P - 2845

A WEB PORTAL USING SERVICE ORIENTED ARCHITECTURE

A PROJECT REPORT

Submitted by

N.PRAVEEN SUNDRA KUMAR	**71205104033**
M.R.SANTHOSH KUMAR	**71205104041**
S.SILAMBARASAN	**71205104046**

in partition fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY, COIMBATORE

ANNA UNIVERSITY: CHENNAI-600 025

APRIL 2009

BONAFIDE CERTIFICATE

Certified that this project report entitled **"A WEB PORTAL USING SERVICE ORIENTED ARCHITECTURE"** is the bonafide work of N.Praveen Sundra Kumar, M.R. Santhosh Kumar, and S.Silambarasan, who carried out the research under my supervision. Certified also, that to the best of my knowledge the work reported herein does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

[illegible]

SIGNATURE

Dr.S.Thangasamy, Ph.D.,

HEAD OF THE DEPARTMENT

Department of Computer Science &Engineering,

Kumaraguru College Of Technology,

Coimbatore-641006.

V.S. Akshaya

SIGNATURE

Mrs.V.S.Akshaya, M.E

SUPERVISOR

Senior Lecturer

Department of Computer Science &Engineering,

Kumaraguru College Of Technology,

Coimbatore-641006.

The candidate with University Register Nos. 71205104033, 71205104041 and 71205104046 were examined by us in the project viva-voce examination held on 28-04-09

[illegible]

INTERNAL EXAMINER

[illegible]

EXTERNAL EXAMINER

DECLARATION

We hereby declare that the project entitled "**A WEB PORTAL USING SERVICE ORIENTED ARCHITECTURE**" is a record of original work done by us and to the best of our knowledge, a similar work has not been submitted to Anna University or any Institutions, for fulfillment of the requirement of the course study.

The report is submitted in partial fulfillment of the requirement for the award of the Degree of Bachelor of Computer Science and Engineering of Anna University, Chennai.

Place: Coimbatore

Date : 28-04-09

[illegible]
(N.Praveen Sundra Kumar)

[illegible]
(M.R. Santhosh Kumar)

[illegible]
(S.Silambarasan)

ACKNOWLEDGEMENT

We extend our sincere thanks to our vice principal, Prof. R.Annamalai, Kumaraguru College Of Technology, Coimbatore, for being a constant source of inspiration and providing us with the necessary facility to work on this project.

We would like to make a special acknowledgement and thanks to Dr. S. Thangasamy, Ph.D., Dean, Professor and Head of Department of Computer Science &Engineering, for his support and encouragement throughout the project.

We express deep gratitude and gratefulness to our guide Mrs.V.S.Akshaya, M.E Senior Lecturer, Department of Computer Science &Engineering, for her supervision, enduring patience, active involvement and guidance.

We would also like to thank our project coordinator, Mrs. P.Devaki, M.S Assistant Professor, Department of Computer Science &Engineering, for her support during the course of our project.

We are most fortunate to have the guidance of Mr. S. Selva Kumar **M.C.A.,** Software Developer, Satyam Private Ltd. We express our sincere thanks to him.

We would like to convey our honest thanks to all members of staff of the Department for their enthusiasm and wealth of experience from which we have greatly benefited.

DEDICATED

TO MY

BELOVED PARENTS

RESPECTFUL STAFF

&

LOVABLE FRIENDS

TABLE OF CONTENTS

ABSTRACT:

In today's world of shortened product cycles and heightened competition, IT's task is to create a flexible environment to ensure that the enterprise is strategically positioned to foster innovation, to respond the changing needs faster than ever before by reducing time to market for new services.

Basically by the usual construction of multi-tier architectures, ***traditional problems*** and ***component based problems*** are encountered.

SOA is a logical way of designing a software system to provide services to either end-user applications or to other services distributed in a network, via published and discoverable interfaces. A well-constructed, standards based Service Oriented Architecture can empower a business environment with a flexible infrastructure and processing environment. SOA achieves this by provisioning independent, reusable automated business process and systems functions as services and providing a robust and secure foundation for leveraging these services. The visionary promise of Service-Oriented Computing is a world of cooperating services where application components are assembled with little effort into a network of services that can be loosely coupled to create flexible dynamic business processes and agile applications that may span organizations and computing platforms.

Service-oriented architectures (SOA) enable enterprises to quickly respond to changing business requirements. The new aspects here include developing and testing applications composed of operational services, deploying and provisioning distributed service-based applications across organizational boundaries in a secure, reliable, and repeatable manner

The services we provide in this portal includes, an elegant and effective service for the Real Estate, where a you can buy, sell and rent properties, lands, individual villas, flats, shops, complexes and offices, which is totally designed using the Service Oriented Architectural approach.

LIST OF FIGURES

LIST OF ABBERVATIONS:

BPEL-Business Processing Execution Language

BSPs- Bus Service Providers

BSRs-Bus Service Requestors

CBE-Common Base Event

COM-Component Object Model

CTQ- Cost, Time, Quality

EA- Enterprise Architecture

EAI-Enterprise Application Integration

EDI-Electronic Data Interchange

ESB- Enterprise Service Bus

HTTP-Hyper Text Transmission Protocol

JSP-Java Servlet Page

MVC-Model-View-Controller

SDO-Service Data Object

SOA – Service Oriented Architecture

SOAP- Simple Object Access Protocol

SOBAs-Service-Oriented Business Applications

SOC-Service-Oriented Computing

WC3-World Wide Web Consortium

WS-Web Service

WSDL-Web Service Description Language

WSO2-Web Service Oxygen

XML-eXtensible Markup Language

CHAPTER-I

1. INTRODUCTION

Most enterprises have made extensive investments in system resources over the course of many years. Such enterprises have an enormous amount of data stored in legacy enterprise information systems (EIS), so it's not practical to discard existing systems. It's more cost-effective to evolve and enhance EIS. But this can be done using Service Oriented Architecture (SOA) provides a cost-effective solution.

1.1 SOFTWARE ARCHITECTURE:

Software architecture is the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.

Software architecture is the blueprint for a solution that satisfies the requirements of a software application. Software applications are developed to meet the needs of users. For example users of airline reservation application have the need to determine the availability of seats, undertaking bookings and make payments. The needs of the users form the basis for arriving at the software requirements of an application.

Software requirements of an application is usually a document that clearly the problem that is addressed by the application in terms of the specific needs and constraints expressed as a set of verifiable and measurable statements. Software requirement defines the problem that the application addresses in terms of what the user gets from the application and not how the problem is solved. In the example of Airline reservation, a statement that expresses the need of the user to make a booking constitutes a requirement a while any description on either the user interface or the storage of booking data in a database constitutes a solution and therefore does not qualify to be a requirement.

The following two types of requirements have to be taken into account in order completely address the need of the users of a software application

1. **Functional Requirement:**

   Functional requirement define the functionality that the application provides.
2. **Non Functional Requirement:**

Non functional requirements define the attributes that are also supported in providing the functionality to the users.

A set of software requirements can be addressed in a number of ways by defining a variety of subsystems and interfaces that leverage the underlying technology platform in different ways. Software architecture is the blueprint that defines the subsystems and interfaces which when implemented fulfill the software requirements and therefore the needs of the users.

1.1a Need for Software Architecture

Software application vary from having a few modules with simple interfaces to large enterprise applications that have large number of modules interfacing with a number of systems requiring several hundred person years of effort. Software applications also evolve over a period of time on the basis of changing business and user needs.

The need for a clearly defined architecture becomes apparent as the size and complexity of application increases. Every development activity involves trade-offs on cost, time and quality (CTQ); larger the application or its complexity greater the number of trade-offs and their impact on the success of the application. Architecture is key to ensuring that the trade-offs are made taking into account objectives of the stakeholders. The software architecture, thus defined, ensures that the available options are analyzed and decision made to satisfy the goals of the stakeholders.

In order to ensure that software application is built to the required quality, established software engineering practices have to be employed for each stage of the software development lifecycle and their effectiveness measured and controlled so that goals of the application are fully met. The software engineering practices require a well-thought out foundation on which development activities can proceed in a planned manner to balance the CTQ. Software architecture serves to be that foundation on which the application can be built.

A well-architected solution whether in software development or civil construction provides,

1. Satisfies the functional requirements of the users;
2. Meets the non-functional requirements;
3. Allows for construction with available technologies and industry practices;
4. Enables development to be based on established engineering principles to achieve the CTQ objectives.

1.1b Objectives of software architecture

The functional and non-functional requirements of an application determined by the analysis by the needs of the users, characterize the problem for which software architecture is developed. Software architecture represents the solution to the problems stated as a set of requirements

The objectives of software architecture is to,

1. Manage complexity by separation of concerns

2. Resolve concerns related to functional requirements by defining a set of layers, modules and interfaces which working together deliver the functionality required by the end users.
3. Ensure that non functional requirements such as performance and scalability are addressed by defining necessary mechanisms.
4. Exploit technology platform capabilities to bring down the effort required to build the application.
5. Leverage reusable assets and frameworks available for the platform to reduce the cost of development.
6. Lower the impact due to change of business requirements, Operating constraints and technology environments.
7. Apply best practices learnt in developing similar applications
8. Represents the layers, modules or subsystems visually to make it easy for different stake holders to validate and confirm that the architecture so defined meets the end users goals.
9. Provide a foundation for designers to elaborate the architecture and generate design artifacts that can form the basis for development.
10. Enable use of consistent technology during software engineering life cycle among architects, designers and developers.

Software architecture can be represented as views. No single view of architecture can elegantly and efficiently represent the various aspects of the architecture and meet the objectives given above. Software architecture is therefore represented in multiple views.

1.2 TYPES OF IT ARCHITECTURE

In the context of an enterprise, software applications are developed, hosted, maintained and used by different stake holders to meet the objectives discussed above. The key types of software architecture that model and address the concerns of different stake holders are as follows.

1.2a Enterprise Architecture (EA)

Enterprise Architecture is a collection of models that defines the structure requires for an organization to achieve its objectives. The scope of EA is business of the organizations and the IT systems needed to support the business along with policies and principles for its governance. From an IT stand point one of the key aspects of EA is that its focus is on enterprise wide concerns and on how IT systems fit into big picture of the enterprise.

Enterprise architects deal with future state business and IT vision explicitly stated and inferred from interactions with cxo's (short form for CEO,CIO,CTO,CFO) and other stake holders in an organization. Using insights in business trends, technology trends, regulations and complaints requirements, and enterprise architects translates business strategies to IT strategies for the enterprise. Several decisions related to rationalization, migration and retiring of application are made by enterprise architects to ensure that IT applications enable and stay aligned with business processes and goals of the enterprise.

The models that represent EA include,

1. Business model
2. Application model
3. Information model
4. Infrastructure model.

There are several frame works for defining EA of which the following are,

1. Zachman's framework
2. Togaf architecture development method framework
3. Federal enterprise architecture framework
4. Treasury enterprise architecture framework
5. C4ISR/DOD framework

1.2b Business Architecture

Business architecture is the blueprint of the business concerns of IT systems. It involves developing the model based on the following;

1. Business drivers

2. Business rules

3. Business processes

Business architecture is defined by Business ARCHITECTS. In developing the v architecture, Business architects initially focused on the current state Business strategy, Business processes, rules and policies and define "as-is" Business architecture for the IT system "to-be" architecture is then developed keeping in mind the v trends, competitiveness and vision for the future state of business.

The following are some of the key standards/frameworks for developing Business architecture,

1. TOGAF's ADM-phase B.
2. IBM's Business architecture description.
3. Microsoft motion.

1.2c Solution Architecture

Solution architecture represents the solution to the business problem that needs to be implemented by IT systems and is key to aligning business with IT. The business problem is stated as a set of business requirements and the solution architecting process involves mapping of functionality envisaged in IT system to technical elements while ensuring that non-functional attributes are suitably addressed.

The focus of the Solution architect is on a specific business area from a functional perspective for which a Solution needs to be developed and from a technical perspective defining those elements which when working together will deliver the required functionality with the defined quality attributes. the Solution architects ensures that the Solution, which is typically one or more IT systems for a business owner, integrates into the target environment, where EA has been defined for an organization that Solution architecture aligns with it. In doing so, the solution architect evaluates "build" vs "buy" options and enables the decision to be made with by-in from key stakeholders.

Before the advent of the component models, software applications are primarily confined to mainframe to super-mini computer environments and it was a common practice to represent software architecture in three different views:

Processing, data and connection. However, enterprise applications developed with component models required more views to bring out additional detail. Solution architecture is, therefore, represented initially at least the following views to describe what the solution is and how it would be implemented,

1. Conceptual view
2. Logical view
3. Physical view.

When the functional and non functional requirements are defined clearly during the software development lifecycle, more views are needed to specify the architecture, so that design activities can elaborate on the architecture to produce detailed designs.

Solution architecture is critical when developing Solutions that are domain-intensive such as, those developed for vertical industry segments. Understanding the domain and mapping the business processes and workflows that represents the functionality of the system to technology elements and meeting integration requirements are key to getting the Solution architecture right and also to the IT systems developed.

1.2d Technical Architecture

Technical architecture as the name suggests, is a blueprint of the solution described in terms of the technical elements, their structure and relationship with each other and with that of the underlying technology platform. The technical vies of a Solution architecture of IT system corresponds to the technical architecture.

Technical architects focused on specific technologies such as J2EE and .NET. Since technical architecture is based on elements that can be constructed through programming, design patterns have been used in the many cases as in the technical architecture. A design pattern is a solution to a recurring problem in specific design separations.

On account of the wide range of technologies in the market place, technical architecture for a given set of requirements can have a number of variants of the product chosen and reusable components are frameworks employed. Also, as technologies get outdated that support for products withdrawn by their vendors, it becomes imperative to revisit the technical architecture for the same set of functional and non functional requirements.

1.2e Infrastructure Architecture

Infrastructure architecture provides the blueprint for underlying hardware operating systems; network, messaging and security that constitute the application of infrastructure on which solutions that meet business needs are hosted.

Infrastructure architects review the technical architecture and focus on sizing and capacity planning of the application hosting and storage system. Identification and mplementation of systems for monitoring, provisioning and fault tolerance are activities taken ıp an infrastructure architect. Definition of policies for application deployment and security compliance also come under their purview.

1.3 SERVICE-ORIENTED ARCHITECTURE

Service-oriented architecture (SOA) development and deployment generally builds on a service view of the world in which a set of services are assembled and reused to quickly adapt to new business needs. This flexibility is seen by many IT organizations as the core value of SOA and has been driving some deep transformations in the way software is being built. Although SOA technology addresses many of the traditional problems of integrating disparate business processes and applications, deploying service-based applications introduces new aspects of the information technology (IT) environment that must be managed. These new aspects include developing and testing applications composed of operational services, deploying and provisioning distributed service-based applications across organizational boundaries in a secure, reliable, and repeatable manner, and tracking the business impact of services on the business processes that those services support.

SOA is an architectural style for building software applications that use services available in a network such as the web. It promotes loose coupling between software components so that they can be reused. Applications in SOA are built based on services. A service is an implementation of well-defined business functionality, and such services can then be consumed by clients in different applications or business processes.

SOA allows for the reuse of existing assets where new services can be created from an existing IT infrastructure of systems. In other words, it enables businesses to leverage existing investments by allowing them to reuse existing applications, and promises interoperability between heterogeneous applications and technologies.

The fundamental goal of SOA is to facilitate business-level software modularity and allow for rapid reuse of application and data logic, enabling the enterprise to be responsive and agile. In reality, most SOA projects are focused on existing application logic, and therefore fail to consider the key implications of integrating data.

CHAPTER-2

2. LITERATURE REVIEW

While most business leaders agree that data is a critical strategic asset, the decision on how to best manage it eludes most enterprises. What is known is that the amount of enterprise data continues to grow, and more often than not it is maintained in an ever-increasing number of locations. For example, a recent IDC report noted that the average company has 49 applications in 14 different databases and typically has no more than 20&percnt; of its customer data residing in any one location.

Given this, data architects have an urgent and important decision to make. What is the best architectural design to enable an enterprise to leverage its data to enhance relationships, increase efficiencies and lower costs. Many data practitioners are stumped when it comes to choosing a master data strategy; some lean toward a more traditional approach that takes advantage of data warehousing principles, while others insist that infusing service-oriented architecture (SOA) can bring about a better result.

During this evolution, three main application architectures can be identified:

- Compact application architecture
- Component application architecture
- Service-Oriented Architecture (SOA)

We are going to take a brief look at these application architectures and outline their characteristics.

2.1 COMPACT APPLICATION ARCHITECTURE

During application development for the single mainframe, there was no clear separation between application layers and no reusable components were used. All the data access, business logic, and user interface-specific code were contained in a single executable program.

This traditional ***compact architecture*** was used because the mainframe computers had specific ***proprietary*** programming languages and formats for accessing and manipulating the data.

All the data access-specific procedures as well as the business logic and business rules code are written in this programming language. At the surface, a user interface is presented to the user for data visualization and manipulation.

To address the issues with the compact application architecture, the ***component-based application architecture*** **was developed.**

2.2 COMPONENT APPLICATION ARCHITECTURE:

In the component application architecture, the application's functionality is defined using *components*. A component is like a black box, a software unit that encapsulates data and code and provides at the surface a set of well-defined interfaces used by other components. Since a component only needs to support a well-defined set of interfaces, it can change the inner implementation details without affecting other components that use its external interfaces. Components that export the same interfaces can be interchanged, allowing them to be *reused* and *tight coupling* to be eliminated. This makes them *loosely coupled* because they don't need to know internal implementation details of one another.

This separation of application functionality using components allows the distribution of development tasks across several developers and makes the overall application more maintainable and scalable. In the Windows environment, the most used component application architecture is the **Component Object Model (COM)**.

Typically, components are grouped into logical layers. For example, an application uses the data access layer to access the different data sources, the business logic layer to process the data according to the business rules, and the presentation layer also known as the user interface layer to present the data to end users. Using well-defined application layers allows for a *modular design*, component decoupling, and therefore the possibility for component reuse.

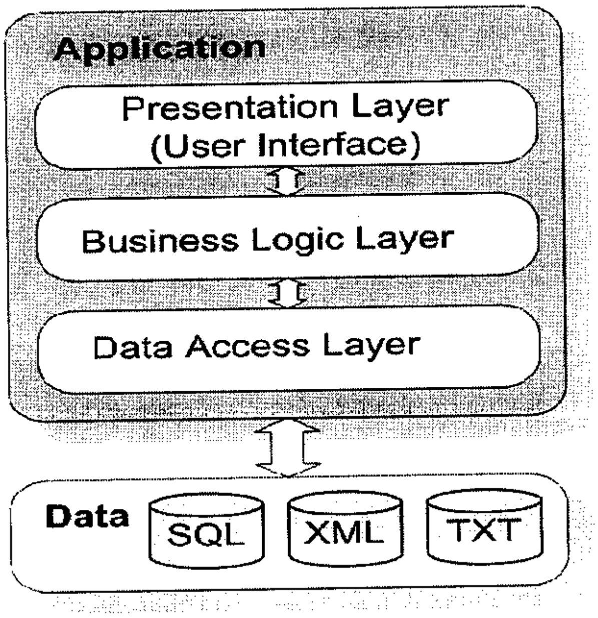


Figure 2.2 Component Architecture

2.2a Data Access Layer

This architecture forms a chain of layers that communicate with one another. The base is the data access layer, which is responsible for querying, retrieving, and updating the data from and to different data sources while providing a uniform data view to the layers above.

2.2b Business Layer

Above the data access layer is the business logic layer. The business logic layer uses the uniform data provided by the data access layer and processes it according to the business rules it contains. The business logic layer doesn't need to know from what source and how the data was obtained. Its purpose is only data manipulation and processing.

2.2c Presentation Layer

At the top of the chain is the presentation layer or the user interface layer. Its purpose is to present the data processed by the business logic layer to end users and to receive input and commands from these end users. The presentation layer will propagate these commands down the chain to the business layer for processing.

2.2d Characteristics

The component application architecture solves many software problems and it has been used extensively in the past. But because software evolves continuously, new requirements introduce new challenges.

Integration between two or more applications running on different platforms would require a middle component-dependent intercommunication layer that is expensive, difficult to build, and reintroduces *tight coupling* between systems, which is what we tried to avoid in the first place. Avoiding building this intercommunication layer would require that the data exchange between these applications be done by a person who will read the necessary data from the source application and write it into the target application.

We need to integrate these systems, and maintain the loose coupling between them. What we need to do, is make these components understand each other, making them to speak the same language. This is where the concept of services and **Service-Oriented Architecture (SOA)** comes into play.

- The architecture of distributed applications, including Windows DNA
- Making up a Web application
- components
- A quick introduction to ActiveX and COM

- The three types of components and why we should use them
- Building Web applications out of components
- Designing a component-based Web application
- **Tiered Applications**

Tiered applications can be characterized by the number of layers that information will pass through on its journey from the data tier (where it is stored in a database typically) to the presentation tier (where it is displayed to the client). Each layer generally runs on a different system or in a different process space on the same system, than the other layers.

- **Two-Tier Applications (Client/Server)**

Let's look briefly at the 2-tier client/server architecture. Typically, we have a user's PC for the client (front-end) and a network server that contains the database (back-end). Logic is divided between these two physical locations. Usually the client contains most of the business logic, although with the advent of stored procedures, SQL language routines allow business logic to be stored and executed on the database server:

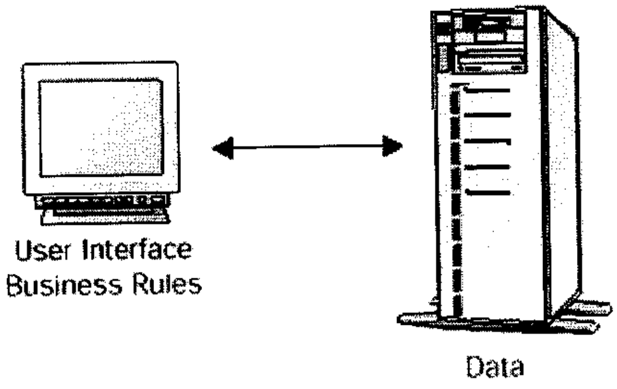


Figure 2.3 2-tire Application

The 2-tier scenario works very well when you have a small business that only uses, or needs, a single data source. However, the goal of most businesses is to grow. As the business grows, so will the database and its requirements. Unfortunately, the 2-tier approach does not scale very well. If the business rules change then the application needs to be rebuilt and redeployed. In addition, there are factors such as the maximum number of simultaneous database connections that prevent this architecture from ever being of much value in a distributed setting with more than a few users.

- **Three-Tier and N-Tier Applications**

Due to the limitations of the 2-tier client-server architecture, distributed applications are often divided up into three or more tiers. Components in each of these perform a specific type of processing – there's a **User Services**(Presentation) tier, a **Business Services** tier, and a **Data Services** tier in a 3-tier application.

The main distinction between this 3-tier architecture and your traditional 2-tier client-server architecture is that, with a 3-tier architecture, the business logic is separated from the user interface and the data source.

Breaking up applications into these separate tiers or sections can reduce the complexity of the overall application, and results in applications that can meet the growing needs of today's businesses. **n-tier** applications are just 3-tier applications that might further sub-divide the standard User Services, Business Services, or Data Services tiers. In any case, an application with more than two tiers can be considered an n-tier application.

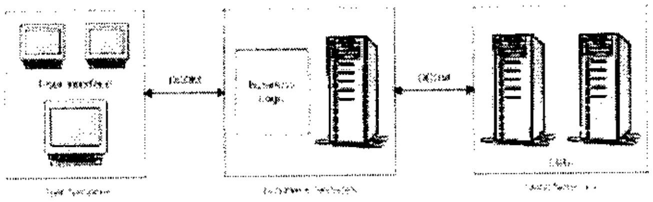


Figure 2.4 N-tire Application

In this type of application, the client should never access the data storage system directly. If it did, it would be circumventing the business rules of the application and would thus be unable to ensure that the data on display to the client was correct.

The separation of the various aspects of an application into n tiers allows for any part of that application to be modified without having to change the other parts, allowing developers to specialize in designing and developing a specific tier or tiers. Similarly, developers can also take advantage of the development tools that specialize in the development of that tier, rather than making use of general purpose tools, which are sufficient to build an entire application but are lacking in terms of powerful features.

