Event-driven Service-oriented Architecture for Dynamic Composition of Web Services

by

Zakir Laliwala
(200221003)

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Information and Communication Technology

DA-IICT

Dhirubhai Ambani Institute of Information and Communication Technology, Gandhinagar

2008

Author’s Declaration

This is to certify that

i) The thesis comprises my original work towards the degree of Doctor of Philosophy in Information and Communication Technology at Dhirubhai Ambani Institute of Information and Communication Technology and has not been submitted elsewhere for a degree.

ii) Due acknowledgement has been made in the text to all other material used.

Signature of Student

Certificate

This is to certify that the thesis work entitled “Event-driven Service-oriented Architecture for Dynamic Composition of Web Services” has been carried out by Zakir Laliwala (200221003) for the degree of Doctor of Philosophy in Information and Communication Technology at Dhirubhai Ambani Institute of Information and Communication Technology under my supervision.

Thesis Supervisor Prof. Sanjay Chaudhary

Abstract

The Business process contains a set of services to fulfill its goal. The Service is a software component to perform a specific activity of a business process. The Business processes are event-driven and change frequently during the life cycle of a process. The state of services should be managed for proper integration during the execution of a business process. Core Web services standards are stateless and do not support event and notification. In today's dynamic environment, changes in business process requirements, terminologies, technologies and policies need to be reflected in the software systems. To provide seamless interoperable integration, automation, execution monitoring, state and notification management of a dynamic business process, scalable software architecture is required.

This thesis proposes event-driven service-oriented architecture by converging the Web services, Semantic web, and grid computing; to model, compose, deploy and execute event-driven dynamic business process. Web service provides loosely coupled integration of information and services for orchestration of a business process. Semantic provides interoperable integration, automated orchestration, negotiation, content based service selection and composition of a business process. Grid business process supports state, notification, service grouping, and policy. Grid provides required middleware support for the execution of a stateful and event-driven dynamic grid business process. We propose event calculus based formal approach for event-driven modeling and rules based approach for dynamic composition. As for the proof-of-concept, agro-produce marketing process is considered. Research experiments are performed using existing open standards, specifications, and tools to realize event-driven service-oriented architecture and its lifecycle.

Acknowledgements

First, I would like to thank my guide, Dr. Sanjay Chaudhary, for his motivation throughout my Ph.D. research work. His encouragement and suggestions provided the required foundation to do research. I express my sincere gratitude to him for showing me a path and for helping me in finding and focusing my research work.

I would like to thank Prof Ashok Amin and Dr. Amit Nanavati, for their comments and suggestions to improve the contents, and the writing of this thesis. I am thankful to my institute Dhirubhai Ambani Institute of Information and Communication Technology (DA-IICT) and Prof. Samaresh Chatterji, Dean (Academic) for providing me an opportunity, financial support, and all sorts of resources to carry out my Ph.D. research work.

I owe special thanks to Mr. Vikram Sorathia, Ph.D. candidate, DA-IICT, for helping me out, for being very friendly, kind and supportive. This work would have not been possible without his help and support. I am grateful for his wonderful cooperation and all the interesting discussions about research, publication, original ideas, and life.

Many people have contributed for the successful completion of this research work. Specially, I would like to thank Prateek Jain, Rahul Kosla, Prita Majumdar, Amit Mahajan, Chaitra Vinus and Amee Desai for helping me in this work and becoming co-authors of research papers. I am thankful to my friends, colleagues and students of DA-IICT, with whom I have shared interesting moments of my life.

Last, but not least, I am thankful to my parents for supporting me and encouraging me during my Ph.D. With their love, and support this work has come into existence. I want to thank my wife Aziza and my son Aman for their love, affection, and patience. I would like to thank my both brothers for their support and help when I needed the most.

Table of Contents

List of Figures

List of Code Snippets

Chapter 1

Introduction

An organization refers to structuring of resources to produce and provide goods and/or services. The basic goal of an organization is to produce goods and services and make them available. Entire organization is viewed as a collection of systems. Hospitals, banks, government agencies and online book stores are some examples of organizations. Each organization comprises of a set of systems such as production, marketing, sales, human resources, accounts and stores. Each system is divided into major processes. These processes are subdivided into smaller processes. For example production might be broken into processes such as manufacturing of the product, verification, packaging and shipment of the product. Processes are essential to understand how system operates. Processes play an important role in the design and realization of systems. For successful operation of an organization, these processes must be integrated and work in coordination.

A business process consists of a set of activities to provide a specified service. An Activity is automated action that indicates what is to be done at a particular step in the process. For example, consider the book procurement process of a book store, where a customer needs to order book(s) from a book store (provider). The service provider processes the order and delivers book(s) to the customer. Once the order is processed, payment is made by the customer using credit card or any other mode of payment. To carry out this process, various activities such as purchase order receiving, book availability checking, credit card validation checking, payment and delivery of books are involved.

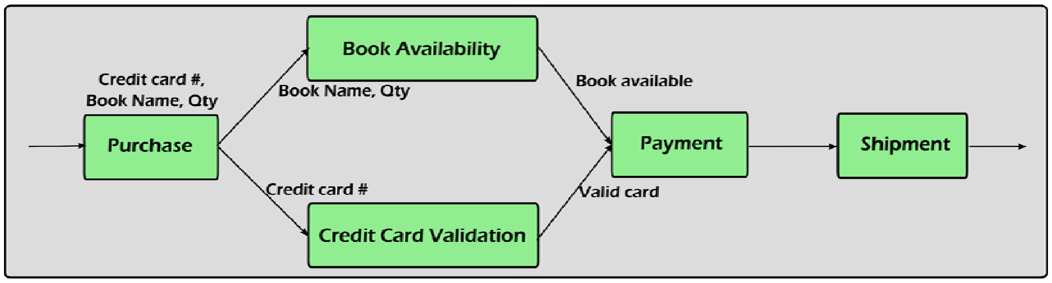


Figure 1: Book procurement process

With the advancement of Internet, organizations express their functionalities in terms of services provided by them. A service requires individual and autonomous unit of activity to be performed. The size and scope of the functionality represented by a service varies. A process is composed of one or more services. Services are performed by a single organization or may interact with services performed by other organizations. For example, book availability checking is performed by book store, while credit card validation is conducted by a credit card company.

Business to business (B2B) interaction provides the connectivity and aggregation of organizations. Business processes interact with each other. For instance, the business processes of a reseller interact with buyer processes. In our book purchase example, if book is not available then book store will order books from resellers as follows.

- The book store sends order to the book reseller.
- The book reseller receives order and starts order processing.
- The book reseller sends an invoice.
- The book store receives the invoice and sends payments. Finally, the book store receives the ordered books.

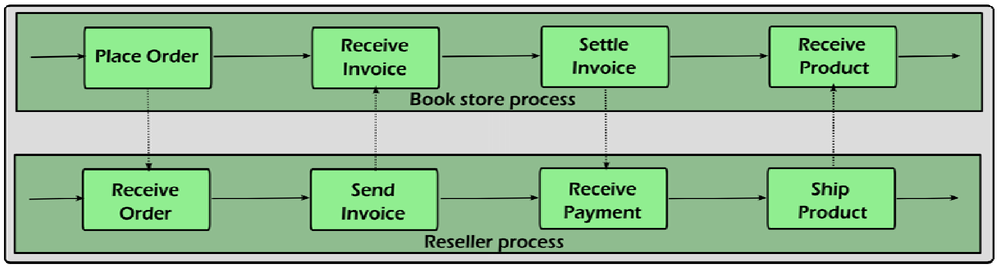


Figure 2: Business to Business Integration

To carry out this process, various parties such as customers, sellers, suppliers, transporters, couriers and various services are involved. The state of the services should be maintained in order to provide the status of a process. For example, customer can check the status of the purchase order and add or delete an item. The overall process also includes various types of policies, service level agreements and events among partners.

