterized only with graphical notations and tools for transformation of models. *Roundtrip engineering* (RTE) is the notion of bi-directional approach to code generation from an abstract model, and MDA is the advancement of this technology wherein the model is the heart of application development. The code is automatically generated from this model and described using standard languages. MDA provides the fundamental features such as model classification in which the business aspects of the model are separated from the details of the platform and refinement of the model in an independent way from the code execution.

MDA encourage usage of system resources in an effective way by providing modeling of software in a platform, network structure and programming language independent manner. In this way, it is achievable to provide a universal approach for integrating complex applications written in different languages and running on different platforms. It offers inclusive interoperability modeling approaches for developing interconnected systems.

MDA defines diverse levels of modeling:

- **Computation Independent Model (CIM):** provides modeling of the system environment independent from its technical structure,
- **Platform Independent Model (PIM):** provides modeling of the application independent from a particular platform,
- **Platform Specific Model (PSM):** extends PIM with particular platform implementation,
- **Platform Model (PM):** provides modeling of a particular platform,
- **Transformation Model (TM):** defines a model for the transformation from a specific PIM to a specific PSM.

The initial step in MDA development process is to create a Platform Independent Model (PIM), expressed using UML or any other well defined modeling language that allows interpretation by a computer. PIM is the logical view of the application structure without any specific technology details, which can be mapped to one or more Platform Specific Models (PSM) that represents the source code implementation and defines the technical platform, for instance COM, CORBA or Enterprise JavaBeans (EJB). The process of converting a PIM into a PSM is called *transformation*. Application development in MDA can be structured around definition of meta-models, a set of models by imposing transformations between models and automation of these models through tools.

MDA allows modeling of well defined interfaces and services in SOA based application development. The following figure represents an example MDA development process.

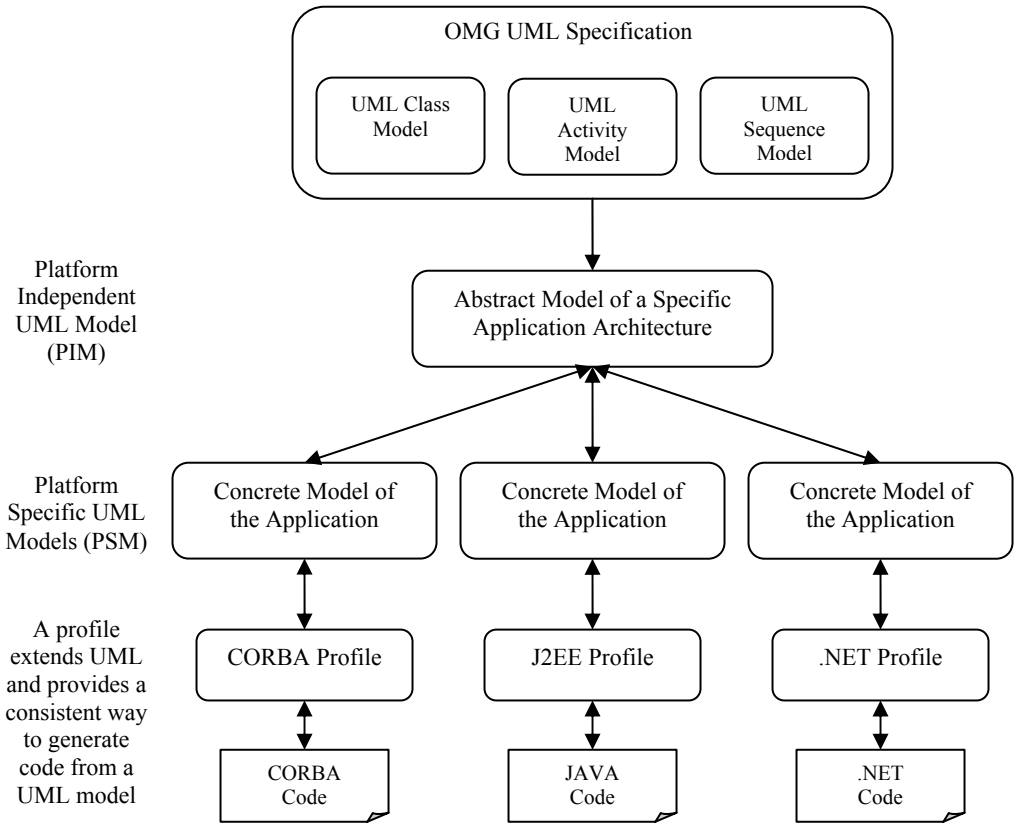


Figure 4.8: Model Driven Architecture Development Process

4.4 Implementation Models for Services

The Middleware Company proposed SOA Blueprints [34] which represent a comprehensive attempt to define and implement an enterprise application architecture that utilize SOA best practices in real-world computing environments.

The SOA Blueprints specifications identify the following SOA implementation patterns for services:

Synchronicity: Services can be invoked in one of two modes, synchronously or asynchronously. The mode chosen for a particular service depends on its potential usage, how long the service takes to run and how reliable the service invocation needs to be.

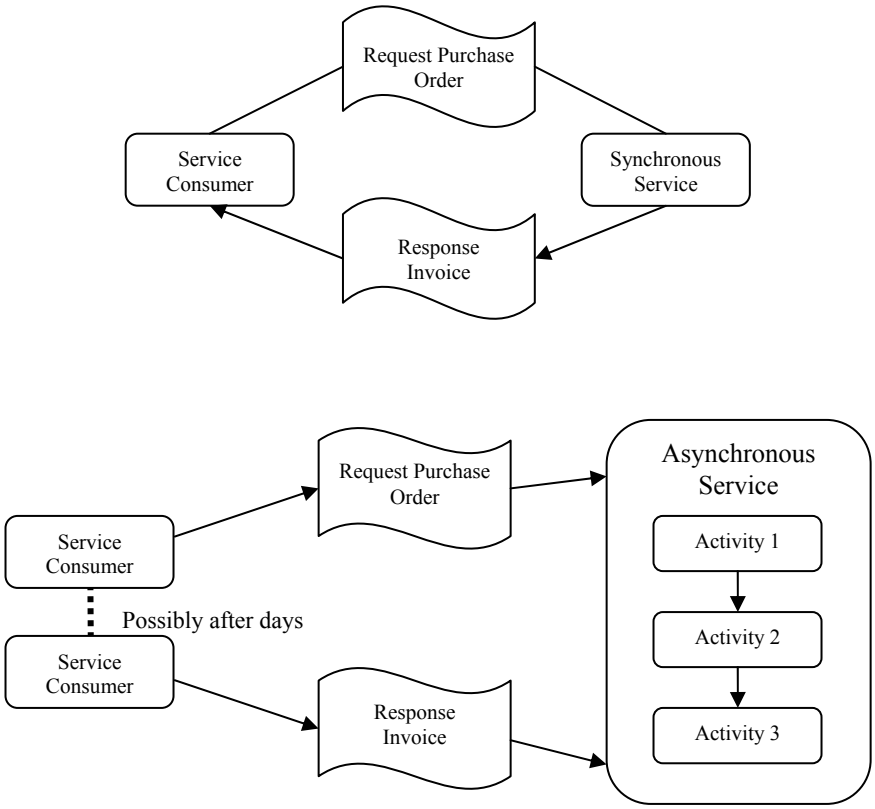


Figure 4.9: Synchronous and Asynchronous Services

Component Services: A component service is a simple atomic action on a simple entity that does not depend on another service to function. As an example database access to a single table can be thought of as a component service.

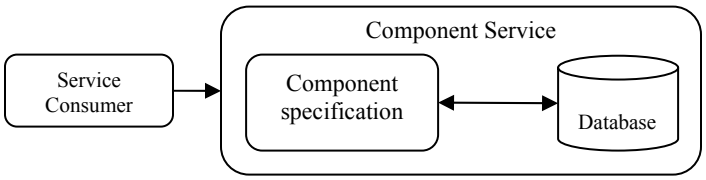


Figure 4.10: Component Service

Composite (Business) Services: A composite service is also atomic in nature, but orchestrates the invocation of component services into a business level process.

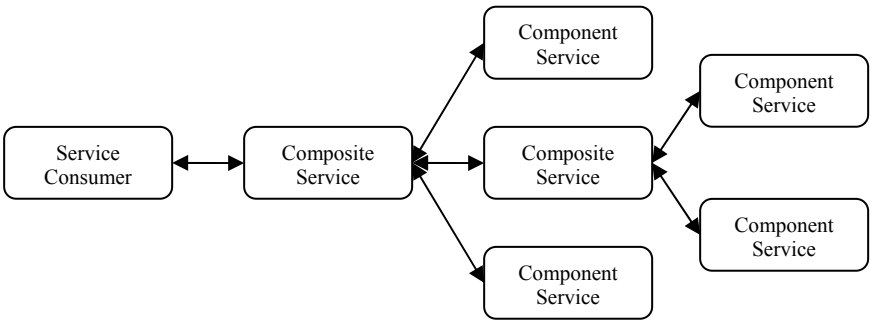


Figure 4.11: Composite Service

Composite services may be invoked synchronously or asynchronously, and may hold internal state while invoking other services. *Serial Service Orchestration* is the invocation of component services in an order, waiting for the completion of one before the next is executed. *Parallel Service Orchestration* involves the execution of services concurrently. Certain sections of a composite service may be concurrent, with a join point at which all services must be complete before moving on.

Conversational (Workflow) Services: A conversational service typically has state attached to it; a certain operation on that service will start the conversation and set some item into a specific state. Subsequent operations may continue the conversation and change the state of the item. The conversation is ended with an operation that sets the item to a final state. Operations within the service may be invoked synchronously or asynchronously and the service needs to have a mechanism for correlating individual operations.

Data Services: A data service provides a mechanism for querying a data source or multiple data sources through a message based request response mechanism. Data services can be combined together to provide a single response containing data from multiple services. The data service is responsible for routing query parameters to the correct source and combining resultant data in the correct response message format.

Publish-Subscribe Services: Publish-subscribe services are the services in which interested parties may request notification of certain events. Some entity manages a list of subscribed parties and publishes notification in the form of a message when the event takes place.

Subscription and subsequent publishing of messages could be through a Message Broker, or managed explicitly by a set of subscribe/unsubscribed messages sent to a subscription manager. Figure 4.12 illustrates Publish-Subscribe services.

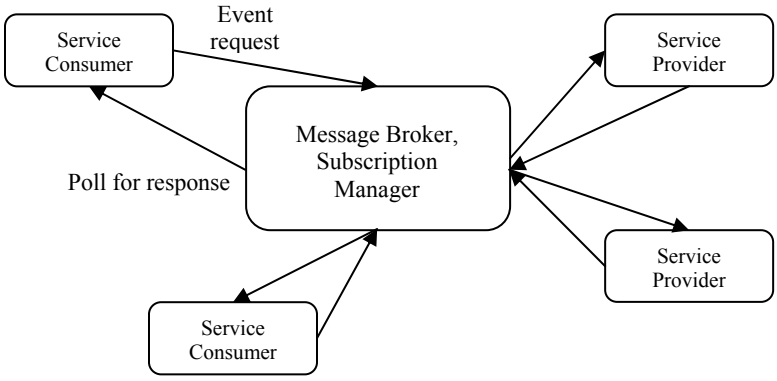


Figure 4.12: Publish-Subscribe Services

Service Brokers: Service Broker is an intermediary service that manages the invocation of a set of registered services based on a set of rules. This incorporates routing of the messages and possibly data transformation between the incoming message and the requirements of the brokered service. A broker may itself be configured to be invoked synchronously or asynchronously.

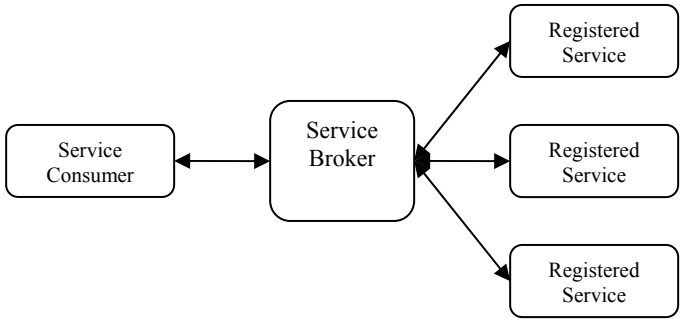


Figure 4.13: Service Broker

The broker may invoke one or many services concurrently depending on how it is configured. If many services are invoked, it may wait for all to complete or just one to complete before notifying the client, if running synchronously.

Exception Handling and Compensating Services: When a service invocation fails with an exception, there usually needs to be some way of handling this failure. A simple mechanism of handling such exceptions is to log or report them via some notification to the invoker of the service.

In a complex business transaction, some action may need to be taken if a service that was expected to succeed as part of the transaction, fails. A *compensating transaction* is a mechanism for undoing some actions that were already completed that are now inconsistent because the service failed.

Interception and Extensibility: Interception is a mechanism for inserting additional functionality into a system without modifying or affecting existing components. Functionality that intersects many aspects of a system can be inserted via interceptors, extending the capabilities of the system.

The Blueprints specification as well defines *interoperability* as a requirement of any service that may be accessed from multiple platforms. It means that the invocation mechanism, message format, data format and security requirements of a service can be interacted with successfully by any SOA implementation.

Service security is another requirement for services to ensure protection of confidential resources. A variety of techniques are available within SOA to ensure security including wire level security, such as HTTP Authentication and HTTPS, as well as message layer security that involves XML Signature and XML Encryption.

The specification does not offer the followings as SOA implementation pattern:

- Overly granular business services,
- Remote access to local services,
- Overuse of XML,
- Usage of loosely coupled services where tight coupling is required.

5. Frameworks for Service Oriented Architecture

A framework comprises tools and components specialized to provide some functionality and aims to solve both technical and non-technical based problem domains. A good framework offers versatile, extensible and easy to use environment for the development of applications. SOA based frameworks apply service oriented principles to its application design and implementations. These frameworks cover necessary infrastructure components for service creation, consistent service-based application development, and other required features.

5.1 SOA Framework Descriptions

SOA frameworks especially based on Web services is rather a new development and consideration area, which requires understanding of SOA principles clearly and able to apply them in order to have a capable framework infrastructure. At present, the industry is moving to the building of sophisticated frameworks which covers all the necessities of service based application development, rather than having tools which fulfill only some functionality of service orientation. The following part will describe the offerings and functionalities of the chosen service based frameworks.

5.1.1 SAP NetWeaver

SAP NetWeaver is an application development and integration platform [35] offered by SAP AG to support *Enterprise Services Architecture* (ESA) through enterprise service orchestration and consumption of services in composite application. Enterprise Services Architecture is SAP's blueprint for building and managing service based applications and providing enterprise solutions by utilizing service oriented architecture principles.

The fundamental feature of the Enterprise Services Architecture is to provide an abstraction of business events and activities from the actual functionality of applications [36]. These business activities are modeled as *enterprise services* and they are differing from application services in a sense that enterprise services are formed by aggregation of Web services and provide building blocks for automating enterprise scale business scenarios. *Composite applications* are developed by composition of existing enterprise services and defined as new applications to support composite level business processes.

SAP NetWeaver provides a platform for the creation, deployment and management of composite applications. It supports enterprise service oriented integration and application platform requirements and provides standard based interoperability with other platforms including .NET and IBM WebSphere.

SAP NetWeaver has tools, methodologies, rules, user interface patterns, and services that provide a unified application development and management platform for enterprises and organizations. It utilizes Internet standards including HTTP, XML, and Web services.

It provides a unified environment for modeling of business processes, model-driven implementation of them, and a repository that includes enterprise business scenarios. It allows integration of diverse SAP and non-SAP software components and development of standard software in which predefined business processes are packaged and formed as an application. mySAP Business Suite and xApps (extended Applications) are examples of such standard software. xApps defines common standard enterprise services such as supply management and merger-and acquisition integration.

SAP NetWeaver involves a variety of components, each of them has specialized focus areas and provides functionalities for a fully integrated development environment. The following figure illustrates abstraction of SAP NetWeaver components [37].

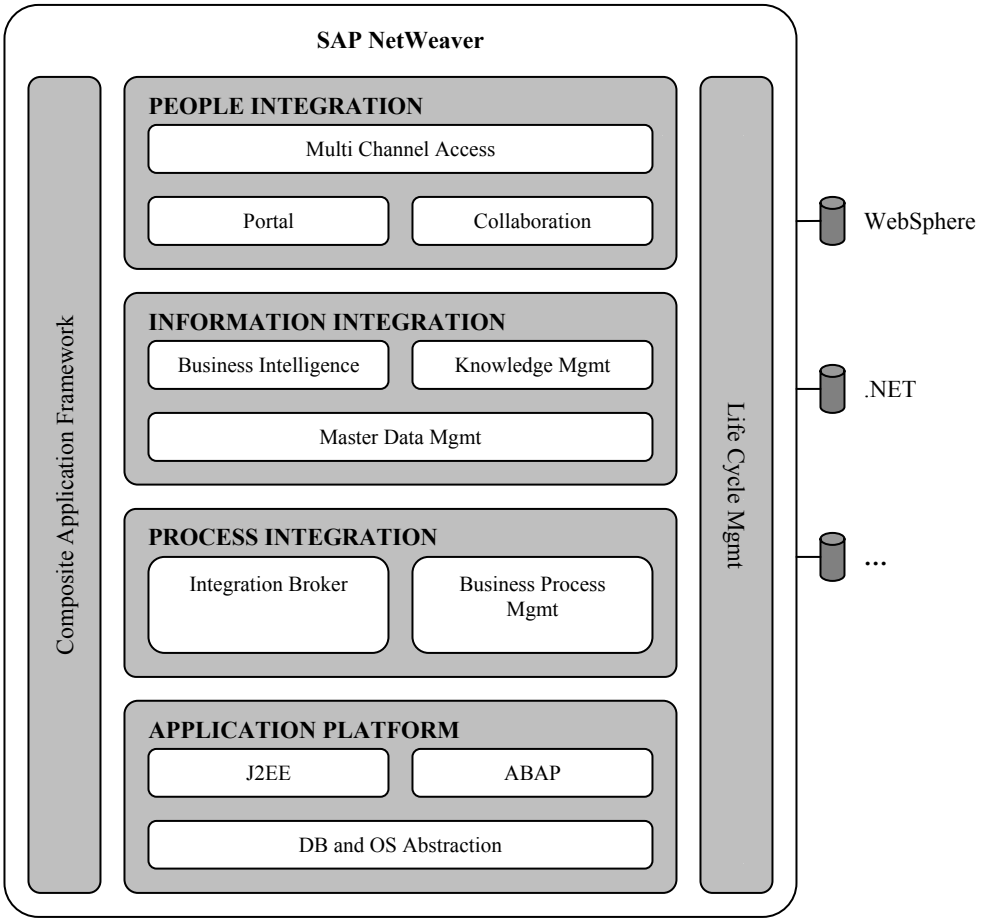


Figure 5.1: SAP NetWeaver Architecture

In general, these components provide application integration and development capabilities for enterprises. Integration components aim to supply tools necessary for application integration and specialized functionalities. Development components are software tools for creation and execution of the application. These components are described as follows:

- ***SAP Enterprise Portal (SAP EP)***

SAP Enterprise Portal (SAP EP) is a process-centric portal framework that supports role-based user interfaces and collaboration of the users with shared folders, forums and email lists. It provides knowledge and content management functionalities for the enterprise by aggregating diverse information from many places into one organized place. SAP EP is based on portlet technology, which SAP calls as *iViews*.

- ***SAP Mobile Infrastructure (SAP MI)***

SAP Mobile Infrastructure (SAP MI) enables accessing to various mobile devices through a single interface. It provides browser based front end for various mobile business applications to be able to access multiple backend connectivity of devices.