Before we look at the three basic types of services in more detail, there is one more variation on the n-tier theme we need to look at.

- **Component Services**

Component-based applications have been the standard method for creating applications for the better part of the last decade. From their earliest days as DDE, then OLE, OLE2, and finally today's ActiveX, component applications for the Windows platform have become widespread. It's not surprising really considering how much easier development is using components, the whole process now has a new model which consists of wiring pre-built components together to create an application. Because every component has a number of publicly accessible functions that other components can discover and use. The end effect is that components are simple to reuse and allow different applications to work with each other.

To help component-based applications grow into enterprise-level applications, Microsoft added the Microsoft Transaction Server to provide services for the development, deployment, and management of component-based distributed applications. By encapsulating this plumbing into a system-level service, it freed developers from having to explicitly add this support to their applications.

With the release of Windows 2000, we saw the introduction of COM+. COM+ unifies the programming models of COM and MTS and makes it even easier to develop distributed applications by eliminating the need to handle both services individually. This allows the developer to build applications faster, easier, and ultimately cheaper by reducing the amount of code required to make use of these component services.

- **Messaging Services**

Since not all parts of a Windows DNA application need to be located on the same computer, or even in the same physical location, it is critical that developers be able to communicate with these disparate systems. And, in the networked applications paradigm, there also needs to be a way for disconnected clients to communicate with the application.

- **Web Services**

The application services of a Windows DNA application need a way to interact with the presentation layer on the client. In this case, that way is the Hypertext Transport Protocol, a mechanism which Microsoft's Web Server, IIS, already provides as a service for anything that cares to use it. In fact, IIS not only supports the delivery of static Web pages to a client, it can also be used to dynamically create client presentations through its integration with Active Server Pages. But

- **Data Services**

As noted earlier, Universal Data Access is Microsoft's strategy for providing access to information across the enterprise. It provides high-performance access to a variety of information sources, including relational and non-relational data, and an easy-to-use programming interface that is both tool and language independent. UDA does not require the expensive and time-consuming movement of data into a single data store, nor does it require commitment to a single vendor's products. It is based on open industry specifications with broad industry support, and works with all major established database platforms. The two primary components that provide data services are ActiveX Data Objects (ADO) and OLE DB, both of which we've already covered in this book.

- **The Structure of the Web**

The structure of the Web is a lattice of HTML-formatted pages, distributed from computers known as Web servers to client computers, and viewed using tools called Web browsers. At the most basic level, there is no difference between a server that has thousands of pages and a server that just has one or two. It is by linking these pages together in some form that a set of pages becomes a Web site, and by adding some additional logic, a Web site becomes a Web application.

- **Web Page**

  The basic unit of a Web interaction is the Web page itself. A Web page is a text file that is marked up using HTML. It is sent to a browser from a Web server based on a request from that browser. The browser parses the information in the HTML file, and the resulting user interface is displayed within the browser itself. The role of the Web server is to listen for a request from the client, parse the request to determine the page that the client requested, retrieve that file from the server's storage area, then transmit that file to the client. At this point, the server forgets everything about sending that file to the client, except for maybe placing an entry into a log file. It is this "connection-less" nature that gives a Web server its scalability, but in turn makes it a challenge to create meaningful applications without some additional support.

- **Web Site**

  A Web site consists of a set of related Web pages grouped together by some means. Generally, a Web site is all of the pages that exist on a server, or within a folder on that server. The correlation between the pages on a site is maintained by the links within each page on the site. The links on each page in the site will take users to other pages within the site. In this way, the pages that make up the site internally maintain the hierarchy of the site. These pages are still subject to the restriction of the Web server architecture, in that the server does not maintain information about the series of requests from a particular client. So while this related set of Web pages that make up a Web site are beginning to look more like an application, there are still some missing components.

- **Web Applications**

  Windows DNA applications are applications in the traditional sense, in that they provide a service to the user of the application. They are different in the way that they are created as well as in the components that make them up. A traditional application requires a special set of files during development, but distributes different outputs. For example, a Visual Basic application has a .vbp project file, multiple .frm, .cls, and .bas files, as well as a set of OCX components that make up the application project. Prior to the application being distributed, these files are compiled into a set of executable files for distribution and execution. The resulting executable does not require the presence of the source code files that were used to develop it.

- **Web Application Design**

  In building a Web application, there are a number of new aspects of application design and development that the developer must take into consideration. While the flexibility of Windows DNA allows for a wide variety of clients accessing a wide variety of services, we will be focusing on a Web application. In a Web application, we will look at using a browser as the primary user interface. The information in our Web application will flow from server to client using the HTTP protocol.

- **The Browser as the User Interface**

  For a Web application, the user interface is presented within a Web browser. This means that the client presentation can either be Browser-Enhanced or Browser-Reliant. The type that you choose for your application should be based on the installed browser base of your target audience. If you can guarantee that most of your users will be using a particular browser level, then you should consider leveraging the features of that browser level. If there are a wide variety of browsers in use, then you may only be able to support a Browser-Reliant client presentation type. One of the advantages of using Active Server Pages to deliver the application is that it has the capability to determine with what browser the current user is accessing the application.

- **HTTP as the Transport**

The communication layer between the client and the server is critical to the design and implementation of the application. The HyperText Transport Protocol (HTTP) defines how the request made by the client is received and handled by the server, and then how the information is sent back to the client. Depending on the type of client presentation being supported, there can be different types of information that flow using this protocol. For the basic support of a Browser-Reliant client, the information that flows over HTTP is limited to the HTML that makes up the page, graphical images to enrich the interface presentation, information-bearing cookies, and possibly some client-side scripting to provide interactivity on the client. With an Enhanced client, special features such as Remote Data Services or COM over HTTP may also be communicated over the transport protocol. But support of these is reliant on the capabilities of the browser at the client. In either case, the HTTP protocol does not understand the concept of a persistent connection between client and server.

- **Component Services**

With the release of COM+, Microsoft Transaction Server has ceased to exist as a separate entity on Windows 2000. Its functionality is now a basic part of the operating system under the moniker "Microsoft Component Services", which should indicate that there's more under this roof than just the management of transactional services and components' lifetimes.

Along with a new name, the Component Services also offer some new functionality in the way that components are handled over MTS. As the operating system service and not the developer handles more and more of these features, our lives should become much simpler in the pursuit of developing robust and scalable components.

- **Component Load Balancing (CLB)**

CLB allows multiple application servers to provide the same COM+ object for use in an application. When that object is needed, the creation request is sent first to the CLB Server which then redirects the request to an appropriate application server, based on certain criteria (like how busy and how far away from the machine running the application it is). The client application then interacts with this server for the lifetime of the component. Load balancing would be implemented at the COM class level but at the time of writing it was unclear as to whether CLB would be included in the initial release of Windows 2000.

- **Queued Components**

It combines the features of COM and MSMQ to provide a way to invoke and execute components asynchronously. Processing can occur without regard to the availability or accessibility of either the sender or receiver. When a client calls a queued component, the call is made to the Queued Components recorder, which packages it as part of a message to the server and puts it in a queue. The Queued Components listener retrieves the message from the queue and passes it to the Queued Components player. The player invokes the server component and makes the same method call.

- **In-Memory Database Support**

The In-Memory Database (IMDB) is a transient, transactional database-style cache that resides in RAM memory and provides extremely fast access to data on the machine on which it resides. IMDB can be used by COM+ applications that need high-speed database lookup capabilities or applications that need to manage transient state. Data inside an IMDB can be accessed using ADO or OLE DB interfaces from a COM+ object. As with load-balanced components, the IMDB technology has not found its way into the first release of Windows 2000.

- **Object Pooling**

Object pooling is an automatic service provided by COM+ that enables you to have instances of a component kept active in a pool, ready to be used by any client that requests the component. Once the application is running, COM+ manages the pool for you, handling the details of object activation and reuse according to the criteria you have specified. In order for components to be pooled, they need to be stateless, have no thread affinity, and be aggregatable.

- **Transaction Manager**

Microsoft Component Services can function as a transaction manager for components. In managing a transaction for an application, Component Services will examine the components participating in the transaction to see what their transactional requirements are. Some components are quite happy to ignore whatever transaction is going on, while others want to or need to participate in a transaction. When a component is developed and deployed, the developer will set this transaction parameter for the component. Component Services will use that information to determine how the component should participate in the transaction, if at all.

- **Component Manager**

In addition to managing components in their interactions with transactions, Component Services can also manage the components themselves. While this functionality was part of MTS, it was generally not what came to mind when looking at the features of MTS. Component Services uses the context of a COM+ component to help manage it during its lifetime.

- **Security Manager**

Component Services provide a number of security features for your COM+ components as well. There are automatic security features, which are available without adding one line of code and are configurable administratively. Other security features can be integrated directly into the development of the component. Role-based security, which can be implemented programmatically or administratively, is the central feature of COM+ security. It allows for security down to the method level of a particular component, allowing all users to access a component, but restricting certain methods of that component to certain users.

- **Data-Centric Components**

The bottom tier of our three-tier architecture is the data access layer. This layer is responsible for integrating with the data sources that our application needs to be able to function. These data sources could be SQL Server or Access databases, Exchange message stores, MSMQ message queues, or UNIX legacy applications. They could exist on the server itself, on some other server on the LAN, or somewhere across the Internet. The data component is not only responsible for encapsulating the access to the data, but also for making the location of that data transparent to the application as well. All that the application needs to do is instantiate and use the component – the component itself figures out the rest.

There are a number of reasons why data-centric components are necessary portions of a three-tier, and therefore a Windows DNA application. Obviously, without a data layer, we would only be left with a two-tier application. But seriously, there are many reasons why encapsulating data access within a component leads to a more robust application.

- **It shields the developer from the inner structure of the database**

This encapsulation is a primary tenet of object-oriented design. If the internal workings of a component are not exposed to the developer using the component, then these inner workings can be changed, updated, enhanced, or replaced depending on the changes to the physical data store below them. It also means developers cannot circumvent security or procedures by changing rows in the database in an incorrect order, etc.

- **It provides consistent data access to different data sources**

By encapsulating the access to disparate data sources in a common interface, developers can use similar methods to access data regardless of where the data actually resides. This means that access to data stored in a SQL Server database can be accomplished using the same methods as data stored in a flat file on a UNIX system.

- **It makes the location of the data transparent**

With the method to physically access the database encapsulated within the component, the location of the data does not matter to the user of the component. They merely access the component, and the component makes the necessary connections to the data source to retrieve the data.

- **Business Components**

The middle of our three tiers is the business component layer – sometimes referred to as the business logic layer or the application layer. No matter what it's called, its job is to provide the functionality for the application. This could mean managing a shopping cart for an e-commerce application, validating the benefits selection of an employee on the HR section of the intranet, or calculating the best route between two locations in a mapping application.

A business component is designed to hide the complicated interactions that a set of business rules need in order to process and also to shield the user interface designer from having to know anything about the underlying data. They simply interact with the methods of the business components to present information entered by the user and interpret the results of the component's processing.

- **Component Application Design**

In building a component-based application, we need to look first at how to design an application that will use components. This will include both a process for breaking the application down into its components as well as how to design the actual components themselves.

Once we're comfortable with that, we must carry on to look at several issues related to building component-based applications for the Web. Specifically, we must address the design of our components and the interfaces we choose to expose in them, how we will tie out components together and the tools we'll use to implement all of the above.

- **Moving to Components**

As we begin to move our application from a traditional monolithic or client/server application, we need to first look at how to break the application functionality into components. This is known as **decomposition** and can be done in a number of different ways. There are many books available on various object design methodologies, so we won't go into a detailed discussion about them here. The key thing to look for in selecting, or creating, a methodology is that it fits in with your manner of doing business, and also does not radically alter your existing development processes, unless you are not happy with the way things are being done now.

There are many advocates for each methodology, but it is up to you as an application designer and developer to select the one that best meets your needs. And if you can't find one that you like, then you should feel free to take parts of existing ones and create your own. As long as you can deliver a design that not only allows you to create an effective application, but also provides a roadmap for others to understand your application, then you have an effective design methodology for yourself.

As we begin to decompose our application into components, the first step is to partition the functionality of the application into the three tiers that make up a component-based application. A good way to begin is to look at each part of the application, and determine which tier it belongs in. If you're finding it difficult to select a particular tier for a piece of the application to fit into, chances are you're not splitting it up into small enough pieces. At this point, it is time to decompose the element you are currently looking at into multiple parts, with the goal that each part will fit nicely within the presentation tier, business logic tier, or data tier.

As these components begin to fall into our design, we still need to maintain some semblance of an application in our minds.

If we don't remember to do this at this time, then when it comes time to wire the components together at the end, the chances are we will have strayed from the application design as a whole. But by keeping in mind what the final goal of the design is, then the components we end up with will readily tie together into an effective application.

Application Design

The application design is important in building a component application. Without a specified set of application requirements, it will be very difficult to create the components to support the application. In designing the overall application, there are a number of methodologies that can be used. For example, in the Use Cases approach, the application design is created by defining how the application will be used in specific instances.

Whichever method is used when we are designing a Web-based application, there are a different set of challenges that must be overcome that are not present in a traditional application. First and foremost, we as application designers have limited control over the tool that a person is using to access the application.

2.3 PROBLEMS IN EXISTING ARCHITECTURES:

The application's functions cannot be re-used. For example, the business functions are written for this particular application/platform only.

- It is difficult to debug the program as it grows, and maintain it as it is deployed. A change to one part of the code could adversely affect other code.
- Security is another problem because the user interface cannot be isolated from the rest of the program.
- Traditional application architecture makes it difficult to integrate applications that reside in different platforms.
- Scalability is all but impossible because it is difficult to spread any part of the application.
- A tightly coupled solution, in which two applications that are being integrated are aware of each others' implementation details, requires custom code that or a human, who reads the output from one application and keys it into another application.
- Having a component-based application design allows the components to be re-used on the same platform and usually the same programming language.
- **Works:**
  - We need to share the business functions throughout heterogeneous platforms.
  - We need to share information across the firewall.
  - We want to share information with our external partners.
- **Required things:**
  - Component-based application needs to speak the same language.
  - Instead of thinking in terms of processes, components, and data, we need to think in terms of services.

CHAPTER 3

3. SERVICE ORIENTED ARCHITECTURE

The use of heterogeneous technologies and applications in corporations is a reality. At a time when resources are scarce, IT shops cannot just throw away their existing applications; rather, they must leverage their existing investments. Service-oriented architecture (SOA) is popular because it lets you reuse applications and it promises interoperability between heterogeneous applications and technologies.

Introduction

SOA is an enabling architecture based on reusable building blocks called business services. Unlike previous architectures, SOA focuses on business processes, rather than technical components. This means that, using SOA and its **enabling technologies**, software developers can deliver business functionality as a set of services that can be deployed and combined to address new business needs at minimum cost and with minimum delay. C

Conceptually, SOA consists of Business Applications, Business Services, IT Services and Data Services. Using an SOA approach, most of the common business logic resides in Business Services while the business logic specific to the business process at hand is in the Business Application. Ideally, most of the Business Application's work is in co-ordinating Business Services to achieve the overall business process. SOA allows developers to construct this Service-oriented Business Applications (SOBAs) by co-ordinating Business Services and IT and Data Services to create true end-to-end business processes. SOBAs comprise co-operating services that may span business boundaries.

In other words, SOA enables software applications to be built as collections of collaborating services that interact without regard to each other's platform, data structures, or internal algorithms. This has recently become possible because the technology required to support such interoperability has 'come of age' and is no longer considered 'bleeding edge'. The enabling technology, which includes Web services, manages all routing issues, differences in technology platforms and differences in message formats. The result is that business systems and underlying services need only minimal knowledge of each other.

The concept of a service is nothing new, but the notion of an SOA has evolved over the past couple of years. It's an architectural style of building software applications that promotes loose coupling between components so that you can reuse them. Thus, it's a new way of building applications with the following characteristics:

- Services are software components that have published contracts/interfaces; these contracts are platform-, language-, and operating-system-independent. XML and the Simple Object Access Protocol (SOAP) are the enabling technologies for SOA, since they're platform-independent standards.
- Consumers can dynamically discover services.
- Services are interoperable.

Figure 3.1 gives a SOA interaction diagram of service-oriented architecture.

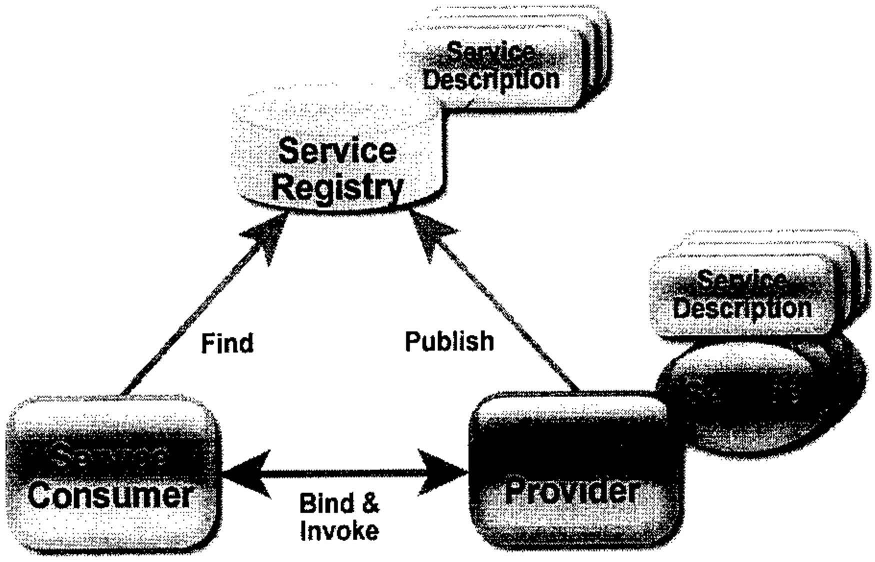


Figure 3.1 interaction pattern

The basic building block of SOA is the *service*. A service is a self-contained software module that performs a predetermined task: "verify a customer's credit history," for example. Services are software components that don't require developers to use a specific underlying technology. As Java developers, we tend to focus on reusing code; thus, we tend to tightly integrate the logic of objects or components within an application. However, SOA promotes application assembly because services can be reused by numerous consumers.

The other key advantage of SOA is that it lets you automate business-process management. Business processes may consume and orchestrate these services to achieve the desired functionality. Thus, new business processes can be constructed by using existing services. For example, a customer order that has been submitted to shipping can be represented by a business process that can asynchronously interact with the requisite services.

Today's IT organizations invariably employ disparate systems and technologies. Most analysts predict that J2EE and .NET will continue to coexist in most organizations and the trend of having heterogeneous technologies in IT shops will continue. Moreover, creating applications that leverage these different technologies has historically been a daunting task. SOA provides a clear solution to these application integration issues by allowing systems to expose their functionality via standardized, interoperable interfaces.

Using SOA offers several key advantages,

- Adapt applications to changing technologies.
- Easily integrate applications with other systems.
- Leverage existing investments in legacy applications.
- Quickly and easily create a business process from existing services.

3.1 BASICS OF SOA

3.1a Evolution of SOA

SOA is the next logical step in the evolution of programming paradigms. Figure 3.2 depicts the evolution of software programming approaches. Software applications were developed on the basis of structured programming approach before the arrival of object-oriented technology.

The development of languages such as Small talk and Microsoft visual basic to support rich user interface resulted in client-server applications. The limitations of client-server applications resulted in enhancements to languages and tools to support N-tier applications. The innovations in managing the lifecycle of objects resulted in development of component models for development of enterprise solutions using languages such as java and C# that could be deployed in application servers such as IBM websphere application server and Microsoft windows platform.

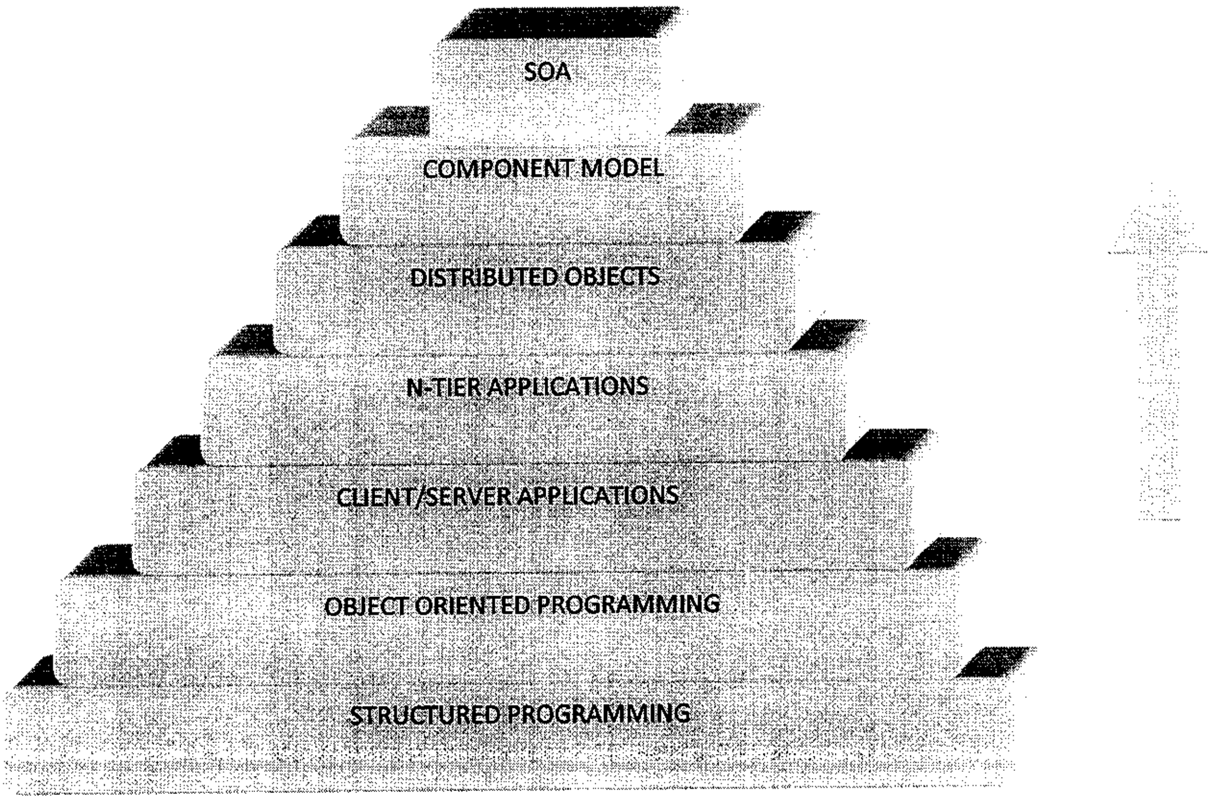


Figure 3.2 Evolution of SOA

3.1 b Dimension of SOA

The dimensions of SOA that enable business transformation in enterprises are :

1. Reuse;
2. Integration;
3. Agility;

Business transformation in enterprises is possible when business processes are implemented on the basis of reusable elements defined from business perspective that can be integrated using an agile IT infrastructure. Organizations are then in a position to make strategic and tactical enhancements to their business processes on the basis of changes in the market place. The use of agile infrastructure enables changes to business processes with relative ease and minimal effort.

Changes to business functionality of any of the business applications do not impact the rest of the application infrastructure of the enterprise. Likewise, changes to the orchestration element of the business process services do not also impact the business applications in well architected SOA solution for an enterprise. Thus the complexity and agility of the business processes can be more easily managed than if they had been "hard-wired" using traditional programming languages.

Integration of service-providers and service-consumers in SOA is addressed efficiently through the Enterpriser service bus (ESB) pattern. It has several advantages over the alternative Hub-and-Spoke architecture used for application integration. In the Hub-and-Spoke architecture too, a centralized "hub" acts a message broker that accepts requests from multiple applications that are connected to it via connectors as "spokes". The hub has capabilities for message transformation, validation, routing and asynchronous message delivery. However, the hub constitutes a single point of failure. Also, most of the vendors provide hub and spoke architecture through extensive proprietary products. Hence, the bus architecture of ESB, with distributed services of message routing, transformation and event handling, provides the capabilities needed for loose coupling and service-level integration between providers and consumers of services more efficiently.