Organizations are becoming more collaborative, distributed, and heterogeneous with the advancement of Information and Communication Technology (ICT). As a result, business processes require integration of distributed heterogeneous services, customers, and providers. When more than one service is required, multiple services can be combined as a single composite service [1]. Business processes are event-driven [2] and events affect the execution of a business process. An event is the specification of a significant occurrence that has a location in time and space. An event changes the state of a service during its execution. Business processes are dynamic in nature because interaction policies, strategies, services, providers and consumers change at a runtime. Identification, selection and composition of services at the runtime of a business process are defined as dynamic composition of a process. Complete execution of a process requires dynamic composition and integration of heterogeneous services according to events. Long running and dynamic business processes require support for state monitoring, transaction management, negotiation, and event notifications. Transaction over a composite service is represented as a set of long running business activities. Transaction management is required to coordinate interactions between consumers and providers, to achieve atomicity and to resolve the conflicts occurring during the execution of a business process. Semantic interoperability among services is desirable when a process spans across the boundaries of multiple business organizations, where vocabulary is different. Because of heterogeneity, dynamic nature, and lack of common vocabulary among business partners, a scalable software architecture is essential. Such an architecture should be capable of providing seamless interoperable integration, automation, execution monitoring, state, transaction, and notification management.

This Thesis aims to focus on dynamic nature of business process and propose event-driven service-oriented architecture (EDSOA) based on the convergence of Web services, Semantic web, and grid computing. Web Services[1] (WS) fulfills the functionality of a business process by integrating distributed heterogeneous services. Dynamic nature of a business process is modeled by capturing events using event calculus and composition schema is generated dynamically by using Event-Condition-Action (ECA) rules. Semantic web provides required support for common vocabulary, interoperability, and automation. Grid provides middleware support for execution of a business process with state, transaction, notification, and life-cycle management of a process.

This chapter is organized as follows. In Section 1.1, outline of Service-oriented computing is given. Section 1.2 provides the overview of service composition, its phases and approaches. Section 1.3 outlines the research issues and challenges of services composition. Section 1.4 presents the proposed approach to resolve the issues of composition and list the contributions of this research work. In Section 1.5 the structure of the thesis is given.

1.1 Service-Oriented Computing

The idea behind the Service-oriented Computing (SOC) is to provide platform and language independent development of software components and applications. The development and acceptance of open standard, framework and languages are the key concepts behind the success of SOC. Services are the key component elements to provide loosely coupled service centric computing. Emerging service-oriented concepts and technologies like Web services, grid computing and Semantic web are seen as efficient and relevant paradigms to integrate computational resources together to provide access to information and processing capability anytime anywhere [3].

Web services has emerged as the next generation of web technology to publish, discover, and invoke the software component as services. It is standardized by World Wide Web Consortium (W3C). As per W3C, Web services is described as: “a software application identified by a Universal Resource Identifier (URI), whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web service supports direct interactions with other software agents using XML-based messages exchanged via Internet-based protocols” [4]. It

[1] In this thesis, the term Web services is referred as a singular term.

provides loosely coupled, interoperable integration of web hosted services and access to wide range of computing devices. Web services is based on three core protocols: Web Services Description Language (WSDL) [5] to describe the service, Simple Object Access Protocol (SOAP) [6] is a communication protocol to access the service, and Universal Description, Discovery, and Integration (UDDI) [7] to register and discover services from the registry. Web services and Service-Oriented Architecture (SOA) are promising paradigms for development of enterprise applications [8]. SOA is software architecture to enable loosely coupled integration and interoperation of distributed heterogeneous systems by using services as component elements.

Grid computing becomes known for wide-area scientific and enterprise applications. It allows runtime selection, integration, and coordination of distributed resources to accommodate dynamic business requirements [9]. It gives scalability and flexibility by following open standards, protocols, and technology like Web services. Modern grid [10] is based on Open Grid services Architecture (OGSA) and Web services Resource Framework (WSRF) [11]. WSRF bridges the gap between Grid services and Web services. This new development makes grid suitable for various kinds of applications like collaborative computing, ubiquitous computing, multimedia applications, and enterprise applications [8, 12, 13].

Semantic web comes as an answer to provide semantic interoperability and automated machine processable system. It is as an extension of current web to provide well defined meaning to the information [14]. It uses ontology to define the concepts of a domain of interest. Semantic approach helps in search, discovery, selection, composition and integration of Web services and also in automation of invocation, composition and execution of services. Service-oriented computing utilizes the Semantic web to provide interoperability and automation benefits to enterprise applications.

1.2 Service Composition

Service composition is a very important service-oriented principle and is known as service assemblies. Composition provides new functionalities by creating new services from existing and distributed services. From a business perspective, business service represents distinct business logic. Service composition is comprised of a set of independent business services. Service itself composes other services and can call other services to accomplish its work. Therefore, each service that participates in composition performs its individual role. Service-orientated principles promote composability. Therefore, services should be designed by keeping composition in mind, so that they can be used for future service compositions.

Service composition contains various phases and it is a complex and challenging task. It is rapidly gaining attention and many solutions and different approaches have been proposed.

1.2.1 Phases of Composition

Following phases are involved in development of services and service composition.

Service Creation: First of all, the service provider should create the service and publish the interface with the description of a service. The description should provide the information about functional and non-functional requirements of a service. It should provide information about methods, inputs, outputs, exceptions, data types and transport mechanisms. Non-functional parameters are related to quality of service, response time, and cost. These parameters are required to evaluate the service.

Service Discovery: Provider should advertise or publish the capabilities of each service, so that later on service consumers can discover any of these services as per their requirements and then communicate with service provider to access discovered services. As number of services increase, discovery and selection becomes complex and essential phase. Directories and registries should be able to manage different versions of services and help to discover the services based on the specified criteria.

Service Composition: When requirement of a requester is not satisfied by a single service, composite service can be created with a set of atomic services. The functionality of atomic services combines by defining the control and data flow and mapping input and output of services. Each service should be designed with proper granularity, so that it can participate in composition.

Service Selection: Similar or identical functionality may be provided by many service providers. A huge set of services, providing identical functionality, may exist. The best available services should be discovered and selected to generate a composite service. Selection is influenced by quality, functional and non-functional requirements of the service.

Execution and Monitoring of Composite Service: After the selection of the best composite service, deploy the components and services and configure the middleware for the execution. Services will be executed in the sequence as specified in the composition schema by passing messages and data from output of a service as input to the next service. Monitoring is required to keep the track of execution of a process, usage of services and exception handling.

1.2.2 Approaches of Services composition

Service composition approaches can be classified in two categories: time based and human intervention based. Based on the time, it can be categorized into two types: static (design time) composition and dynamic (run time) composition [15-17]. Human intervention based composition can be categorized into two types: manual (human driven) and automated (machine driven) [15].

Static composition is an approach, where business processes, business partners and services are known at design time and do not change frequently. Application designer will manually generate the composition schema by selecting and integrating services at design time. Static composition is useful to provide complex interaction pattern among known components. Leading commercial products such as Oracle BPEL Process Manager, IBM WebSphere Business Modeler, BEA Web logic, Microsoft Biztalk are supporting static composition.