- ***SAP Business Intelligence (SAP BI)***

SAP Business Intelligence (SAP BI) provides mechanism for gathering of information from enterprises and evaluation of this information to form into meaningful and functional small units of data. Businesses realize this data through tools that create reports and queries from SAP BI repository. It creates a unified view of information from many resources. SAP BI is useful for performance determination and some other types of measurements. SAP BI is integrated with SAP EP in a way that the result data of some operation can be seen as iViews within the portal framework, or the content of an iView can be stored to SAP BI repository.

- ***SAP Master Data Management (SAP MDM)***

SAP Master Data Management (SAP MDM) provides data integrity and consistency across the enterprise. It harmonizes the information obtained from distributed environments to build real-time master data warehouses and repositories. *Master data* is defined as the data that is not belong to any business transaction, such as address information and product descriptions. SAP MDM merges the content and centrally manages the master data to have reliable information integration across business network.

- ***SAP Exchange Infrastructure (SAP XI)***

SAP Exchange Infrastructure (SAP XI) is a message oriented middleware which provides a framework for routing the flow of messages between each application, if it is required by usage of adapters, with supporting security and guaranteed delivery.

It is used to integrate processes, facilitate invocation of remote functionalities and e-business interactions. SAP XI has its own integration directory in which the process descriptions are stored centrally. As well, it provides central configuration of the system and collaboration knowledge.

- ***SAP Composite Application Framework (SAP CAF)***

SAP Composite Application Framework (SAP CAF) is a model-driven development environment for creating composite applications with utilizing *SAP NetWeaver Visual Composer*, a tool that provides user interface modeling environment for SAP CAF. SAP CAF contains a metadata repository that includes object definitions, process descriptions and other related information. It is a runtime framework with modeling and generation tools for developing composite applications which operates on existing services and applications.

- ***SAP Web Application Server (SAP Web AS)***

SAP Web Application Server (SAP Web AS) is a toolset which provides the execution environment at runtime for other components of SAP NetWeaver. It executes Java and ABAP (Advanced Business Application Programming) code, and supports Web service development with model driven user interface.

- ***SAP NetWeaver Developer Studio***

SAP NetWeaver Developer Studio is an Eclipse based integrated development environment for building programs in Java language with additional editing, building and debugging programs. It also supports creation of user interfaces for wireless and handheld devices, and developing portal interfaces.

- ***SAP Solution Manager***

SAP Solution Manager provides a framework for configuration and management of an application through its life cycle, from installation to deployment and monitoring. It is also capable of monitoring processes and upgrading the application structure.

Enterprise Services Architecture (ESA) is the SAP's blueprint based on Service Oriented Architecture (SOA). SOA supports an abstract model, in which the three main entities, including service provider, service consumer and registry, collaborate to form service oriented application structure. ESA extends the single service description to enterprise services which provides abstraction of business activities or events [38]. Enterprise services aggregates Web services to form meaningful building blocks for automation of enterprise-scale business scenarios. Composite applications are developed from enterprise services by composing functionality and information from existing systems to support new business processes.

All enterprise services are built with Web services standards and can be described in a central repository. These services are created and managed by tools provided by SAP NetWeaver. SAP NetWeaver provides comprehensive integration and application platform by utilizing its tools and components to design, build, implement, and execute enterprise business processes with ensuring open standards support and interoperability with other platforms.

SAP NetWeaver provides Web services based runtime infrastructure, allows development of user interfaces, and integration of processes and applications, and provides a common ESA service bus for communication, transaction handling, and debugging. SAP NetWeaver components are integrated with each other in a loosely coupled way and collaborates for building service oriented application structure.

The following figure illustrates the functional areas of SAP NetWeaver components within the ESA enterprise [39].

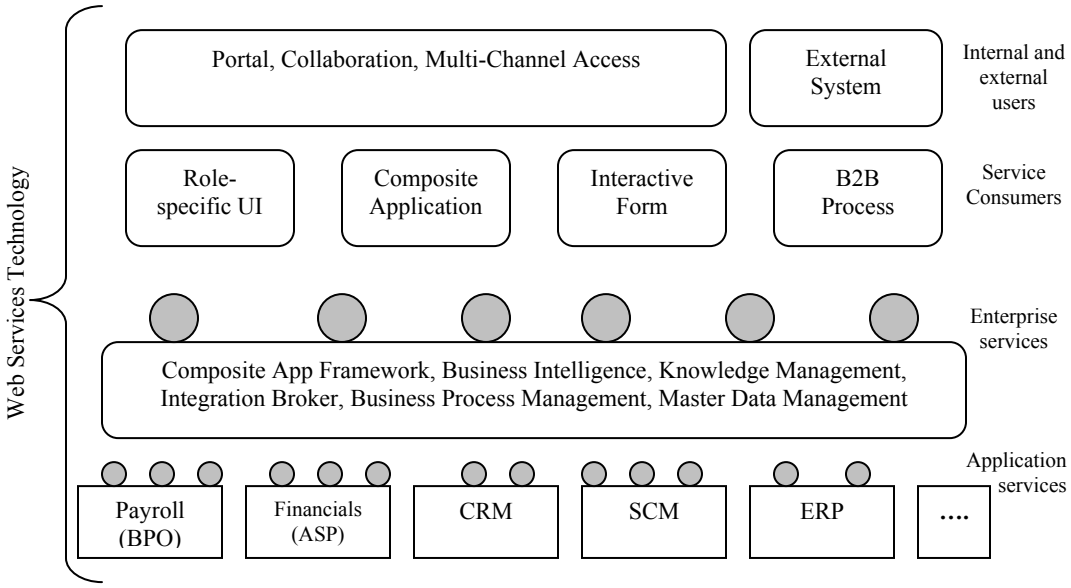


Figure 5.2: Functional Areas of SAP NetWeaver in Enterprise Services Architecture

SAP NetWeaver allows the development of service based application structures. Using only SAP NetWeaver to create services does not guarantee the creation of service oriented architectures, as it is needed to follow the principles of SOA.

An important entity of SOA is the registry, which gathers the offered services from enterprise and allows service consumers to discover them. SAP NetWeaver supports UDDI and allows creation of different service repositories, including repositories for application services available for enterprise services development, enterprise services available within the organization, and enterprise services available to the outside of the organization.

The following diagram, illustrates the fundamental service publishing and consuming concepts of SOA within the SAP NetWeaver enabled enterprise. The figure is based on the previous figure, and it expresses related specific functional areas of SAP NetWeaver.

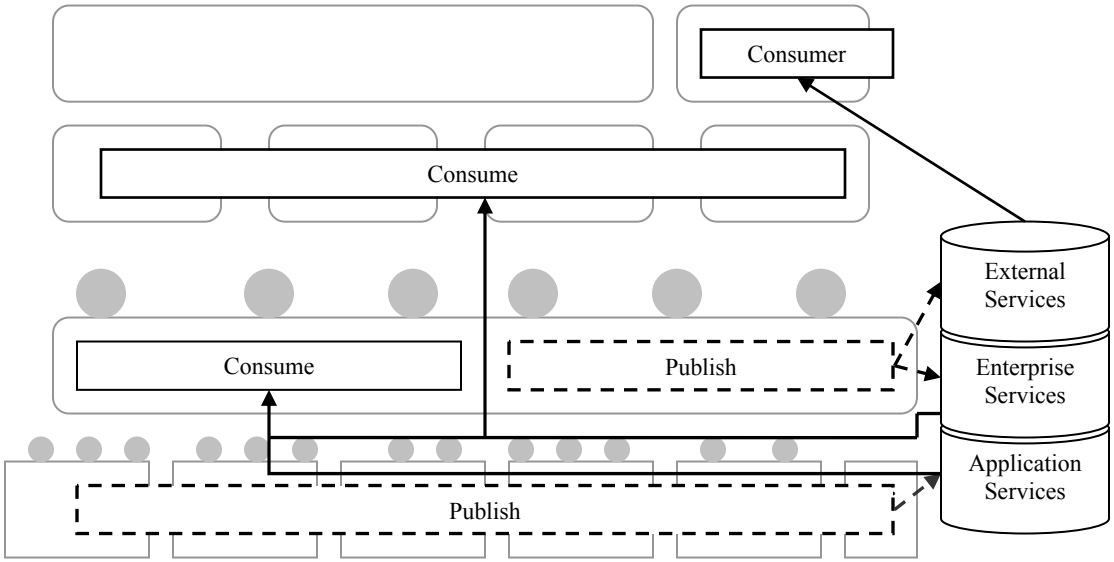


Figure 5.3: Service Interactions in SOA Based Enterprise Services Architecture

SAP NetWeaver is a comprehensive platform which enables and manages the orchestration of services, mapping, and data transformation within the enterprise between the provider and the consumer of the enterprise services.

5.1.2 Apache Beehive Project

Beehive is a lightweight, metadata driven and open source framework proposed by Apache Software Foundation for J2EE and SOA based applications [40]. It leverages the features of JDK 5.0, particularly JSR 175 metadata annotations [41] and builds on essential Apache projects including Tomcat, Struts, and Axis.

Originally evolved from BEA WebLogic Workshop product, Beehive aims to simplify enterprise Java application development and provide a model for building service oriented architectures.

Beehive offers three application building unit: *controls* framework for creating and consuming J2EE components, a simplified *metadata driven Web services* development framework, and Struts based *Java Page Flow* technology for creating Web based user interfaces and applications. These components can be built individually or all in the same application in order to have a system able to access enterprise resources and integrate various modules through supporting services.

- ***Controls***

The Control architecture is a lightweight component framework based on annotated JavaBeans, which provides client model for accessing a variety of J2EE resource types. Controls have unified access to Enterprise Java Beans (EJBs), JMS Queues and Topics, Web services, databases via JDBC, and enterprise resources via JCA.

The framework provides functions such as Java Bean based client access, configuration through JSR-175 metadata and external configuration data, automatic resource management, context services, and an extensible authoring model for creating new Control types. The goal of Controls is to enable collaboration where the base J2EE distributed system architecture and other related components can be designed and built, then assembled into exposed Web user interfaces, Web services, or individual applications.

The definition of a new resource type in the Control architecture is composed of three distinct classes:

- **Control Public Interface:** it is source file that defines the set of operations, events, extensibility model, and properties associated with the Control type.
- **Control Implementation Class:** it is source file that provides the implementation of the operations and extensibility model described by the Control Public Interface.
- **Control Bean Generated Class:** it is a code-generated Java Bean class that is derived from the Control Public Interface and the Control Implementation Class by a Control compiler.

These three classes work together to fulfill the runtime responsibilities. The client interacts with the Control by invoking operations defined on the Control Public Interface. Control Bean Generated Class is automatically created by the compiler and manages accessing remote resource.

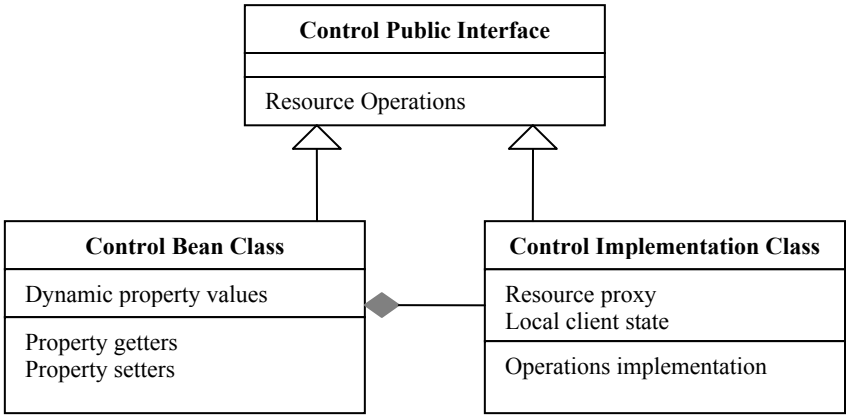


Figure 5.4: Relationships of Classes in Controls Architecture

The following diagram illustrates Control architectural elements and the process flow.

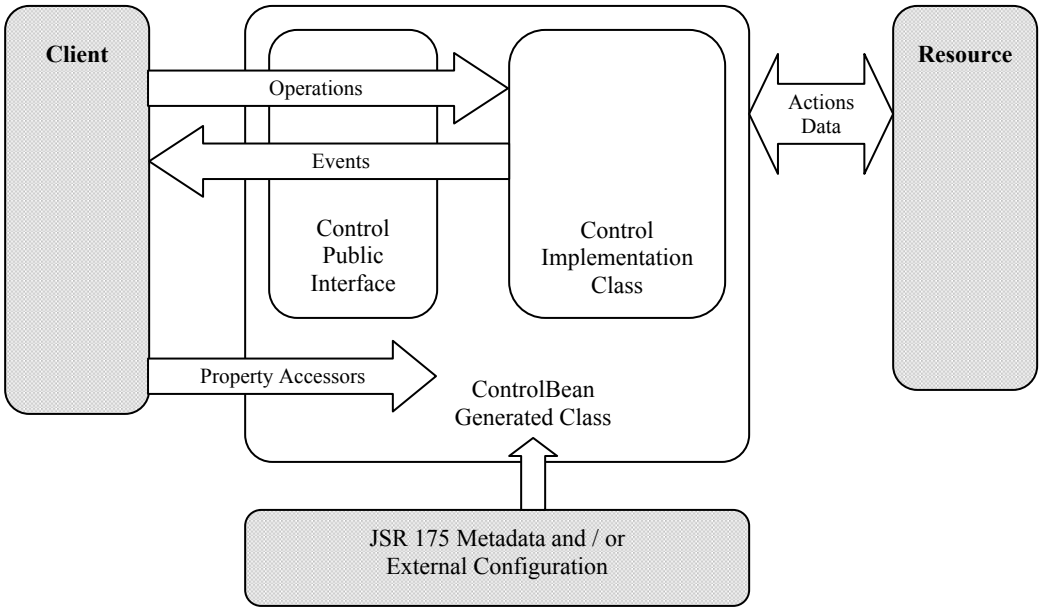


Figure 5.5: Control Architectural Elements and Flow

The Controls architecture supports a composition model, based on *JavaBeans Runtime Containment and Services Protocol*. This means that it is possible for new types of Control Beans to be defined that are built through composition of one or more other types. In this model, JavaBeans are associated with a Bean-Context that manages the composition hierarchy and also manages any contextual services requested by the contained beans.

The Controls architecture offers two client models with slightly different characteristics: in the *programming model* the client takes explicit responsibility for construction of Control instances and event routing, whereas in *declarative model*, the Control container provides initialization and routing services on behalf of the client by the help of JSR-175 metadata annotations.

The Controls architecture provides a packaging model to distribute Controls that offer client access to provided services and components. An *Integrated Development Environment (IDE)* can discover packaged controls to present them in a list of available resource types for client use.

- ***Java Page Flow***

Java Page Flow (JPF) is a web application framework based on *Apache Struts* [42] with a group of Java Server Pages (JSP) to present the visual interface to users and a Java class (Controller) that drives and coordinates the application.

Java Page Flow is based on JSR-175 metadata and supports features such as state management and integration with Controls, Java Server Faces, and other modules within the application. It has the following characteristics:

- Page Flows are state-full
- Page Flows are modular
- Page Flows are nest-able
- Supports rich data binding and integration possibilities
- Action and Exception Handling

Java Page Flow is built on top of Apache Struts, and can cooperate and interact with it inside a Web application. Page flows leverages the features of Struts, on the other hand overcome the difficulties faced with Struts based application development. Page flows are as well based on ***Model-View-Controller*** (MVC) design pattern that separates application functionality into three distinct roles: *Model* represents the business data and the rules associated with accessing and modifying the data, *View* is the user interface, and *Controller* manages and controls the user interaction with model and view layers within the application.

The interception point of the user with the Web application is defined by *action* methods of the particular controller class. An action can be navigation of the user to a specified JSP page, performing conditional logic, validating data and handling exceptions that arise in the application. The business logic in the controller class is separated from the presentation through JSP files, which provides an easy-to-use framework for building dynamic Web applications.

- ***Metadata Driven Java Web Services***

Beehive Web services are the implementation of the JSR-181 specification [43], which simply the Web service development process and offers implementations of the most common Web service features, including conforming to basic SOAP and WSDL standards, separating the public contract and the private implementation, and creating asynchronous communication between the Web service and its clients.

The fundamental aspect of Beehive Web service development is that the functionality of an ordinary Java class is exposed as a Web service by specifying metadata annotations that decorate the class and its methods. The metadata annotations replace the deployment descriptors and encapsulate the Web service APIs.

The specification does not specify some advanced concerns such as security, internationalization and localization in current specification. Beehive is in Beta 1.0 version and still evolving. An additional development is the *Pollinate project* (http://www.eclipse.org/pollinate/) which enables usage of Eclipse IDE plug-ins to easily build applications based on the Apache Beehive application framework.

5.1.3 Rogue Wave Lightweight Enterprise Integration Framework

The Lightweight Enterprise Integration Framework (LEIF) is a lightweight, cross-platform framework offered by Rogue Wave Software for integrating C++ applications with other applications built on variety of technologies across the enterprise [44]. It is based on standard technologies including XML and Web services and enables process-based communication across platforms such as .NET and J2EE.

LEIF framework consists of three tiers that deal with different aspects of system integration [45]:

- The **Data Tier** provides for data handling through C++ interface. A code generator produces C++ classes for a particular XML document type as defined by an XML schema, allowing to process XML documents through C++.

- The **Network Tier** provides for transmitting and consuming network data. A code generator produces Web service clients able to send requests to any well-defined and locatable Web service on the network.

- The **Service Tier** provides for development of network services, including but not limited to Web services. A code generator produces service code that hides the details of XML, SOAP, and network protocols, allowing developers to concentrate on business logic. A C++ servlet container hosts the completed services.

The architecture of LEIF is based on these three layers which facilitate integration of legacy code as services and allowing SOA applications. Currently it is in version 2.1, LEIF is able to dynamically transport messages, and implement WSDL message patterns entirely. It allows asynchronous and synchronous message interaction with improved XML Schema and WSDL support. The following figure illustrates LEIF tiers.

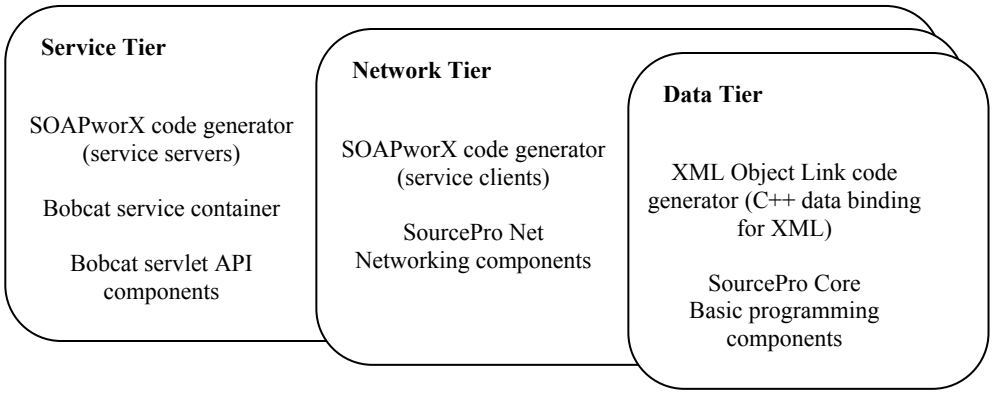


Figure 5.6: LEIF Tiers

LEIF provides browser based user interface to its code generators from a single point of access. Through using this interface, LEIF provides creation of different types projects, including Data tier project, which generates a C++ data binding from an XML Schema, Network tier project, which generates client code for a Web service from a WSDL file, Service tier project, which generates server code for a Web service from a WSDL file, and finally C++ servlet framework for generation of a framework for a servlet that can run in the Service Tier container.