SOA, therefore, has reusability, agility and integration as three dimensions that are critical to the business transformation in organizations. The use of SOA techniques can also bring about their possibilities. As organizations go through mergers and acquisitions and also make collaborative arrangements with other enterprises for competitive edge, restructuring of the business takes place that would result in business process supported by existing business applications to be moved to different/newer applications within the same organization or to those of the collaborating partners outside the organizational boundaries. SOA techniques allow for the degree of agility such business transformations demand by maintaining focus on the business processes and their orchestration rather than be limited by the technology used for implementing the applications.

3.1c Key Components of SOA

The key components of SOA would be those that support the three dimensions discussed in previous section.

Service is the basic component of service orientation that provides reuse. A service exposed by a service provider is coarse-grained in the business functionality it provides and is defined by a service contract and a data contract. Service consumers invoke an operation published by a service as a part of the service contract and provide data as per the data contract.

Services can be implemented using a number of technologies such as web services. Web services are described using WSDL and are invoked using XML messages embedded in a SOAP envelope over a standard protocol such as the HTTP.

While web services (SOAP over HTTP) is the most popular technology to implement a service, it is important to note that the use of web services does not amount to SOA. The following four tenets of service orientation need to be met:

1. Service boundaries are explicit and services interact through explicit message passing over well defined boundaries.
2. Services are autonomous in terms of data isolation and loose coupling.
3. All interactions are based on service contract, data contract ant associated policies.
4. Service compatibility is based on policy expressions (such as those of WS-policy) when service contract cannot completely specify all aspects of service interaction.

3.2 ENTERPRISE SERVICE BUS (ESB)

An Enterprise Service Bus is an architectural pattern that acts as the mediator/broker between service provider and the consumer. It forms the communication infrastructure for services to interact with one another across locations, supporting different transports, and across organizational boundaries. It provides integration capability through content routing, transformation and delivery between service providers and the consumers and has several advantages over alternative integration approaches such as Hub and Spoke.

ESB is an architectural pattern which by definition does not impose a specific implementation approach. The current scenario is that major vendors have tools/products that support the ESB concept and through ESBs from different vendors support the core requirements of content routing, transformation and delivery, they are significantly different from one another in the implementation aspects.

The following gives the summary of the key concepts of the ESB and defines the Integration model for it, including key user roles. These roles are fulfilled using meta-data that describes the service endpoints, such as the service interface and policy requirements and capabilities. The ESB manages this meta-data through a registry, which supports configuration, connection, matchmaking, and discovery of service endpoints. Some typical mediation patterns that are used to satisfy endpoint policies are explored, and usage patterns are described in which the ESB is used to implement real SOAs. The essential characteristics of an Enterprise Service Bus (ESB) are their: the meta-data that describes service requestors and providers, mediations and their operations on the information that flows between requestors and providers, and the discovery, routing, and matchmaking that realize a dynamic and autonomic SOA. In particular, ESB provides the tools and runtime infrastructure to realize the promise of SOA formulated in the iconic ''publish-find-bind'' triangle (see Figure 3.3) that was popular in the early days of the SOA revival caused by Web Services

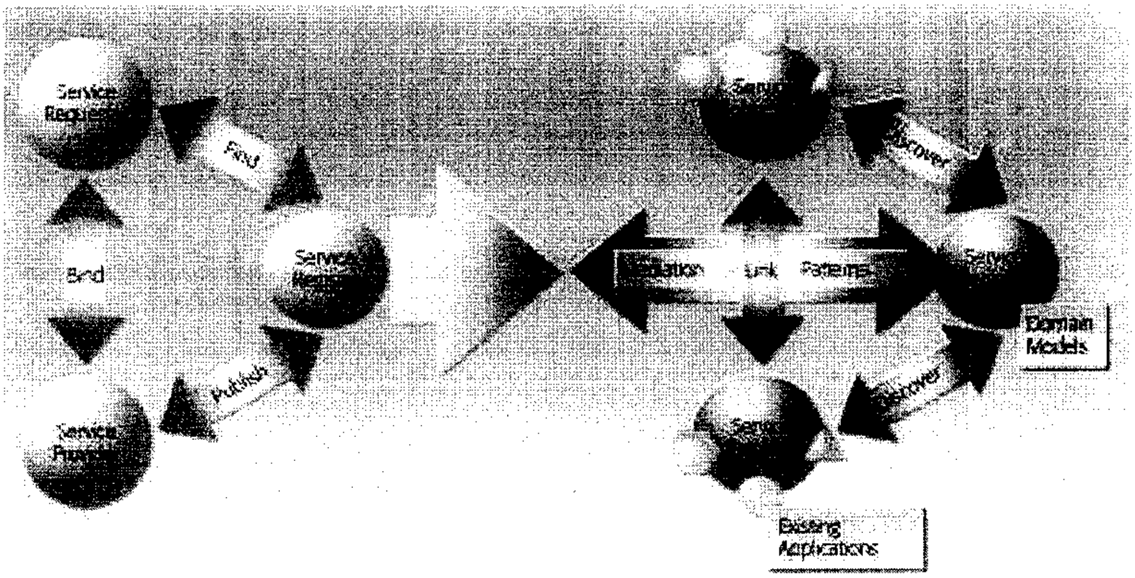


Figure 3.3 ESB underpinnings for SOA

As illustrated in Figure 3.3, the ESB manages and exploits meta-data describing interaction endpoints as well as the domain models used to describe the capabilities of those endpoints; it supports configuration of links that bridge between capabilities demanded by service requestors and those offered by service providers, dynamically matching requestors with providers and in the process establishing and enacting contracts between those interaction endpoints.

3.2a ESB in a Nutshell

The ESB enables an SOA by providing the connectivity layer between services. The definition of a service is wide; it is not restricted by a protocol, such as SOAP (Simple Object Access Protocol) or HTTP (Hypertext Transfer Protocol), which connects a service requestor to a service provider; nor does it require that the service be described by a specific standard such as WSDL (Web Services Description Language), though all of these standards are major contributors to the capabilities and progress of the ESB/SOA evolution. A service is a software component that is described by meta-data, which can be understood by a program. The metadata is published to enable reuse of the service by components that may be remote from it and that need no knowledge of the service implementation beyond its published meta-data. Of course, a well designed software program may use meta-data to define interfaces between components and may reuse components within the program. The distinguishing feature of a service is that the meta-data descriptions are published to enable reuse of the service in loosely coupled systems, frequently interconnected across networks.

The ESB populates the registry with meta-data about services in three different ways. When services are deployed to the runtime environment,

- They can be simultaneously and dynamically added to the ESB
- Meta-data associated with components already deployed can be explicitly added to the ESB or
- The ESB can discover services and service interactions that are already deployed and incorporate meta-data describing them in the registry.

Note that the ESB is the infrastructure for interconnecting services, but the term ESB does not include the business logic of the service providers themselves or the requestor applications, nor does it include the containers that host the services. Hosting containers and free-standing applications are enabled for interaction with ESBs with varying levels of integration, depending on the range of protocols and interoperability standards supported. After the ESB has delivered its payload to a container, its responsibilities are fulfilled. Within the container, the service invocation may be redirected among the machines in a cluster, or it may be responded to from a local cache.

The service requestor, both in its application logic and in its deployment, does not need to have any awareness of the physical realization of the service provider. The requestor does not need to be concerned about the programming language, runtime environment, hardware platform, network address, or current availability of the service provider's implementation. In the ESB, not even a common communication protocol need be shared. The requestor connects to the bus, which takes responsibility for delivering its requests to a service provider, offering the required function and quality of service. Not surprisingly, the infrastructure of the bus is itself virtualized, allowing it to grow or shrink as required by the network and workload which it is supporting. The flexibility that comes from an SOA, and the virtualization it implies, is fully realized by the dynamic nature of the ESB. All the meta-data, conditions, and constraints used to enable a connection from a requestor to a provider can be discovered, used, and modified at runtime..

A service requestor might select a reduced level of assured delivery and see an improved level of performance as the ESB determines that it can use a different delivery protocol. This flexibility is available as a direct consequence of the role of the ESB registry. Because all relevant meta-data for the service provider and service requestors has been placed in the ESB registry, it can be subsequently discovered and used to make dynamic changes. To achieve much of this flexibility, the ESB accepts requests as messages, then operates on them, or "mediates" them, as they flow through the bus.

In the case of message formats, this is usually achieved through a schema definition. For other service properties, policy statements, which may describe the encryption algorithms to be used or the requirements for auditing, can be associated with the meta-data of the service provider and requestor. The ESB consults this meta-data at runtime and can reconfigure the mediations between requestor and provider to match the requirements. By annotating a policy for the service providers in the ESB registry, the system administrator can, for example, ensure that the services meet the company's new privacy guidelines. Thus the ESB implements an autonomic SOA, reacting to changes in the services it connects. One of the major uses of mediations is in systems management. Mediations can be deployed in the ESB environment to enable request and response messages to be monitored as they flow through the system, enabling service-level management or problem determination.

The ESB also provides tools to configure the interactions between services—to display the services available in the ESB, to interconnect them, to add policy requirements to a service or group of services, to identify mismatches in the endpoints, and to associate mediations to correct these, either explicitly or through automatic reconciliation of their policy declarations. Much of the preceding discussion uses the terms service requestor and service provider, as is appropriate for the ESB. Service requestors and service providers are equal partners in the interaction, with the requestor simply being the endpoint that initiated the interaction. The interaction may continue with either endpoint sending or receiving messages.

The ESB supports many different types of program interaction: one-way messages as well as requests and responses, asynchronous as well as synchronous invocation, the publish/subscribe model, where multiple responses may be generated for one subscribe request, and complex event processing, where a series of events may be observed or consumed to produce one consequential event. The ESB is also, in principle, transport and protocol ''agnostic,'' with the capability to transform messages to match the requestor's preferred formats to those of the provider. In practice, most ESBs support SOAP/HTTP, which reinforces its role as an interoperability standard. They also support a range of other transports and protocols, some for use by service requestors and providers connected by the ESB, and some for internal communication within the ESB.

3.2b ESB Integration Model

Having established the basic concepts and features supported by the ESB, we focus on ESB-based SOA solutions. The ESB integration model captures those aspects of the overall solution that are relevant for the ESB tools and infrastructure to facilitate interactions between ESB-managed service endpoints. It enables the various user roles involved in creating and managing those solutions to express the ESB relevant information that they contribute or monitor, and the ESB runtimes to manage interactions accordingly. In essence, the model contains metadata describing service endpoint requirements, capabilities, and relationships, including information describing the specific details of interaction contracts. Not all user roles use the model directly. A business analyst, for example, might define a set of key performance indicators (KPIs) that need to be translated into events produced by the underlying implementation artifacts and potentially into parameters of a service interaction contract. Users in the architecture and design space, such as a solution architect, might define service capabilities and requirements in more abstract terms than those required by the ESB integration model. Other roles, such as the application developer, create and use service capabilities and requirements for the integration model. The user roles that are the most interesting from the ESB perspective are those of the integrator and the solution administrator. We do, however, hint at relationships to products and standards where appropriate. Integration specialists assemble business solutions from a set of service components. They do not have to understand the implementation details of those components (process coordination, existing applications, interactive tasks, etc.); all they need to understand is the capabilities offered by components and the requirements of the components they use, with respect to other components.

The ESB service registry provides the required information about those components and, together with the ESB runtime, enables integration specialists to perform component-assembly tasks, selecting components required to implement the solution, resolving dependencies those components might have on other components, and interposing the mediations required to make components interact. Solution administrators deploy and customize the solutions they get from their integration specialist colleagues: they may be given a set of component relationships which they simply adopt; they may choose to override the defaults defined by an integration specialist; they may have to compensate for the fact that an integration specialist has not resolved certain variables of a solution; or they may have to reconfigure a previously deployed solution due to changes in the solution environment. These tasks are usually done in the component development environment, but the ESB enables flexible configuration through late binding by providing this service to the solution-administrator role. The key to enabling this flexible configuration and reconfiguration of solutions is the explicit declaration of capabilities and requirements of service interaction endpoints.

3.2c ESB Service Registry

Simply put, the ESB plays two main roles in the service endpoint matchmaking game—the service registry manages all relevant meta-data about interaction endpoints (see Figure below), and it also takes care of the matchmaking between those endpoints. This section discusses the service registry; the next section discusses the matchmaking. The ESB is a service registry in the sense that it manages not only meta-data about the service interaction endpoints involved in the SOA, but also information about domain models. This information establishes a common understanding of services beyond the scope of visibility of an individual service requestor or provider. The ESB captures information that can be used to better understand the practical content of the registry: domain models representing general knowledge about a topic area, independent of the specific domain applications represented as services in the registry. As before, we use a very generic definition of the term domain model because we want to establish an implementation-independent model for the ESB. Our definition covers domain models as simple as a topic space or a simple taxonomy that classifies events exchanged in publish/subscribe style interactions; it includes standard message sets used in specific industries or a set of "generic business objects" covering a specific application domain; and it extends to moderately complex ontology's describing concepts and their relations in a particular topic space. In the ESB integration model, domain models are used to establish semantics of the practical meta-data artifacts that the ESB cares about. In many "local ESB" scenarios, little or no domain knowledge needs to be formalized—the user community involved simply knows the semantics, and a simple hint in the form of well-named interfaces and messages suffices.

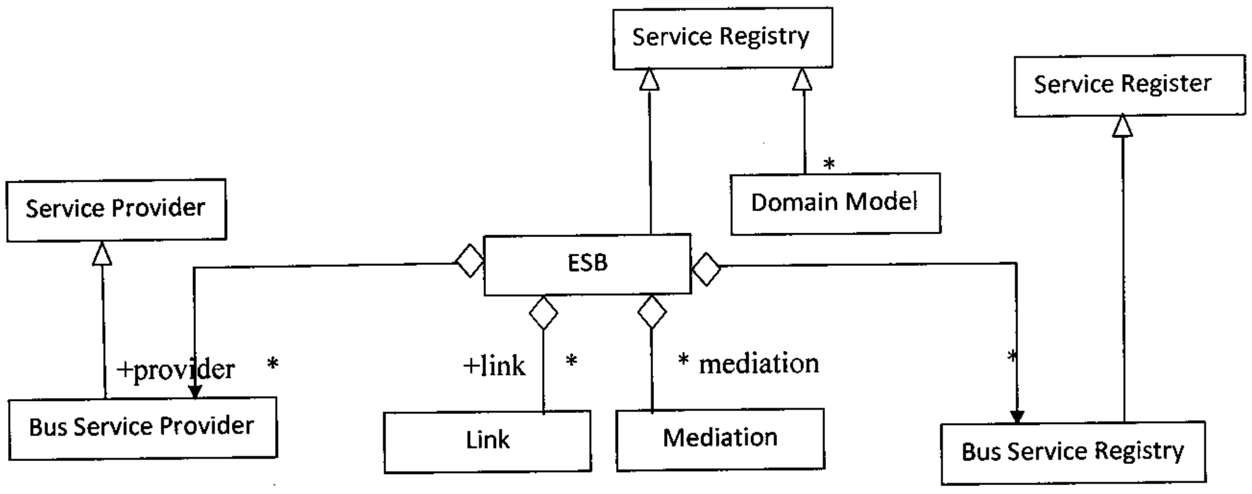


Figure 3.4 ESB Service Registry Content

This enables service-level managers to observe the status of a system, based on knowledge about events which adhere to the Common Base Events (CBE) standards proposal. Anthologies can establish a deeper semantic understanding of the interaction endpoints and facilitate use of more sophisticated ways to identify possible interactions between them. The main objective of managing domain models and establishing semantic understanding of interaction endpoints is to enable matchmaking between those endpoints the more explicit knowledge there is about capabilities and requirements of the endpoints, the more automated the matchmaking can be. The main role of the ESB registry is to manage metadata about the interaction endpoints themselves. To participate in ESB-managed interactions, endpoints need to register with the ESB. The ESB model represents registered service requestors as bus service requestors (BSRs) and registered service providers as bus service providers (BSPs). Service providers that are not registered as BSPs are invisible to the ESB for interaction partner selection. When a BSP or BSR is registered, its service interface and policy declarations are captured in the ESB service registry. At this point it is also possible to provide additional information about the service endpoint that might not have been provided with the original declaration. Semantic annotation is one example. Documenting or discovering relationships between the newly registered service and other artifacts in the service registry is another. Like any good service registry, the ESB provides the following features:

- Discovery and management of meta-information about interfaces and capabilities of existing applications that can be used as building blocks for integration solutions. This includes analysis of legacy applications to discover meta-information about their interfaces, policies, and behavioural constraints, as well as exploitation of object discovery agents to capture meta-information about packaged applications.

- Management of meta-information about services. This includes WSDL declarations as well as WS Policy declarations describing capabilities provided by services or required by service requestors and also BPEL-defined declarations of behavioural constraints for services (abstract BPEL processes) or actual behaviour of those services .
- Management of domain models describing general knowledge about an application domain relevant for SOA-based business integration scenarios. Examples include industry-standard message sets, generic business objects, anthologies encoded in the OWL Web Ontology Language, and "contracts" for SOA interactions.
- Discovery and management of relationships between "real world" artifacts representing existing applications, service declarations, and domain models.
- Enabling generation of artifacts for the ESB runtime. Examples include Service Data Object (SDO) schema, maps between service interfaces for ESB mediations, and application adapters.
- Management of runtime meta-information for matchmaking between service requestors and service providers (e.g., service declarations with policies and compatibility rules) and SDO schema with annotations.

3.3 ORCHESTRATION AND BPEL:

Business process orchestration brings agility to the enterprise. To create business process consisting of a set of services, a workflow is defined that invokes the appreciate service at each stage in the business process. To execute a business process, the workflow defined is "orchestrated". Orchestration controls the process-level integration and automation of services. The following briefly describe about the nature and functionalities of BPEL.

Business process execution language (BPEL) is gaining wide acceptance as the orchestration standard. BPEL is a language to specify business process as a workflow. Major vendors/open-source projects support BPEL and have orchestration engines that automate the process flow.

3.3a Introduction to BPEL

BPEL (Business Process Execution Language for Web Services, also WS-BPEL, BPEL4WS) is a language used for composition, orchestration, and coordination of web services. It provides a rich vocabulary for expressing the behaviour of business processes. In this chapter, we introduce BPEL, define its role in the **SOA** (Service Oriented Architecture), and explain the process-oriented approach to SOA and the role of BPEL. We also provide short descriptions of the most important BPEL servers—the run-time environments for execution of business processes specified in BPEL—and compare BPEL to other business process languages. We discuss the following features in the following pages:

• Discussing the role of business processes and their automation

• Overview web services, ESB (Enterprise Service Bus), and SOA

• Discussing the composition of services

• Explaining the role of BPEL in web service composition

• Explaining the most important BPEL features

• Overview of BPEL orchestration servers

• Comparing BPEL with other standards

Business Processes Matter

Enterprise applications and information systems have became fundamental assets of companies. Companies rely on them to be able to perform business operations. Enterprise information systems can improve the efficiency of businesses through automation of business processes. The objective of almost every company is that the applications it uses should provide comprehensive support for business processes. This means that applications should align with business processes closely.

Although this requirement does not sound very difficult to fulfill, the real-world situation shows us a different picture. Business processes are usually of dynamic nature. Companies have to improve and modify, act in an agile manner, optimize and adapt business processes to their customers, and thus improve the responsiveness of the whole company. Every change and improvement in a business process has to be reflected in the applications that provide support for them. Only companies where applications can be quickly and efficiently adapted to the changing business needs can stay competitive on the global market.

Automation of Business Processes

Based on what we have said so far, we can conclude that for efficient automation of business processes through IT we need to,

• Provide a standardized way to expose and access the functionality of applications as services.

• Provide an enterprise bus infrastructure for communication and management of services, including message interception, routing, transformation, etc.

• Provide integration architecture between the various services and existing and newly developed applications used in business processes.

• Provide a specialized language for composition of exposed functionalities of applications into business processes.

3.3b Service Composition with BPEL

Here we will get familiar with the BPEL concepts and discuss composing services with BPEL. We will show how to develop executable business processes. In a nutshell, we:

- Discuss service composition with BPEL
- Explain how business processes are defined in BPEL
- Get familiar with core concepts including:
    - ➢ The structure of BPEL process definitions
    - ➢ Invoking web services
    - ➢ Synchronous and asynchronous processes
    - ➢ Partner links
    - ➢ The role of WSDL
    - ➢ Important activities and other constructs
- Define an example BPEL process

3.3c Developing Business Processes with BPEL

BPEL uses an XML-based vocabulary that allows us to specify and describe business processes. With BPEL, you can describe business processes in two distinct ways:

- **Executable business processes** specify the exact details of business processes and can be executed by a BPEL engine. In the majority of cases we will use BPEL to specify executable processes.

- **Abstract business processes** specify only the public message exchange between parties, without including the specific details of process flows. They are not executable and are rarely used.

Executable business processes are processes that compose a set of existing services. When we describe a business process in BPEL, we actually define a new web service that is a composition of existing services. The interface of the new BPEL composite web service uses a set of port types, through which it provides operations like any other web service. To invoke a business process described in BPEL, we must invoke the resulting composite web service.

In a typical scenario, the BPEL business process receives a request. To fulfill it, the process then invokes the involved web services and finally responds to the original caller. Because the BPEL process communicates with other web services, it relies heavily on the WSDL description of the web services invoked by the composite web service.

Anyone developing BPEL processes requires a good understanding of WSDL and other related technologies. BPEL introduces WSDL extensions, which enable us to accurately specify relations between several web services in the business process. These relations are called **partner links**. The following figure shows a BPEL process and its relation to web services (partner links):

Any BPEL process specifies the exact order in which participating web services should be invoked. This can be done sequentially or in parallel. With BPEL, we can express conditional behavior, for example, a web service invocation can depend on the value of a previous invocation. We can also construct loops, declare variables, copy and assign values, define fault handlers, and so on. By combining all these constructs, we can define complex business processes in an algorithmic manner. We can describe deterministic as well as non-deterministic flows. Because business processes are essentially graphs of activities, it is sometimes useful to express them using **UML** (Unified Modeling Language) activity diagrams. To understand how business processes are defined in BPEL, we look at the core concepts in the next section.

Core Concepts

A BPEL process consists of steps. Each step is called an **activity**. BPEL supports basic and structured activities. Basic activities represent basic constructs and are used for common tasks, such as those listed below:

- Invoking other web services, using <invoke>
- Waiting for the client to invoke the business process through sending a message, using <receive> (receiving a request)
- Generating a response for synchronous operations, using <reply>
- Manipulating data variables, using <assign>
- Indicating faults and exceptions, using <throw>
- Waiting for some time, using <wait>
- Terminating the entire process, using <terminate>

We can then combine these and other basic activities and define complex algorithms that exactly specify the steps of a business process. To combine basic activities BPEL supports several structured activities. The most important are:

- Sequence (<sequence>) for defining a set of activities that will be invoked in an ordered sequence

- Flow (<flow>) for defining a set of activities that will be invoked in parallel
- Case-switch construct (<switch>) for implementing branches
- While (<while>) for defining loops
- The ability to select one of a number of alternative paths, using <pick>

Each BPEL process will also define partner links, using <partnerLink>, and declare variables, using <variable>.

To provide an idea of how a BPEL process looks, we show below a very simple BPEL process, which selects the best insurance offer from several.

3.3d Synchronous/Asynchronous Business Processes

We have already mentioned that the BPEL modeled business process is exposed as a web service. The BPEL process itself can be synchronous or asynchronous. A synchronous BPEL process returns a response to the client immediately after processing and the client is blocked for the whole duration of the BPEL process execution.

An asynchronous BPEL process on the other hand does not block the client. To return a result to the client, an asynchronous process uses a callback, similar to any other web service. However, it is not required that such a BPEL process returns a response.
This brings us to the conclusion that the type of BPEL process we choose is very important. Most real-world processes are long running, so we model them as asynchronous. However, there may also be processes that execute in a relatively short time, or processes where we want the client to wait for completion. We model such processes as synchronous.
We know that both first wait for the initial message, using a <receive>. Both also invoke other web services, either synchronously or asynchronously. However, a synchronous BPEL process will return a result after the process has completed. Therefore, we use a <reply> construct at the end of the process.