Dynamic composition is an approach, where business partners, consumers, and services are changing at runtime. New and better services may become available and partner policies are likely to change dynamically. Business process should be flexible and adaptable and should provide service selection based on user requirements and context. The service composition schema is generated dynamically and it does not require human intervention for composition. Therefore, dynamic service composition is useful for applications where components, services and users are dynamic, such as mobile computing, grid computing and ubiquitous computing. SWORD [18], eFlow [19], and StarWSCoP [20] are few examples of dynamic service composition systems.

Manual composition is an approach, where designer (human) can manually design or model the workflow and interactions among components for generating the composition schema. Business Process Execution Language (BPEL) [21] is an example of manual composition approach. It is a lower level process modeling and execution language where designer design the flow using control constructs like if-then, switch case, fork, while-loops etc. It is a widely accepted standard for manual composition of services.

Automated composition is a semantic based approach, where it processes the data and generates the composition schema. Semantic web gives well defined meaning of information and makes it machine processable. It uses ontology to provide description of functional and non-functional properties of services, to carry out automated discovery, selection and composition. Several research efforts both in academia and in industry have proposed a number of automated planning and semantic based techniques for automated composition (SHOP2 [22], Medjahed et al. [23], Berardi et al. [24]). OWL-S [25] and WSDL-S [26] are known standards to provide semantic in Web services composition.

Model driven service composition is Unified Modeling Language (UML) [27] based approach. Due to lack of dynamic and adaptive composition support in BPEL, model driven approach is proposed and it is similar to business rule driven service composition. It uses UML to provide higher level of abstraction and Object Constraint Language (OCL) based business rules to describe process flow. This approach can be static or dynamic. Orriens et al [28] and Zhang et. al [29] have introduced model driven dynamic web services composition.

Adaptive service composition is a dynamic composition approach, where composite services need to be adaptive. It should be capable to adjust according to the changes in the environment, requirements and the context of a user. It is an advanced service composition approach, where service selection and composition is done dynamically based on user context, constraints and preferences. Complex applications also support negotiations to provide optimal solution. eFlow and Ardagna et. al [30] are based on adaptive composition approach.

1.3 Research Issues and Challenges of Service Composition

Web services standards, specifications and languages do not support all the workflow patterns. Core Web services is stateless and does not have notation for event and notification. Standards and specifications of Web services are syntax oriented and lack semantic support to make the business processes machine processable.

Services composition is a very important aspect to realize enterprise integration and business process. For efficient composition of services, descriptions of functional and non-functional properties of services are to be defined. Business process modeling language to model the business process and to generate the composition schema as per the events is required. Runtime orchestration of the services should be based on the events and the requirements of a process under execution. Dynamic generation of composition schema, dynamic selection of services and runtime life-cycle management of a business process is needed. To achieve state, notification and execution monitoring during the execution of composite services, middleware support is required. Issues related to Web services and service composition are identified as under:

- **Service Description and Discovery:** For efficient composition, discovery, and selection of services, description of functional and non-functional properties of services are required. Interoperability, incompatible vocabularies and semantic interoperability among service providers and consumers should be resolved.
- **Modeling of Composite Service:** Business process modeling deals with design and execution of a business process. Modeling language needs to model the business process as per the events. Language needs to specify the flow of a process, services to be combined and the order in which services are to be executed. It also specifies the parameters, conditions, and events required for invoking a service. For dynamic business process, modeling language should provide support for dynamic schema generation, dynamic service selection and runtime life-cycle management of a business process.
- **Adaptive Composition:** Composite service should be adaptive to the changes (state, event and execution) likely to occur during the execution. It should be flexible to adjust to the dynamic enterprise environment. Enterprises are changing constantly, new service provider with better Quality of Service (QoS) may become available at any time, old or

previous version of services may be removed, existing services may withdraw execution or throw an exception during the execution. Business process should have the ability to manage such changes. The required support should be available in the form of middleware.

- **Dynamic service composition:** B2B interaction with dynamic and automated business processes is quite challenging. Most of the real world business processes keep changing. Dynamic business processes require dynamic discovery of services, dynamic service selection, and dynamic schema generation. Business processes behave according to various rules and policies. For generating composition schema at runtime, rules, and policies need to be considered. Once the composition schema is generated, services should be selected on the basis of various criteria such as functional and non-functional requirements, QoS, and consumer preferences. Services are distributed and require constant monitoring for exception handling, state, event and execution management.
- **State Management:** Web services are fundamentally stateless. Composite service is a series of services, where result of one service depends on prior service(s) and/or prepares for a subsequent service. A service acts upon stateful resources based on messages it receives and sends. It uses messages to determine the processing behavior of a process. Business process involves the transaction, which again constantly initiates the changes in the state of a process under execution. Business processes require runtime coordination, asynchronous integration, notification mechanism and state and transaction management. Core Web services standards lack the notion of state, stateful interactions, resource lifecycle management and notification of state changes. Web services themselves are not capable enough to provide required functionalities and various specifications have been proposed to achieve missing functionalities within Web services. Flexible and reliable ACID (Atomicity, Consistency, Isolation, Durability) transaction in a long running process and loosely coupled environment is a challenging issue [31, 32].
- **Execution Monitoring:** Execution of a business process requires scalable software architecture with workflow execution monitoring support. Web services are published and maintained by respective organizations. For the complete execution of a business

process, a common controlling mechanism to monitor the execution and lifecycle of a process is required. Again conflicting standards and specifications and lack of middleware support have raised challenges to resolve these issues.

- **Event and Notification:** Business services are event driven. Different specifications and mechanisms have been proposed to achieve eventing and notification in SOA. Two major specifications exist: WS-Notification and WS-Eventing. Microsoft has published WS-Eventing [33] and IBM and HP have published WS-Notification as a collection of specifications to address the same problem. These specifications use different approaches and terminology to address the same issues. These two specifications provide overlapping features. Apart from these specifications, Web Services stack [34] is flooded with numerous specifications related to routing, addressing, reliable messaging, transaction, orchestration etc. While developing a real life application, the challenge is to carefully select these specifications and to ensure their coherence after the implementation.
- Apart from these broader issues, issues related to coordination, transaction, security and performance are likely to arise due to dynamic and heterogeneous nature of composite service. Challenges such as inability to ensure scalability, robustness, and QoS related issues of Web services make it unfit for mission-critical and certain kinds of business applications [35-38].

1.4 Proposed Approach

Semantic web has originated from an Artificial Intelligence (AI) domain to provide knowledge-centric computing environment [39]. Grid computing provides support for data and computation intensive large-scale distributed computing system. Web Services and SOA aims to provide language and machine independent, loosely coupled services and architecture for integration of distributed heterogeneous components. Researchers and developer found relationships among these technologies to achieve collaboration, cooperation among scattered components with flexible and scalable open standard based global scale architecture.

1.4.1 Convergence of Web services, Semantic web and Grid computing

With the advancement of Web services and Semantic web, recent research merges these two paradigms as Semantic Web Services (SWS). Semantic Web Services aims to automate discovery, composition, invocation, and execution of Web services and to make Web services machine processable and interoperable [40]. Various semantic markup languages and standards have been proposed for the annotation of Web services. Among important standards; OWL for Services (OWL-S) [25], Web Service Modeling Ontology (WSMO) [41] and WSDL-S [26] are the known standards.

Global Grid Forum's (GGF) proposed OGSA as a convergence of SOA and grid. OGSA uses Web services as a core component to expose its core functionalities and combine the SOA and grid computing for business and scientific applications. With the replacement of Open Grid Service Infrastructure (OGSI) with WSRF, OGSA is now based on Web services standards such as WSRF, WS-Notification [42], and Web Services Distributed Management (WSDM) [43]. WSRF is proposed as a standard to converge Web services and Grid service [3].