5.2 Evaluation of Frameworks

Capabilities of the framework are important for the development of a sophisticated service oriented application, as naturally the application may expand its structure through accessing various resources in enterprise and handling different data formats, platforms and proprietary software modules. The framework should support visual, declarative and service driven programming model and allow orchestration of services and development of composite applications.

SOA based frameworks may include tools for service creation, modeling of services and user interfaces, integration of processes and connectivity to databases, legacy and other type of applications. These tools provide runtime for building and maintaining the services within the SOA across disparate and heterogeneous environments. Process information is essential in such environment to achieve reusing of service logic in an effective manner and formation of dynamic execution atmosphere.

SAP NetWeaver is a sophisticated and well-developed platform which covers all the necessities needed for the development of a service based application. The main drawback is that it is vendor specific and may oblige the developer to use SAP specific proprietary modules within the framework. Although the components offered within SAP NetWeaver are essential and required for building an enterprise level SOA application, for a small size project, all these components may not be required.

Apache Beehive project is still evolving and in current version it may not be applicable for the development of standalone applications. It provides open source and standards based development environment, which gives flexibility to the developer. Beehive focuses on core requirements of J2EE platform and enhances the way of application development through offered components. Ease of use feature of Beehive is essential for building service based enterprise applications.

The key characteristic of Rogue Wave LEIF framework is the usage of C++ programming language for service development. LEIF provides an environment for integration of C++ applications through utilizing service logic. The weakness of the framework is that it does not support direct mechanism for creation of processes and orchestration of services. While the user interface is not a required component for SOA, still it is a capability of a framework to allow remote client access for an application. LEIF does not provide this feature as well.

The capabilities of the frameworks are summarized in the following table.

	SAP NetWeaver	Apache Beehive	Rogue Wave LEIF
Framework Ownership	Vendor Specific	Open Source	Vendor Specific
Service Development Support	Supports Java and ABAP based Web service development	A specialized Java Web service development tool built upon Apache Axis project	A component that creates C++ service from its WSDL document – limited support for service creation (need WSDL to build the service)
Language Interfaces for Service Invocation	Java and ABAP Languages	Java Language	C++ Language
Metadata and Data Handling Support	Supports metadata and provides advance level data handling features	Supports standard based metadata management through usage of JSR-175 Specification	Fully XML and XML Schema based data handling – no direct mechanism for metadata definitions
User Interface / Portal Support	Supports through SAP Enterprise Portal (SAP EP)	Supports through Java Page Flow (JPF) Framework	No Web based User Interface support
Process and Application Integration Support	Specialized frameworks and tools for process creation, management and application integration support	Controls are specialized components used for accessing resources and applications – process information can be created by utilizing Controls and JPF.	No direct support for modeling process information; supports integration of applications through created services.
Modeling Approaches and Supporting Tools	Specialized tools for user interface and process flow modeling	No specialized tool support, but it is possible to derive meta-models for user interface and process flow	No specialized tool support for definition of process flow
Specialization	A platform for development and management of model-driven service oriented applications	A framework for development and management of model-driven service oriented applications	A framework specialized for integrating diverse application and business logics through services.

Table 5.1: Comparison of SOA Frameworks

6. Case Study: A Prototypical SOA Implementation

The aim of this case study is to provide a prototypical SOA implementation in which the fundamental requirements of service based application development are investigated and proved. Therefore, a sample Student Information System (SIS) has been developed to illustrate essential characteristics of SOA including utilization of service based infrastructure, interoperability of different platforms and software modules, service based integration, and reusability of loosely coupled service business logic.

The case study is prototypical, which means that it is satisfactory practical and demonstrates the core aspects of SOA based application. The application utilize Web service technologies in order to allow communication and interaction of two diverse and independently located applications, and build an architecture that can tolerate changing of the infrastructure, consent to flexible extensibility of the application by adding other business logics from enterprise, and provide and consume the offered services to form a service based interaction of applications.

Student Information System (SIS) is based on an example scenario which envisages that the system provides students an online access to resources related to their educational activities and targeted information needed to carry out certain educational transactions. Technically speaking, SIS is an independent and standalone application which assembles the required services for a student from enterprise and allows students to consume them to fulfill their educational requirements.

A typical Student Information System may contain activities and functions such as address and profile updating, student unofficial grade summary, adding and dropping courses and course schedules. SIS is designed and implemented to support the following functions: sign-in to system, choosing the program and department that offer services for students, and registration-dropping of courses that a particular study department provides.

SIS is the system which allows end-users to consume gathered services. To build a SOA, it is needed to have another computing system, or module, which provides services for students to consume them. For this aim, the *infoAsset Broker System* is chosen, which is a system platform for the construction of personalized content portals in Internet as well as corporate knowledge portals in Intranet [46]. It is developed by close cooperation between infoAsset AG (http://www.infoasset.de/) and Software System Institute (STS) research department (http://www.sts.tu-harburg.de/) of Technical University of Hamburg-Harburg (TUHH). In this case study, a Web service is developed which interacts with the Broker system to obtain the required information and passes this information to SIS by applying service oriented architecture principles.

6.1 Description of Implementation Infrastructure

SIS is implemented by using Apache Beehive open source project, which offers a framework for construction of service based applications. The Web service application is as well developed using Metadata-driven Web Service component of Beehive framework. Within the implementation infrastructure, these three applications, which are Broker, SIS and Broker Web service, collaborate together to achieve formation of a single application logic. The end user only face with the SIS graphical user interfaces without having knowledge about Broker and Broker Web service implementation.

The following figure illustrates the implementation infrastructure:

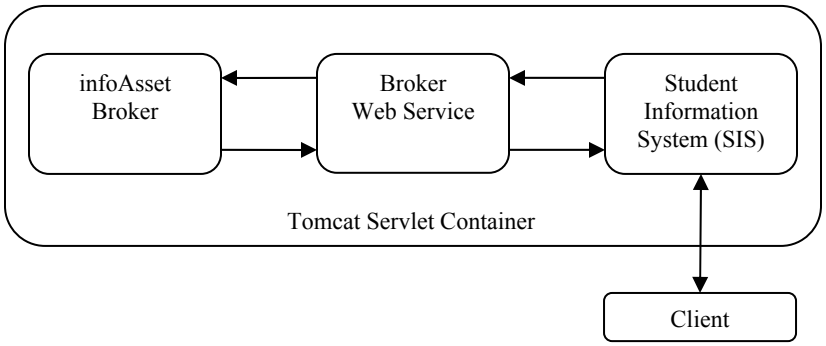


Figure 6.1: Case Study Implementation Infrastructure

6.1.1 InfoAsset Broker Architecture

The infoAsset Broker is Java based and platform independent server software which communicates with client through Web browsers, PDAs, or WAP based interfaces, by utilizing HTTP protocol. Core Broker is supported with File systems, data bases and content management information repositories. Database access is achieved through Java Data Base Connectivity (JDBC) and corresponding APIs.

The Broker composed of four main components, and each of them is separated through clearly defined interfaces. These components are described as follows:

- **Web Server component:** it is an integrated Web server to the core Broker and handles the communication between client and the system. It routes the requests to the corresponding handler and returns back the results.

- **Handler component:** it handles the incoming client requests and routes these requests to specified Broker services. Handler component works together with the Template component to generate client responses. A handler can be in two types: *visible handler* is for creating a visible output within a template, and *invisible handler* which changes the state of some object or figures out some data.

- **Template component:** generates the user interface for the client in the requested output format, which can be HTML, WML, or other type. Templates support multiple languages.

- **Services component:** it is one of the main component of Broker. The Services object instance exist only once within the Broker and provides access for business objects, called as *Asset* [47]. There are many kind of asset can be defined, such as person, document, and group. *An asset container* is responsible for the life cycle of specified asset kind and called as the plural form of asset type, such as persons, documents and groups.

Figure 6.2 demonstrates the components of Broker within the Broker client-server architecture [46].

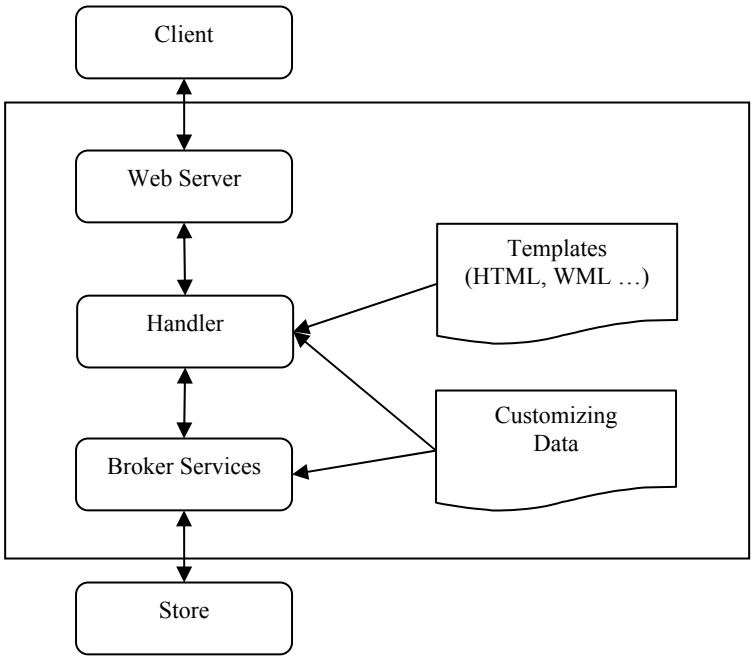


Figure 6.2: The infoAsset Broker Architecture with its Components

Within the Broker, there are already defined asset kinds including Course, CalendarItem and Program, which are required business objects for the case study. Accessing these objects within Broker is done through *Services object*, a singleton object for the whole core Broker system. Services object can be accessible from handler, asset or an asset container. In order to access Services object from outside of the Broker system, a new mechanism is required. The difficulty for the integration of Broker system with Web service implementation is that Broker has its own integrated Web server and as these applications do not have a common runtime environment, they can not communicate and allow object passing with each other's system infrastructure.

6.1.2 Integration of InfoAsset Broker with Apache Tomcat

The Web service provides the functionalities of Broker to the enterprise and allows consuming of Broker business logic by other services and applications to accomplish the desired functionalities. The Web service implementation is Beehive based standalone application running on Apache Tomcat Servlet Container [48]. In order to be able to access Services object of Broker system from Web service implementation, both of these systems are required to be executed in common runtime environment.

Integration of Broker system with Tomcat Web server has been achieved in Lars Diestelhorst's Master Thesis project [49]. For this aim, two source files have been created, called as *BrokerContextListener.java* and *BrokerServlet.java*. The idea of integration is that Tomcat notifies the BrokerContextListener at startup, which then initializes the actual Broker object and registers the created instance of Broker at the current context. When the BrokerServlet needs to handle a request from client, the Broker instance is retrieved from the current context. The following sequence diagram illustrates this process.

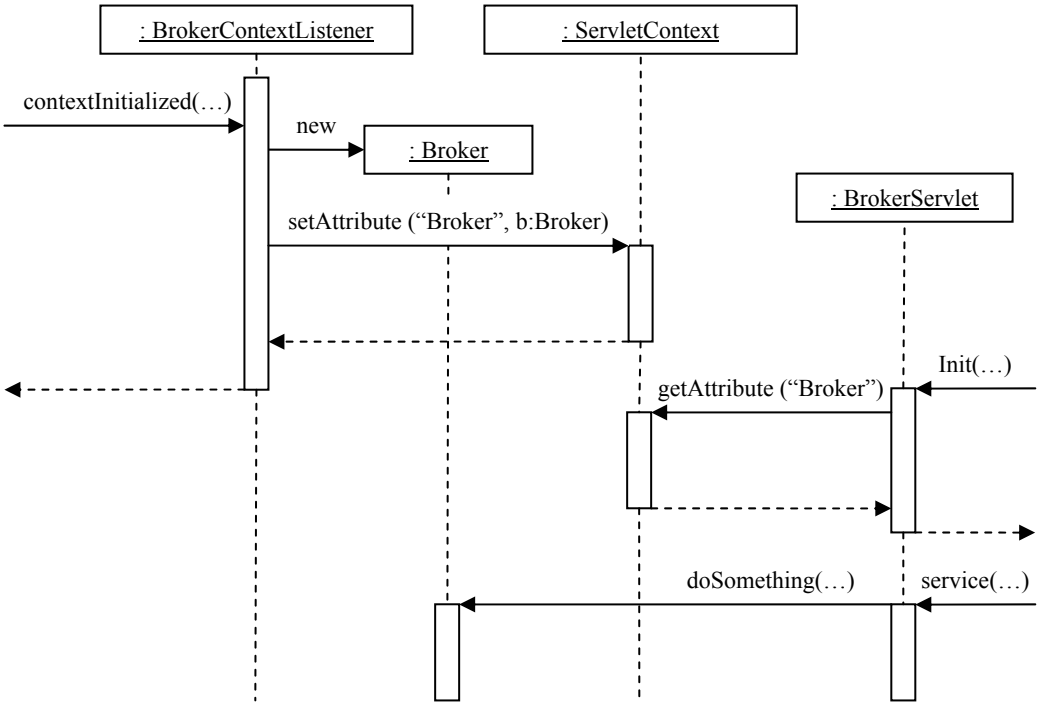


Figure 6.3: Sequence Diagram for Broker – Tomcat Integration

Each Web application running under Tomcat Servlet container has the ability to share the same context and communicate with each other through object passing. In order to access Services object of Broker, the object is defined as a global resource in *JNDI (Java Naming and Directory Interface)* Context, and other required configuration is done to reference the object from other Web applications running under Tomcat.

6.1.3 Broker Web Service Development

Broker Web service application utilizes the *Learning Interfaces* of Broker system. These interfaces are located in the package *de.infoasset.broker.interfaces.learning* and propose many useful asset kinds including course, calendarItem and studyProgram. The detailed explanations of these interfaces can be accessible from Leif M. Koch's Master Thesis [50].

Broker Web service is Beehive based Web service implementation which access the Services object of Broker and utilize the Learning and other required interfaces. The Web service implementation infrastructure comprises of one source file which executes the service logic with the file extension .jws (Java Web Service), and other utility files. The configuration of Web service environment is done with JSR-181 Java Metadata Annotations, as shown in the following source code snapshot.

```
package de.infoasset.broker.webservices;
……
@WebService
@SOAPBinding(style=SOAPBinding.Style.RPC, use=SOAPBinding.Use.ENCODED)
public class BrokerWebService {
        @WebMethod
        public boolean login(String user, String password) {
        boolean loggedIn = false;
        Person p = AccessServices.getServices().getPersons().getPersonByLogin(user);
        if (password.equals(p.getPassword())){
                loggedIn = true;
        }
        return loggedIn ;
        }
……..
}
```

Figure 6.4: Sample Broker Web Service Code with Metadata Annotations

The following class diagram represents the operations of Broker Web service.

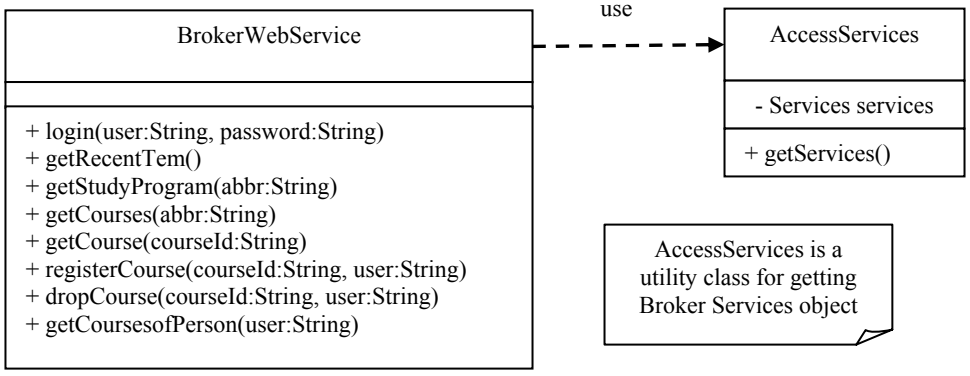


Figure 6.5: Class Diagram for Broker Web Service Operations

The following is the simplified sequence diagram which illustrates SIS, Broker Web service and Broker interaction for the Login process.

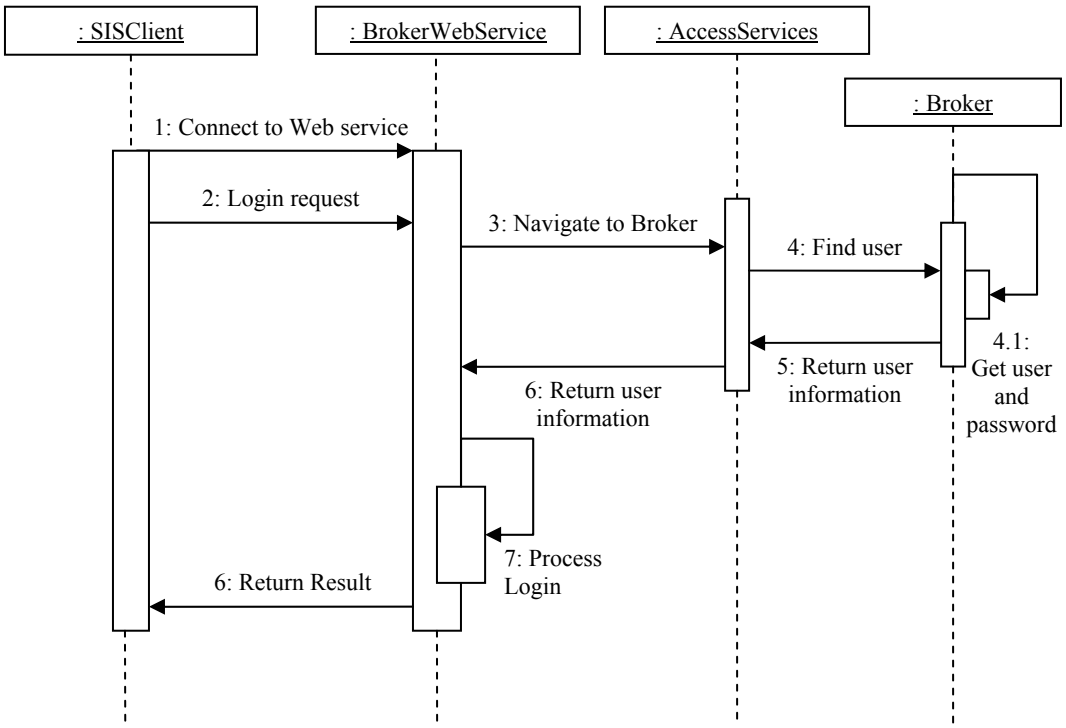


Figure 6.6: Sequence Diagram for Login Process

6.1.4 Student Information System Web Application

Student Information System (SIS) is the Web application which provides user interface to allow the students to login to the system and register or drop the offered courses for the current term. SIS is developed using Beehive Java Page Flow (JPF) technology. The application includes JSP pages to present the graphical interface to the user, a Controller which navigates and manages the JSP pages, and JavaBean files required for the interaction with the Broker Web service.

SIS Web application is based on Model-View-Controller Java design pattern, in which the JSP pages acts as the View, the Controller processes the user's request and interacts with the Model, which is the Broker.

The overall architecture of Case Study qualifies the fundamental requirements of Service Oriented Architecture. The Broker provides its business logic to the outside through Broker Web service, and SIS consumes the Broker Web service to gain the functionality of Broker. The following figure illustrates the simplified SIS application structure and its relationships with Broker and Broker Web service.