3.3e Advanced BPEL

We are now familiar with defining business processes, invoking web service operations sequentially and in parallel, defining partner links, defining variables, and assigning values. However, using BPEL for complex real-world business processes requires additional functionality. Sometimes the activities of a business process need to be performed in loops. Often activities might have links that would affect the execution order. This is usually the case with concurrent flows. Sometimes we will have to wait either for a message event or an alarm event to occur.

One very important aspect of business process modelling is fault handling. Particularly in business processes that span multiple enterprises and use web services over the Internet, we can assume that faults will occur quite often due to various reasons, including broken connections, unreachable web services, unavailability of services, and so on. If business processes do not finish successfully, we might need a way to undo the partial work. This is called compensation and is one of the features of BPEL.

We now look at these and other advanced BPEL features including:

- BPEL activities not covered in the previous chapter, such as loops, delays, and process termination
- Fault handling
- Scopes and serialization
- Compensation
- Events and event handlers
- Concurrent activities and links
- The business process lifecycle
- Correlations and message properties
- Dynamic partner links
- Abstract business processes
- A model-driven approach for generating BPEL processes from UML activity diagrams

3.4 LAYERS OF SOA

Like any distributed application, service-oriented applications are multi-tier applications and have presentation, business logic, and persistence layers. Figure 2 provides a typical architecture for a service-oriented application. The two key tiers in SOA are the services layer and the business process layer.

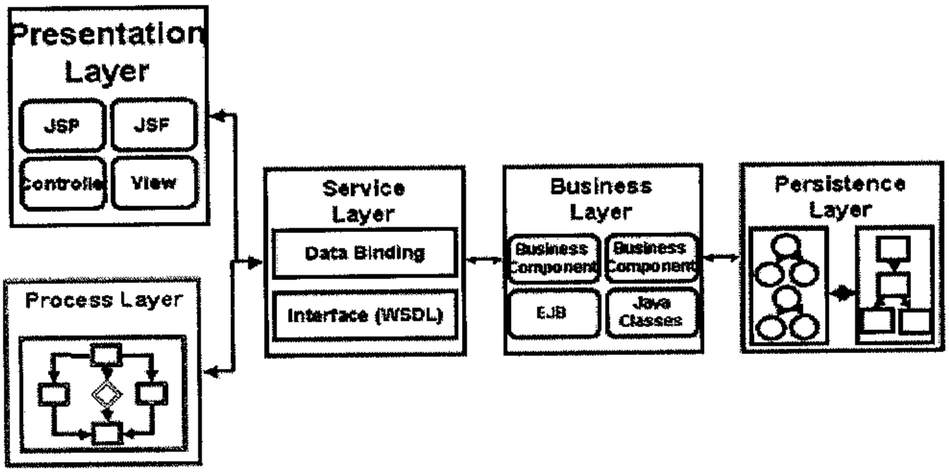


figure 3.5 Layers of SOA

3.4a The Service Layer

As we discussed earlier, services are the building blocks of service-oriented applications. Thus, services are somewhat analogous to Java objects and components such as EJBs. Unlike objects, however, services are self-contained, maintain their own state, and provide a loosely coupled interface.

The greatest challenge of building a service-oriented application is creating an interface with the right level of abstraction. While analyzing your business requirements, carefully consider what software components you want to build as a service. Generally, services should provide coarse-grained functionality. For example, the software component that processes a purchase order is a good candidate for publication as a service, as opposed to a component that just updates an attribute of a purchase order.

You have two choices when building a service: the top-down approach or the bottom-up approach. The top-down approach requires that you identify and describe the messages and operations your service provides and then implement the service. This approach is recommended when you're building a completely new service, as it lets you choose your preferred implementation technology. This approach also promotes the most interoperable services, since you can avoid implementation artifacts that may preclude interoperability (for example, data types that may not have an interoperable representation).

The bottom-up approach is quite popular because it lets you reuse your existing investment in business components. For example, vendors provide the tools that let you expose a PL/SQL-stored procedure that checks whether a customer is entitled to a discount as a service.

The most important aspect of a service is the service description. When using web services as the implementation technology for SOA, Web Services Description Language (WSDL) describes the messages, types, and operations of the web service, and is the contract to which the web service guarantees it will conform.

3.4b Business Process Layer

Another promise of SOA is that you can build a new application from existing services. The main benefit that SOA brings is the standardization of business process modeling, often referred to as *service orchestration*. You can build a web-service-based layer of abstraction over legacy systems and subsequently leverage them to assemble business processes. In addition, SOA platform vendors are providing tools and servers to design and execute these business processes. This effort has been standardized by an OASIS standard named **Business Process Execution Language (BPEL)**; most platform vendors are adhering to this standard. BPEL is essentially a programming language but is represented in XML.

Here's an overview of a BPEL-defined process:

```
<process>

 <!– Definition and roles of process participants -->
 <partnerLinks> ... </partnerLinks>

 <!- Data/state used within the process -->
 <variables> ... </variables>

 <!- Properties that enable conversations -->
 <correlationSets> ... </correlationSets>

 <!- Exception handling -->
 <faultHandlers> ... </faultHandlers>

 <!- Error recovery – undoing actions -->
 <compensationHandlers> ... </compensationHandlers>

 <!- Concurrent events with process itself -->
 <eventHandlers> ... </eventHandlers>

 </process>
```

BPEL provides:

- Partnerlinks for the services with which the process interacts.
- Variables for the data to be manipulated.
- Correlations to correlate messages between asynchronous invocations.
- Faults for message definitions for problems.

- Compensation handlers to execute in the case of problems.
- Event handlers that let the process deal with anticipated events in a graceful fashion.

Although the BPEL syntax is rather straightforward, a graphical representation of a business process is preferable, so you'll benefit from a GUI design tool to assemble business processes from existing services. Thus, creating business processes is a relatively simple task if you understand your business processes and you've deployed services available for your use. The **Oracle BPEL Designer,** which can be used as either as an Eclipse or JDeveloper plug-in, helps you design business processes, making it easier to design and develop services.

In addition, you'll need a high-performance processing environment (or server) to execute the generated business processes, and a management tool to test and monitor the status of these deployed processes. Most SOA platform vendors such as Oracle and IBM now have a robust platform to deploy business processes. For example, Oracle provides the **Oracle BPEL Process Manager** to deploy and execute business processes and Oracle BPEL Console to test and monitor business processes.

3.4c Presentation Layer: The Data Binding Problem

The presentation layer is very important from a user perspective. This layer can be built with technologies such as JSP, JSF, portlets, or standalone Java clients. For developers, it's an uphill battle to achieve loose coupling between the presentation layer and the business logic or service layers. Several Model-View-Controller (MVC) frameworks, which have been in use for a long time, allow for loose coupling between the view, or presentation layer, and the model that supplies the data and business logic. This lets you change either the view or model with minimal impact. This does help us achieve the type of loose coupling we want between the presentation and service layers. The main problem, however, is that there's no standard way of binding data between different kinds of clients (such as JSP, Java clients, and services such as EJB or web services), and clients have to know the exact underlying implementation of the service layer.

3.4d Best Practices

Here are a few best practices to follow when building service-oriented applications:

1. Do not jump into implementation technologies just because of the hype. Carefully consider whether web services make sense for your organization. Building a service-oriented application using traditional technologies such as RMI may be more appropriate for your use cases than using web services.

2. Modularity of services is very important. Do not build fine-grained services, as this will result in a tightly coupled, brittle infrastructure.

3. For most applications, interoperability is important. Adhere to the WS-I best practices--and even then, make sure to test for interoperability with your preferred clients.

4. If it doesn't make sense to use web services for your application, there are alternatives. Many BPEL process managers, such as Oracle BPEL Process Manager, let you use native protocols.

3.5 ENABLING TECHNOLOGIES:

Technologies that enable SOA include:

Web Services

A common misconception is that business services are the same as web services. In fact, while web services are highly interoperable, making them excellent for composite business applications that span business boundaries, they are only one of several options available for SOA. Conversely, adopting web services standards does not automatically give you SOA. Web Services is based on a messaging protocol and provides standards for interoperability between different software applications running on different platforms. It also provides a language (WSDL) for describing loosely coupled services irrespective of the contexts in which such services will be used and a language for publishing and discovering services (UDDI).Web services make the distinction between a service and the agent that implements it, with the view that new agents can be substituted while continuing to deliver the same service.

Business Process Management

In implementing business processes, business applications execute rules to determine the flow of control through the application. Traditionally such rules were embedded in application code and could only be maintained by IT staff. In recent years, Business Process Management technologies have emerged that allow people with minimal IT skills to manage business process rules. Business Process Management products provide a graphical interface for defining business processes by specifying what services to call when and under what conditions to call them. It is possible to achieve a similar result by encoding the processing rules in a business rules engine though in this case the logic flow is still embedded in the business application. A number of standards have emerged in the business process management area: including Business Process Execution Language for Web Services (BPEL4WS or BPEL). It is an XML based language developed by WC3 that describes how services (specifically web services) are co-ordinated to create processes.

Integration technologies

Integration technologies provide the plumbing that enables applications and services to talk to each other. They include: **Message orientated middleware** - providing support for sending messages between different technology platforms, but without any intelligent routing, data transformation or adaptor connectivity. **Enterprise Service Bus (ESB) products** - performing a function similar to EAI products but with a smaller footprint and fewer bells and whistles. ESBs claim to have better support for open standards. Their lightweight bus architecture affords them flexibility and scalability and they usually come at a lower cost than full feature EAI products.

CHAPTER 4

IMPLEMENTATION AND TESTING

Service Oriented Architecture (SOA) promises to implement composite applications that offer location transparency and segregation of business logic. Location transparency allows consumers and providers of services to exchange messages without reference to one another's concrete location. Segregation of business logic isolates the core processes of the application from other service providers and consumers. Together, these features let the user to replace or upgrade individual components in the composite application without affecting other components or the process as a whole. Moreover, we can independently specify alternative paths through which the various parts of the composite application exchange messages.

Here we have implemented a real estate service by using service oriented concept.
The services we provide in this portal includes, an elegant and effective service for the Real Estate, where a you can buy, sell and rent properties, lands, individual villas, flats, shops, complexes and offices, which is totally designed using the Service Oriented Architectural approach.

4.1 PROPOSED APPLICATION

Our application is an online real-estate service committed to helping user make wise and profitable decisions related to buying, selling, renting and leasing of properties, in India and key global geographies. This will provide a fresh new approach to our esteemed users to search for properties to buy or rent, and list their properties for selling or leasing.

It promises to be the most preferred way of finding your dream property and we are committed to help you make a wiser property decision, as a buyer or a seller. We want our esteemed users to realize over time that it is an expert friend, who can help them make some of their biggest and important decisions in life in a sharper, faster and easier way.

This team is committed to the following:-

1) Understand the needs and concerns of Individuals, Brokers, Builders and Corporate, and provide them a common platform for realizing maximum benefits from real-estate
2) Provide a superior real-estate experience by making it easier, faster, secure and more accurate to find buyers for your valuable property, and sellers for the dream property you have been waiting for
3) Give our customers complete control through easy to use interfaces and features that can help them identify, filter and contact potential buyers and sellers for properties

4.2 SOFTWARES AND LANGUAGES USED

To implement the services we have to have,

- Netbeans IDE 6.5
- Oracle database
- Apache tomcat 6.0
- Jsp pages

Netbeans IDE 6.5

The NetBeans Platform provides an application's common requirements, such as menus, document management, and settings. Building an application "on top of NetBeans" means that, instead of writing applications from scratch, you only provide the parts of your application that the NetBeans Platform doesn't already have. At the end of the development cycle, you bundle your application with the NetBeans Platform, saving user time and energy and resulting in a solid, reliable application.

Oracle database 10g

Oracle Database Express Edition (Oracle Database XE) is a free, smaller-footprint edition of Oracle Database. Oracle Database XE is easy to install and easy to manage.

With Oracle Database XE,

- Administrate the database
- Create tables, views, and other database objects
- Import, export, and view table data
- Run queries and SQL scripts
- Generate reports

Apache Tomcat 6.0

Apache Tomcat is an implementation of the Java Servlet and Java Server Pages technologies. The Java Servlet and Java Server Pages specifications are developed under the Java Community Process.

Apache Tomcat is developed in an open and participatory environment and released under the Apache Software License.

Apache Tomcat powers numerous large-scale, mission-critical web applications across a diverse range of industries and organizations.

4.3 IMPLEMENTATION STEPS

1. Identifying the business process services.
2. Identifying SOA Platform.
3. Refining BPEL.
4. Build web pages.
5. Expose as SERVICES.
6. Integrate and test.

Business process services

Service Oriented Computing (SOC) is a new computing paradigm that utilizes services as the basic constructs to support the development of rapid, low-cost and easy composition of distributed applications even in heterogeneous environments. The visionary promise of Service Oriented Computing is a world of cooperating services where application components are assembled with little effort into a network of services that can be loosely coupled to create flexible dynamic business processes and agile applications that may span organisations and computing platforms.

SOC is being shaped by, and increasingly will help shape, modern society as a whole, especially in the areas of dynamic and on-demand business, health and government services. The subject of Service Oriented Computing is vast and enormously complex, spanning many concepts and technologies that find their origins in diverse disciplines that are woven together in an intricate manner. In addition, there is a need to merge technology with an understanding of business processes and organizational structures, a combination of recognizing an enterprise's pain points and the potential solutions that can be applied to correct them. The material in research spans an immense and diverse spectrum of literature, in origin and in character. As a result research activities at both worldwide as well as at European level are very fragmented. This necessitates that a broader vision and perspective be established one that permeates and transforms the fundamental requirements of complex applications that require the use of the Service-Oriented Computing paradigm. Service composition, service management and monitoring and service-oriented engineering.

In the case of Service oriented architecture implementation, identifying business process services will be the first process step. After deciding which format of application we have to list the services which we are going to provide through that application. While collecting the services of our real estate website, we can have,

- **Buy**
- **Sell**
- **Authentication**
- **Authentication verified**
- **Sign-in**
- **Main(co-coordinating all the above services)**

Identifying SOA Platform

So after getting the services, we must choose an appropriate platform to implement SOA, that is we must have options such as,

- SOA.
- BPEL design.
- WSDL.
- DATABASE connection.
- PROXY settings and
- SERVER connection.

We have chosen **netbeans IDE 6.5** which is having all the options specified above.

Steps involved :-(netbeans IDE 6.5)

1. From the main menu, choose File > New Project.

2. In the Categories list, select the SOA node.

3. In the Projects list, select the BPEL Module node and click Next.

4. Optionally, click on Browse button to specify a location where you want this project to be created. For this tutorial accept the default location.

5. In the Project name field, type "real estate" and click finish.

6. Now do the orchestration for each service specified in previous section by using business process execution language.

7. For each BPEL design there will be the automatic creation of WSDL file to publish in the registry.

8. Start the Apache Tomcat server after adding the jsp pages to run.

9. Connect the database.

10 Set the proxy settings.

11. Build and run the BPEL module.

Refining BPEL

BPEL (Business Process Execution Language for Web Services, also WS-BPEL, BPEL4WS) is a language used for composition, orchestration, and coordination of web services. It provides a rich vocabulary for expressing the behaviour of business processes. In the platform of netbeans, it has option for designing BPEL orchestration for all the services specified above.

To design the BPEL we have some variables in that to use, such as

- Invoking other web services, using <invoke>
- Waiting for the client to invoke the business process through sending a message, using <receive> (receiving a request)
- Generating a response for synchronous operations, using <reply>
- Manipulating data variables, using <assign>
- Indicating faults and exceptions, using <throw>
- Waiting for some time, using <wait>
- Terminating the entire process, using <terminate>
- We can then combine these and other basic activities and define complex algorithms that exactly specify the steps of a business process. To combine basic activities BPEL supports several structured activities. The most important are:
- Sequence (<sequence>) for defining a set of activities that will be invoked in an ordered sequence
- Flow (<flow>) for defining a set of activities that will be invoked in parallel
- Case-switch construct (<switch>) for implementing branches
- While (<while>) for defining loops
- The ability to select one of a number of alternative paths, using <pick>

Each BPEL process will also define partner links, using <partnerLink>, and declare variables, using <variable>. And for each BPEL design there must be the connection of database, if required. This is also done in nebeans itself. So while designing BPEL to compose services, description of those services also will be done by generating WSDL files according to the BPEL files.

Building web pages

To design the presentation layer, we have to have the web pages in our real estate web site. Here we have designed jsp pages with database connection for,

- Home page for proreal, which has login form and recent uploads registered by the end user, after the verification is done by the moderator.
- Sign up form.
- Individual home page for each end-user.
- Searching the required Land
- Intimating the owner by customer.
- Registering the land by providing some details.
- Administrative page to verify and authenticate the registered land. This is especially for moderator. After the verification only land details will be displayed to the public.

These are the pages which will be running through Apache tomcat server.

Exposing as SERVICES

To expose the created BPEL and WSDL files we must provide SOA interaction pattern here, that is **the relationship between service-consumer, service-provider and service registry.**

Service Consumer

Consumer will be the end-user of our website who is expecting all the services provided in the case of SOA. She/he has the relationship between service provider and service registry. He gets the relation to service provider through XML messages, in same way gets the relationship to service registry by setting proxy configurations.

Service Provider

Provider will be referring the developer side. They are the one who generates and provides all the services. Generally to bring the concept of SOA interaction pattern, service providers will register or publish those services into service registry by specifying its URL. The relation between the service consumer and service provider will be made by ESB which is provided by WSO2.

Service Registry

Registry will be referring the repository of all the services which has the description of each service. Here we use WSO2 registry to store all the web services.WSO2 is an SOA open source company which provides the registry, ESB etc. from the service registry service consumer can discover the required service.

Integration:

Integration is the final step of SOA web portal implementation. Here the designs and codes which are all done in previous steps such as,

- Web pages.
- Service Registry.
- Enterprise service bus.
- BPEL and WSDL files.
- Database connection.
- Apache tomcat server.

Finally build and run for all the test cases by giving and testing different type of input to the portal.

CHAPTER-5

CONCLUSION & FUTURE WORKS

Service-oriented architecture (SOA) helps organizations more easily transform their business processes for high performance by simplifying the underlying information systems.

Service-oriented architecture gives existing systems the flexibility and agility to respond to a business environment which is changing rapidly. Service-oriented architectures allow businesses and governments to capitalize interoperability, dynamic discovery of services, loose coupling, and reusability. Because of its standards based approach, it gives alignment to Business and Technology.

Here we have shown this architecture by taking the application of real estate business process. It is an online real-estate service committed to helping you make wise and profitable decisions related to buying, selling, renting and leasing of properties, in India and key global geographies.

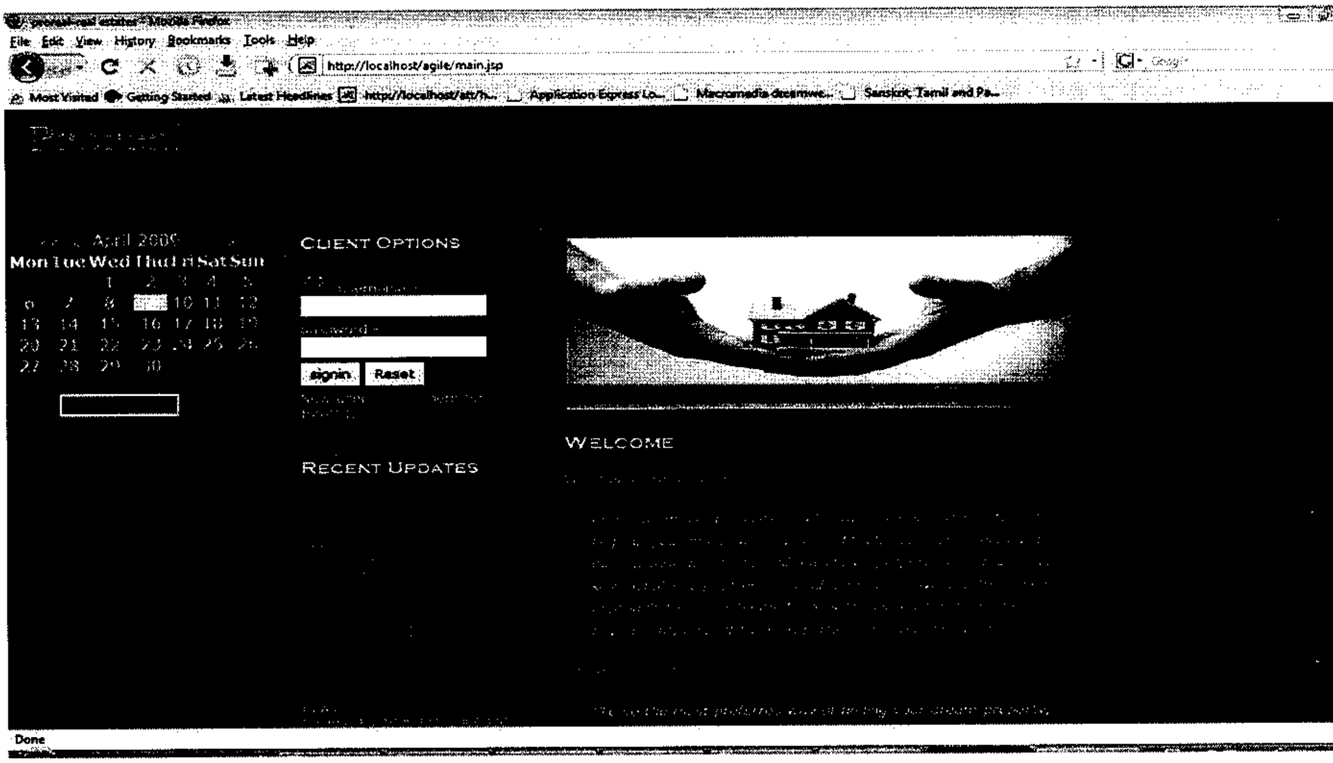
FUTURE WORK:

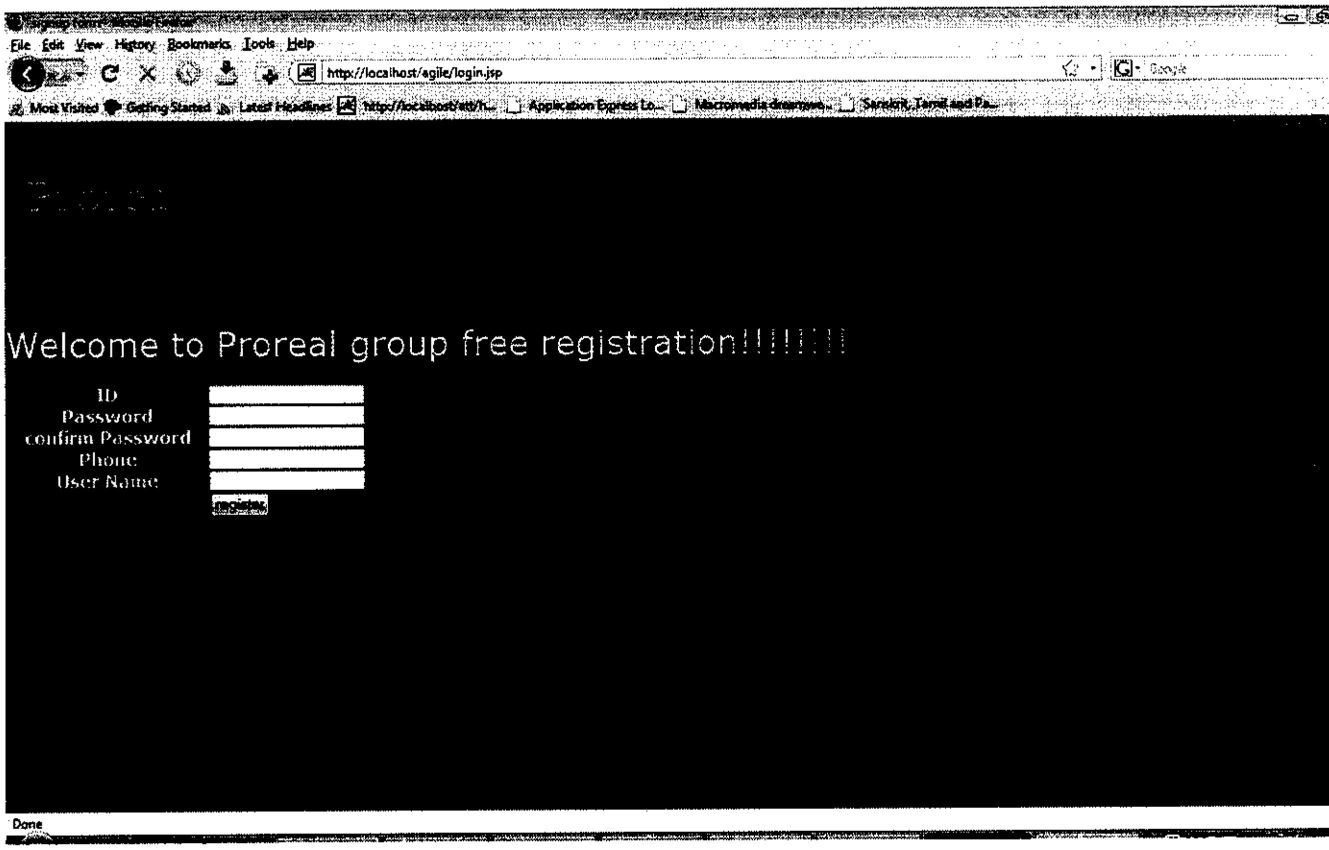
In the case of implementation process, we have implemented SOA by showing the interaction process of services. But still we shall upgrade our implementation by showing following details,

- Upgrading ESB by adding more proxy services.
- Adding more number of services in presentation layer.
- Testing is to be done to increase the efficiency.