The Semantic Grid [44] is described as an extension of grid computing where metadata is used to describe resource, information and services of the grid. Semantic helps in discovery, sharing and collaboration of resources. It also helps to achieve automation, to make services and process machine processable, and to enable cooperation between man and machine. Semantic provides knowledge in a grid environment, where intensive data and information integration are involved. At present Semantic grid is evolving and it is at experimental level. It lacks support in terms of framework, standards and architecture.

We plan convergence of Web Services, Semantic web and grid computing to resolve the issues of enterprise applications as shown in figure-1. Web services provides loosely coupled integration of scattered services. Semantic provides the knowledge and vocabulary of a domain and rules to design composition of Web services. Grid provides middleware support to achieve state, transaction, notification, execution, monitoring, and scalable enterprise architecture. We propose EDSOA to facilitate dynamic composition, negotiation, state, transaction, notification and middleware support for the automation of business processes as shown in figure-15.

1.5 Contributions

There exist various approaches for service composition but the aim of this thesis is to propose EDSOA to achieve event-driven dynamic Web services composition with the convergence of three emerging paradigms: Web services, Semantic web and grid computing.

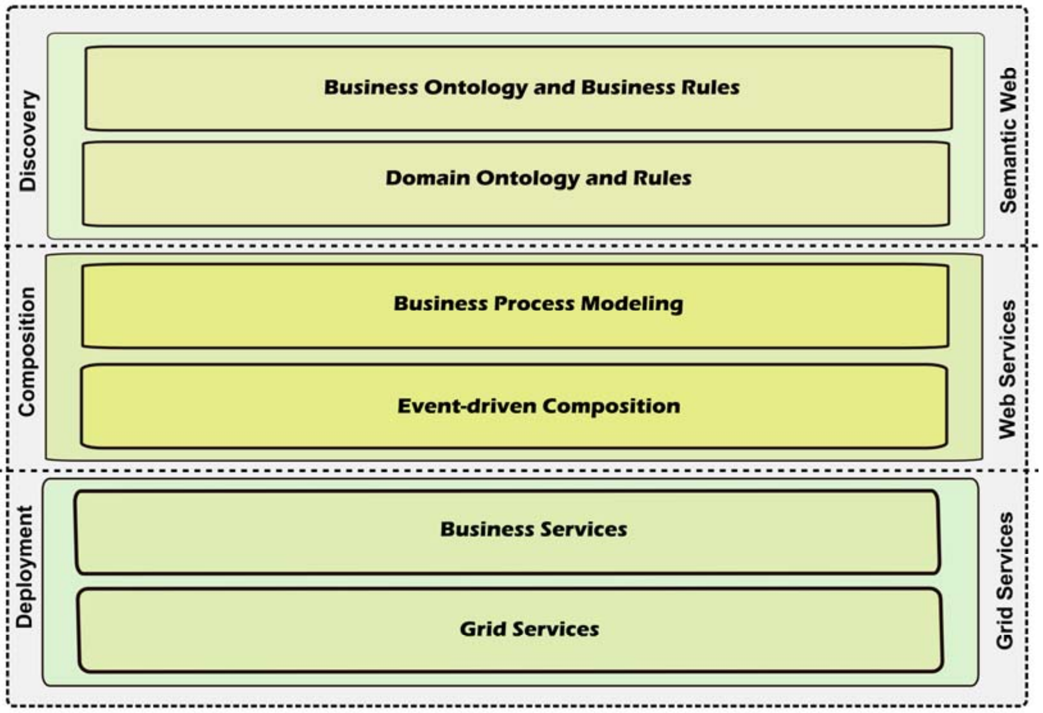


Figure 3: Convergence of Web services and Grid computing

More precisely, the major contributions of this thesis are as under:

- Event-driven Service-oriented Architecture (EDSOA) [45-47]: As an extension of SOA we have proposed EDSOA to capture the requirements of event-driven dynamic business process and to achieve event-driven dynamic composition of a business process.
- Realization of EDSOA [47]: We have proposed convergence of Web Services, Semantic Web, and Grid computing to achieve EDSOA. Web services provides loosely coupled integration of information from distributed sources. It also provides orchestration of heterogeneous services [48, 49]. Semantic provide interoperability for orchestration of a business process. Ontology and rules are used for event correlation, automated negotiation and discovery, and delivery of personalized context and location based recommendation [50-52]. Grid provides middleware for workflow execution and to

achieve state, transaction, notification, execution, monitoring and scalable enterprise architecture [45, 53].

- Modeling and Composition [46, 54]: We have proposed Event Calculus based approach to model the event-driven business process. Dynamic composition schema is generated based on events using ontology, rules, backward and forward chain algorithm.
- Grid Business Process [55]: A business process is likely to span across various distributed services. Development, deployment and execution of integrated services come with the challenge of its inherent heterogeneity. The software and hardware infrastructures are also heterogeneous. Proposed standards and specifications are conflicting, not yet matured and face many difficult challenges. We have proposed Grid Business Process to fulfill dynamic business process requirements.
- Service Grouping and Group Notification [56]: Business processes require integration with distributed heterogeneous services. Business processes are running in parallel and interacting with multiple services, partners and customers as per the requirement and policy. There is a need to aggregate information from multiple resources or services, to provide better query, search and group notification. We have proposed event-driven service grouping and group notification of stateful services.
- Policy-driven Grid Business Process [57]: There are a number of factors that both service provider and service consumer should consider before they interact with each other. In Web service selection phase both functional and non-functional requirements need to be considered. Composition of services requires dynamic services discovery and dynamic service selection. Service discovery and selection are depending on business services metadata, policy and event associated with the services. There is a need of dynamic services selection based on runtime environment such as content (semantics), context (event) and contract (policy). We have proposed event-driven dynamic services selection based on event, policy and semantic. Dynamic service selection will help in dynamic composition of business process and to deliver the efficient services to consumer as per their business context and request.

- For implementation of end-to-end EDSOA, we have evaluated different strategies of SOA. We have tested the capabilities of Web service, Semantic web and Grid computing standards and specifications. We have shown the effective integration of different tools and techniques for the development of enterprise application.

1.6 Organization of the Thesis

This thesis proposes an EDSOA and event-driven dynamic composition methods to provide state and notification management during the execution of business processes. An outline of the organization of this thesis is shown as follows:

Chapter 2 provides an overview of concepts of Service-oriented computing, standards and specifications related to Web services, Semantic web, Grid computing and Web services composition.

Chapter 3 discusses the proposed event-driven SOA lifecycle, EDSOA and realization of architecture using Agro-produce marketing use case scenario.

Chapter 4 presents a generic approach of event calculus based event-driven modeling and ECA rules based approach for event-driven dynamic composition and automatic schema generation.

Chapter 5 describes the concepts and implementation of our approach for grid business process. It presents the usage of grid to achieve the required middleware support for the execution of workflow and to achieve state and notifications during the execution of a business process. It also discusses the implementation of prototype system and development of grid business services using existing standards, protocols, and tools.

Chapter 6 shows the deployment and execution of grid business process prototype and discusses experiments and results.

Chapter 7 summarizes our work and discusses the future direction of this research work.

Chapter 2
Concepts, Standards and Related Technologies

Service-oriented Computing is based on open standard, open framework, platform independent and language independent technologies. This chapter gives an overview of concepts of Service-oriented computing, namely, Web services, Semantic web, grid computing, service composition standards, and specifications. The SOA concepts and the main research work published in the literature are explained in this chapter.