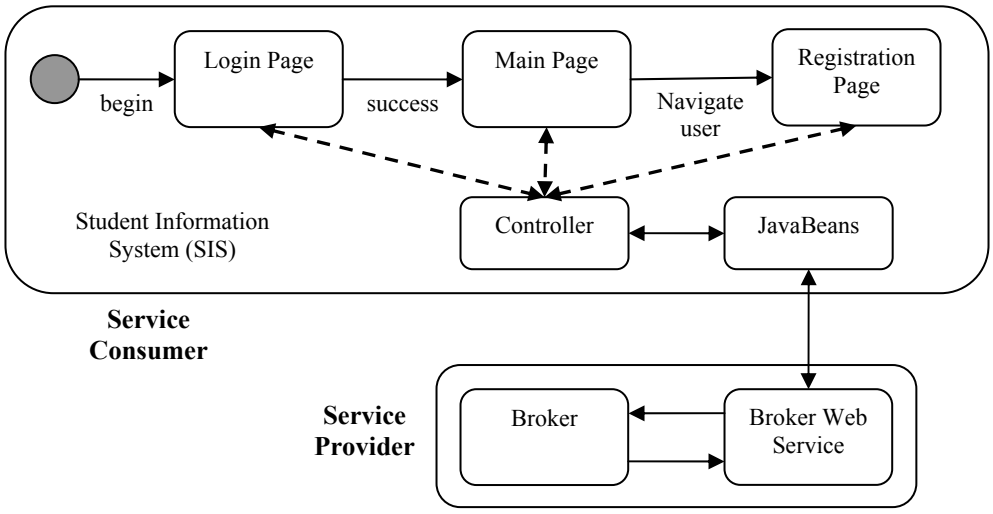


Figure 6.7: Simplified SIS Application Structure within Service Oriented Architecture

The following screenshots aim to introduce and illustrate the SIS and Broker applications that discussed within case study.

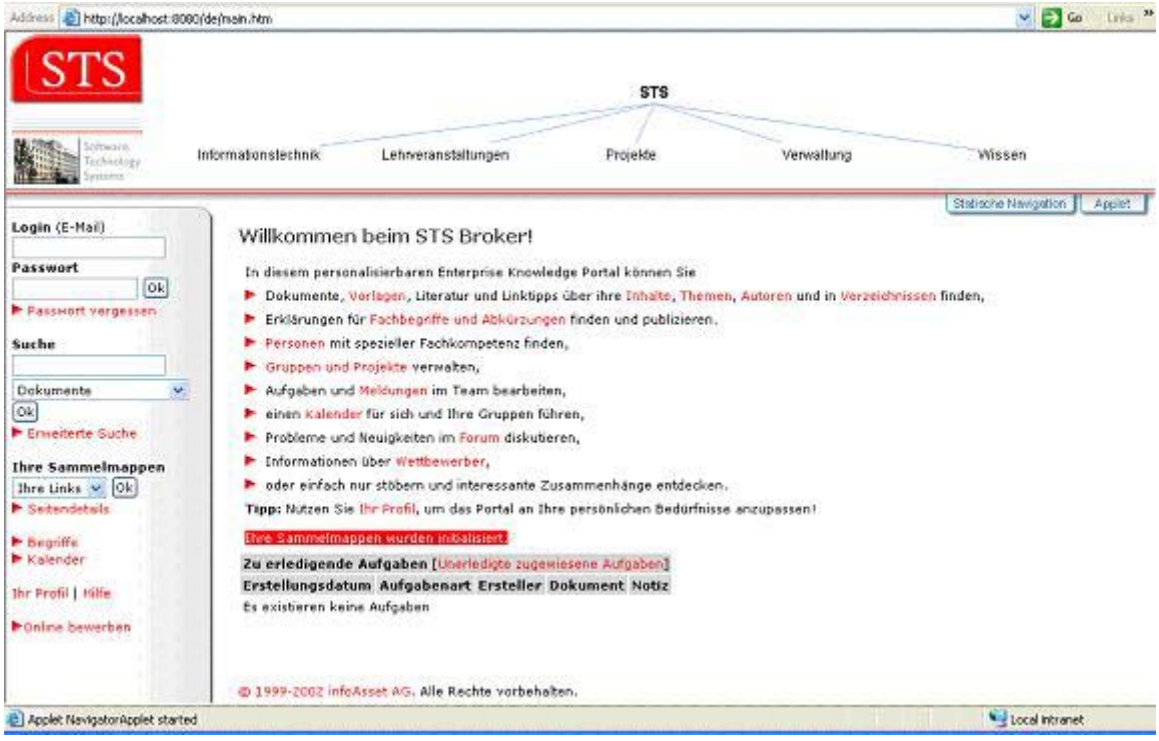


Figure 6.8: Screenshot for infoAsset Broker Main Page

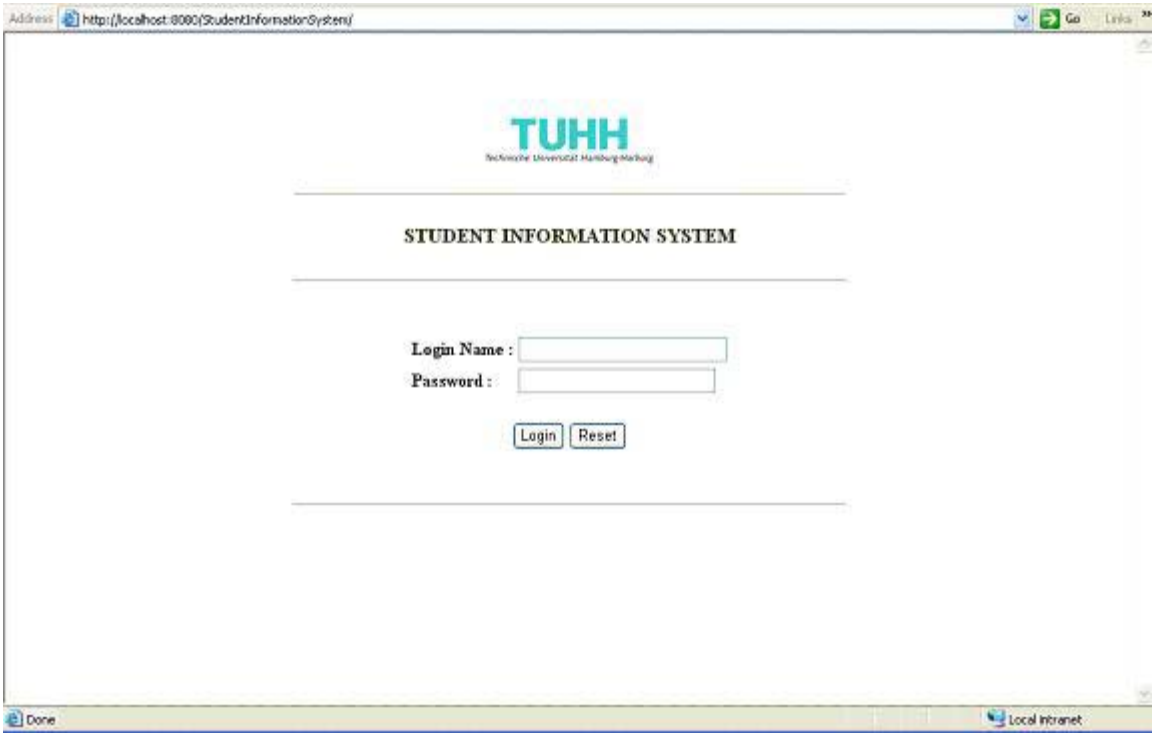


Figure 6.9: Screenshot for Student Information System Login Page

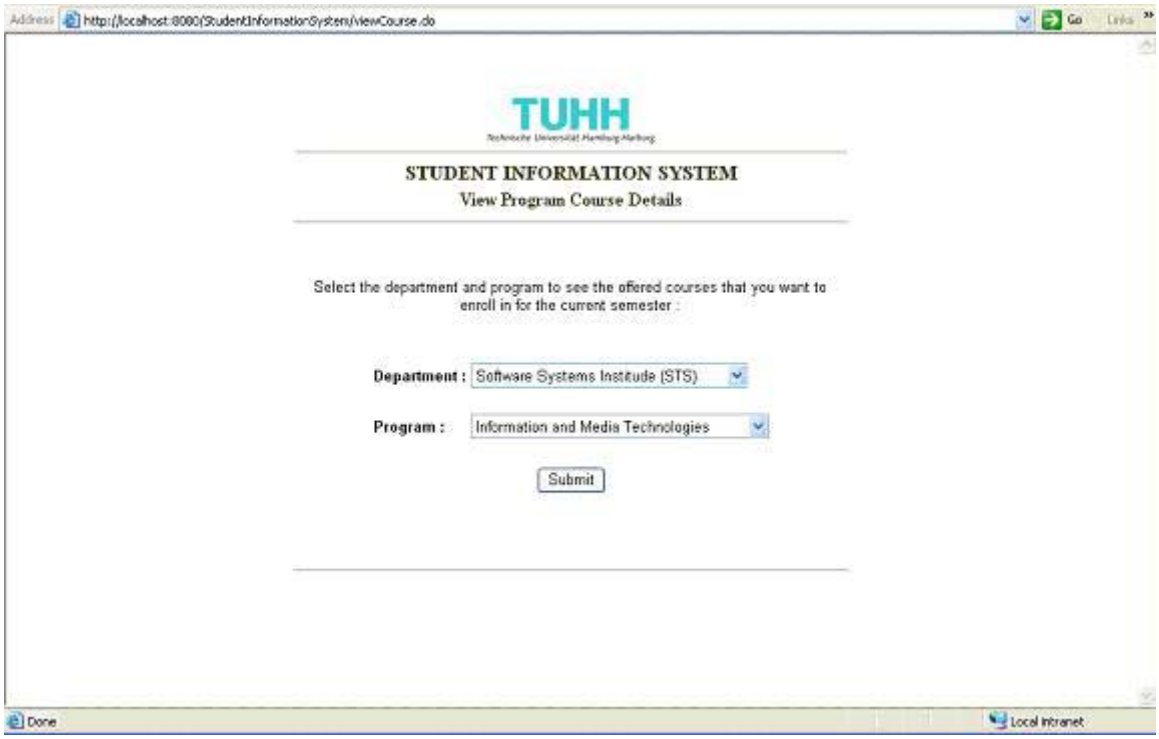


Figure 6.10: Screenshot for Student Information System Main Page

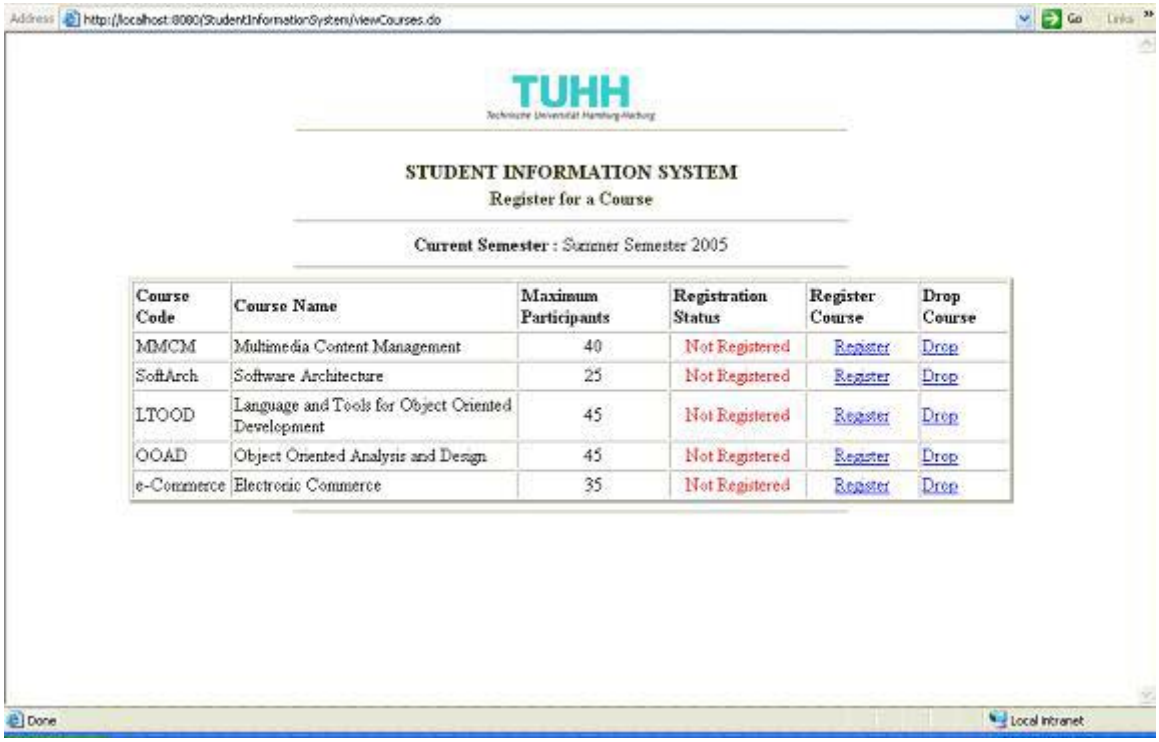


Figure 6.11: Screenshot for Student Information System Registration Page

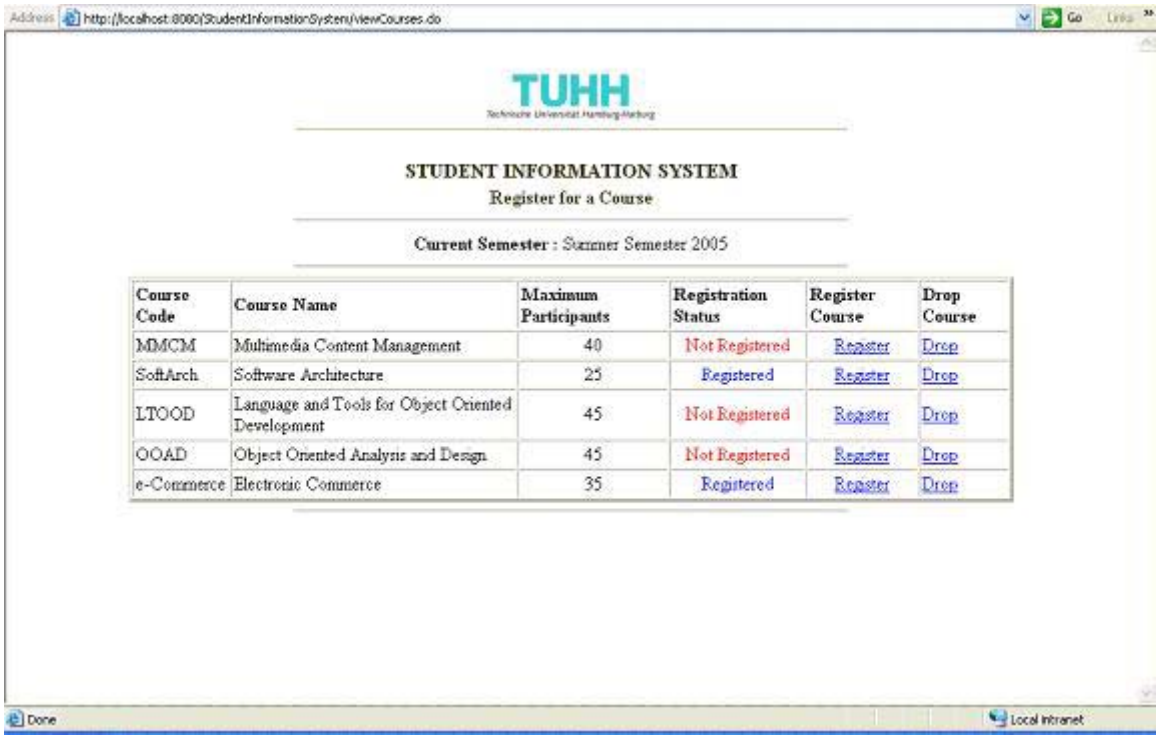


Figure 6.12: Screenshot for Student Information System Registration Processing

6.2 Extensibility of the Architecture

The architecture built for this case study is based on Service Oriented Architecture, and it is likely to extend it by integrating other application's business logic and Web services. For this aim, the availability of Amazon Web services [51] are analyzed and investigated.

Amazon does not utilize SOAP style Web service; rather it is based on *Representational State Transfer (REST)* style interface. REST is based on XML query protocol and operates on HTTP request-response model.

The main difference between REST and SOAP style Web services is that REST Web services are *resource oriented*, in which the resources can be identified and located by a Universal Resource Identifier (URI), and the operations are defined by the HTTP specification [52], on the other hand, *activity oriented* services focus on actions rather than on the resources upon which they act.

7. Strategies and Concepts for Service Orientation in Enterprise

Service orientation in enterprise is an approach to build an application structure in which the services that an organization provides to its clients, customers, communicating partners and other organizations are the fundamental assets for the needs of business. SOA implementation brings to enterprises competitive advantages in a sense that services can easily react to the changing of business requirements and allows structuring an agile and responsive system.

SOA implementation surrounds different approaches and strategies that an organization may follow when building an enterprise application. Especially SOA with Web services are nowadays way of implementation environment which replaces the traditional application development and integration approaches.

The elements of SOA implementation environment involves service oriented modeling, service oriented integration, business process management systems, and service development and management, as shown in following figure.

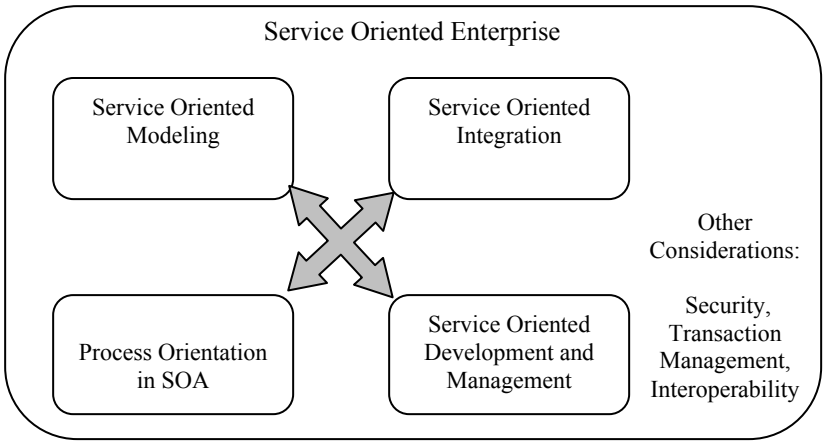


Figure 7.1: Service Oriented Enterprise

The architecture of software should provide the required infrastructure for integrating different parts of the enterprise to form an application without human involvement. SOA provides this feature with the usage of reusable services. However, the architecture can be structured in a variety of ways. In the following part, different approaches to enterprise integration will be discussed by means of mainly Web services as a SOA implementation technology.

7.1 Service Oriented Architectures for Enterprise Integration

Integration of business systems running on different locations is the most challenging aspect of application development in an enterprise. As the requirements of businesses grows to support e-commerce, Supply Chain Management, Customer Relationship Management (CRM), usage of sophisticated Internet technologies, and other necessities of the organizations, service based architectures to application integration and exchanging information within the enterprise are gaining momentum.

Application integration requires understanding of the business domains and necessities of the enterprise to provide a long term, enabled solution and build up architecture applicable for different computing environments. To develop such architecture which involves real-time combination of several applications and functionalities is a complex process. Traditional approaches to application integration such as direct combination of applications and databases are not sufficient to prove this aim. SOA solves this problem by providing services that utilizes applications broken down to component level and leverages combination of several approaches and technologies.

Nowadays approach to application integration is moving towards from information-oriented to service-based integration. *Information-oriented* approach involves direct integration of databases and information-producing, proprietary APIs, without any need to change the application structure [53]. It deals with the simple exchange of information between two or more systems. *Service-based* integration combines the applications at the service level including its structure, transactions and distributed objects by leveraging Internet technologies to form composite applications, processes, and flow of information from many resources in enterprise.