SAMPLE SCREEN

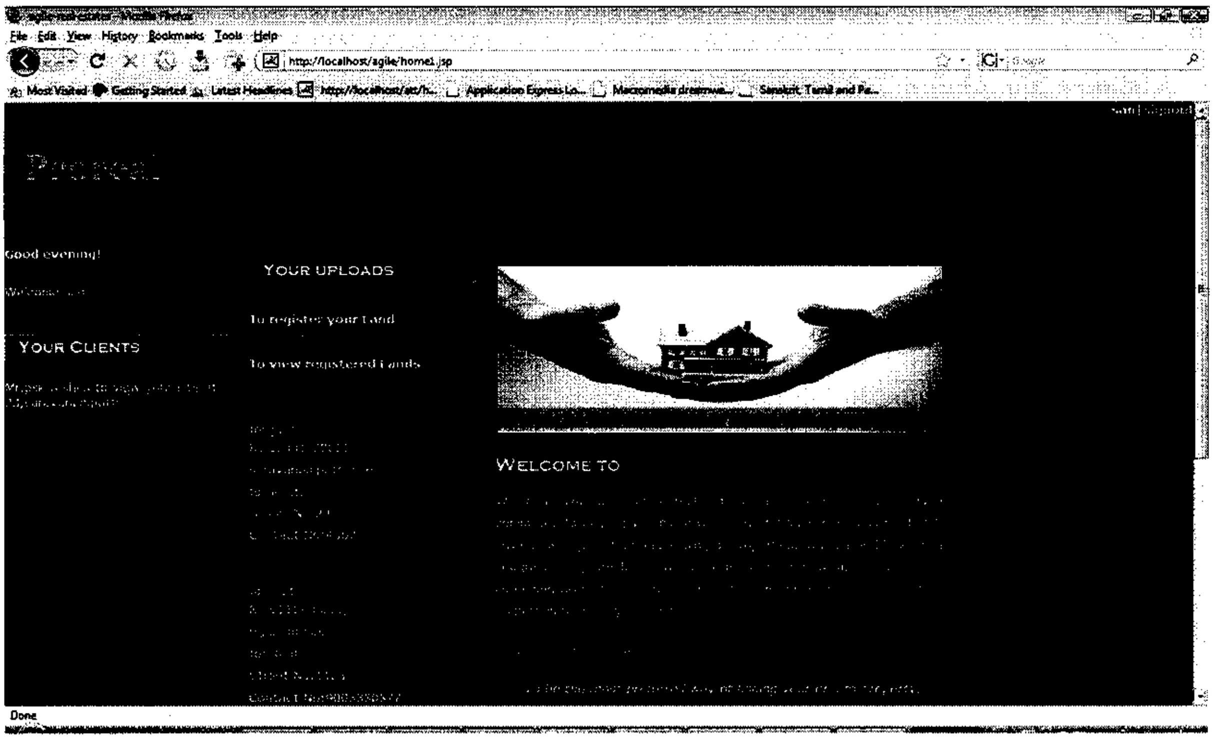
WEB PAGES


File Edit View History Bookmarks Tools Help
http://localhost/agile/main.jsp
Most Visited Getting Started Latest Headlines http://localhost/att/h... Application Express Lo... Macromedia dreamwe... Sanskrit, Tamil and Pa...
April 2009
Mon Tue Wed Thu Fri Sat Sun
Client Options
signin
Reset
Recent Updates
Welcome
Done


File Edit View History Bookmarks Tools Help
http://localhost/agile/login.jsp
Most Visited Getting Started Latest Headlines http://localhost/att/h... Application Express Lo... Macromedia dreamwe... Sanskrit, Tamil and Pa...
Done

Welcome to Proreal group free registration!!!!!!!!

ID	
Password	
confirm Password	
Phone	
User Name	
	register



http://localhost/agile/home1.jsp

Good evening!

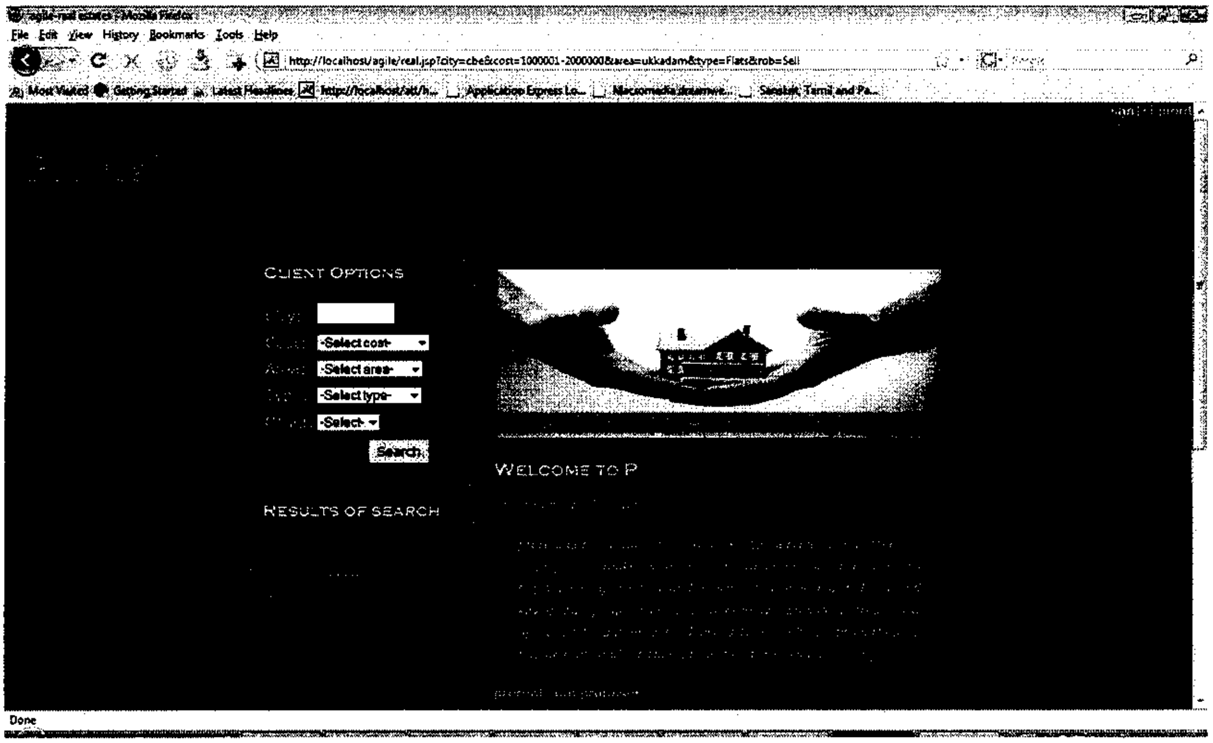
Your Clients

Your uploads

To register your Land

To view registered Lands

Welcome to



http://localhost/agile/real.jsp?city=cbe&cost=1000001-2000000&area=ukkadam&type=Flats&rob=Sell

Client Options

-Select cost-

-Select area-

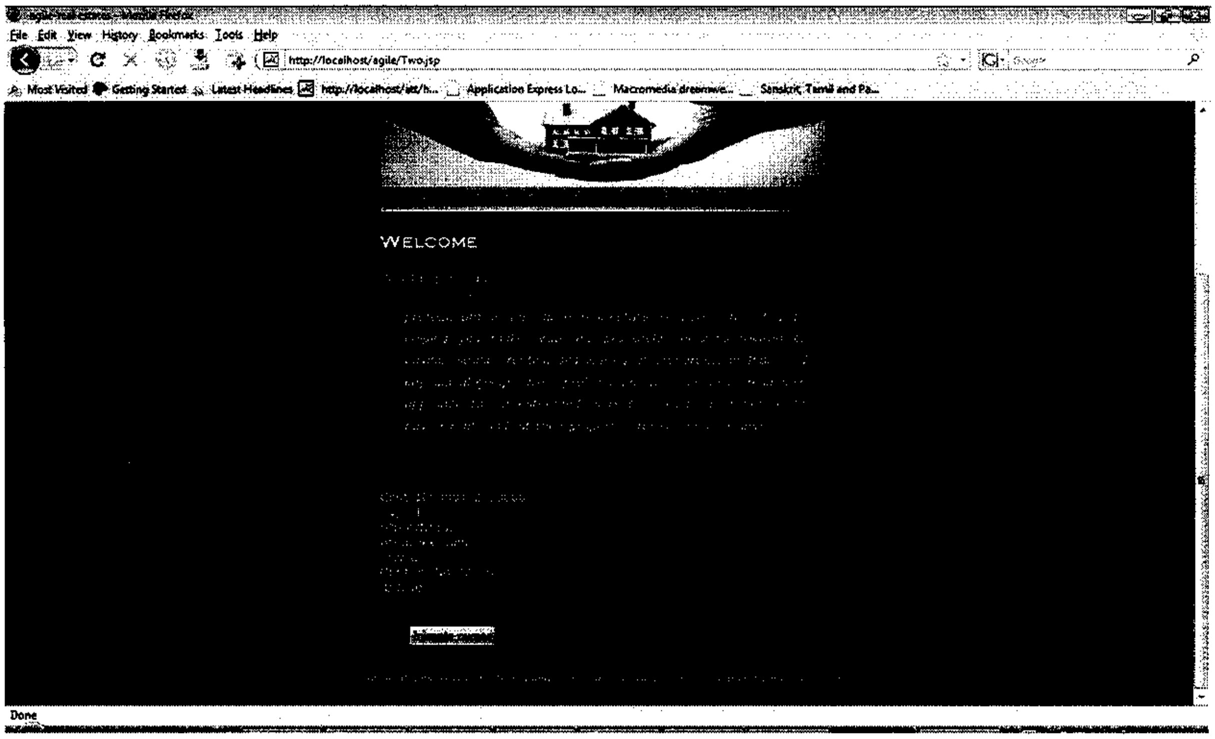
-Select type-

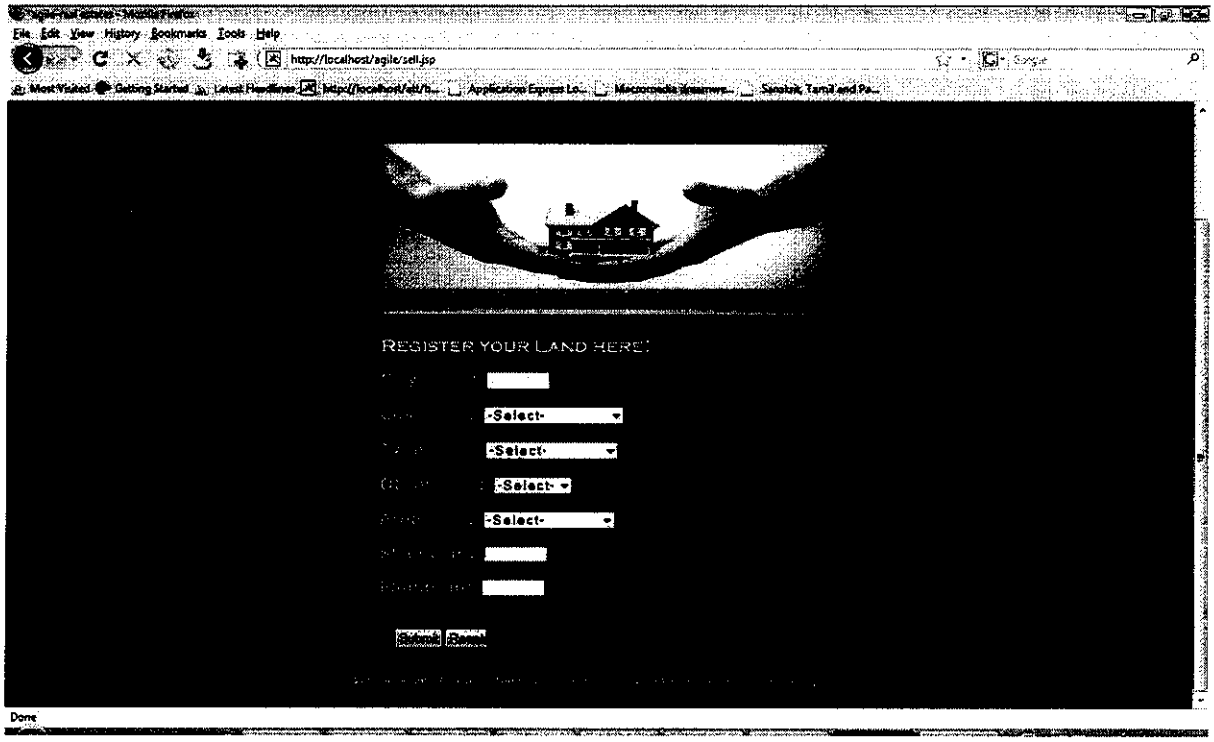
-Select-

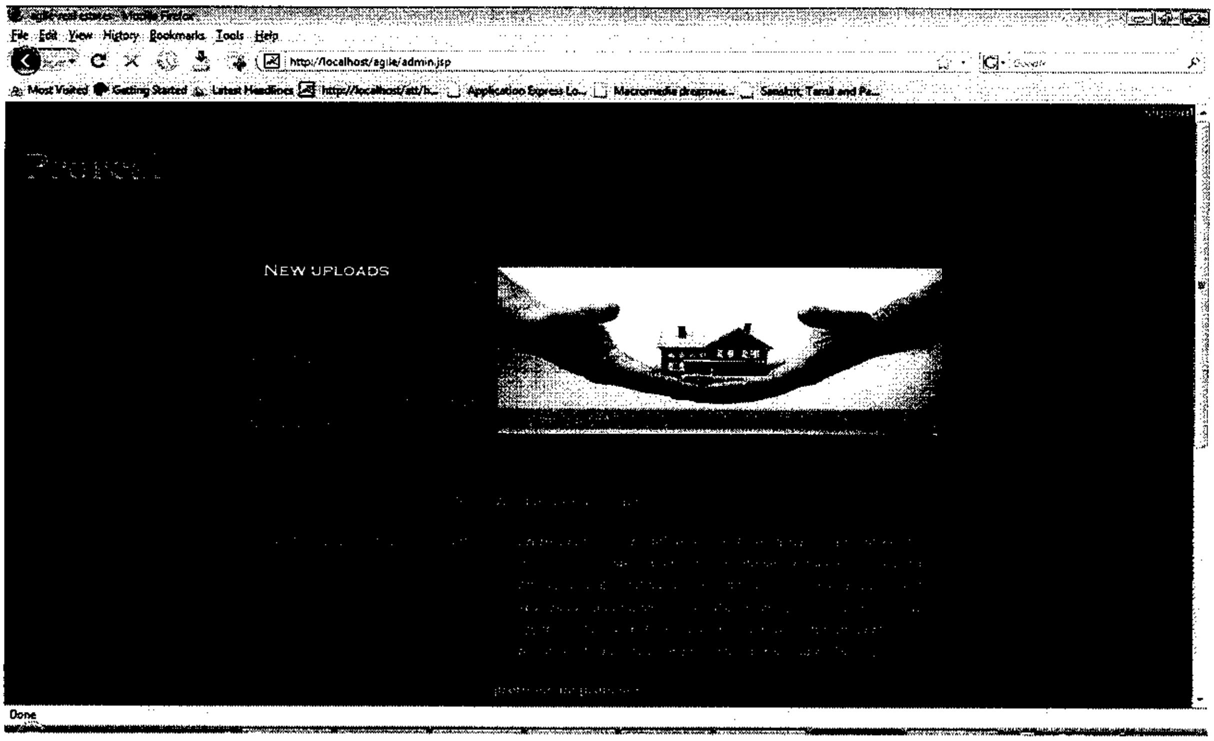
Search

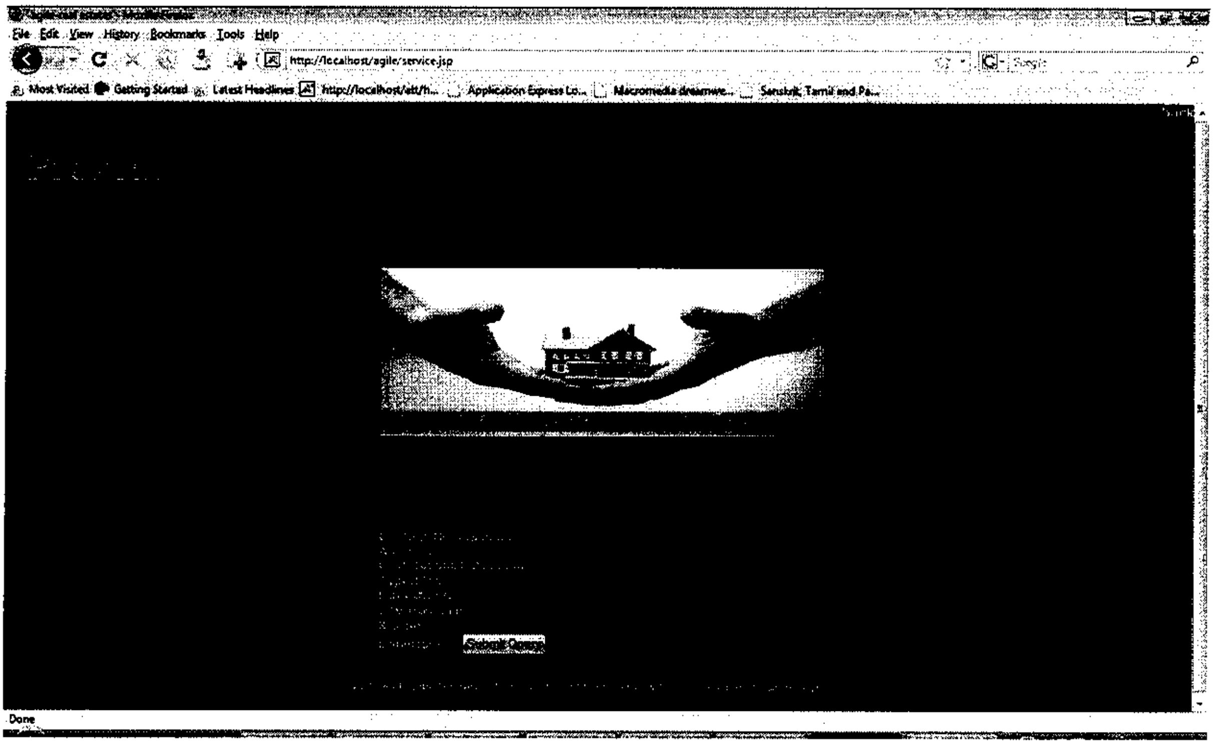
Results of search

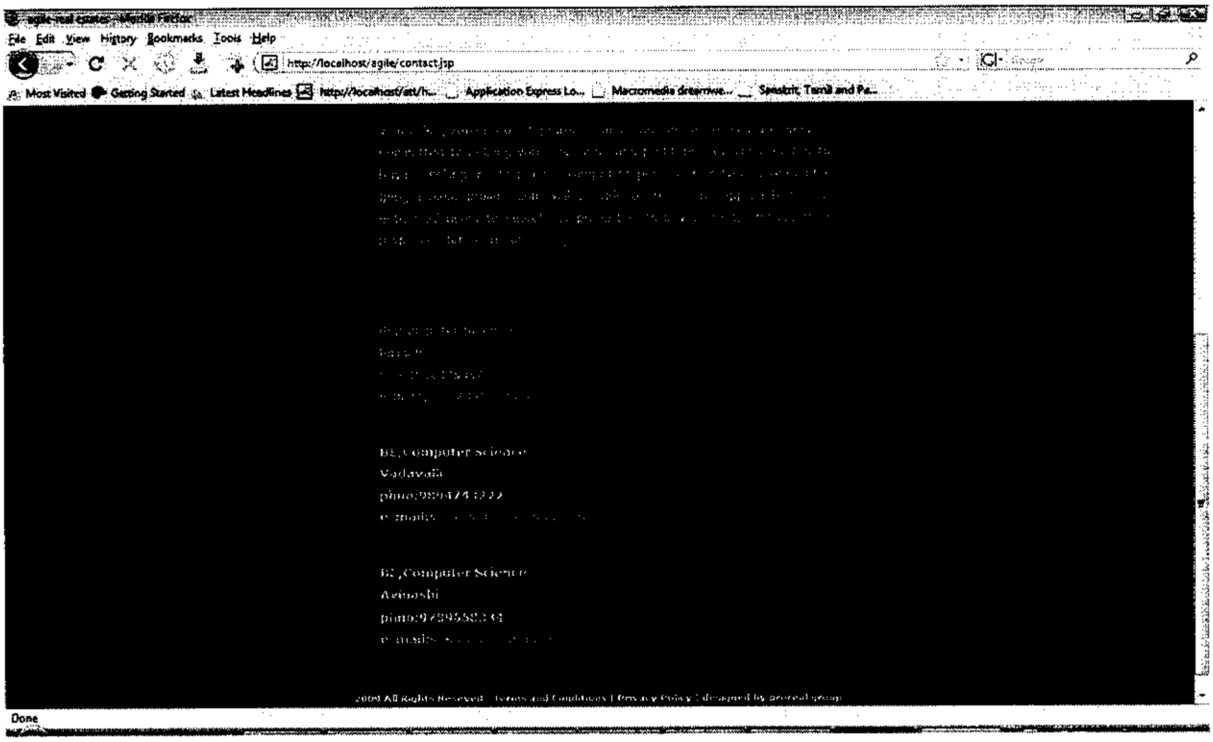
Welcome to P


http://localhost/agile/Two.jsp
WELCOME
Done


http://localhost/agile/sell.jsp
REGISTER YOUR LAND HERE!
-Select-
-Select-
-Select-
-Select-
Submit
Reset
Done


File Edit View History Bookmarks Tools Help
http://localhost/agile/admin.jsp
Most Visited
Getting Started
Latest Headlines
http://localhost/att/h...
Application Express Lo...
Macromedia dreamwe...
Sanskrit, Tamil and Pa...
NEW UPLOADS
Done


File Edit View History Bookmarks Tools Help
http://localhost/agile/service.jsp
Most Visited
Getting Started
Latest Headlines
http://localhost/att/h...
Application Express Lo...
Macromedia dreamwe...
Sanskrit, Tamil and Pa...
Submit Query
Done



http://localhost/agile/contact.jsp

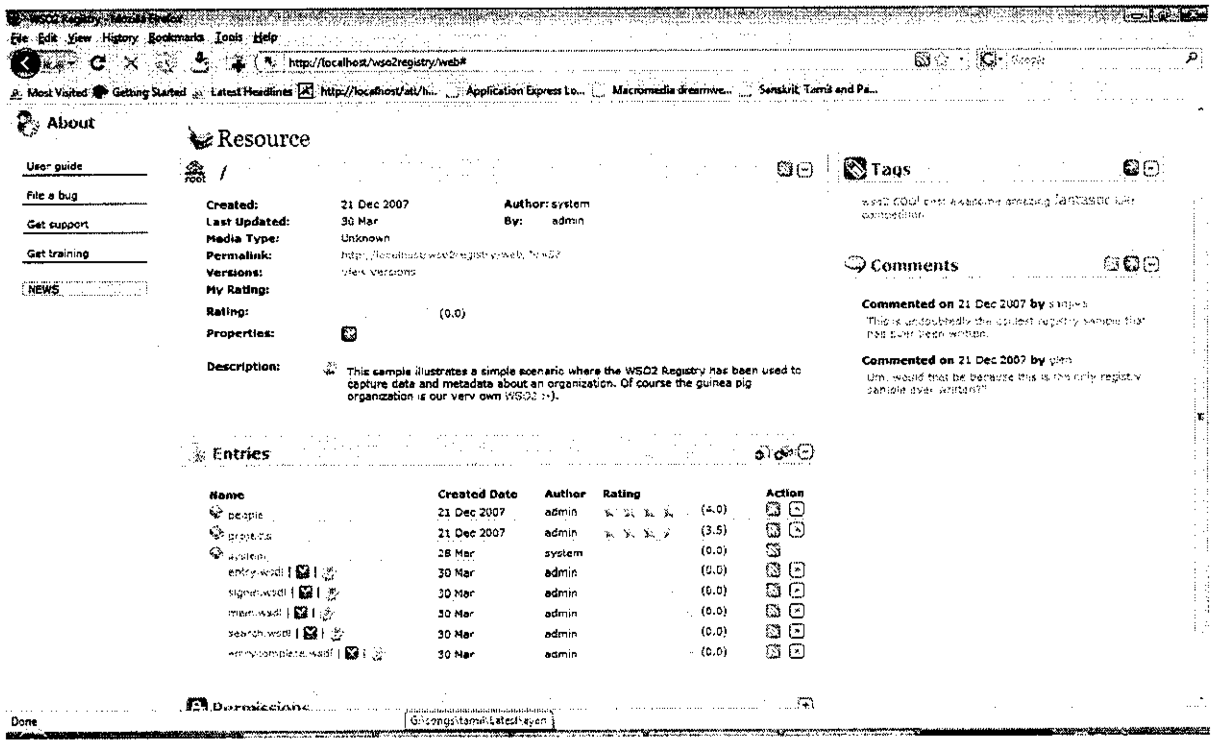
[illegible]

BE,Computer Science
Vadavalli
phno:9894743322
e-mail: [illegible]

BE,Computer Science
Avinashi
phno:9789555334
e-mail: [illegible]

2009 All Rights Reserved · Terms and Conditions | Privacy Policy | designed by [illegible] group

REGISTRY



http://localhost/wso2registry/web#

About

User guide
File a bug
Get support
Get training
NEWS

Resource

/

Created:	21 Dec 2007	Author:	system
Last Updated:	30 Mar	By:	admin
Media Type:	Unknown		
Permalink:	[illegible]		
Versions:	View versions		
My Rating:			
Rating:	(0.0)		
Properties:			

Description: This sample illustrates a simple scenario where the WSO2 Registry has been used to capture data and metadata about an organization. Of course the guinea pig organization is our very own WSO2 :-).