Section 2.1 gives an introduction of SOA. Section 2.2 gives overview of core Web services standards and section 2.3 discusses second generation Web services standard. Sections 2.4 and 2.5 provide overview of standards related to Semantic web and Semantic web services. Section 2.6 discusses the new development in the area of grid computing and section 2.7 presents the related work, standards for Web services composition and discusses the most widely used composition standards and benefits.

2.1 Introduction of Service-Oriented Architecture

Service-oriented Architecture is a conceptual architecture and a distinct architecture design approach to design and develop business information system. SOA represents an open, flexible, composable architecture that uses services as core components. Services are autonomous, interoperable, discoverable and reusable software components. Following are some of the key requirements imposed by SOA principles [58].

2.1.1 Principles of Service-oriented Architecture

Loose coupling: Service provider and consumer should maintain a high-level contractual relationship to reduce dependencies and tight coupling.

Platform independence: Service provider and consumer should be independent of the implementation, programming language and hardware of the interacting components.

Discoverability: Service should be described by the service contract so that it can be discovered, accessed and reused.

Flexible configuration: Service should be flexible and composable so that they can be combined with each other to generate composite service.

Reusability: Services should be designed at a higher abstraction level. It should be coarse grained to reduce dependencies and to promote reuse.

SOA can be classified in different ways, based on the method that we used to develop services. Web services are becoming more popular and widely accepted standard to develop services for SOA. Web service is defined as a distributed software component accessible over the web through Uniform Resource Locators (URL). As per the online technical dictionary Webopedia, The term Web services is described as: “a standardized way of integrating Web-based applications using the XML, SOAP, WSDL and UDDI open standards over an Internet protocol backbone. XML is used to tag the data, SOAP is used to transfer the data, WSDL is used for describing the services available and UDDI is used for listing what services are available” [59]. WSDL, SOAP, and UDDI standards are known as a first generation Web services standards or core Web services standards. Web services community has done significant research work to extend the first generation standards to provide more functionality such as security, reliability, interoperability, transaction management and distributed management. In the following section both first and second generation Web services standards are discussed with details.

2.2 Core Web Services Standards

Web services is a new paradigm to provide services on web. Various standards and specifications related to Web services are published to support required functionalities. The three protocols (SOAP, WSDL and UDDI) complete the triangle (publish, find and bind) of Web services domain as shown in figure-4. Service provider creates service and publishes it using WSDL by registering in services registry. Service consumer finds the service in the registry using UDDI and then interacts with service provider using SOAP, to bind and access the service. These core standards are accepted by the World Wide Web Consortium (W3C) and supported by all the vendors. Besides above-mentioned core standards, additional standards are under development, which are referred as WS-* or Web services second generation standards to support other functionalities such as security, reliable messaging, transaction, state, notification and so on. In this section we have briefly described the core Web services standards.

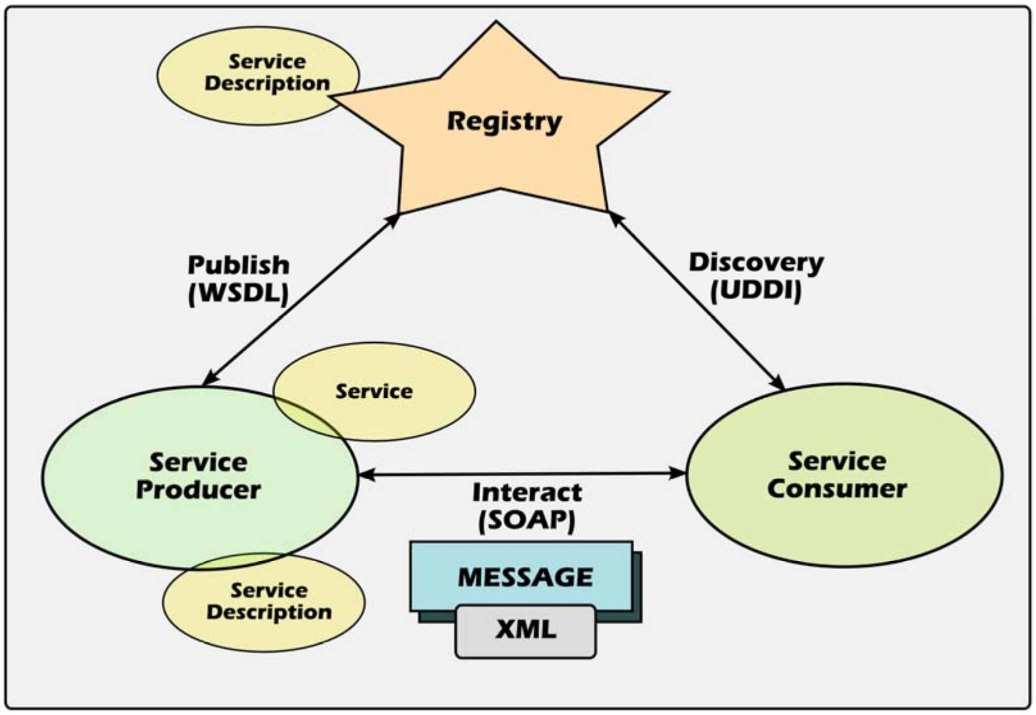


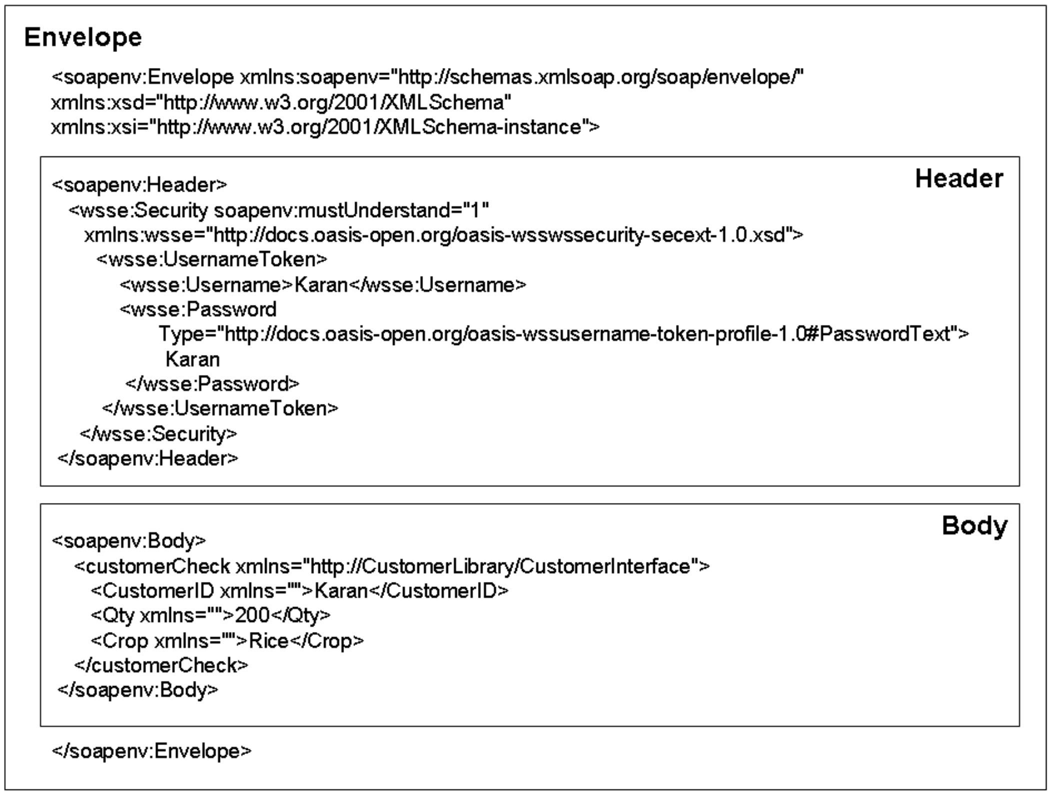
Figure 4: Web Service Interaction Pattern

2.2.1 SOAP

Simple Object Access Protocol (SOAP) [6] is an XML-based protocol to define message exchange mechanism to invoke the remote services. It describes how to package information in a structured and typed manner using XML, so that it can transmit over the web. It provides simple, stateless, one-way and lightweight mechanism to transmit information over various transport protocols (HTTP, SMTP, and SIP) and different platforms. SOAP supports one-way asynchronous messaging and two-way synchronous messaging interaction pattern. Interaction pattern (document-style or RPC-style) should be encoded within the SOAP document.