Service based integration allows well-defined interfaces exposed as services and available through Internet for an application. It may also require necessary changing in both target and source application structures, which makes service based integration exclusive and more sophisticated than traditional approaches. Nevertheless, this approach offers high-quality opportunities when implementing solutions to integration problems. It broadens the application domain from an individual department to the enterprise-wide resources to structure of an enabled architecture satisfying business needs.

With services, it is achievable to have an environment in which the functionalities of the businesses are executed autonomously, accessing information is straightforward and rapid, and utilizing of enterprise resources without restriction of its physical locations is undemanding. Although this intention is not fully applicable yet to current computing system implementations, as the technology cultivates and the methodologies become more mature, application integration based on service orientation allows this target realizable in the future. Apart from its technologic aspects of the application integration, the business with SOA has the ability to do a real-time business and satisfy the customer needs and expectations.

Application integration is primarily categorized into *Business-to-Business Application Integration* (B2B), *Enterprise Application Integration* (EAI) and *Business Process Management (BPM)* technologies. All these approaches provide techniques to deal with application semantics, accurate information routing between applications, and processing of this information to clarify integration behavior.

7.1.1 E-business Integration

Business-to-Business (B2B) Integration is the combination of two or more application logics to share the information flow and application functionalities in an automated manner between different communicating partners. The partners individually have defined rules and regulations for distribution of the services and application logics in enterprise.

E-business integration as well shows variety in the level of application integration which means that the integration involves business rules integration and sharing a collaboration middleware, and takes place at information and process level [54]:

- *Information integration* approach provides a platform for exchanging relevant business data and documents to support business proposals. It is the initial step in application integration, and message brokers, data replication and migration engines utilizes this technique.

- *Business rules integration* approach allows developers to expose existing rules or methods to other applications that may need them to support a virtual e-business system. As an example, a CORBA implementation between parties provides a standard mechanism to share application services.

- *Process integration* provides a set of processes that function above both business rules and information integration. It offers a model that forms a layer and resides on top of middleware and allows both logical and physical information flows over existing business systems.

- *Collaboration middleware* allows collaboration of geographically dispersed workgroup with the opportunity to share messages and other information in real time to support a business need. As an example, customer relationship management, online customer service, and virtual product development support this kind of application development. Collaboration uses a centralized set of middleware to manage the movement of information, and shares same of common characteristics with process automation.

Traditional e-business integration mostly involves utilizing an API that is strongly-typed and not flexible to support the changing of the businesses. The requirement for dynamic e-business integration involves having a computing structure which allows integration of partner systems through enabled interfaces.

The interfaces that support B2B integration should be aware of the processes and data flow within the organization. The simple form of integration is to support data exchange and extracting information from one of the application databases, processing it and updating the information in another one. As the needs for compound application structure become comprehensible for an enterprise, service-based integration provides mechanisms to share common business logic and create composite applications by leveraging services gathered from many remote and heterogeneous systems. The tools and techniques of B2B integration give the business partners the opportunity to create an infrastructure in which the services can be created, tested and deployed. If it is required, this infrastructure should allow the enterprise applications be changed in order to support business needs.

The initial technological development to B2B integration is ***Electronic Data Interchange*** (EDI) [1] which was established to enable the data exchange electronically over the network. It defines the format of the messages and the way of exchanging data between involving partners to communicate. EDI does not define network details and protocols for the enterprise. The organizations using EDI have to build their own private networks and allow other parties to connect into them. This approach was quite common in the earlier period; however, it was expensive to implement EDI and resulted in proprietary application structures and closed systems.

EDI is used mainly by large organizations and before the invention of Internet it was seen as the only option for e-business and communication between two geographically or physically distributed points. The following figure illustrates the EDI approach.

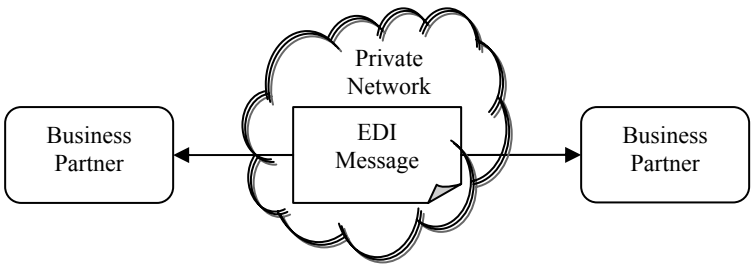


Figure 7.2: EDI in Enterprise

Before the development of Web services, *CORBA*, *COM/DCOM* and *RMI* are the common approaches for B2B transactions subsequent to EDI. These technologies provide better solutions for enterprise communication; however, they have as well difficulties which make them not a widely-accepted solution for enterprise integration.

The advantages that these technologies bring to enterprise are the usage of Internet, practicing the common agreed network protocol, approved service interfaces and progressing of communication infrastructures from information-oriented to service-based integration approaches. The main disadvantage is they are not flexible for usage of firewalls, and do not provide enhanced interoperability options.

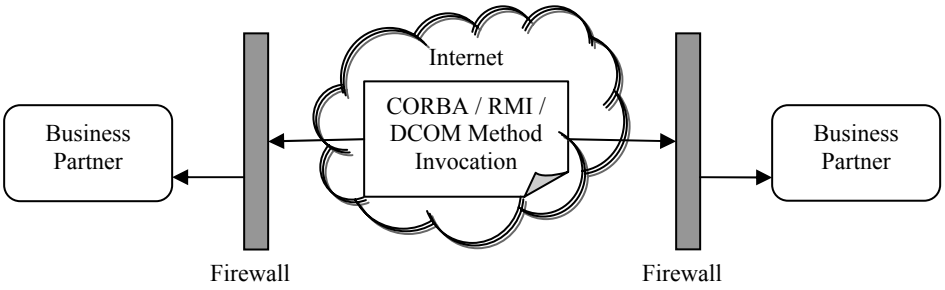


Figure 7.3: CORBA, RMI and DCOM in Enterprise

Web services technology solves some of the problems faced with the technologies discussed above when building B2B integration applications. SOAP, one of the technologies of Web services, standardizes the data format and the protocol by applying XML, which is a platform-independent language for description of data, and uses other standard Internet technologies. Web services provide set of specifications for the interoperability and ease of integration of diverse applications for business-to-business communication.

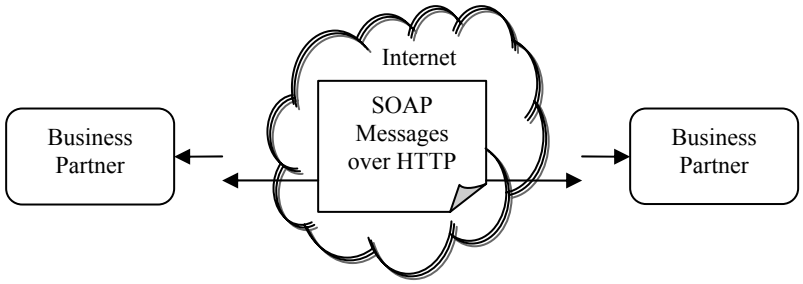


Figure 7.4: Simple Object Access Protocol (SOAP) in Enterprise

The main drawback with Web services is that they do not provide the capabilities to express business collaboration and process information. For this aim, ***electronic business XML (ebXML)*** has been proposed by United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT) joint with Organization for the Advancement of Structured Information Standards (OASIS) in 2001 [55].

ebXML proposes an XML based framework for e-business conversation, message interactions within computing environments and definition of business processes between diverse organizations. It provides a common structures and message syntax across all enterprise, and standard services for security, error handling and exceptions.

The framework utilizes a business process model, which can be done with UML or other modeling languages, used for capturing process information. The ebXML *Business Process Specification Schema* (BPSS) is produced as a result of this process modeling by applying some modeling techniques and tools, and generated as an XML document for the representation of use cases.

The *Collaboration-Protocol Profile* (CPP) is an XML document which describes the organization's functional and technical properties including the information about the business processes, protocol details for the transportation of the data and messaging, and the security aspects of the computing environment. It is used by other organizations to discover the offered services and engage in collaboration with the organization.

The *Collaboration-Protocol Agreement* (CPA) is as well an XML document used for definition of system level agreement for data interchange between partners based on their CPP documents. When both of the parties agree on the common CPA, communication and conversation can take place.

The offered services and processes can be discovered using *ebXML registry*, which demonstrates similarities with UDDI specification. ebXML registry stores information about organization, its services, business semantics, and processes for other partners to discover and consume them. Both ebXML registry and UDDI are the specifications of OASIS consortium and offer overlapping functionalities, except for some differences in the architectures.

The *ebXML Messaging Service Specification* provides a mechanism to handle the flow of information between organizations, and describes the characteristics of exchanged business messages. The message package in ebXML is the extension of SOAP messages and done by specifying schemas for the SOAP header. The package contains the SOAP envelope and the payload, which describe any required organization specific information.

The messaging service is required at runtime for the collaboration to ensue. The following figure illustrates simplified steps of the partner interactions in ebXML.

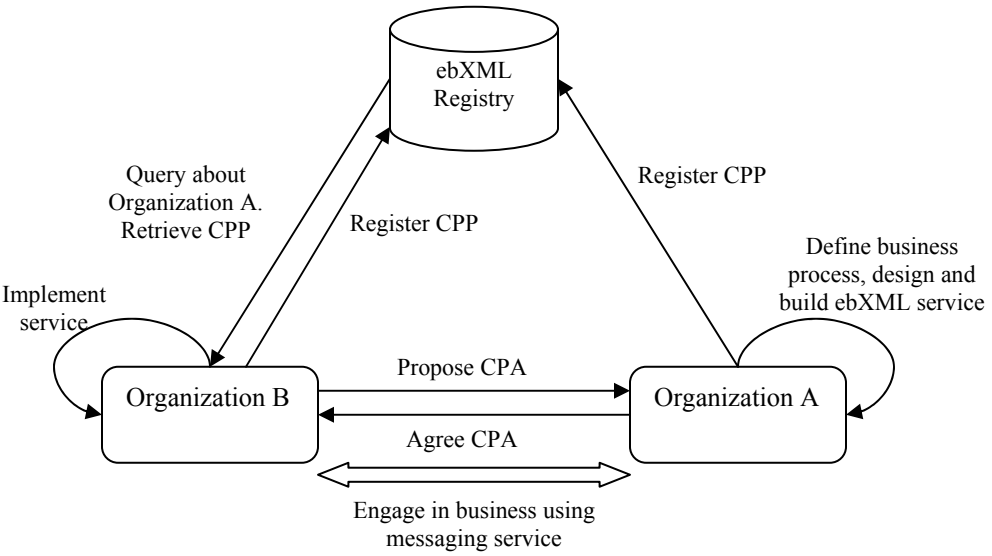


Figure 7.5: Business Collaboration Steps with ebXML

The technologies for e-business integration are in continuing development to support business requirements in a sophisticated manner. The following table compares the discussed technologies in terms of supporting e-business integration.

EDI	CORBA, RMI COM/DCOM	Web Services	ebXML
Message based communication Selecting message format applicable for both business partners Runs on private network Expensive to implement Information oriented integration	RPC based communication Standardization of protocol (e.g. IIOP) Usage of Internet, except for configuration of firewalls Offered standard services Service based business rules integration Lack of interoperability options	Standardization of data format and communication protocol Usage of open and standard Internet technologies Interoperability options Standard based message and service description Registry of services Service based integration	Capabilities to express business collaboration and process information Registry of services and processes Reliable and secure messaging Offered standard services Process based integration

Table 7.1: Comparison of e-business Integration Technologies

Web services include basically the technologies for service creation, description and registry of services. ebXML overlaps some aspects of the Web services and provides the utilization of processes for an effective and operable e-business integration. However, ebXML could not achieve to be an extensively accepted and common solution for the needs of the organizations. Business Process Management System technologies utilize Web services and offer additional technologies for business collaboration and usage of process information. These concepts and technologies will be discussed in the following section.

7.1.2 Enterprise Application Integration

Enterprise Application Integration (EAI) is a specialized form of integration technique in which separate applications can unify to build a cooperating application structure. These applications can share and exchange the information among diverse homogeneous and heterogeneous enterprises to provide application logic in order to fulfill the needs of the businesses and organizations.

An enterprise in general includes many specialized software that may have propriety architectures, and they do not allow easily sharing of its business logic and functionalities with other applications. As an example, packaged applications including SAP and Oracle ERP, CRM (Customer Relationship Management) and SCM (Supply Chain Management) applications, legacy systems and Portal implementations are dedicated applications that work individually to perform a particular function for the organization. Integration and interoperability of these applications is demanding as all of them have different interfaces and each system produces disparate information.

A likely approach to combine these separate applications is to use particular technologies that focus on the required functionality of the application and produce solutions along with the needs of the business. For instance, *Java Database Connection* (JDBC) can be used to call another application's database tables, or *Business Application Programming Interface* (BAPI) from SAP allows other applications to access SAP packaged application software. Using *J2EE Connector Architecture* (JCA) is one technique to integrate these interfaces with other applications [56].

EAI offers mainly two topologies for integrating applications [57]:

- **Application-to-Application integration:** This approach is based on point-to-point integration and suitable if there are not many applications to integrate, as it combines the applications directly and bind one interface to other one to use the functionality and processes that it provides.

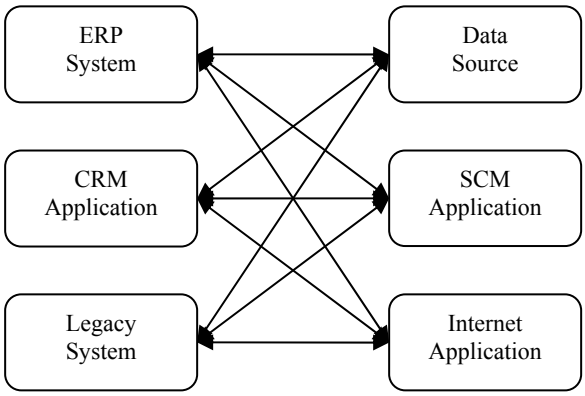


Figure 7.6: Point-to-Point EAI Topology

- **Integration Broker:** Because of the complexity of application-to-application integration method, integration broker approach, also known as "hub-and-spoke" model, provides a centralized system in which each application that needs to communicate with another one initially send the message to the broker, which converts and routes the message to the recipient application. The broker holds software components called as *Adapters* for transformation of the messages in order to be comprehensible by application interfaces. This approach is more loosely coupled compared to point-to-point integration method of EAI.

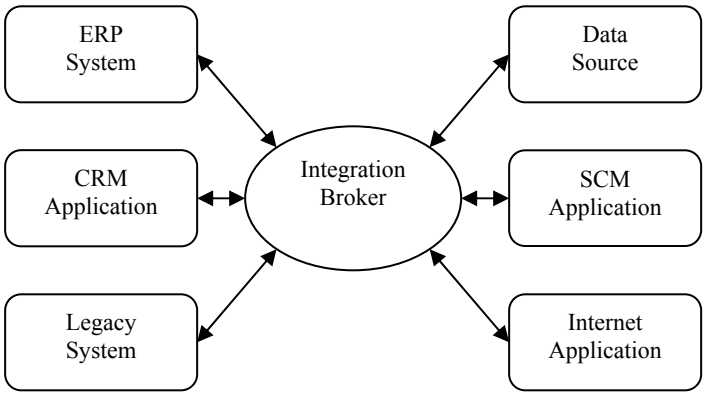


Figure 7.7: Integration Broker EAI Topology

A variation of Integration Broker method, called as ***Integration Bus*** approach, is architected to form an integration bus in which each application has its own adapters and talk to a common, centralized system.

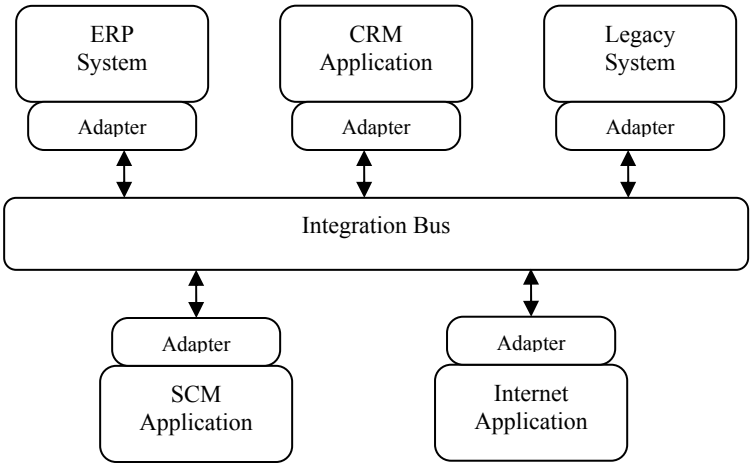


Figure 7.8: Integration Bus EAI Topology

The main drawbacks for EAI products are that they are expensive to implement and dependent on considerations and effort so not to have failures in building an integration solution. These products are proprietary applications which make them difficult to interoperate with other applications and expand the application structure to support an enterprise wide functional scope.

Web services provide better solutions to integrate diverse applications by delivering open and standard technologies which replace the traditional EAI products. Service oriented solution to EAI reestablishes multiple application architectures that rely on varied programming languages and operation environments by offering reusable and platform independent services.

Web services can be used as an individual solution to integrate two or more application logic. The principles of service orientation oblige that the traditional three-tier application structure provides an abstraction of service layer on top of its business logic to connect to various other applications including tightly coupled GUIs, mobile devices, and process engines. Services enable to share the information without affecting the business logic of the application, however, service orientation requires that the application provides necessary infrastructure for services to operate.

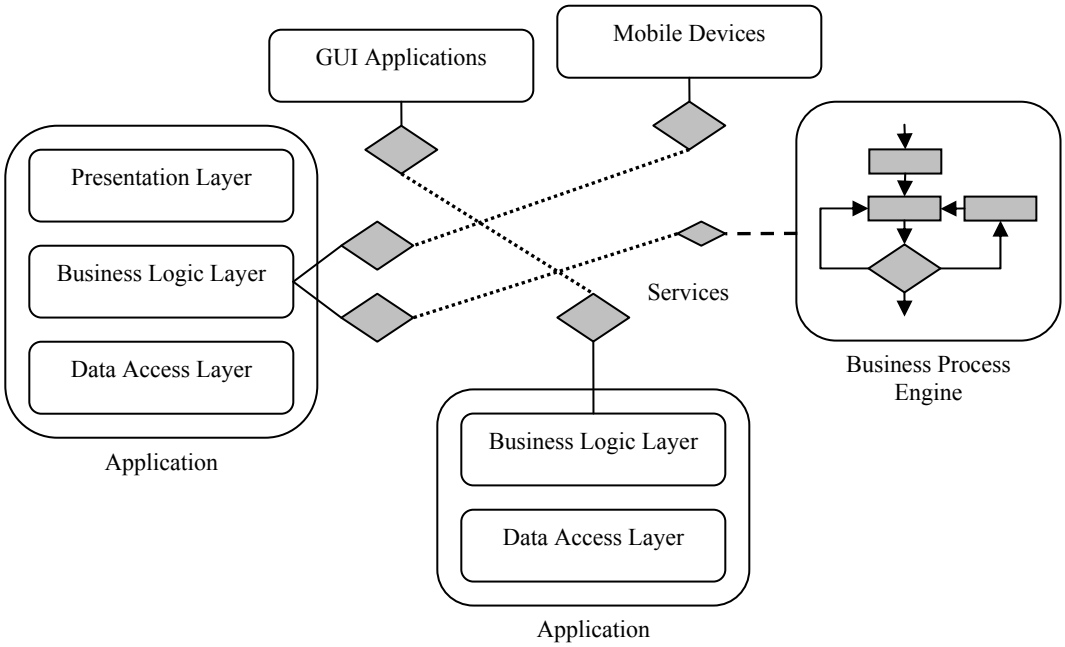


Figure 7.9: Web Services for Enterprise Application Integration

Using Web services in EAI provides straightforward solutions with considering business requirements and strategies for an economical and undemanding application structure in which reusable services are used for integration logic rather than proprietary integration products. Web services connect the systems directly through common data format and standards and allow application modules to use common service repository that stores and retrieves individual service descriptions. Applications discover the required services from this repository and easily bind to them by SOAP messages.