Entries

Name	Created Date	Author	Rating	Action
people	21 Dec 2007	admin	(4.0)	
projects	21 Dec 2007	admin	(3.5)	
system	28 Mar	system	(0.0)	
entry.wsdl	30 Mar	admin	(0.0)	
signin.wsdl	30 Mar	admin	(0.0)	
main.wsdl	30 Mar	admin	(0.0)	
search.wsdl	30 Mar	admin	(0.0)	
entrycomplete.wsdl	30 Mar	admin	(0.0)	

Permissions

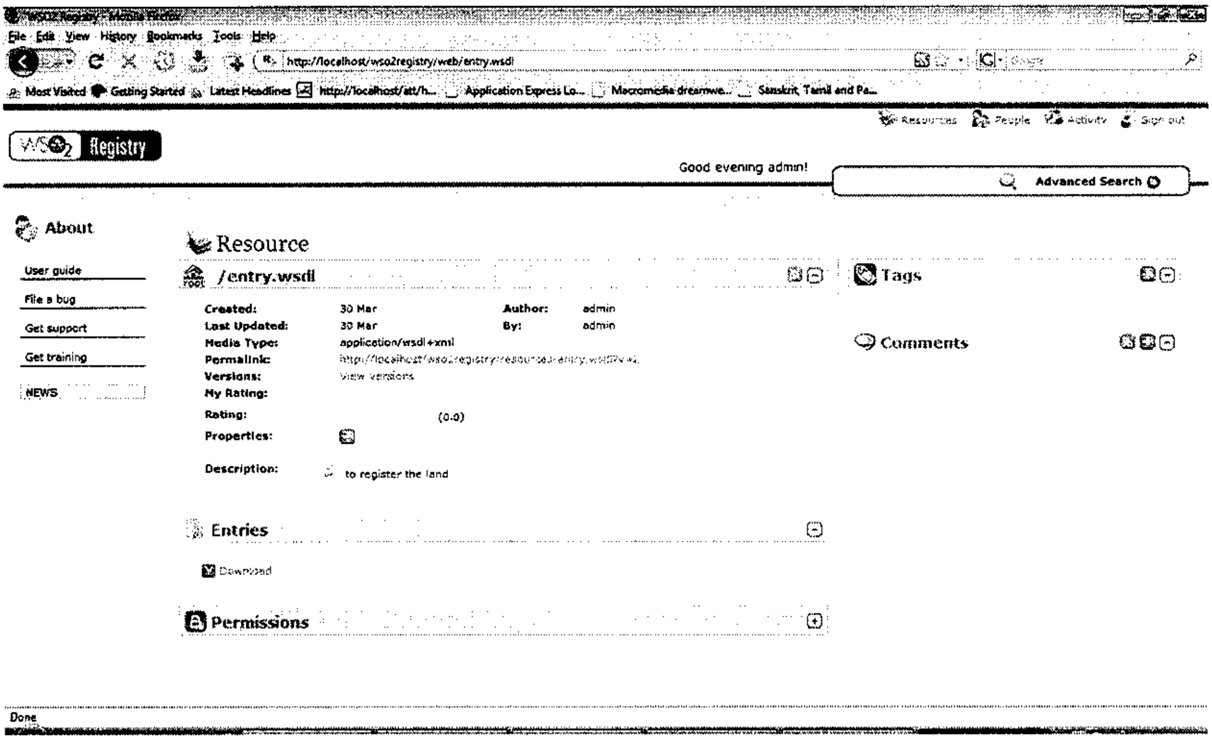
Tags

wso2 cool best awesome amazing fantastic sample [illegible]

Comments

Commented on 21 Dec 2007 by [illegible]
This is undoubtedly the coolest registry sample that has ever been written.

Commented on 21 Dec 2007 by [illegible]
Um, would that be because this is the only registry sample ever written??


WSO2 Registry - Mozilla Firefox

http://localhost/wso2registry/web/entry.wsdl

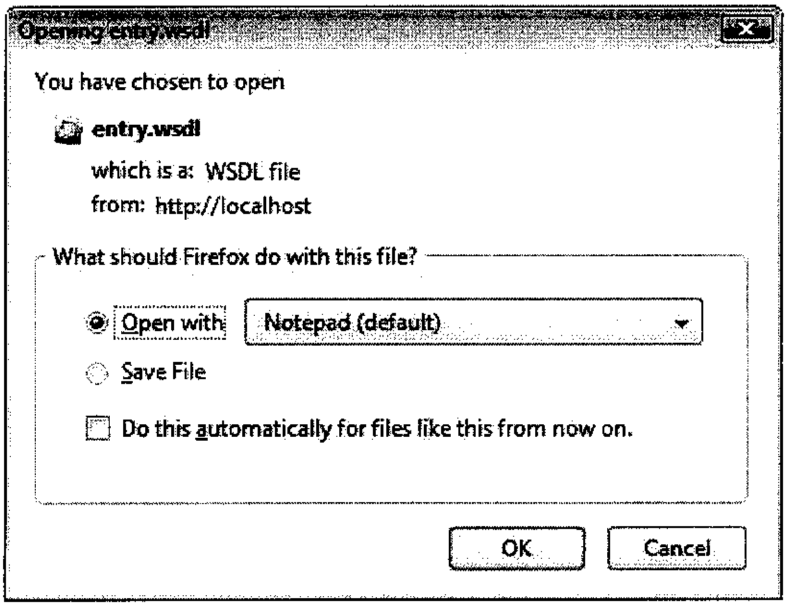
Good evening admin!

/entry.wsdl

Created: 30 Mar **Author:** admin
Last Updated: 30 Mar **By:** admin
Media Type: application/wsdl+xml
Permalink: http://localhost/wso2registry/resources/entry.wsdl?v=4
Versions: View versions
My Rating:
Rating: (0.0)
Properties:

Description: to register the land

File Edit View History Bookmarks Tools Help
Most Visited Getting Started Latest Headlines http://localhost/att/h... Application Express Lo... Macromedia dreamwe... Sanskrit, Tamil and Pa...
Resources People Activity Sign out
WSO2 Registry
Advanced Search
About
User guide
File a bug
Get support
Get training
NEWS
Resource
Tags
Comments
Entries
Download
Permissions
Done


Opening entry.wsdl

You have chosen to open

entry.wsdl

which is a: WSDL file
from: http://localhost

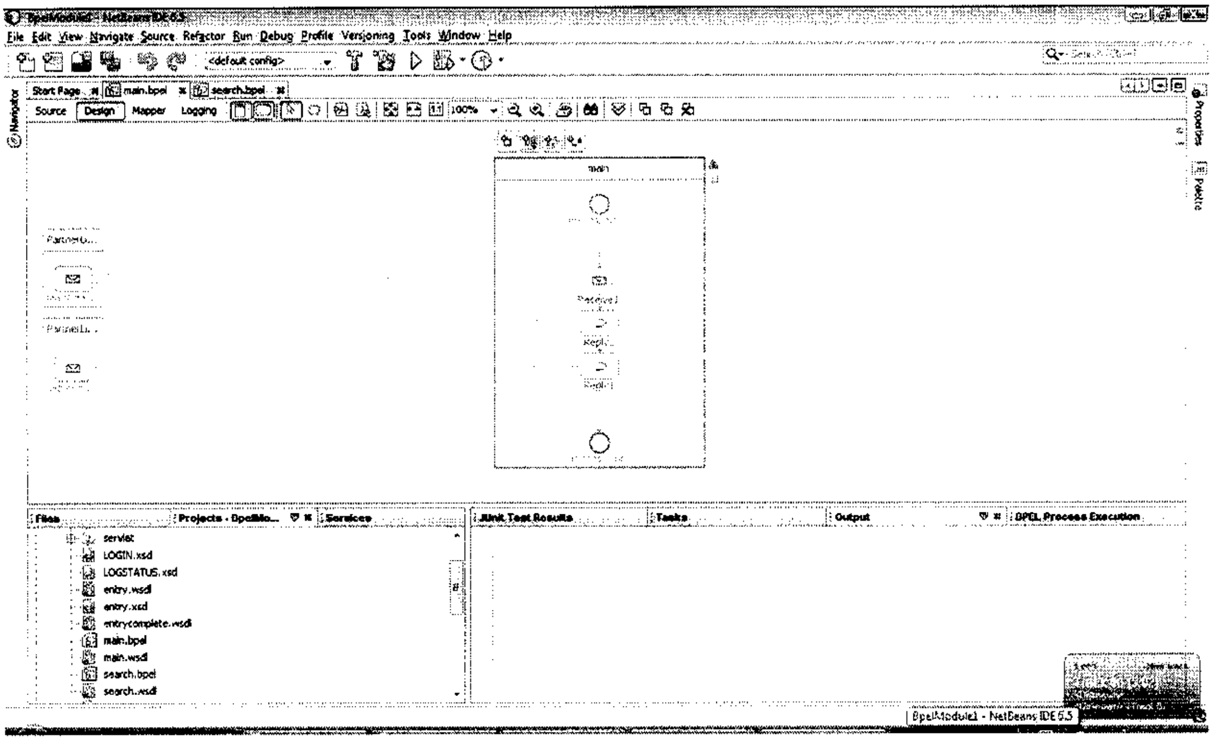
What should Firefox do with this file?

[✓ Open with] Notepad (default)

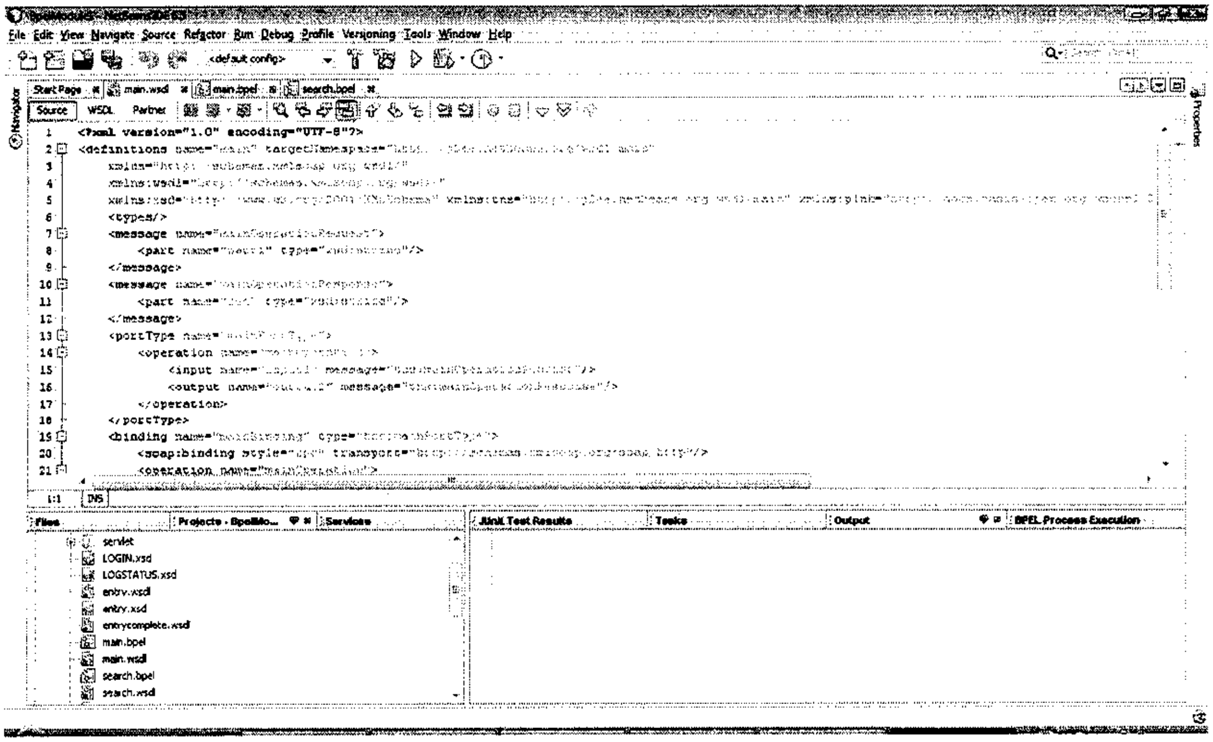
Save File

☐ Do this automatically for files like this from now on.

BPEL DESIGN

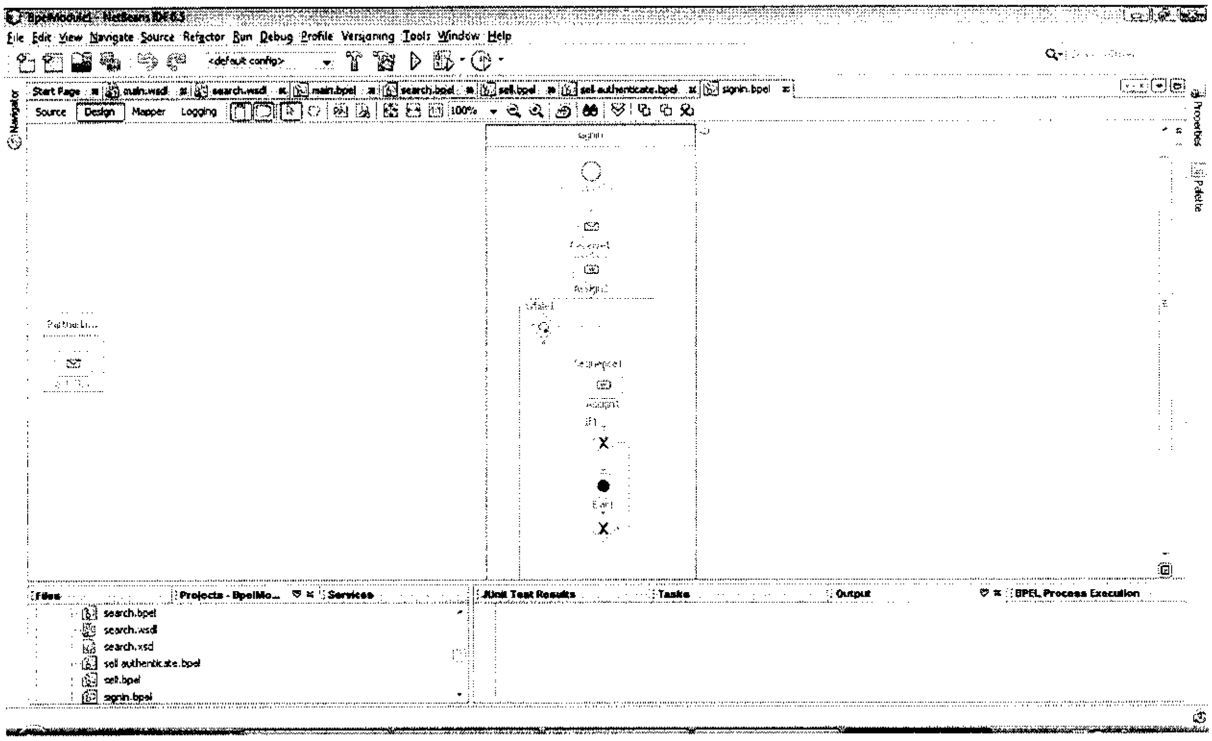

BpelModule1 - NetBeans IDE 6.5
File Edit View Navigate Source Refactor Run Debug Profile Versioning Tools Window Help
<default config>
Start Page main.bpel search.bpel
Source Design Mapper Logging
100%
main
Receive1
Reply1
Reply1
PartnerL...
PartnerL...
Files
Projects - BpelMo...
Services
JUnit Test Results
Tasks
Output
BPEL Process Execution
Navigator
Properties
Palette

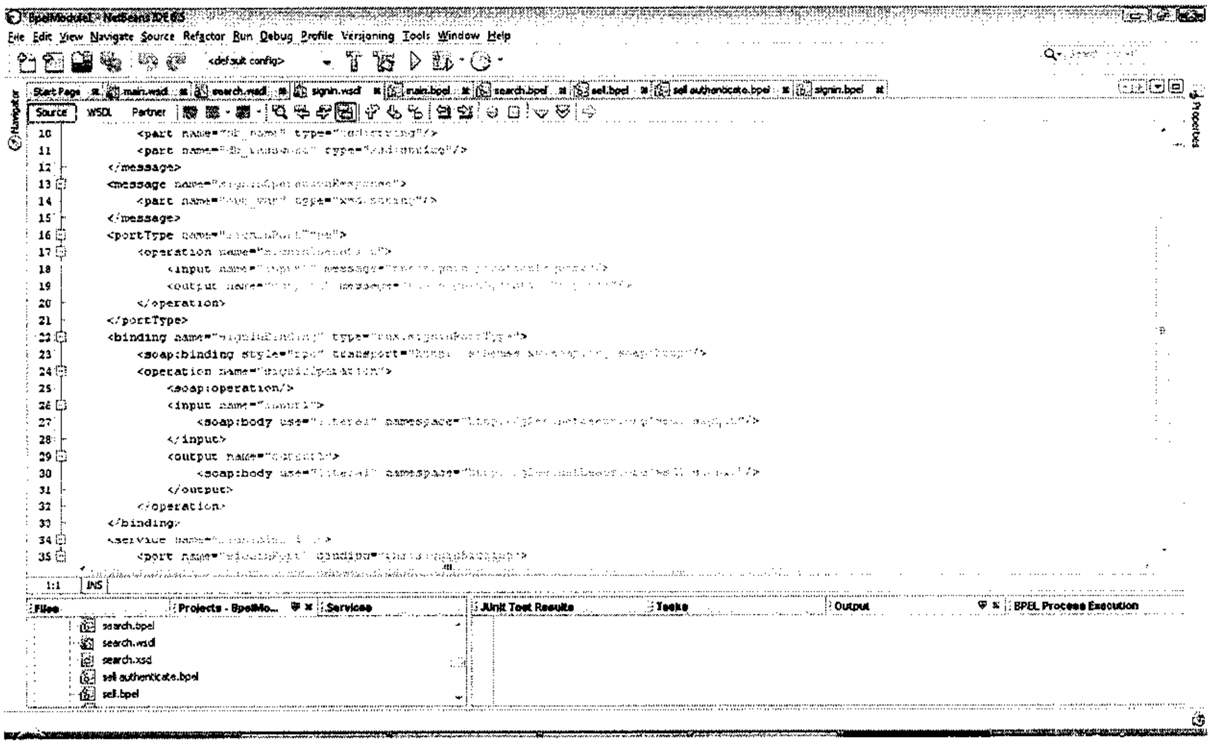
servlet
LOGIN.xsd
LOGSTATUS.xsd
entry.wsdl
entry.xsd
entrycomplete.wsdl
main.bpel
main.wsdl
search.bpel
search.wsdl


BpelModule1 - NetBeans IDE 6.5
File Edit View Navigate Source Refactor Run Debug Profile Versioning Tools Window Help
<default config>
Start Page main.wsdl main.bpel search.bpel
Source WSDL Partner
1:1 INS
Files
Projects - BpelMo...
Services
JUnit Test Results
Tasks
Output
BPEL Process Execution
Navigator
Properties