SOAP used concepts of envelope to specify the message format. SOAP message is an XML document that consists of three parts: envelop as root element which contains optional header and mandatory body part. Header adds extended processing and controlling information and body is the last element of the message which receives the final message and processes it.

Header block is a child element of a header element. WS-* specifications are using header block to provide new messaging features.


Envelope
Header
Body

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<soapenv:Header>
  <wsse:Security soapenv:mustUnderstand="1"
    xmlns:wsse="http://docs.oasis-open.org/oasis-wsswssecurity-secext-1.0.xsd">
     <wsse:UsernameToken>
        <wsse:Username>Karan</wsse:Username>
        <wsse:Password
              Type="http://docs.oasis-open.org/oasis-wssusername-token-profile-1.0#PasswordText">
               Karan
         </wsse:Password>
      </wsse:UsernameToken>
   </wsse:Security>
 </soapenv:Header>

<soapenv:Body>
   <customerCheck xmlns="http://CustomerLibrary/CustomerInterface">
      <CustomerID xmlns="">Karan</CustomerID>
      <Qty xmlns="">200</Qty>
      <Crop xmlns="">Rice</Crop>
   </customerCheck>
 </soapenv:Body>

</soapenv:Envelope>
```

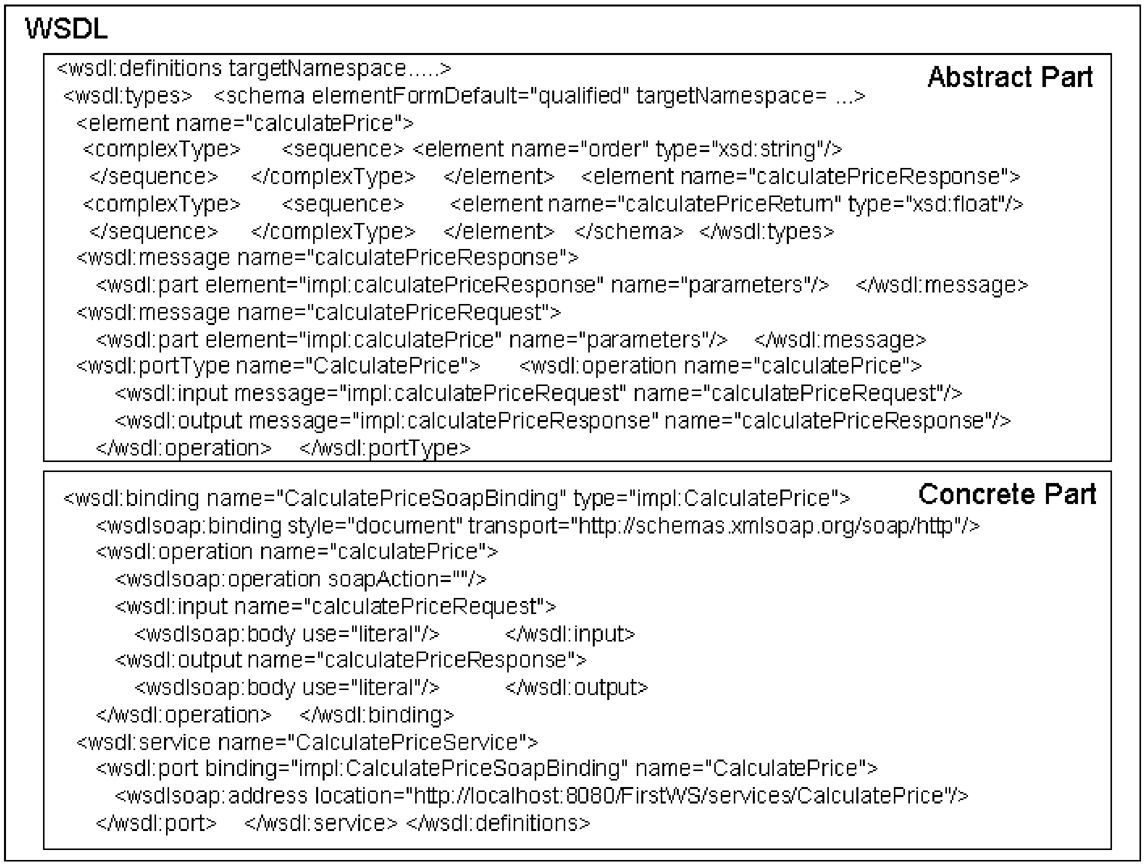
Figure 5: SOAP Envelope

2.2.2 WSDL

Web Services Description Language (WSDL) [5] provides mechanism to describe the service in terms of interface. Before invoking a service, we need to know certain information regarding service like, location of a service, input parameters to invoke service and message passing transport protocol. All of this information is provided by WSDL. WSDL is an XML-based standard and supported by all vendors. It provides information to others about services, such as what services can do and how to access the services over SOAP and other protocols. Service provider publishes services in the registry to describe the functions performed by services and information about messages to be sent and received by these services. It provides loosely coupled and flexible integration model of services.

WSDL is made up of two parts: Abstract part and concrete part. Abstract part describes the operational behaviour of services by providing receive and send messages. It consists of type, message and port type elements. Concrete part describes how and where to access a service implementation. It contains binding and services elements.

WSDL represent information in two ways either document-oriented or RPC oriented. WSDL supports wide range of message interaction patterns like: one-way, request-response, notification, solicit type. WSDL document provides syntactic or structural terms but does not provide semantics of messages which are exchanges by services. To provide semantic support in WSDL, semantic web services standards and language like DAML-S, OWL-S and WSDL-S have been proposed, which are discussed in section 2.5.


WSDL
Abstract Part
Concrete Part

```
<wsdl:definitions targetNamespace.....>
 <wsdl:types>   <schema elementFormDefault="qualified" targetNamespace= ...>
   <element name="calculatePrice">
    <complexType>      <sequence> <element name="order" type="xsd:string"/>
     </sequence>    </complexType>    </element>    <element name="calculatePriceResponse">
    <complexType>      <sequence>      <element name="calculatePriceReturn" type="xsd:float"/>
     </sequence>    </complexType>    </element>   </schema>   </wsdl:types>
   <wsdl:message name="calculatePriceResponse">
     <wsdl:part element="impl:calculatePriceResponse" name="parameters"/>    </wsdl:message>
   <wsdl:message name="calculatePriceRequest">
     <wsdl:part element="impl:calculatePrice" name="parameters"/>    </wsdl:message>
   <wsdl:portType name="CalculatePrice">      <wsdl:operation name="calculatePrice">
       <wsdl:input message="impl:calculatePriceRequest" name="calculatePriceRequest"/>
       <wsdl:output message="impl:calculatePriceResponse" name="calculatePriceResponse"/>
     </wsdl:operation>    </wsdl:portType>
```