An additional approach to apply Web services in EAI field is to use ***Enterprise Service Bus (ESB)***, a middleware solution offered from vendors such as BEA, IBM, SAP, IONA and Systinet, and capable of providing infrastructural elements to distributed services on the network. The ESB approach offers asynchronous, message oriented communication infrastructure that deals with message routing, orchestration, and service management. It allows loosely coupled, document oriented message exchange between diverse and independent systems.

ESB provides certain capabilities [58], which includes routing, addressing, synchronous and mainly asynchronous messaging infrastructure, connectivity to EAI middleware, protocol transformation, language interfaces for service invocation, security options such as authentication, authorization and confidentiality, messaging processing framework, modeling options for services such as business object models, public versus private models for B2B integration, and development and deployment tooling, to connect new and existing software applications within and across enterprise, with a rich set of features and enabling management and monitoring of interactions between applications.

Organizations are using ESB as it provides common and standard-based communication and integration services which facilitate interoperability between applications executed on different platforms and with different programming models. The architecture of ESB is centered on a bus, and applications and services plug into the bus using standards based technologies such as SOAP, HTTP, JMS (Java Messaging Service), and JCA (Java Connector Architecture) [59]. The following figure illustrates the ESB architecture.

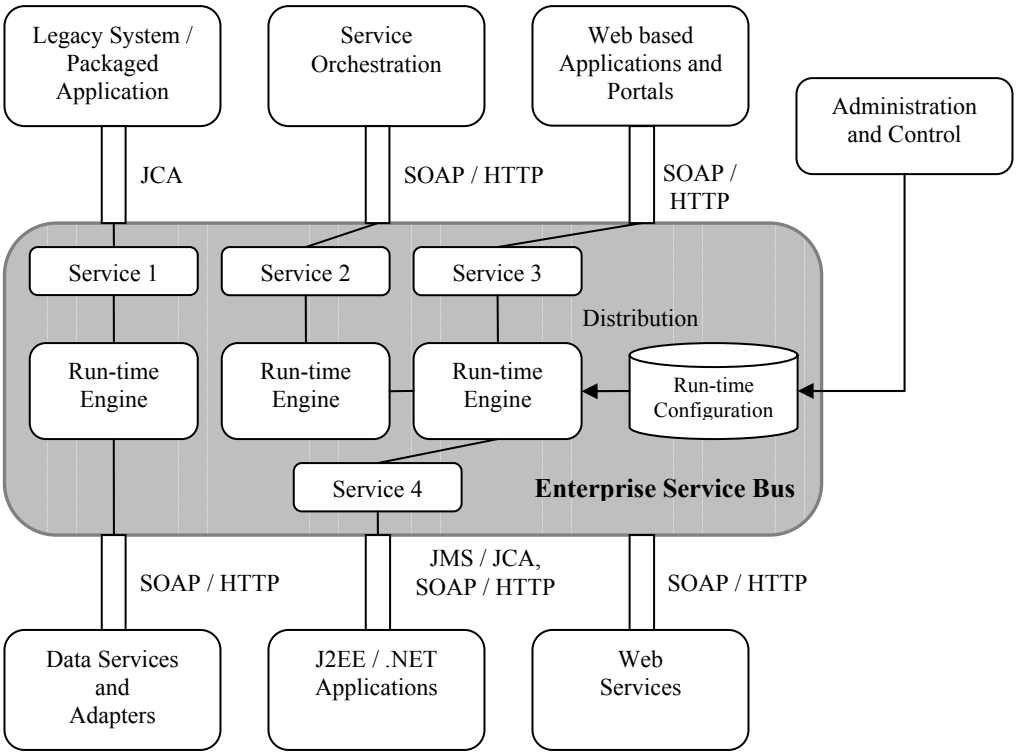


Figure 7.10: Enterprise Service Bus Architecture

The basic ESB architecture usually contains a distributed ESB engine, responsible for message delivery and service invocation as well as quality of services on these operations, distributed ESB services, a run-time configuration, and centralized administration control. Typical distributed ESB services are service locator/routing directory, transactional support, routing rules, mediations, and transformations. Additional ESB services can include security and monitoring.

Run-time configuration defines configuration and distribution of ESB components, which includes engine and services, and manages ESB metadata. Centralized administration and control is usually provided through the GUI that allows viewing and modifying ESB metadata. Service providers and consumers connect to the ESB either directly through the ESB interfaces or using bus adapters.

ESB can be seen as the replacement of traditional EAI products by applying service oriented integration principles. The foremost architectural consideration of SOA is to provide a registry in which service consumers can discover service providers and bind to them to execute the services. SOA does not offer a central control mechanism for service consumers and providers, on the other hand, the communication takes place peer-to-peer interaction of individual parties. ESB is particular case of SOA wherein the registry notion is not present; however, still it is capable of articulating other significant characteristics of service oriented architecture.

7.1.3 Portal Oriented Integration

Portal oriented integration aims to provide a single user interface for internal applications and external enterprise computing systems without requiring a direct integration of them. It aggregates diverse applications and forms a single access to these resources within the enterprise mostly through a Web browser. The portal application has the capability to access packaged applications, databases and other resources in enterprise by the usage of Web server, middleware technologies and Web enabled frameworks.

The main difference of portal oriented integration methodologies with other integration approaches is that portal orientation allows aggregating of information from multiple enterprise systems to a single application or an interface without requiring real-time exchange of information. A portal can be thought of a content aggregator customizable by the end-user in terms of the applications that the portal contains and look-and-feel aspects of the user interface. Portals are quite common especially in B2B area in which the information is exchanged in an automated manner with the coordination of the end-user. Yahoo [60] is an example portal application which gathers information and resources from many sites within the enterprise and Internet for its users.

The remarkable development in Portal-based integration is to use Web services for aggregation of content from different resources. ***Portlet*** and ***Web Services for Remote Portlets*** [61] specifications defines the way to use Web services to generate mark-up fragments which contains diverse application logics gathered from enterprise within the portal application. A *portlet* is defined as user facing, interactive Web component managed by a container, can process requests and generate dynamic content for the user. Portlets communicate with users through the hosting portal server. Typically, portal servers maintain a catalog of available portlets from which end users can select them for placement on portal pages. The Java Portlet Specification (JSR-168) [62] defines a standard API for J2EE based portal platforms.

The portal invokes the requested portlets through the portlet container. The portal framework creates the portal page with the fragments generated by the portlets and returns the page to the end-user. The following figure illustrates this process [63].

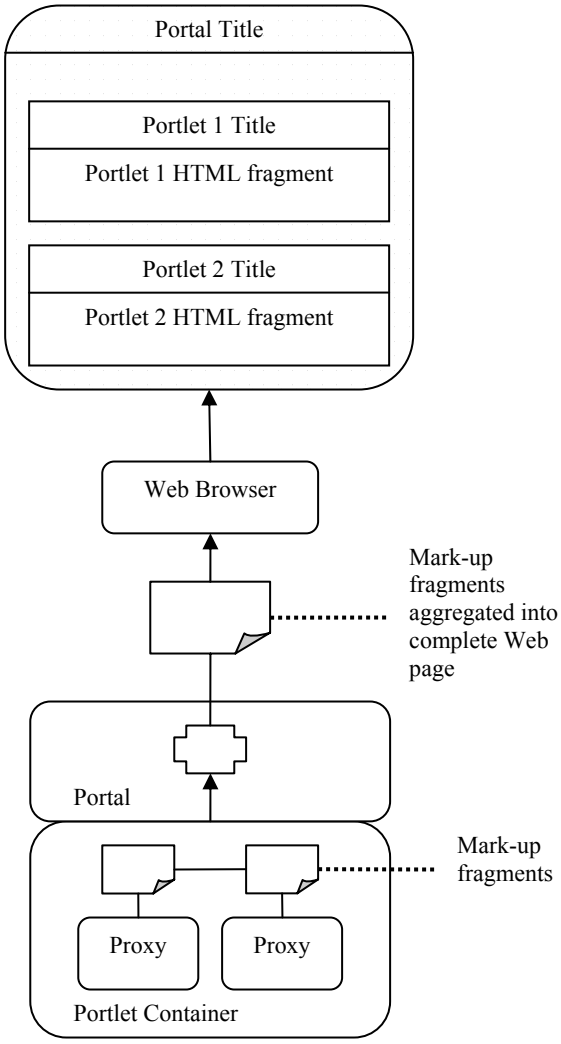


Figure 7.11: Aggregating Mark-up Fragments from Local Portlets

Integration of content and applications into portals has been a challenging task which requires significant programming effort. Portal frameworks allow dynamic integration of business applications in a straightforward manner by accessing to the remote portlets which interact with the required service and application logic. Web Services for Remote Portlets (WSRP) specification defines the methodology for a portal to display remotely-running portlets inside the portal page without requiring any additional programming by the portal developer. Leveraging WSRP within service oriented architecture provides a powerful combination in which presentation-oriented portlet fragments can be discovered and reused without engaging in additional development or deployment activities.

Once a remote portlet has been published, portal administrators can use their portal administration tools to search the registry for Web services that implement the WSRP interface and make some of the matching portlet Web services available for their users by adding them to their portal's portlet registry [64]. End-users can select the portlets from the registry to be displayed on their personal pages just like local portlets.

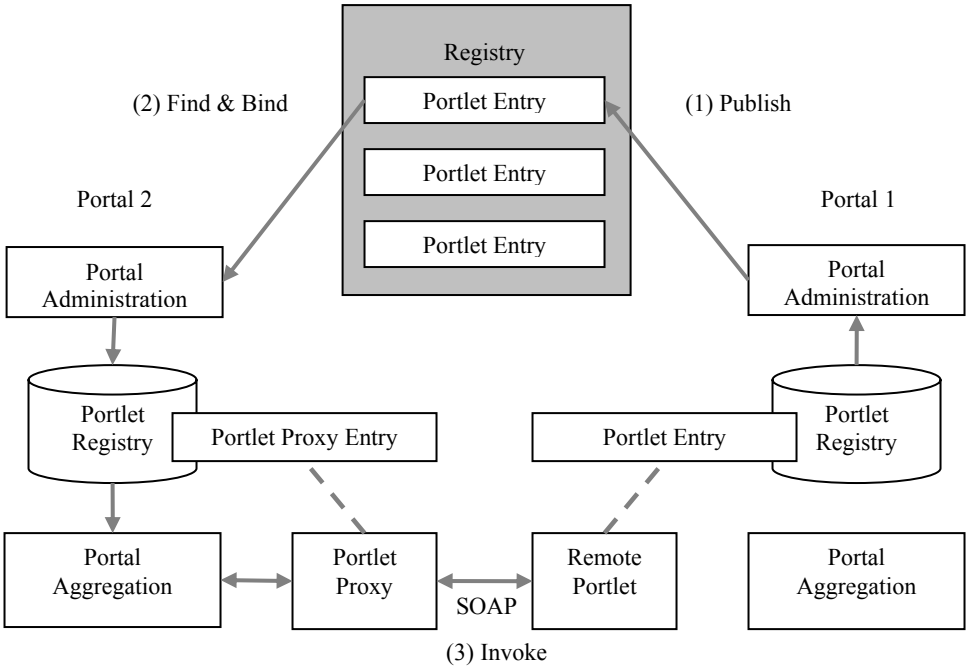


Figure 7.12: Publishing, Finding, and Binding WSRP Services

This mechanism discussed above allows presentation-based, interactive Web services to plug into portal framework for the aim of creating a single interface which includes the functionalities offered by the providers in the Internet. WSRP allows interoperability of portals by providing remote access to resources in enterprise in a standardized manner.

7.1.4 Business Process Oriented Integration

Business process oriented integration involves integration of application logics encapsulated in remote systems through well-defined business processes located as an upper layer on top of existing set of services within the enterprise. Process oriented integration provides mechanism for the activation of processes in proper and sequential order to fulfill the functionalities that the business needs. These processes are valuable for the businesses as they express the accurate flow of information and the execution order of business applications. They are built as a separate layer on existing applications including object-based systems, packaged applications and services, and manage and control these enterprise resources when they are executed. Processes are the assets of the businesses in which the required functionalities are described as small, well-designed units by allowing data exchange between participating systems.

A ***Business Process*** is defined as a set of activities that have to be executed in some order to accomplish a certain functionality or business goal within the environment of the organizational structure and the enterprise. The process definition describes network of activities, their relationships, participating units including applications, organizations and people, data flow between activities, and properties that surrounds the process such as conditions for a process to start and end of the execution. ***Business Process Management (BPM)*** provides an infrastructure for the design, deployment, execution, maintenance, and monitoring of business processes. BPM systems supplies necessary tools for interpretation of process definitions, modeling, development and management of processes while they are executed. The system ensures that the sequences of work items are assigned to appropriate participants and resources are invoked where required.

Workflow is the automation of business processes. It can be seen that BPM systems offer similar descriptions for the automation of processes provided by the Workflow Management Coalition (WfMC), and become widespread especially after invention of Web services. The Workflow Management Coalition, a non-profit, international organization, identifies the common product structure and proposes specifications for the use of workflow through common terminology, allowing interoperability and connectivity between products [65]. Workflow management system defines and creates a workflow instance from its process definition, which requires a *process definition tool*, and executes this instance through the use of software, called as *workflow engine*. Process definition tool is necessary to create and demonstrate the process description in a form that the engine is able to execute. This tool utilizes a process definition language to model the process definition in textual or graphical form or in a formal language notation.

BPM offers strategies that focus on definition of business processes and integration of them within and between enterprises, rather than the development of tightly coupled individual application structure. Business process integration involves integrating of several applications by utilizing various metadata, platforms, and processes.

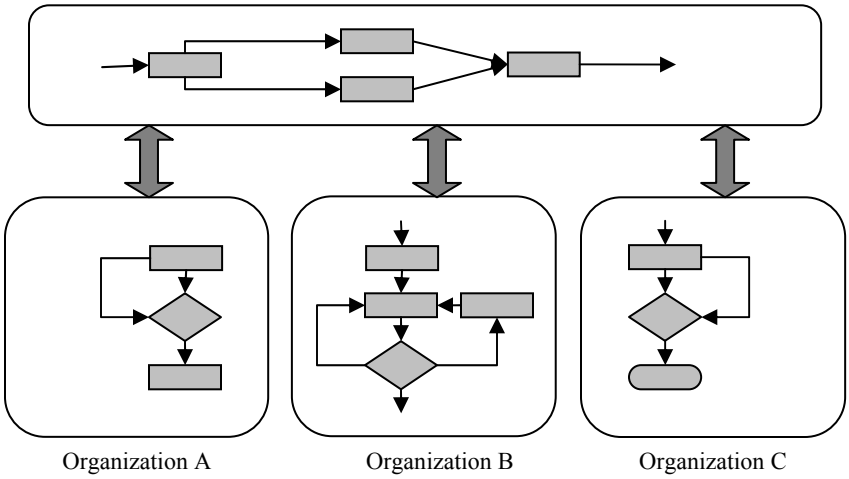


Figure 7.13: Process Based Integration Approach

Web Services are the accepted technology for the interpretation of business processes. BPM systems use Web services as the technology for the description of business processes, and offer a new approach which replace traditional workflow systems that applies proprietary process definition languages and runtime environments. ***Orchestration*** and ***Choreography*** of Web services express combination of Web services together in a consequential order to create an executable business process. The difference between orchestration and choreography is that orchestration refers to an executable business process which is controlled by one of the involved business parties; however choreography describes the collaborative work of each involved parties within the whole interaction of web services.

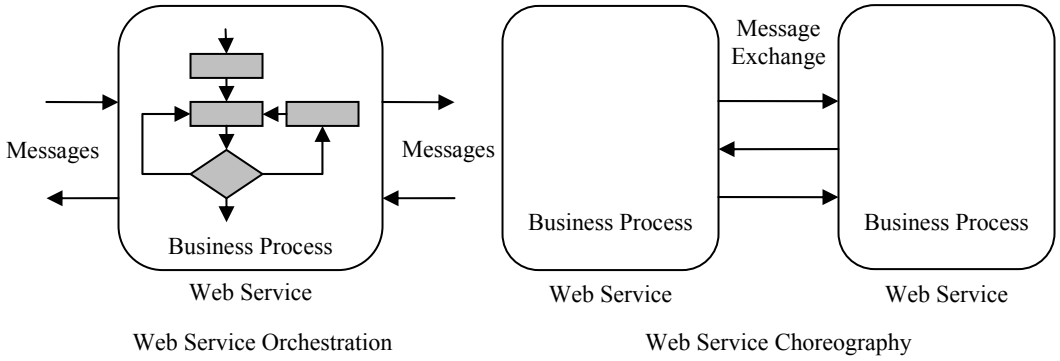


Figure 7.14: Web Service Orchestration and Choreography

Process definition language is used to describe the sequencing order in which the participating Web services are invoked to accomplish a business function, and how the operations of Web services are correlated to form a business conversation, as these operations are stateless by nature. Although various process definition languages proposed from organizations and vendors, still there is no standard and universally accepted language for the description of business processes. Each of these languages has different strength and weakness to express the business process.

One of the languages used for process definition is the ***Web Service Choreography Interface (WSCI)***, [66] an XML-based language proposed jointly from Intalio, Sun Microsystems, SAP and BEA Systems. The language describes the flow of messages exchanged among interacting Web services by providing a global, message-oriented view of a process definition.

It is a choreography language, which means it describes the observable behavior between Web services without dealing with the definition of an executable business process and transactional properties. WSCI describes the interdependencies among the Web service's operations so that any client can understand how to interact with such service in the context of the given process, and can anticipate the expected behavior of such service at any point in the process's lifecycle.

WSCI describes the details of the behavior of the Web service within a process whose execution can be initiated by the receipt of a message. A single WSCI interface describes the message exchange from the point of view of the consequent Web service. The following figure describes the relationship between WSCI interface and Web services.

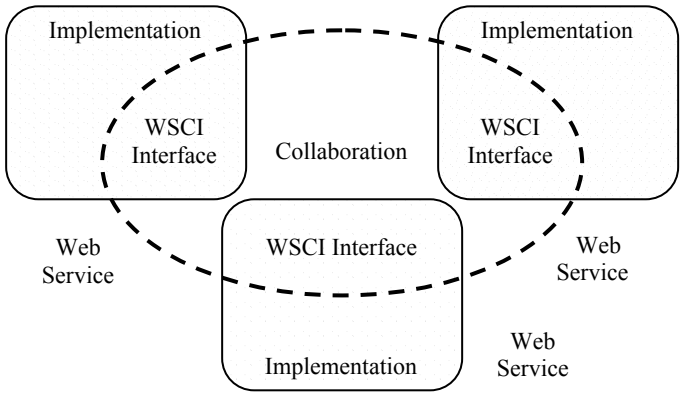


Figure 7.15: WSCI Interface and Web Services

Most common process definition language is the ***Business Process Execution Language for Web Services (BPEL4WS)***, [67] a specification written jointly by IBM, BEA, Microsoft, SAP, and Siebel. It is a union language which features from IBM's Web Service Flow Language (WSFL) and Microsoft's XLANG. BPEL4WS utilizes an XML-based grammar to create process definition and is layered on top of WSDL to describe the necessary Web service components for defining the messages being exchanged, operations being executed and the required port types.