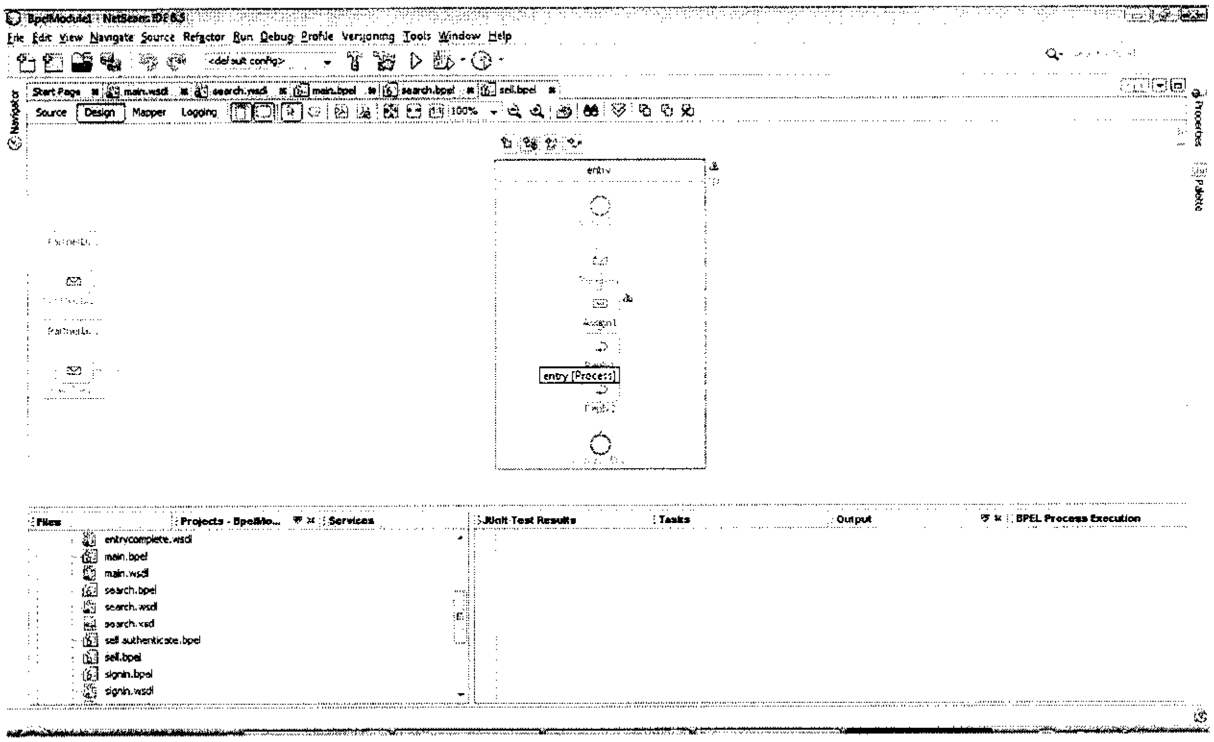
```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="[illegible]" targetNamespace="[illegible]"
    xmlns="[illegible]"
    xmlns:wsdl="[illegible]"
    xmlns:xsd="[illegible]" xmlns:tns="[illegible]" xmlns:plnk="[illegible]"
    <types/>
    <message name="[illegible]">
        <part name="[illegible]" type="[illegible]"/>
    </message>
    <message name="[illegible]">
        <part name="[illegible]" type="[illegible]"/>
    </message>
    <portType name="[illegible]">
        <operation name="[illegible]">
            <input name="[illegible]" message="[illegible]"/>
            <output name="[illegible]" message="[illegible]"/>
        </operation>
    </portType>
    <binding name="[illegible]" type="[illegible]">
        <soap:binding style="[illegible]" transport="[illegible]"/>
        <operation name="[illegible]">
```

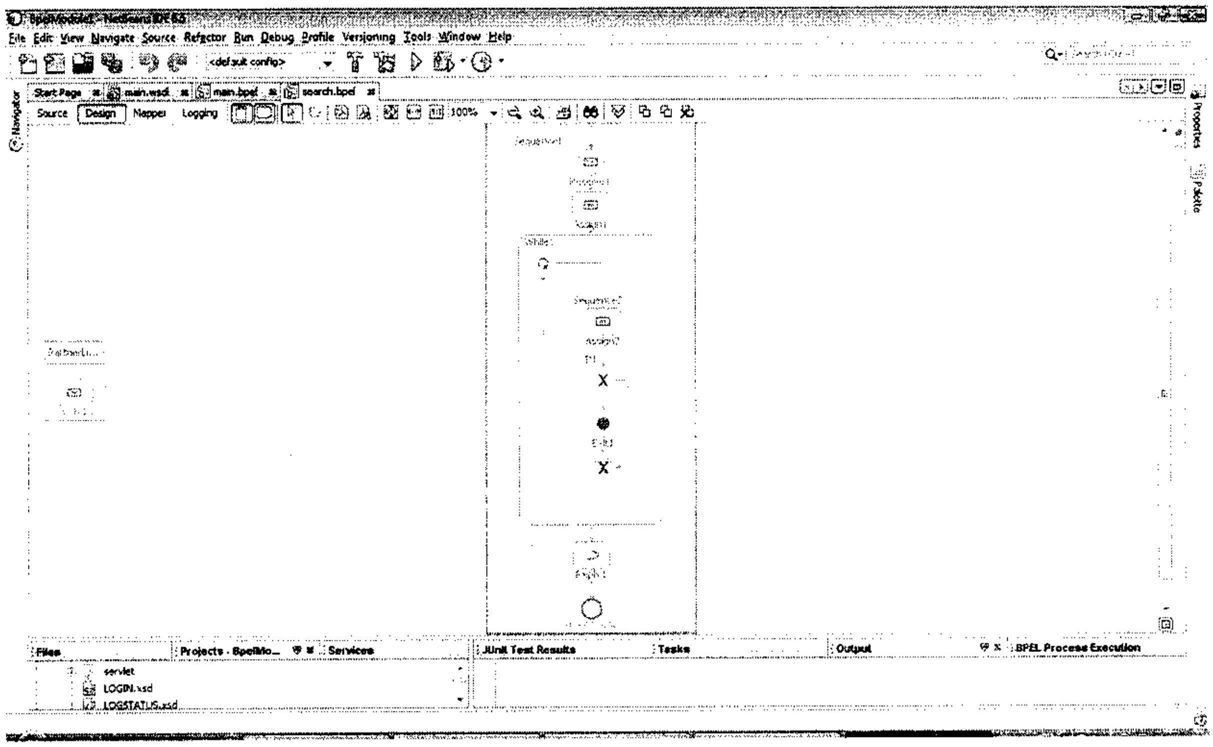
servlet
LOGIN.xsd
LOGSTATUS.xsd
entry.wsdl
entry.xsd
entrycomplete.wsdl
main.bpel
main.wsdl
search.bpel
search.wsdl

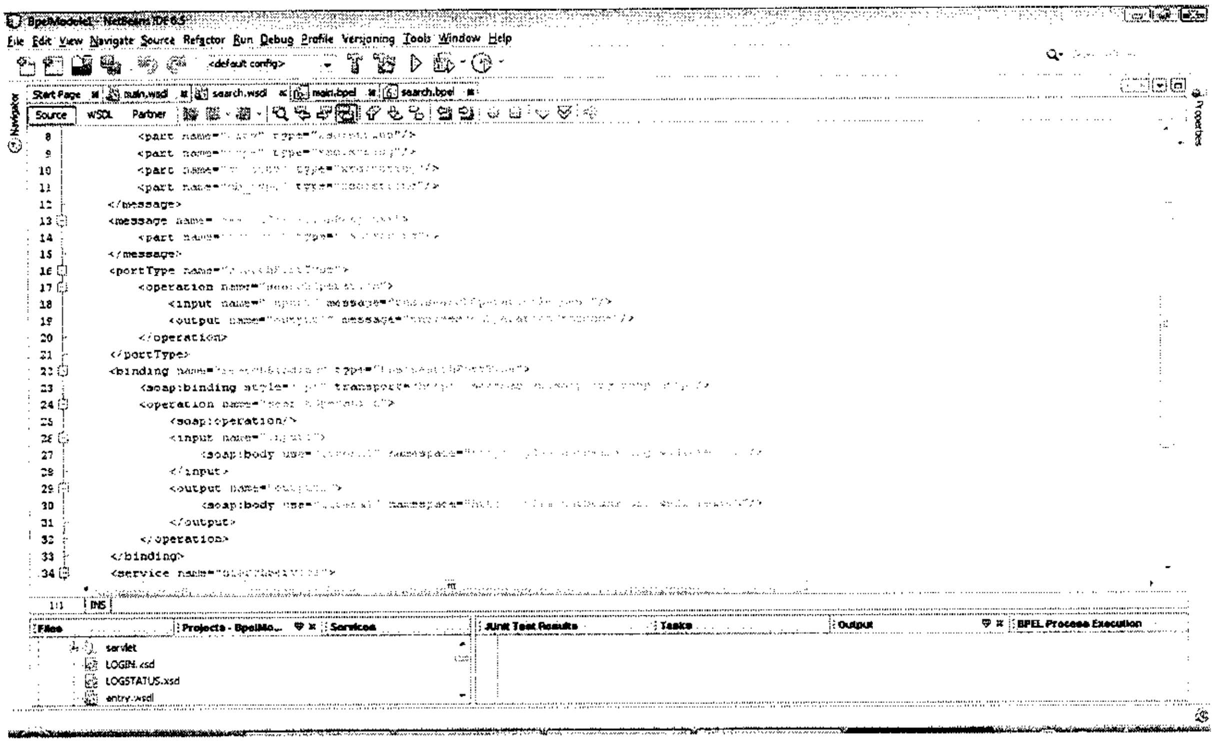

BpelModule1 - NetBeans IDE 6.5
File Edit View Navigate Source Refactor Run Debug Profile Versioning Tools Window Help
<default config>
Start Page
main.wsdl
search.wsdl
main.bpel
search.bpel
sell.bpel
sell authenticate.bpel
signin.bpel
Source
Design
Mapper
Logging
100%
Navigator
Properties
Palette
Files
Projects - BpelMo...
Services
search.bpel
search.wsdl
search.xsd
sell authenticate.bpel
sell.bpel
signin.bpel
JUnit Test Results
Tasks
Output
BPEL Process Execution
Receive1
Assign1
Sequence1
Assign2
If1
Exit


BpelModule1 - NetBeans IDE 6.5
File Edit View Navigate Source Refactor Run Debug Profile Versioning Tools Window Help
<default config>
Start Page
main.wsdl
search.wsdl
signin.wsdl
main.bpel
search.bpel
sell.bpel
sell authenticate.bpel
signin.bpel
Source
WSDL
Partner
Navigator
Properties
1:1
INS
Files
Projects - BpelMo...
Services
search.bpel
search.wsdl
search.xsd
sell authenticate.bpel
sell.bpel
JUnit Test Results
Tasks
Output
BPEL Process Execution

```
            <part name="[illegible]" type="[illegible]"/>
            <part name="[illegible]" type="[illegible]"/>
        </message>
        <message name="[illegible]">
            <part name="[illegible]" type="[illegible]"/>
        </message>
        <portType name="[illegible]">
            <operation name="[illegible]">
                <input name="[illegible]" message="[illegible]"/>
                <output name="[illegible]" message="[illegible]"/>
            </operation>
        </portType>
        <binding name="[illegible]" type="[illegible]">
            <soap:binding style="[illegible]" transport="[illegible]"/>
            <operation name="[illegible]">
                <soap:operation/>
                <input name="[illegible]">
                    <soap:body use="[illegible]" namespace="[illegible]"/>
                </input>
                <output name="[illegible]">
                    <soap:body use="[illegible]" namespace="[illegible]"/>
                </output>
            </operation>
        </binding>
        <service name="[illegible]">
            <port name="[illegible]" binding="[illegible]">
```

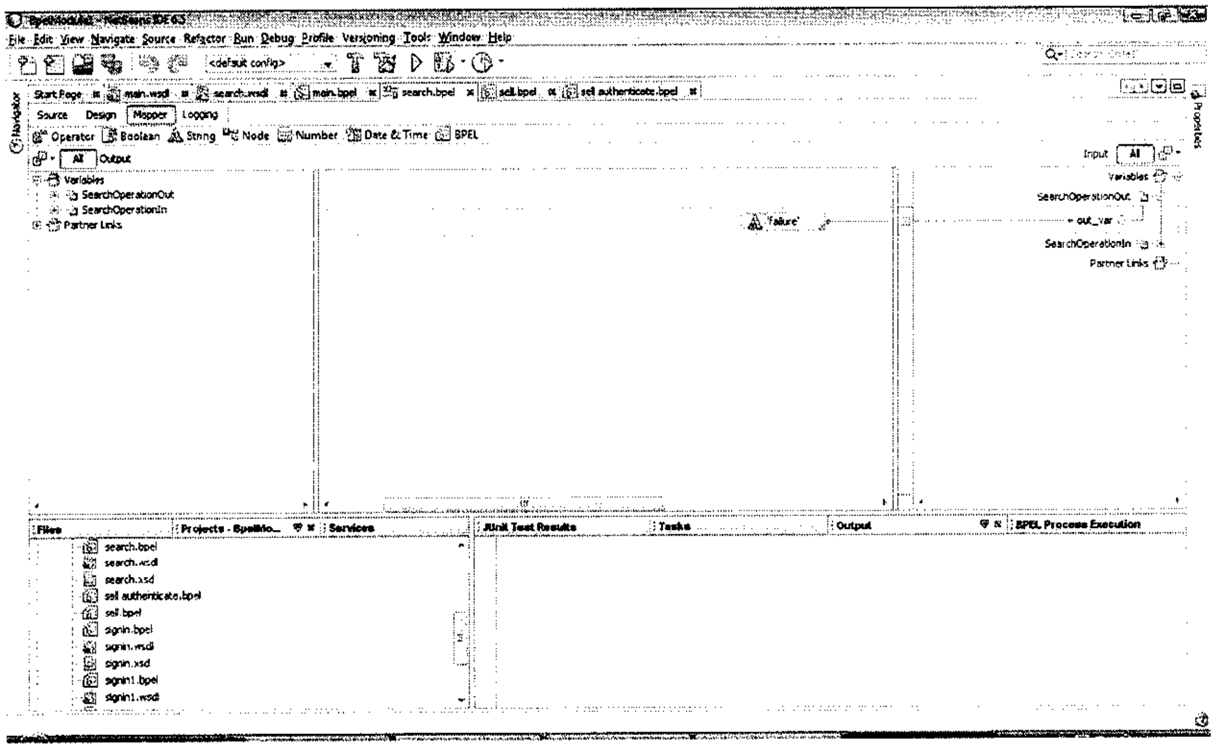

BpelModule1 - NetBeans IDE 6.5
File Edit View Navigate Source Refactor Run Debug Profile Versioning Tools Window Help
<default config>
Start Page
main.wsdl
search.wsdl
main.bpel
search.bpel
sell.bpel
Source Design Mapper Logging
100%
Navigator
Properties
Palette
entry
Assign1
entry [Process]
Files
Projects - BpelMo...
Services
JUnit Test Results
Tasks
Output
BPEL Process Execution
entrycomplete.wsdl
main.bpel
main.wsdl
search.bpel
search.wsdl
search.xsd
sell authenticate.bpel
sell.bpel
signin.bpel
signin.wsdl


BpelModule1 - NetBeans IDE 6.5
File Edit View Navigate Source Refactor Run Debug Profile Versioning Tools Window Help
<default config>
Start Page
main.wsdl
main.bpel
search.bpel
Source Design Mapper Logging
100%
Navigator
Properties
Palette
Assign1
While1
Assign2
Files
Projects - BpelMo...
Services
JUnit Test Results
Tasks
Output
BPEL Process Execution
servlet
LOGIN.xsd
LOGSTATUS.xsd

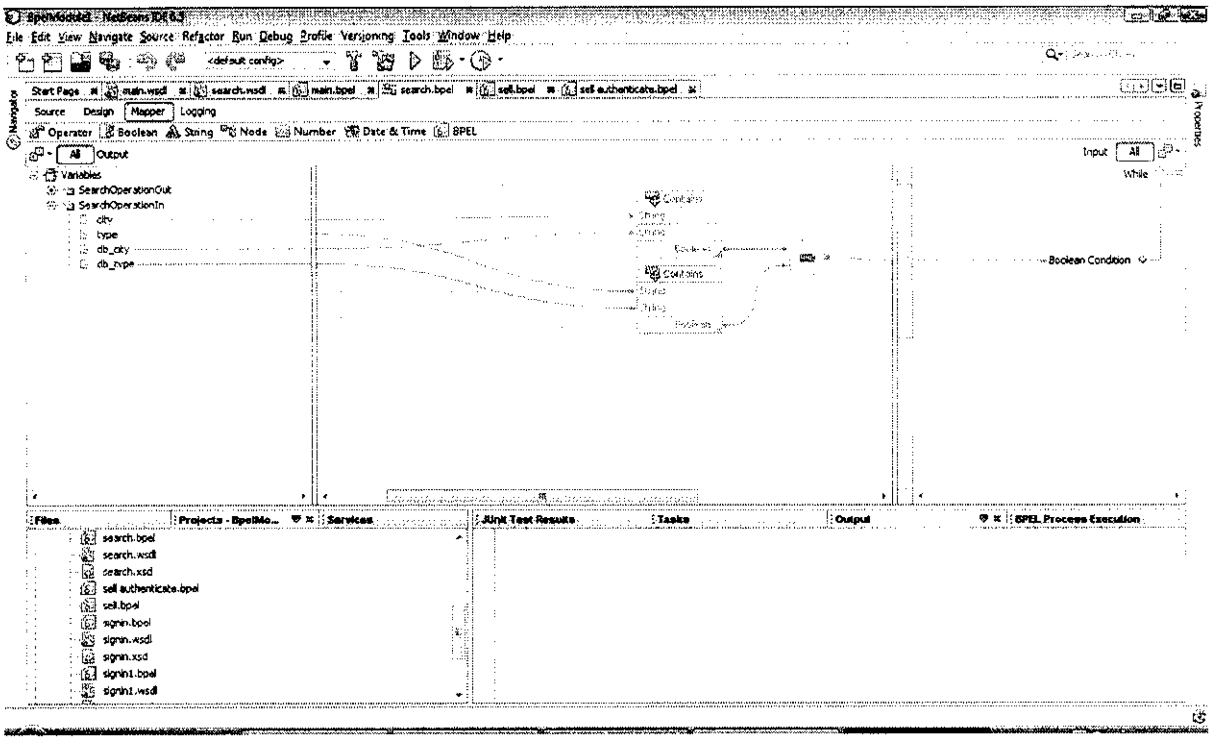


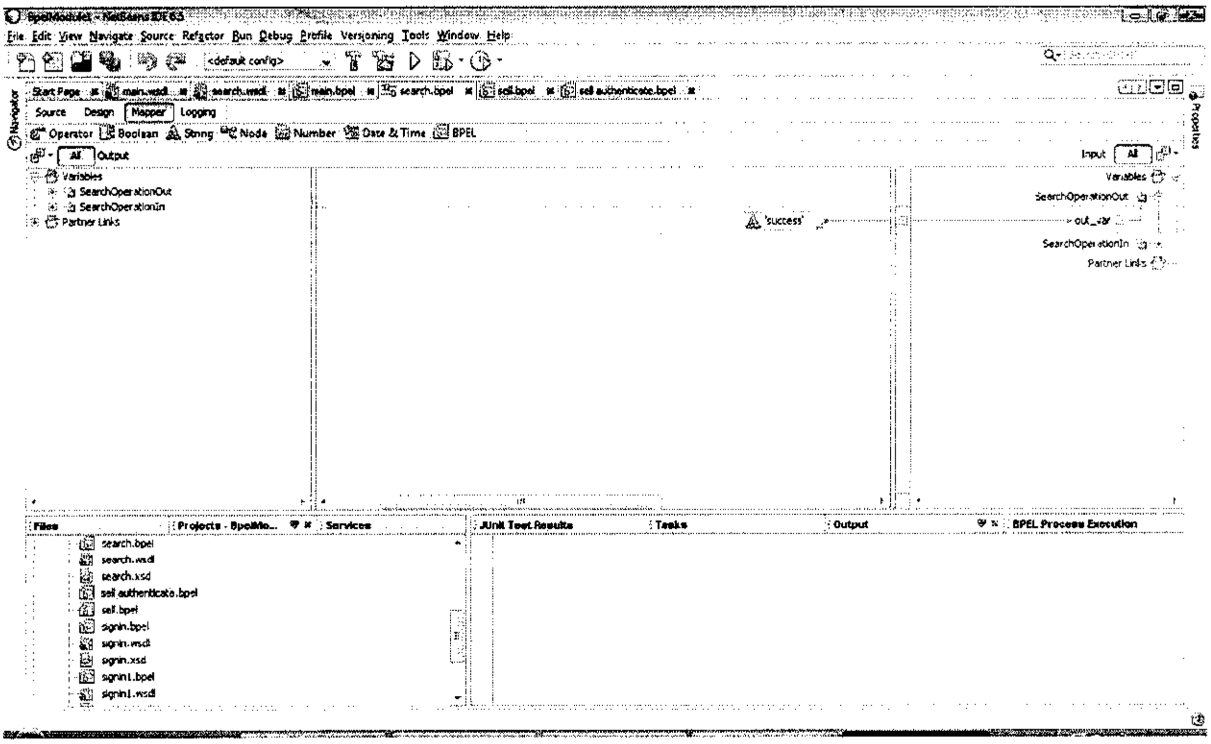
```
            <part name="[illegible]" type="[illegible]"/>
            <part name="[illegible]" type="[illegible]"/>
            <part name="[illegible]" type="[illegible]"/>
            <part name="[illegible]" type="[illegible]"/>
        </message>
        <message name="[illegible]">
            <part name="[illegible]" type="[illegible]"/>
        </message>
        <portType name="[illegible]">
            <operation name="[illegible]">
                <input name="[illegible]" message="[illegible]"/>
                <output name="[illegible]" message="[illegible]"/>
            </operation>
        </portType>
        <binding name="[illegible]" type="[illegible]">
            <soap:binding style="[illegible]" transport="[illegible]"/>
            <operation name="[illegible]">
                <soap:operation/>
                <input name="[illegible]">
                    <soap:body use="[illegible]" namespace="[illegible]"/>
                </input>
                <output name="[illegible]">
                    <soap:body use="[illegible]" namespace="[illegible]"/>
                </output>
            </operation>
        </binding>
        <service name="[illegible]">
```

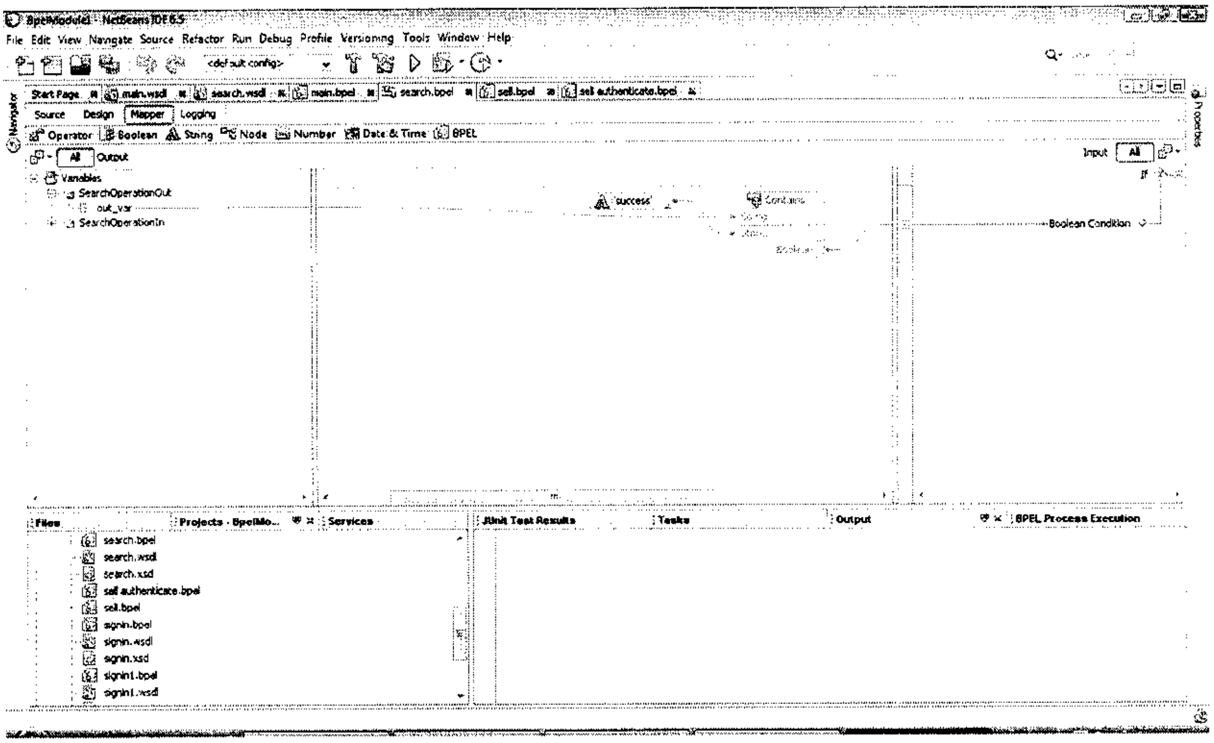
BpelModule1 - NetBeans IDE 6.5
File Edit View Navigate Source Refactor Run Debug Profile Versioning Tools Window Help
<default config>
Start Page
main.wsdl
search.wsdl
main.bpel
search.bpel
Source
WSDL
Partner
Navigator
Properties
1:1
INS
Files
Projects - BpelMo...
Services
JUnit Test Results
Tasks
Output
BPEL Process Execution
servlet
LOGIN.xsd
LOGSTATUS.xsd
entry.wsdl