```
 <wsdl:binding name="CalculatePriceSoapBinding" type="impl:CalculatePrice">
     <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
     <wsdl:operation name="calculatePrice">
       <wsdlsoap:operation soapAction=""/>
       <wsdl:input name="calculatePriceRequest">
         <wsdlsoap:body use="literal"/>         </wsdl:input>
       <wsdl:output name="calculatePriceResponse">
         <wsdlsoap:body use="literal"/>         </wsdl:output>
     </wsdl:operation>    </wsdl:binding>
   <wsdl:service name="CalculatePriceService">
     <wsdl:port binding="impl:CalculatePriceSoapBinding" name="CalculatePrice">
       <wsdlsoap:address location="http://localhost:8080/FirstWS/services/CalculatePrice"/>
     </wsdl:port>    </wsdl:service> </wsdl:definitions>
```

Figure 6: Sample WSDL document

2.2.3 UDDI

Universal Description, Discovery, and Integration (UDDI) [7] standard specify the mechanism to register and find the services. It provides common plan to publish the services and to search the services either manually or programmatically. It provides programmatic and GUI interface to search and query the registry. It gives the list of the existing services and the information or pointers to access the particular service. It helps to realize the benefits of Web services such as reuse of services, dynamic selection, runtime binding, and composition of services.

UDDI registry can be public or private. It comprises of four types of descriptions: business entity, business services, binding templates, and tModels. Business entity describes basic information about the service provider. Business service describes one or more services offered by the business entity. Binding template describes binding information to use a particular service. tModel provides pointer to the service description in case of Web services.

UDDI contains six sets of Application programming interfaces (API) for users to publish services, search services, and to exchange information. It contains Publisher API, Inquiry API, Subscription API, Security API, Replication API and Custody and Ownership Transfer API.

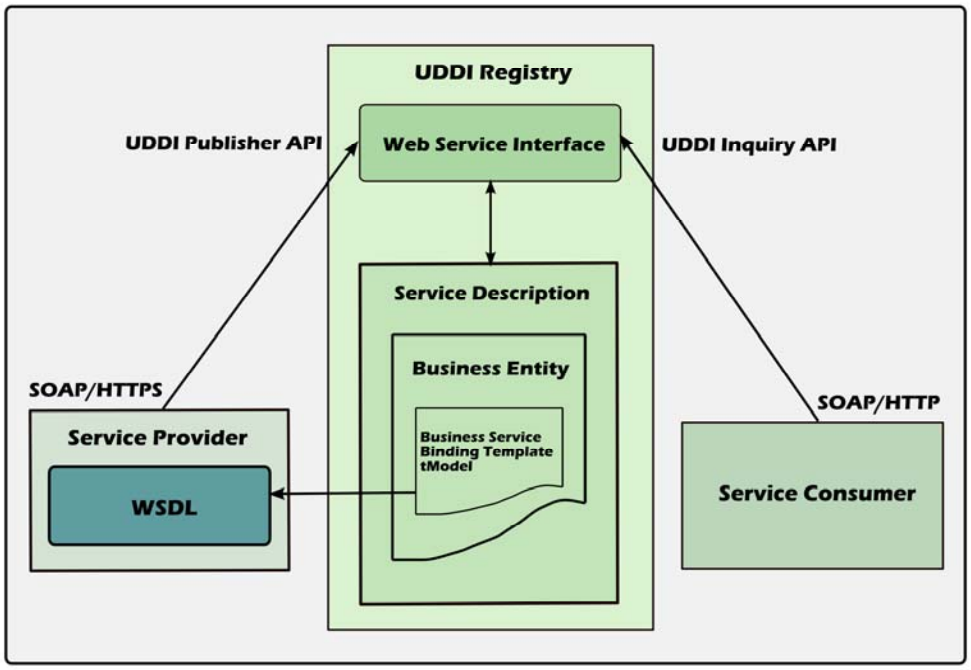


Figure 7: SOAP, WSDL and UDDI Interaction

2.3 WS-I* Standards and Specifications

WS-* is a second generation Web services specifications built upon the core WS standards as shown in figure-8. The reason behind these specifications is to add more functionality to SOA domain and make it viable solution for large distributed enterprise applications. It adds the specifications to achieve transaction, notification, execution management, coordination, composition, security and reliability.

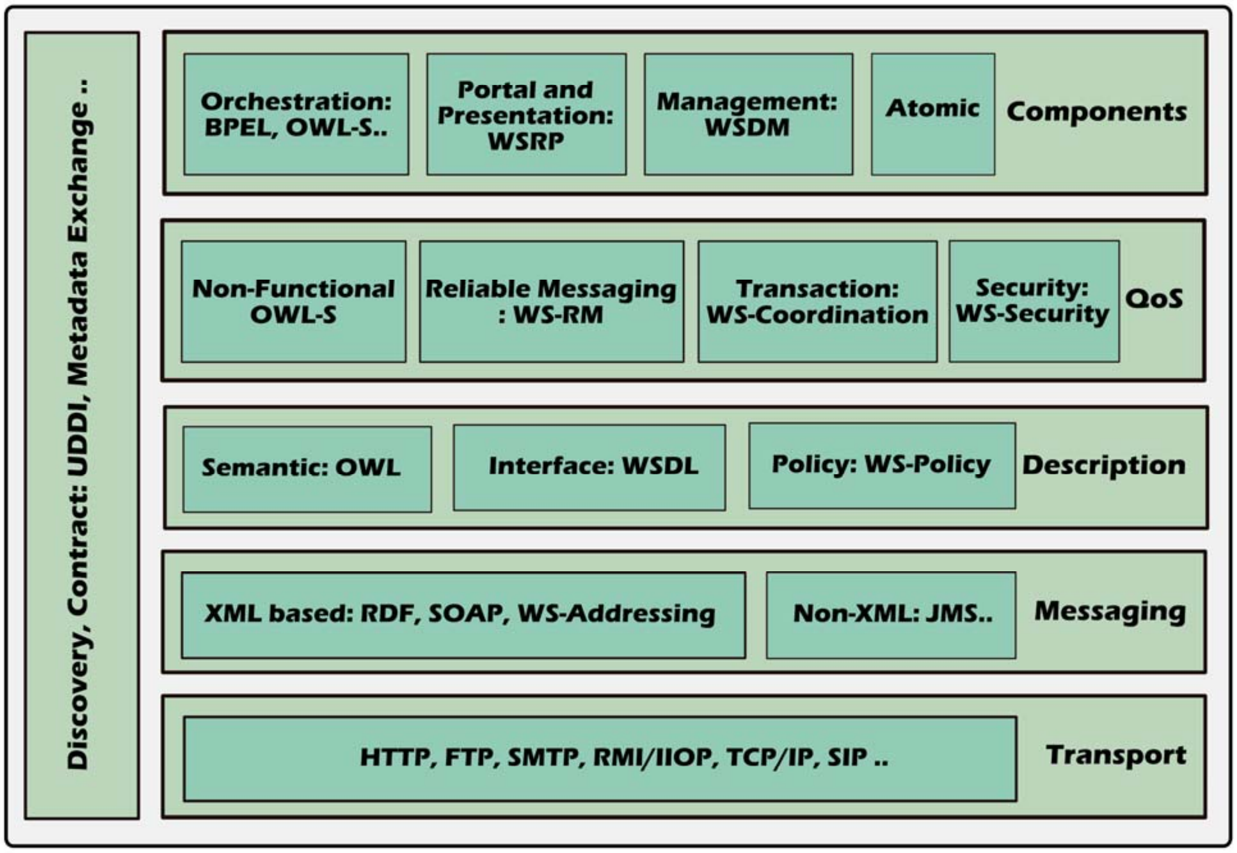

Discovery, Contract: UDDI, Metadata Exchange ..
Orchestration: BPEL, OWL-S..
Portal and Presentation: WSRP
Management: WSDM
Atomic
Components
Non-Functional OWL-S
Reliable Messaging : WS-RM
Transaction: WS-Coordination
Security: WS-Security
QoS
Semantic: OWL
Interface: WSDL
Policy: WS-Policy
Description
XML based: RDF, SOAP, WS-Addressing
Non-XML: JMS..
Messaging
HTTP, FTP, SMTP, RMI/IIOP, TCP/IP, SIP ..
Transport

Figure 8: The WS-* standards and specifications stack

2.3.1 WS-Addressing

The WS-Addressing [60] specification defines a standard for incorporating message addressing information into Web services messages. WS-Addressing provides a uniform addressing method for SOAP messages travelling over synchronous and/or asynchronous transports.

SOAP does not provide a standard way to specify where a message is going, how to return a response, or where to report an error. WS-Addressing defines the standard way to route a message over multiple transports or direct a response to a third party. For example, a client application might send a request over Java Message Service (JMS) and ask to receive the response through e-mail or SMS. To enable these kinds of applications, WS-Addressing incorporates delivery, reply-to and fault handler addressing information into a SOAP envelope.

WS-Addressing also defines a standard for including service-specific attributes within an address for use in routing the message to a service or for use by the destination service itself. These attributes are particularly useful for the creation of stateful web services, which are services that can receive a series of requests from a particular client and remember state information between requests. WS-Addressing introduces two new constructs for Web services vocabulary: endpoint references and message addressing properties. "Endpoint" is an established term for a destination at which a Web service can be accessed. Endpoint references are a new model for describing these destinations.

2.3.1.1 Endpoint Reference (EPR)

Endpoint references are defined as a complex type in the WS-Addressing schema. The endpoint reference type contains an address (a URI), reference properties, reference parameters, a port type, a service name and policy elements (defined by the WS-Policy specification). The only required element of an endpoint reference is the address. The other elements of an endpoint reference are all optional, making it easy to use only what you need. The port type and service name elements are very similar to their WSDL counterparts. A service is a named collection of ports. As in WSDL, the port type and service name are QNames (qualified names) in WS-Addressing. The port type and service name in a WS-Addressing endpoint reference are meant to provide compatibility with WSDL rather than to replace it entirely.

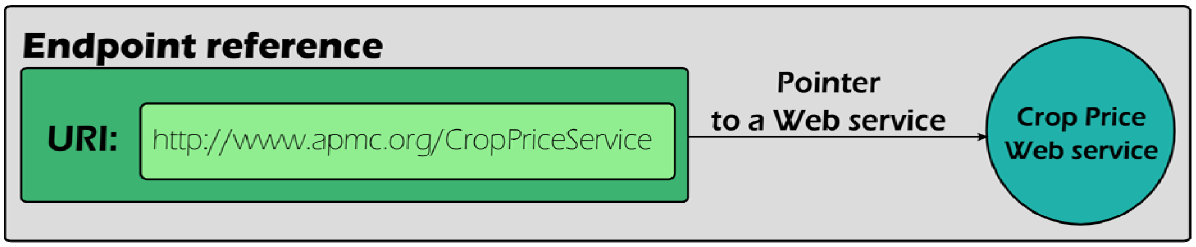


Figure 9: Endpoint reference