The language is used to support the two separate usage scenarios:

- ***Abstract process*** is for definition of the business protocol role and identification the message exchange behavior between different parties involved in the protocol with hiding their internal behavior.

- ***Executable process*** identifies the actual behavior of a participant in a business interaction by defining the sequential order of the Web service executions between each business partners. It defines how many service interactions with these partners are coordinated and as well introduces systematic mechanisms for dealing with business exceptions and processing faults.

BPEL4WS process definition contains a set of elements which describes the control flow, asynchronous interactions, correlation, faults, compensation and other components within the business process. The process definition defines process in terms of its interactions with *partners*. A partner may provide services to the process, call for services from the process, or contribute in a two-way interaction with the process. *Partner links* identify the shape of a relationship with a partner by defining the message and port types used in the interactions in both directions.

The following figure identifies the relationship between BPEL4WS process and its partners.

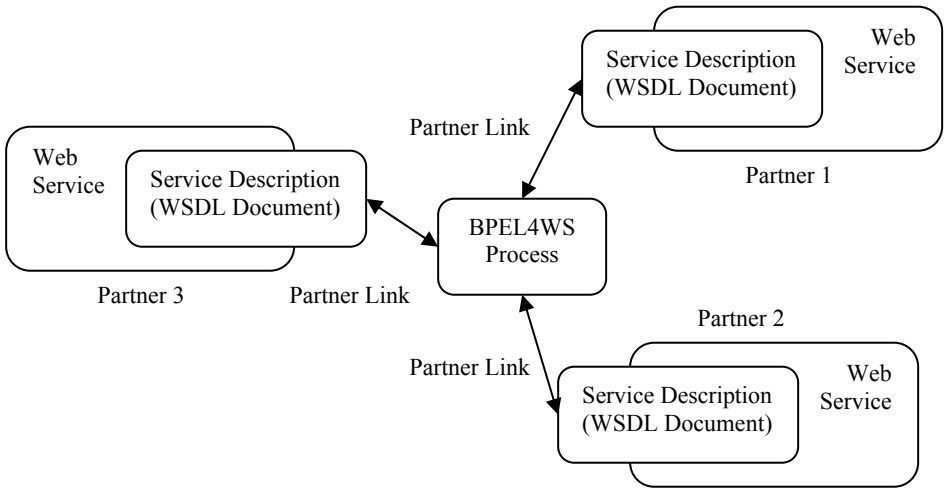


Figure 7.16: BPEL4WS Process and Partners

There are other various specifications exist, such as *WS Choreography Model*, or process definition languages, such as *Business Process Modeling Language (BPML)*, which support and extend Web services technologies in order to allow composition of services to have state-full and long-running interaction between business partners in the enterprise. BPM systems with Web services provides tools and well-defined strategies for information passing between participating systems, modeling and visualizing of business processes in application and enterprise level, and automation of them.

Process integration is more sophisticated than other approaches in a sense that an instance of a business process covers many traditional application integration solutions and extends the boundaries from communication of two or more application to the enterprise level integration by providing a *process model*. The process model provides an abstraction for the business rules and values, and determines how each system should interact with each other in order to achieve the business goal.

Process based integration provides a satisfactorily integration approach for businesses considering the requirements of the enterprise and allowing real-time integration with the enabling middleware and points of combination. It is more business centric approach which gives business individuals the possibility to configure the process model without requiring demanding changes in the implementation model. As the processes constructed from lower level services within the enterprise architecture, it is possible to build a hierarchy of processes, from lower-level processes to higher-level processes, which deals with specific and small-sized application integration, and makes it straightforward to maintain individual activities of the process instance.

7.1.5 Realization of SOA through Integration Approaches

SOA implementation can range from very primitive to very sophisticated realizations consistent with the needs of the enterprise and organizations. The following table points up different levels of SOA implementations.

Basic SOA	It is a realization of SOA that simply communicates through messages from system to system. The concept of service is not applicable. EDI implementations are examples of basic SOA.
Preliminary SOA	It is an approach to SOA implementation in which the notion of service exist, however, the execution environment does not apply the principles of SOA completely. As an example, Enterprise Service Bus (ESB) implementations are lack of registry of services, which makes them not a true SOA realization.
Central SOA	It is the SOA implementation that covers the all the requirements and principles for a completed SOA realization. It is capable of moving information from source to target systems, as well transformation and routing of services. EAI with Web services offers central level SOA implementations.
Advanced SOA	It is an advanced level realization of SOA which involves common directory for discovering actual services, and allows brokering and managing them. It provides real-time implementation of services and leverages application behavior as if it is implemented locally. Portal based integration based on WSRP is the example of advanced SOA.
Sophisticated SOA	It is the SOA implementation which allows composition and orchestration of services in order to build an application structure layered on top of services to achieve specific business goal. It integrates diverse applications in enterprise and creates a flow of services in the form of business processes. Business process oriented integration is an example approach for a sophisticated SOA implementation.

Table 7.2: Realization of SOA Implementations in Enterprise

7.2 Considerations for SOA Implementations

SOA implementation in enterprise is challenging as it involves fulfillment of many additional features, functions, and quality of service (QoS) requirements to have an enabled and proper execution environment. Core Web service technologies, such as SOAP, WSDL and UDDI provide the basic background for a service to operate, however, in enterprise level these technologies have to be extended to encompass enabling execution of many types of applications and integration of them in order to accomplish the complete vision of service enabled enterprise.

Additional technologies, such as security, transactions, and reliable messaging, are required considerations for an enterprise that builds on Web services, as the standardization of these technologies are still in continuing progression. The organizations including the World Wide Web Consortium (W3C) and Object Management Group (OMG) define and propose various specifications for Web services to cover all the required functionalities that an enterprise may need for successful construction of SOA.

Quality of service (QoS) requirements for a Web service includes the following considerations [68]:

- **Availability:** it is defined as the probability that a service is available at a particular time.
- **Accessibility:** it is defined as the capability of a service to reply incoming requests. It represents the successful instantiation of the service at a point in time.
- **Integrity:** it represents how the Web service maintains the correctness of the interaction in respect to the source. Proper execution of Web service transactions provides the correctness of interaction.
- **Performance:** it is the responsiveness of the system measured with the time required to execute some function. It is calculated in terms of throughput and latency. Throughput represents the number of Web service requests served at a given time period. Latency is the round-trip time between sending a request and receiving the response.
- **Reliability:** it represents the degree of being capable of maintaining the service and service quality. In another sense, reliability refers to the assured and ordered delivery for messages being sent and received by service requestors and service providers.

Quality of Service (QoS) features improves the significance of SOA environment and makes Web services better suited for use in more kinds of SOA-enabled applications. Not every service based application require extended features for their execution, however, in some cases, the application may not be functioned properly if some aspects of the environment is not advanced, such as ensuring reliable messaging environment. Reliable messaging guarantees that one or more messages were received the appropriate number of times and followed correct message exchange pattern. Competing specifications are proposed for Web services, such as *WS-Reliability* and *WS-ReliableMessaging*, to ensure message delivery and ordering without duplications.

Other specifications in messaging area cover event notification and publish/subscribe mechanism and extend the asynchronous messaging capability of Web services. Such proposals include *WS-Eventing* and *WS-Notification*. Notification delivers messages through an intermediary often called a *message broker* or *event broker*. Subscribers identify the topics for which they wish to receive messages. Publishers send messages to the channels or topics on which subscribers are listening. Notification is a messaging mechanism that can be used to set up broadcast and publish/subscribe messaging.

7.2.1 Service Control and Management

Service management is a required activity for an organization, as services may need changes from time to time and if the number of services that an organization provides to its partners is many, it is essential to define a mechanism to control and manage them. In a dynamic service oriented environment, the messages a service accepts, the routing of messages from one service to another and the usage area of the service may change over time. Management covers all these aspects before putting the services into production.

Service management and monitoring includes policy configuration, distributed *Service Level Agreements (SLA)* supervision between business partners, metadata management, provisioning and routing of services. These capabilities involve both reporting and changing the configuration parameters for a service which requires to find out what applications are using the service at a given point in time. This is necessary to understand the growing demand on the service for capacity planning and to provide information about service utilization.

SLA is formed between two parties and states how services will be used and accounted for and any prerequisites for use. Metadata management includes the description information about a Web service necessary to construct a message body including its data types and structures, and message headers so that a service consumer can invoke the service. Metadata specifications include *WS-Addressing*, *WS-Policy*, and *WS-MetadataExchange* and are necessary for the correct operation of an SOA based on Web services. WS-Addressing defines endpoints and reference properties associated with endpoints in communication patterns. WS-Policy is a framework which includes policy declarations for various aspects of security, transactions, reliability, and quality of service requirements with a WSDL definition. WS-MetadataExchange is for querying and discovering metadata associated with a Web service, including the ability to fetch a WSDL file and associated WS-Policy definitions.

7.2.2 Transaction Management

Transaction is defined as multiple operations on persistent data are completed as a unit, whether succeed or fail together. Transaction processing technologies provide mechanisms to recover for an application to a known state after some failures or inconsistencies. Enterprise platforms, including J2EE, have transactional guarantees on their behavior. For J2EE, Java Transaction API (JTA) provides standard API for accessing the transactional capabilities provided in compliant software. In most cases, for Web services, the underlying execution environment provides required transaction processing capabilities. However, still it may be needed to have transactional context for Web services so that multiple services can be grouped into a larger transactional unit and can be coordinated across multiple execution environment. Web service transaction specifications extend the transaction processing technologies by adjusting the two-phase commit protocol for Web services, and define new extended transaction protocols for supporting compensation-based and long-running business transactions.

The specifications in this area include:

WS-Transactions:

- **WS-AtomicTransactions:** defines ACID transactions for a standard two-phase commit protocol and short-lived executions.
- **WS-BusinessActivity:** defines transactions for uncertain commit and compensation-based undo protocols for longer-lived executions.
- **WS-Coordination:** defines the management and coordinator for the WS-Transaction family of protocols and their variations.

WS-Composite Application Framework (WS-CAF):

- **WS-Context:** defines a context management system for generic context.
- **WS-CoordinationFramework:** defines a management and coordinator for the basic context specification and the transaction protocols defined in the WS-TransactionManagement specification.
- **WS-TransactionManagement:** defines three transaction protocols for the pluggable coordinator: ACID, long-running compensation based transactions, and business process management.

Another specification is the *Business Transaction Protocol (BTP)*, proposed by OASIS and specifies that in a business transaction no single party controls all resources needed; rather parties manage their own resources but coordinate in a defined manner to accomplish the work scoped by a transaction. BTP provides two types of transaction: *atoms* are business transactions where all participants have to agree before a transaction can be committed, which means all participants in an atom are guaranteed to see the same ending to the transaction; and *cohesions* which provides a central coordinator that reviews the status of each member of the transaction. Even if some of the members cannot successfully commit the transaction the coordinator can still decide to allow the remaining members to commit.

These specifications overlap some functionality, such as they are all centered on an extended coordinator notion, define atomic and compensation based long running transactions. WS-CAF divides context management into a separate specification and adds another protocol specifically designed for business process management. BTP proposal defines a loosely coupled protocol that ensures that multiple Web service interactions are correctly propagated and shared. The adoption of these new transaction protocols will allow improved business process execution environment and standardization of these technologies for the needs of enterprise.

7.2.3 Security

Security is a requirement in an enterprise to protect the application structure against various threads. For Web services, as the proposals in this area are still in progression, it is important to ensure the security by providing some mechanisms for message exchange and surrounding execution environment.

Basic mechanisms include encryption, authentication, authorization and logging for problem definition. Internet security technologies comprise Secure Socket Layer (SSL) and secure HTTP (HTTPS), which provides basic encryption-level security possibilities. With J2EE, platform security is provided by the Java Authentication and Authorization Service (JAAS) and supporting infrastructure.

XML has its own XML-based security technologies which can be applicable for Web services as well. These technologies protect XML data and the message is secured while it is exchanging as a SOAP message. *XML Encryption* provides confidentiality by ensuring that the content of the document cannot be captured and read by unauthorized persons. *XML signature* is another technique which provides integrity by ensuring that service providers can determine whether or not the documents have been altered while transportation and received only once.

The main proposal for Web services is ***Web Service Security (WS-Security)***, which provides a framework for message level security on an end-to-end basis for Web services messages. WS-Security headers include the ability to carry authentication formats such as Kerberos tickets and can use XML Encryption and XML Signature technologies for further protecting the message content. Other specifications exist to extend the WS-Security specification:

- **WS-SecurityPolicy:** it defines the security requirements of a Web service, so the consumer of the service preserves them.
- **WS-Trust:** it defines the ways to build a trustable and secure environment by using required security tokens from trusted sources.
- **WS-SecureConversation:** it describes the requirements for multiple Web service invocations to maintain a constant context and have a secure session.
- **WS-Federation:** it defines how to build a federated session so that a Web service may require to be authenticated once in multiple security domains.

Security is important and required to control access to Web services and ensure the confidentiality and integrity of Web services data, especially when multiple Web services are executed collectively. Although there is no common standard yet for securing Web services, the technologies discussed above will enhance the security possibilities of Web service deployment environment.

7.3 Grid Computing

Grid Computing is based on a set of fundamental services that allow end users and applications to share information and resources in heterogeneous computing environments. Grid computing discipline involves the actual networking services and connections of a potentially unlimited number of ubiquitous computing devices within a "grid" [69]. End user sees computing resources as one large system and is able to have single access to these resources, including software, data files, services, and licenses.

Grid participants share the distributed and coordinated heterogeneous resources in a virtualized form. The resource virtualization can be organized into virtual organizations, each one sharing its own resources cooperatively as a larger grid. Participants of the grid can be members of several real and virtual organizations.

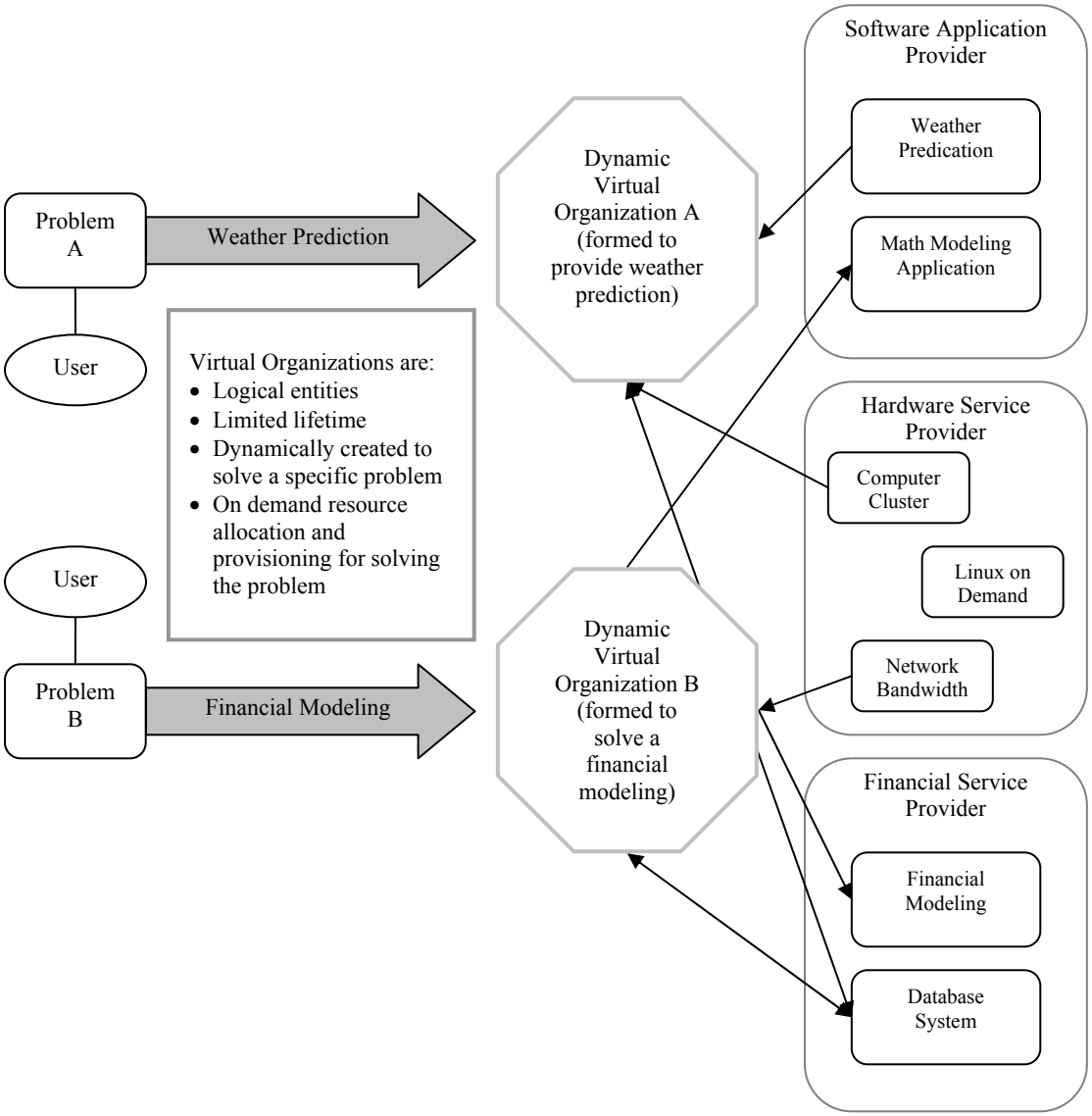


Figure 7.17: Resource Virtualization in Grid Computing

Grid computing is based on open set of standards and protocols. The Open Grid Services Architecture (OGSA) Working Group proposed a service oriented approach for the definition of the Grid Architecture, a technology independent blueprint [70] for the implementation of the required services and their specific characteristics.

8. Conclusion

Service Oriented Architecture (SOA) provides a common platform and execution environment for heterogeneous applications built with diverse technologies. Service orientation supports interoperability of these applications by hiding their internal structures from each other, and creates a flexible and loosely coupled distributed systems.

Service oriented development is the natural evolution of component based software development, and services are created from components. Service development is the central part of service oriented application structure, and it is challenging as the interface design for a service should be enough competent to support the needs of internal computing systems, as well as the other unknown consumers from external organizations and enterprise.

SOA offers creation of flexible, loosely coupled and reusable service logics, which makes service based development valuable for many organizations. The businesses gain benefit from SOA implementations, as the services are capable of expressing not only technical properties of computing environment, but also can focus on representing and solving of specialized business functionalities and problem domains. Remotely located services can be composed into business processes, and the process flow is essential for an organization to fulfill its business activities in an effective and computerized way.

SOA implementations in enterprise provide straightforward integration of application logics and capable of building sophisticated application structures for the needs of businesses and organizations. SOA is best realized in enterprise by Business Process Management System technologies, as the process information is constructed by utilization of individual services, and the business process is located on top of services. Enterprise Application Integration based on Web services is as well an effective approach to SOA realization as it replace the traditional middleware technologies and provides improvement in building application structures.

Although SOA is independent from any specific technology, it enhance its vision with Web service technologies, however, these technologies are still evolving and support for mature SOA implementation based on Web services is the upcoming target and working area of many organizations and computing environments.

Within this thesis study, a prototypical service based application has been developed to illustrate the fundamental characteristics of SOA. At the basic level, the case study proves the interoperability of different platforms, reusability of loosely coupled service business logic, and consumption of service from other software modules to form a service based interaction of applications.

SOA can be implemented by combination of different tools which provides specialized functionalities for the construction of SOA execution environment. A rather new trend is to offer frameworks which support all required tools and models within the same infrastructure. Currently in the industry many vendor specific SOA based frameworks exist with diverse offerings, including integration platforms, SOA management and monitoring suites, and collection of service design, creation and modeling tools. Unfortunately, the open source developments in this area are still evolving and most of the current implementations do not support complete SOA framework requirements.