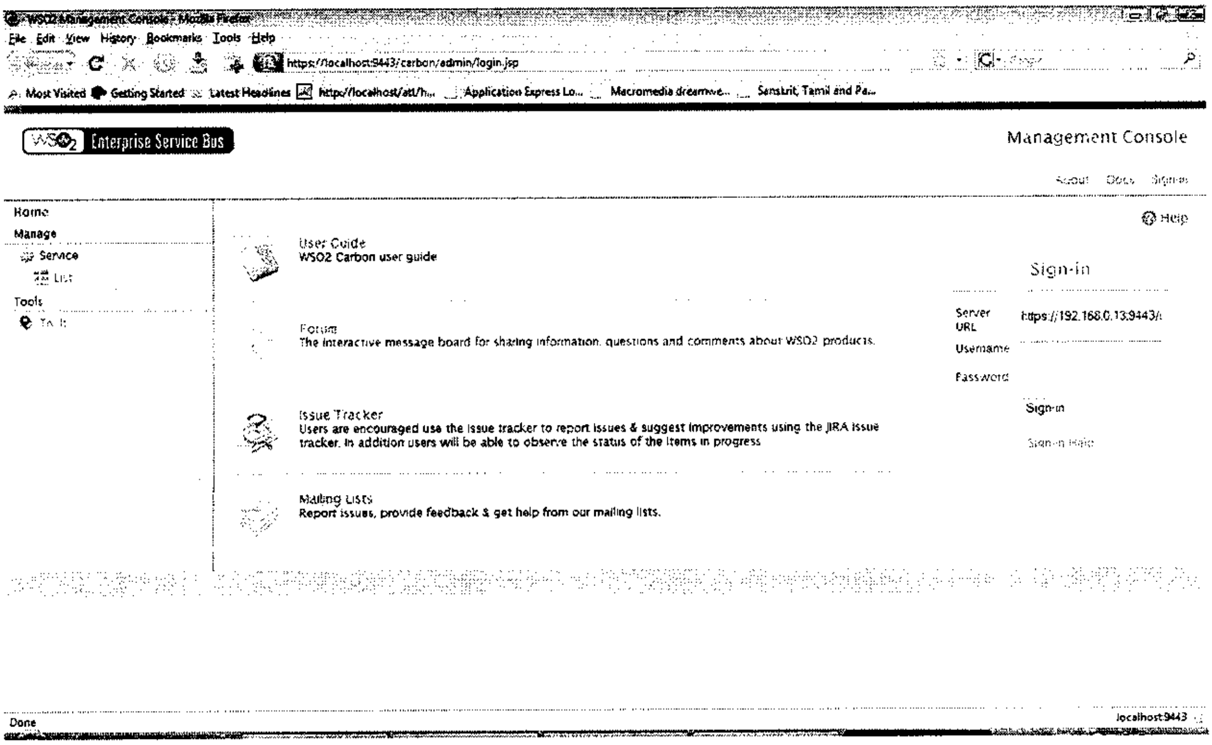
BpelModule1 - NetBeans IDE 6.5
File Edit View Navigate Source Refactor Run Debug Profile Versioning Tools Window Help
<default config>
Start Page
main.wsdl
search.wsdl
main.bpel
search.bpel
sell.bpel
sell authenticate.bpel
Source
Design
Mapper
Logging
Operator
Boolean
String
Node
Number
Date & Time
BPEL
All
Output
Variables
SearchOperationOut
SearchOperationIn
Partner Links
'failure'
Input
All
Variables
SearchOperationOut
out_var
SearchOperationIn
Partner Links
Navigator
Properties
Files
Projects - BpelMo...
Services
JUnit Test Results
Tasks
Output
BPEL Process Execution
search.bpel
search.wsdl
search.xsd
sell authenticate.bpel
sell.bpel
signin.bpel
signin.wsdl
signin.xsd
signin1.bpel
signin1.wsdl


BpelModule1 - NetBeans IDE 6.5
File Edit View Navigate Source Refactor Run Debug Profile Versioning Tools Window Help
<default config>
Start Page
main.wsdl
search.wsdl
main.bpel
search.bpel
sell.bpel
sell authenticate.bpel
Source
Design
Mapper
Logging
Operator
Boolean
String
Node
Number
Date & Time
BPEL
All
Output
Input
Variables
SearchOperationOut
SearchOperationIn
city
type
db_city
db_type
Contains
String
Boolean
While
Boolean Condition
Files
Projects - BpelMo...
Services
JUnit Test Results
Tasks
Output
BPEL Process Execution
search.bpel
search.wsdl
search.xsd
sell authenticate.bpel
sell.bpel
signin.bpel
signin.wsdl
signin.xsd
signin1.bpel
signin1.wsdl
Navigator
Properties


BpelModule1 - NetBeans IDE 6.5
File Edit View Navigate Source Refactor Run Debug Profile Versioning Tools Window Help
<default config>
Start Page
main.wsdl
search.wsdl
main.bpel
search.bpel
sell.bpel
sell authenticate.bpel
Source
Design
Mapper
Logging
Operator
Boolean
String
Node
Number
Date & Time
BPEL
All
Output
Input
Variables
SearchOperationOut
SearchOperationIn
Partner Links
'success'
out_var
Files
Projects - BpelMo...
Services
JUnit Test Results
Tasks
Output
BPEL Process Execution
search.bpel
search.wsdl
search.xsd
sell authenticate.bpel
sell.bpel
signin.bpel
signin.wsdl
signin.xsd
signin1.bpel
signin1.wsdl
Navigator
Properties


BpelModule1 - NetBeans IDE 6.5
File Edit View Navigate Source Refactor Run Debug Profile Versioning Tools Window Help
Start Page main.wsdl search.wsdl main.bpel search.bpel sell.bpel sell authenticate.bpel
Source Design Mapper Logging
Operator Boolean String Node Number Date & Time BPEL
All Output
Variables
SearchOperationOut
out_var
SearchOperationIn
'success'
Contains
Boolean Condition
Input
Files Projects - BpelMo... Services
search.bpel
search.wsdl
search.xsd
sell authenticate.bpel
sell.bpel
signin.bpel
signin.wsdl
signin.xsd
signin1.bpel
signin1.wsdl
JUnit Test Results Tasks Output BPEL Process Execution

ESB

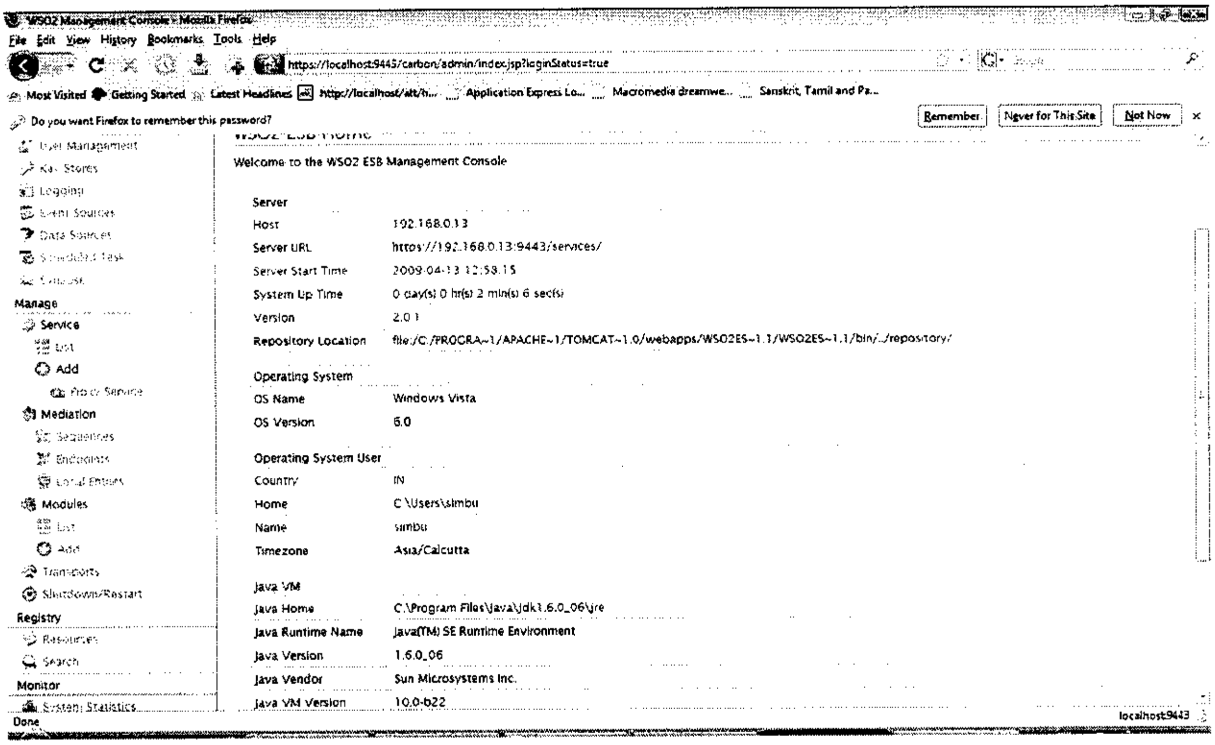

WSO2 Management Console - Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://localhost:9443/carbon/admin/login.jsp
Most Visited Getting Started Latest Headlines http://localhost/atl/h... Application Express Lo... Macromedia dreamwe... Sanskrit, Tamil and Pa...
WSO2 Enterprise Service Bus
Management Console
About Docs Sign-in
Home
Manage
Service
List
Tools
Try It
Help
Sign-in
Server URL
https://192.168.0.13:9443/
Username
Password
Sign-in
Sign-in Help
Done
localhost:9443

User Guide
WSO2 Carbon user guide

Forum
The interactive message board for sharing information, questions and comments about WSO2 products.

Issue Tracker
Users are encouraged use the issue tracker to report issues & suggest improvements using the JIRA issue tracker. In addition users will be able to observe the status of the items in progress

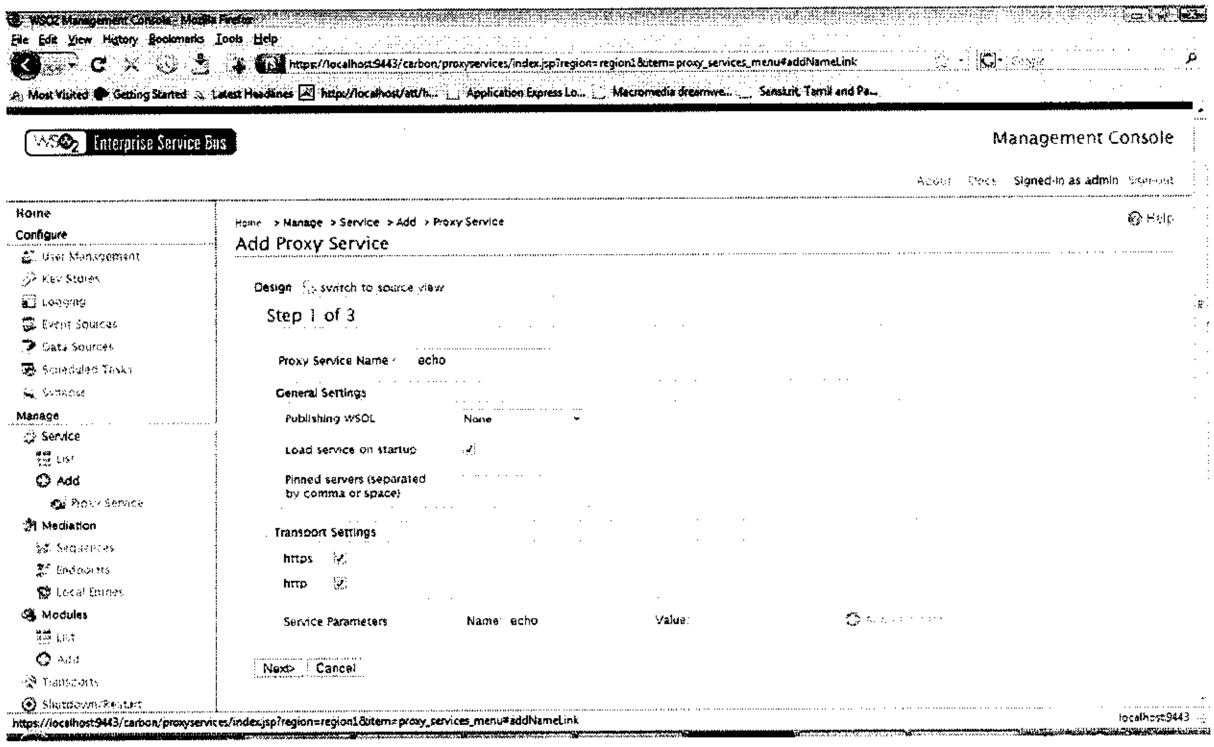
Mailing Lists
Report issues, provide feedback & get help from our mailing lists.



Welcome to the WSO2 ESB Management Console

Server	
Host	192.168.0.13
Server URL	https://192.168.0.13:9443/services/
Server Start Time	2009-04-13 12:58:15
System Up Time	0 day(s) 0 hr(s) 2 min(s) 6 sec(s)
Version	2.0.1
Repository Location	file:/C:/PROGRA~1/APACHE~1/TOMCAT~1.0/webapps/WSO2ES~1.1/WSO2ES~1.1/bin/../repository/
Operating System	
OS Name	Windows Vista
OS Version	6.0
Operating System User	
Country	IN
Home	C:\Users\simbu
Name	simbu
Timezone	Asia/Calcutta
Java VM	
Java Home	C:\Program Files\Java\jdk1.6.0_06\jre
Java Runtime Name	Java(TM) SE Runtime Environment
Java Version	1.6.0_06
Java Vendor	Sun Microsystems Inc.
Java VM Version	10.0-b22

WSO2 Management Console - Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://localhost:9443/carbon/admin/index.jsp?loginStatus=true
Most Visited Getting Started Latest Headlines http://localhost/att/h... Application Express Lo... Macromedia dreamwe... Sanskrit, Tamil and Pa...
Do you want Firefox to remember this password?
Remember
Never for This Site
Not Now
User Management
Key Stores
Logging
Event Sources
Data Sources
Scheduled Tasks
Manage
Service
List
Add
Proxy Service
Mediation
Sequences
Endpoints
Local Entries
Modules
List
Add
Transports
Shutdown/Restart
Registry
Resources
Search
Monitor
System Statistics
Done
localhost:9443



Home > Manage > Service > Add > Proxy Service

Add Proxy Service

Design switch to source view

Step 1 of 3

Proxy Service Name: echo

General Settings

Publishing WSDL: None

Load service on startup: ✓

Pinned servers (separated by comma or space):

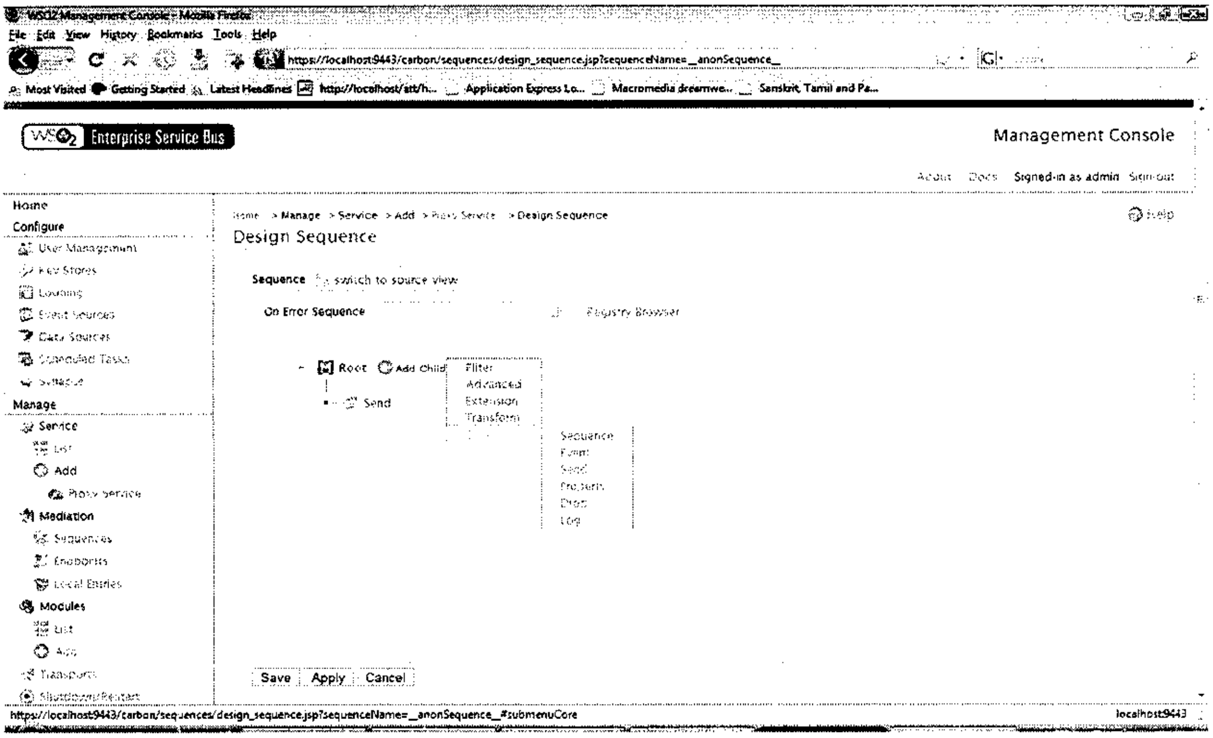
Transport Settings

https: ✓

http: ✓

Service Parameters Name: echo Value:

WSO2 Management Console - Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://localhost:9443/carbon/proxyservices/index.jsp?region=region1&item=proxy_services_menu#addNameLink
Most Visited Getting Started Latest Headlines http://localhost/att/h... Application Express Lo... Macromedia dreamwe... Sanskrit, Tamil and Pa...
WSO2 Enterprise Service Bus
Management Console
About Docs Signed-in as admin Sign-out
Help
Home
Configure
User Management
Key Stores
Logging
Event Sources
Data Sources
Scheduled Tasks
Manage
Service
List
Add
Proxy Service
Mediation
Sequences
Endpoints
Local Entries
Modules
List
Add
Transports
Shutdown/Restart
Next>
Cancel
https://localhost:9443/carbon/proxyservices/index.jsp?region=region1&item=proxy_services_menu#addNameLink
localhost:9443



WSO2 Management Console - Mozilla Firefox

File Edit View History Bookmarks Tools Help

https://localhost:9443/carbon/sequences/design_sequence.jsp?sequenceName=__anonSequence__

Most Visited · Getting Started · Latest Headlines · http://localhost/att/h... · Application Express Lo... · Macromedia dreamwe... · Sanskrit, Tamil and Pa...

WSO2 Enterprise Service Bus — Management Console

About · Docs · Signed-in as admin · Sign-out

Home > Manage > Service > Add > Proxy Service > Design Sequence

Design Sequence

Sequence — switch to source view

On Error Sequence — Registry Browser

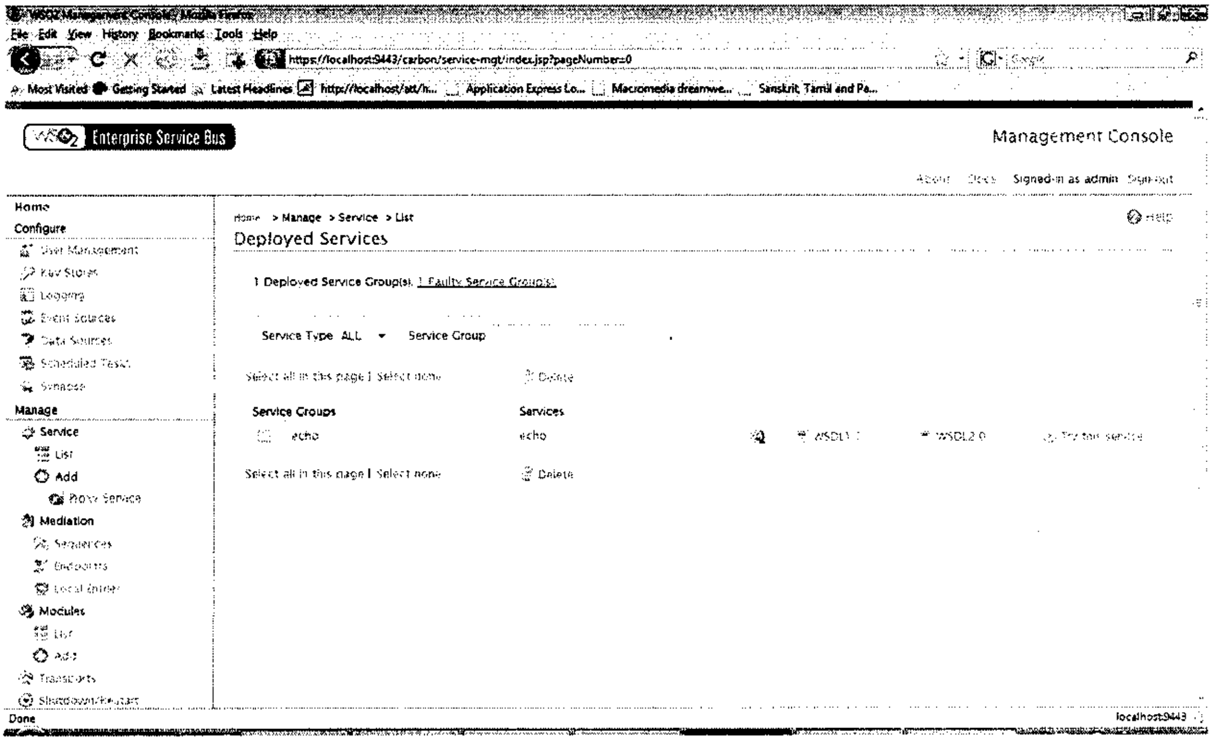
Root · Add Child: Filter, Advanced, Extension, Transform

Send

Sequence, Event, Send, Property, Drop, Log

Save · Apply · Cancel

https://localhost:9443/carbon/sequences/design_sequence.jsp?sequenceName=__anonSequence__#submenuCore



WSO2 Management Console - Mozilla Firefox

File Edit View History Bookmarks Tools Help

https://localhost:9443/carbon/service-mgt/index.jsp?pageNumber=0

Most Visited · Getting Started · Latest Headlines · http://localhost/att/h... · Application Express Lo... · Macromedia dreamwe... · Sanskrit, Tamil and Pa...

WSO2 Enterprise Service Bus — Management Console

About · Docs · Signed-in as admin · Sign-out

Home > Manage > Service > List

Deployed Services

1 Deployed Service Group(s). 1 Faulty Service Group(s).

Service Type ALL · Service Group

Select all in this page | Select none · Delete

Service Groups	Services				
echo	echo		WSDL1.1	WSDL2.0	Try this service

Select all in this page | Select none · Delete

REFERENCES

Websites:

- www.w3.org
  This is the website especially for understanding web services.
- www.SOA.com
  It provides the basic concept of Service Oriented Architecture.
- www.netbeans.org
  It gives the working details of Netbeans 6.5 IDE.
- www.wso2.com
  WSO2 provides its open source registry and ESB.
- www.roseIndia.com
  It is a general website to get the basic details of required concepts.
- www.wikipedia.com
  It is a website to study the basic definitions.
- www.IBM.com
  IBM provides its open source registry and ESB.

Books:

- Shankar Kambhampaty's Service Oriented Architecture for Enterprise applications
- Erik T.Ray's Learning xml
- James Goodwill's Pure JSP -- Java Server Pages - A Code-Intensive Premium reference(SAMS)
- Danny Goodman's Java - JavaScript Bible Gold(2001)
- Ethan Cerami's Web Services Essentials

Papers:

- Siew Poh Lee ,Lai Peng Chan, Eng Wah Lee's Web Services Implementation Methodology for SOA Application
- D. F. Ferguson and M. L. Stockton's Service-oriented architecture Programming model and product architecture
- Christian Emig, Kim Langer, Karsten Krutz, Stefan Link, Christof Momm, and Sebastian Abeck's **The SOA's Layers.**
- Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar and Frank Leymann's ***Service-Oriented Computing Research Roadmap***