SOA is becoming more adoptable and operable as the organizations continue to support service based implementations. In the future, SOA will have a significant influence on the development of enterprise application infrastructure with the facilitation of developing technologies.

8.1 Further Studies

SOA has a broad influence in each stage of application development including analysis and design of individual services, service deployment, creation of new applications from services and implementation of services through service oriented strategies and approaches. SOA obliges and applies its principles and architectural considerations to these stages of application development. Within this context, it is possible to elaborate and broaden the specialization of the applicable concepts and topics discussed within the thesis study. As an example, Model Driven Architecture is an emerging concept which can be analyzed and explained in more detailed way as a separate study with considering its relationships with SOA.

Another further study can be related with Grid Computing, as it is a distinct topic which has its own characteristics and strategies. Grid Computing is based on SOA, and it can be a separate study to describe what it is with surrounding features and concepts.

Appendix: List of Web Service Specifications

Several organizations are proposing specifications for the interoperability of diverse Web service implementations and the adoption of Web services successfully. Here some of the specifications will be described and organizations that propose them will be introduced.

Organization	Description	Specifications	Web Site
World Wide Web Consortium (W3C)	A consortium of several member organizations that progressing standards for the Internet.	SOAP, WSDL, WS-Choreography, WS-Addressing, XML Encryption, XML Signature ……	www.w3c.org
Web Services Interoperability Organization (WS-I)	A consortium of mostly vendor companies focusing on web services interoperability and compatibility.	WS-Security WS-Transaction WS-Coordination WS-Attachments WS-Inspection WS-Referral WS-Routing ……	www.ws-i.org
Organization for the Advancement of Structured Information Standards (OASIS)	A consortium that focuses on the development of e-business standards.	UDDI, WS-Security, WS-BPEL, WS-Composite Application Framework, WS-Notification, WS-Reliability, Web Services Policy Language, Web Services for Remote Portlets, Web Services Distributed Management, Web Services Resource Framework …….	www.oasis-open.org

Table 1: Organizations That Propose Web Service Standards

Web Services Addressing (WS-Addressing)	**Description:** provides transport-neutral mechanisms to address Web services and messages, to identify Web service endpoints and to secure end-to-end endpoint identification in messages. WS-Addressing replaces earlier proposals called WS-Routing and WS-Referral.
	Organization: World Wide Web Consortium
	Web site: http://www.w3.org/Submission/ws-addressing/
Web Services Policy Framework (WS-Policy)	**Description:** a framework that includes policy declarations for various aspects of security, transactions, and reliability. WS-Policy provides a general purpose model and corresponding syntax to describe and communicate the policies of a Web Service. • **WS-PolicyAssertions** specifies a set of common message policy assertions that can be specified within a policy. • **WS-PolicyAttachment** specifies three specific attachment mechanisms for using policy expressions with existing XML Web Service technologies.
	Organizations: IBM, BEA, Microsoft, SAP
	Web site: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-policy.asp
Web Services Eventing (WS-Eventing)	**Description:** provides a protocol that allows Web services to subscribe to or accept subscriptions for event notification messages.
	Organizations: BEA, Microsoft, TIBCO, IBM
	Web site: http://www-128.ibm.com/developerworks/webservices/library/specification/ws-eventing/
Web Services Notification (WS-Notification)	**Description:** defines a set of specifications that standardize the way Web Services interact using the notification pattern.
	Organization: OASIS
	Web site: http://ifr.sap.com/ws-notification/ws-notification.pdf
Web Services Reliability (WS-Reliability)	**Description:** a generic and open model for ensuring reliable message delivery for Web Services with a chosen level of quality of service (QoS).
	Organization: OASIS
	Web site: http://www.oasis-open.org/committees/wsrm/charter.php

Table 2: List of Web Service Specifications (continue)

Web Services Reliable Messaging (WS-ReliableMessaging)	**Description:** describes a protocol that allows messages to be delivered reliably between distributed applications in the presence of software component, system, or network failures.
	Organizations: IBM, BEA, Microsoft, and TIBCO
	Web site: http://www-128.ibm.com/developerworks/library/specification/ws-rm/
Web Services Resource Framework (WSRF)	**Description:** defines a generic and open framework for modeling and accessing stateful resources using Web Services. Additional related specifications that have been developed that will be considered by OASIS for the WSRF. • **WS-ResourceProperties:** This defines how the data associated with a stateful resource can be queried and changed using Web Services technologies. • **WS-ResourceLifetime:** This defines two ways of destroying a WS-Resource: immediate and scheduled. This allows designers flexibility to design how their Web Services applications can clean up resources no longer needed. • **WS-BaseFaults:** This defines an XML Schema type for a base fault, along with rules for how this fault type is used by Web Services. • **WS-ServiceGroup:** This defines a means by which Web Services and WS-Resources can be aggregated or grouped together for a domain specific purpose.
	Organization: OASIS
	Web site: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf
WS-MetadataExchange	**Description:** defines three request-response message pairs to retrieve three types of metadata: one retrieves the WS-Policy associated with the receiving endpoint or with a given target namespace, another retrieves either the WSDL associated with the receiving endpoint or with a given target namespace, and a third retrieves the XML Schema with a given target namespace. Together these messages allow incremental retrieval of a Web service's metadata.
	Organizations: BEA, IBM, Microsoft, and SAP
	Web site: http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-metadataexchange.pdf
Web Services Dynamic Discovery (WS-Discovery)	**Description:** defines a multicast discovery protocol to locate services. By default, probes are sent to a multicast group, and target services that match return a response directly to the requestor.
	Organizations: BEA, Microsoft, Canon and Intel
	Web site: http://msdn.microsoft.com/library/en-us/dnglobspec/html/WS-Discovery.pdf

Table 2: List of Web Service Specifications (continue)

Web Services Security (WS-Security)	**Description:** describes enhancements to SOAP messaging in order to provide quality of protection through message integrity, and single message authentication.
	Organization: OASIS
	Web site: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss
Web Services Security Policy (WS-SecurityPolicy)	**Description:** defines security assertions detailing a Web service's requirements so that the service requester can meet them.
	Organizations: Microsoft, IBM, VeriSign
	Web site: http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-securitypolicy.pdf
Web Services Trust (WS-Trust)	**Description:** defines how to establish overall trust of the security system by acquiring any needed security tokens from trusted sources.
	Organizations: IBM, BEA, VeriSign
	Web site: http://www-128.ibm.com/developerworks/library/specification/ws-trust/
Web Services Secure Conversation (WS-SecureConversation)	**Description:** defines how to establish and maintain a persistent context for a secure session over which multiple Web service invocations might be sent without requiring expensive authentication each time.
	Organizations: IBM, BEA, VeriSign
	Web site: http://www-128.ibm.com/developerworks/library/specification/ws-secon/
Web Services Federation (WS-Federation)	**Description:** defines how to bridge multiple security domains into a federated session so that a Web service only has to be authenticated once to access Web services deployed in multiple security domains.
	Organizations: IBM, BEA, VeriSign, Microsoft
	Web site: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-federation.asp
WS Choreography Model	**Description:** describes the data and the relationships between them to define a choreography that describes the sequence and conditions.
	Organization: World Wide Web Consortium
	Web site: http://www.w3.org/TR/2004/WD-ws-chor-model-20040324/

Table 2: List of Web Service Specifications (continue)

WS Choreography Description Language	**Description:** an XML-based language that describes peer-to-peer collaborations of Web Services participants by defining their common and complementary observable behavior
	Organization: World Wide Web Consortium
	Web site: http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/
Business Transaction Protocol (BTP)	**Description:** an XML specification of a protocol for managing complex, B2B transactions over the Internet.
	Organization: OASIS
	Web site: http://www.oasis-open.org/committees/business-transactions/documents/primer/Primerhtml/BTP%20Primer%20D1%2020020602.html
WS-Composite Application Framework (WS-CAF)	**Description:** an open, multi-level framework for standard coordination of long-running business processes across multiple, incompatible transaction processing models and architectures. It is divided into three parts: • **WS-Context**: defines a standalone context management system for generic context (that is, for non-transaction protocol contexts such as security, device and network IDs, or database and file IDs). • **WS-CoordinationFramework:** defines a coordinator for the basic context specification and the pluggable transaction protocols in the WS-TransactionManagement specification. • **WS-TransactionManagement:** defines three transaction protocols for the pluggable coordinator: ACID, long-running actions (compensation), and business process management.
	Organization: OASIS
	Web site: http://developers.sun.com/techtopics/webservices/wscaf/primer.pdf
Web Services Atomic Transaction (WS-AtomicTransaction)	**Description:** provides the definition of the atomic transaction coordination type that is to be used with the extensible coordination framework described in the WS-Coordination specification.
	Organizations: IBM, Microsoft, BEA
	Web site: http://msdn.microsoft.com/library/en-us/dnglobspec/html/WS-AtomicTransaction.pdf

Table 2: List of Web Service Specifications (continue)

Web Services Coordination (WS-Coordination)	**Description:** describes an extensible framework for providing protocols that coordinate the actions of distributed applications. The framework enables existing transaction processing, workflow, and other systems for coordination to hide their proprietary protocols and to operate in a heterogeneous environment.
	Organizations: IBM, Microsoft, BEA
	Web site: http://msdn.microsoft.com/library/en-us/dnglobspec/html/WS-Coordination.pdf
Web Services BusinessActivity (WS-BusinesActivity)	**Description:** provides the definition of the business activity coordination type that is to be used with the extensible coordination framework described in the WS-Coordination specification.
	Organizations: IBM, Microsoft, BEA
	Web site: http://msdn.microsoft.com/library/en-us/dnglobspec/html/WS-BusinessActivity.pdf
Business Process Modeling Notation (BPMN)	**Description:** provides a graphical notation for expressing business processes in a Business Process Diagram (BPD).
	Organization: BPMI.org
	Web site: http://www.bpmn.org/Documents/BPMN%20V1-0%20May%203%202004.pdf
Business Process Modeling Language (BPML)	**Description:** a meta-language for the modeling of business processes, and provides an abstracted execution model for collaborative and transactional business processes.
	Organization: BPMI.org
	Web site: http://www.bpmi.org/BPML.htm

Table 2: List of Web Service Specifications

These specifications listed above are based on Web services and aims to extend Web service functionalities in order to provide a satisfactorily execution environment and development of application structure. As well, there are many other specifications exist.

References

[1] James McGovern, Sameer Tyagi, Michael E. Stevens, Sunil Mathew *Java Web Services Architecture*, Morgan Kaufmann Publishers, 2003

[2] L. Bass, P. Clements, R. Kazman *Software Architecture in Practice*, Addision-Wesley, 1997

[3] Raphael Malveau, Thomas J. Mowbray *Software Architecture: Basic Training*, Prentice Hall PTR, Apr. 16, 2004

[4] E. Yourdon, L. Constantine *Structured Design*, Prentice Hall, 1975

[5] G. Booch *Object Oriented Design with Applications*, Benjamin-Cummings Publishing, 1990

[6] C. Szyperski *Component Software: Beyond Object-Oriented Programming*, Addision-Wesley, 1998

[7] Dirk Slama, Karl Banke, Dirk Krafzig *Service Oriented Architecture: Inventory of Distributed Computing Concepts*, Prentice Hall PTR, Dec. 10, 2004

[8] Bieber, Carpenter, Stevens *Jini Technology Architectural Overview*, Sun Microsystems, 2001

[9] World Wide Web Consortium (W3C), *Web Services Architecture*, 11 February 2004, http://www.w3.org/TR/ws-arch/

[10] Ali Arsanjani *How to Identify, Specify and Realize Services for your SOA (Part II and III)*, IBM, 2005, http://www.ebizq.net/topics/dev_tools/features/5632.html

[11] Sun Microsystems, *Jini Network Technology*, http://www.sun.com/software/jini/

[12] Sun Microsystems, *Java 2 Platform Enterprise Edition Specification*, Version 5.0 http://java.sun.com/j2ee/

[13] Jeff A. Estefan, *Exploring Open Software Standards for Enterprise e-business Computing*, August 28, 2000, IBM International Technical Support Organization.

[14] Sun Microsystems, *Java Message Service Specification*, Version 1.1 http://java.sun.com/products/jms/docs.html

[15] Sun Microsystems, *Remote Method Invocation Architecture and Functional Specification*, http://java.sun.com/j2se/1.4.2/docs/guide/rmi/spec/rmiTOC.html

[16] Microsoft Corporation, *Component Object Model (COM)*, http://www.microsoft.com/com/default.mspx

[17] Sara Williams, Charlie Kindel *The Component Object Model: A Technical Overview*, Microsoft Corporation, October 1994.

[18] Microsoft Corporation, *DCOM Technical Overview*, November 1996

[19] Object Management Group, *Common Object Request Broker Architecture: Core Specification*, Version 3.0.3, March 2004 http://www.omg.org/technology/documents/corba_spec_catalog.htm

[20] World Wide Web Consortium (W3C), *Extensible Markup Language (XML) Specification*, Version 1.0, 04 February 2004, http://www.w3.org/TR/REC-xml/

[21] World Wide Web Consortium (W3C), *SOAP Specification*, Version 1.2, 24 June 2003, http://www.w3.org/TR/soap/

[22] World Wide Web Consortium (W3C), *Web Services Description Language Specification*, Version 1.1, 15 March 2001, http://www.w3.org/TR/wsdl

[23] OASIS UDDI Technical Committee, *Universal Description, Discovery and Integration Specification*, Version 3.0.2, 2004, http://uddi.org/pubs/uddi_v3.htm

[24] M. Endrei, J. Ang, A. Arsanjani, S. Chua, P. Comte, P. Krogdahl, M. Luo, T. Newling *Patterns: Service Oriented Architecture and Web Services*, IBM Redbook, April 2004

[25] M. Fowler *UML Distilled: Applying the Standard Object Modeling Language*, Addision-Wesley, 1997

[26] B. Meyer *Object Oriented Software Construction*, Prentice Hall, 1997

[27] Boris Lublinsky *SOA Design: Meeting in the Middle*, August 20, 2004

[28] Greg Lomow, Eric Newcomer *Introduction to SOA with Web Services*, Addison Wesley, 2005

[29] Lawrence Wilkes, Richard Veryard *Service Oriented Architecture: Considerations for Agile Systems*, CBDI Forum, April 2004

[30] Object Management Group, *Unified Modeling Language Specifications* Home page: http://www.uml.org/

[31] Object Management Group, *Unified Modeling Language Specification*, Version 2.0, October 8, 2004

[32] Simon Johnston, *UML 2.0 Profile for Software Services*, IBM, 13 April 2005

[33] Object Management Group, *Model Driven Architecture Specifications*, Home page: http://www.omg.org/mda/specs.htm

[34] The Middleware Company, *SOA Blueprints Specification*, Version 0.5, June 2004 http://www.middlewareresearch.com/soa-blueprints/

[35] SAP AG, *SAP NetWeaver*, http://www.sap.com/solutions/netweaver/index.epx

[36] Franz-Josef Fritz *an Introduction to the Principles of Enterprise Services Architecture (ESA)*, 2004, SAP AG.

[37] Laurie Nolan *Realization of the ESA Vision*, 2004, SAP AG

[38] Kartik Iyengar, *Creating a Comprehensive Collaborative Platform with SAP NetWeaver, Part II*, SAP Developer Network, 11 January 2005

[39] Kaj van de Loo, *Implementing an Enterprise Services Architecture*, 27 July 2004

[40] Apache Software Foundation, *Beehive Project*, Version 1.0 http://incubator.apache.org/beehive/

[41] Sun Microsystems, Java Community Process, JSR 175: *A Metadata Facility for the Java Programming Language*, 30 September 2004 http://www.jcp.org/en/jsr/detail?id=175

[42] Apache Software Foundation, *the Apache Struts Web Application Framework*, http://struts.apache.org/

[43] Sun Microsystems, Java Community Process, JSR 181: *Web Services Metadata for the Java Platform*, 27 Jun, 2005, http://www.jcp.org/en/jsr/detail?id=181

[44] Rogue Wave Software, *The Lightweight Enterprise Integration Framework (LEIF)*, evaluation version 2.1, http://www.roguewave.com/products/leif/index.cfm

[45] Rogue Wave Software, *the Lightweight Enterprise Integration Framework (LEIF)* software documentation

[46] infoAsset AG *The infoAsset Broker Technical White Paper*, 7 April 2001

[47] Holm Wegner, *The infoAsset Broker Documentation*, 2000

[48] Apache Software Foundation, *Tomcat Project*, http://jakarta.apache.org/tomcat/

[49] Lars Diestelhorst, *Wissensmanagement für die Entwicklung von Feststoffprozessen: Assetorientierte Analyse, Design und Implementierung eines Information-Brokers*, Software System Institute, Technical University Hamburg-Harburg, Master Thesis, 24 June 2004

[50] Leif M. Koch, Wissensportalsoftware als Learning Management System, Technical University Hamburg-Harburg, Master Thesis, October 2002

[51] Amazon Web Services Home Page, www.amazon.com/webservices

[52] James M. Snell, *Resource-Oriented vs. Activity-Oriented Web Services*, Oct. 2004

[53] David S. Linthicum *Approaching Application Integration*, Addision Wesley, Feb. 13, 2004

[54] David S. Linthicum *Approaching E-business Integration*, Addision Wesley, Nov. 13, 2000

[55] UN/CEFACT, OASIS, *ebXML Technical Architecture Specification*, v1.0.4, 16 February 2001, http://www.ebxml.org/specdrafts/approved_specs.htm

[56] Abraham Kang, *Enterprise Application Integration Using J2EE*, 9 August 2002

[57] Manoj Seth, *Web Services – A Fit for EAI (Part 1 and 2)*, October 2002 http://www.developer.com/tech/article.php/10923_1489501_1

[58] Martin Keen, Amit Acharya, Susan Bishop, Alan Hopkins, Sven Milinski, Chris Nott, Rick Robinson, Jonathan Adams, Paul Verschueren, *Patterns: Implementing an SOA Using an Enterprise Service Bus*, IBM Red Book, July 2004

[59] Naveen Balani, *Model and Build ESB SOA Frameworks*, 15 March 2005

[60] Yahoo Portal Application: www.yahoo.com

[61] OASIS, *Web Services for Remote Portlets Specification*, v1.0, August 2003 http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrp

[62] Sun Microsystems, Java Community Process, JSR 168, *Portlet Specification*, v1.0, October 2003, http://www.jcp.org/en/jsr/detail?id=168

[63] Bryan Castle, *Introduction to Web Services for Remote Portlets*, 15 April 2005

[64] Thomas Schaeck, *Web Services for Remote Portlets (WSRP) Whitepaper*, 22 September 2002, IBM Corporation

[65] Workflow Management Coalition, *the Workflow Reference Model*, Document Number WFMC-TC00-1003, January-1995, http://www.wfmc.org/

[66] World Wide Web Consortium (W3C), *Web Service Choreography Interface (WSCI) Specification*, Version 1.0, 8 August 2002, http://www.w3.org/TR/wsci/

[67] OASIS, *Business Process Execution Language for Web Services (BPEL4WS) Specification*, Version 1.1, 5 May 2003, http://dev2dev.bea.com/webservices/BPEL4WS.html

[68] A. Mani, A. Nagarajan *Understanding Quality of Service for Web Services*, January 2002, IBM DeveloperWorks

[69] Joshy Joseph, Craig Fellenstein, *Introduction to Grid Computing*, Prentice Hall PTR, Apr 16, 2004

[70] Global Grid Forum (GGF), *the Open Grid Services Architecture*, Informational Document, version 1.0, 29 January 2005, http://www.globus.org/ogsa/