A significant aspect of an endpoint reference is the ability to attach data from your own XML namespace via reference properties or reference parameters. Both of these elements are collections of properties and values that you can use to incorporate elements from your own XML namespace (or any XML namespace) into the endpoint reference. The key distinction between a reference property and a reference parameter is not the format but the intended usage.

A reference property is used to identify the endpoint at which a service is deployed. Two endpoint references that share a URI but specify different reference property values represent two different services. Reference properties are used to dispatch a request to the appropriate service. For example, one might deploy two different versions of a service and have requests specify a target version in their reference parameters. When a service receives a request and fulfills it, its behaviour should not vary in response to the reference properties.

In contrast, reference parameters are meant to identify resources managed by a particular service. Reference parameters tell a service which resources to handle. They do not identify the service. Two endpoint references with different reference parameters do refer to the same service.

2.3.1.2 Message Addressing Properties

As explained above, endpoint references are used within message addressing properties. The message addressing properties define the full set of addressing information that can be attached to a SOAP message. Most of the fields are optional; the only required fields are the ‘To’ and Action fields, each of which specifies a URI. In an HTTP request, these would be the same URI.

In a non-HTTP request, the To URI may differ from the Action URI. The request is delivered to the To URI. The Action URI indicates the action to be taken. The Action URI should represent a service corresponding to a WSDL port type. The To URI specifies the "where" and the Action URI specifies the "what". In addition to the required To and Action URIs, the message addressing properties include several optional elements. A ReplyTo endpoint must be specified only when the sender expects a response, but it can be used to route that response to any valid endpoint. FaultTo is always optional and routes SOAP fault messages to specified endpoint references. Additionally, consumers of a service can use a From endpoint reference element to identify themselves to the service. Explicitly separating the message source endpoint, expected reply endpoint, and fault handling endpoint helps WS-Addressing support a variety of messaging models beyond the simple request/reply interactions we typically associate with web services.

When a reply is expected, whether it is expected by the sender or by a third endpoint specified in the ReplyTo header, a MessageId element must also be present. The message ID is a unique URI. Because web services can be used over unreliable transports, it is possible that an endpoint will receive duplicate copies of a message. The message ID can be used to avoid processing the same message twice.

When a service receives a message addressed using WS-Addressing, it will also include WS-Addressing headers in the reply message. The message ID of the original message becomes a RelatesTo element in the reply's address. At present, the only supported relationship type is "Reply." If a client is sending multiple web services requests and receiving asynchronous responses, possibly over different transports, the RelatesTo element provides a standard way to associate incoming replies with their corresponding requests.

2.3.2 WS-ResourceFramework

Web Service Resource Framework (WSRF) [11] is a set specification approved by Organization for the Advancement of Structured Information Standards (OASIS) for various aspects related to stateful Web services. It provides definition of message exchange pattern as to how stateful Web services should be created, addressed and destroyed [61]. WSRF defines conventions within the context of established Web Services standards for managing 'state' so that applications can reliably share changing information and discover, inspect and interact with stateful resources in a standard and interoperable way. WSRF is a collection of four specifications, namely WS-ResourceProperties, WS-ResourceLifetime, WS-ServiceGroup and WS-BaseFaults. It also refers to two related specifications, namely WS-Notification and WS-Addressing.

2.3.2.1 WS-ResourceProperties

WS-ResourceProperties [62] (WS-RP) describes properties of resources and the association between Web services and stateful resources. This relationship is described as the Implied Resource Pattern. In the implied resource pattern, messages to a Web service include a component that identifies a stateful resource to be used in the execution of the message exchange. The composition of a stateful resource and a Web service under the implied resource pattern is termed as a WS-Resource. Different WSRF resource patterns are discussed in our previous work [55].

WS-RP specification standardizes the means by which the definition of the properties of a WS-Resource may be declared as part of the Web service interface. The declaration of the WS-Resource's properties represents a view on the resource's state. The projection is defined in terms of a resource properties document. This resource properties document serves to define a basis for access to the resource properties through the Web service interface.

This specification also defines a standard set of message exchanges for the retrieval, modification, deletion and subscription for notification when the value of a resource property changes. The set of properties defined in the resource properties document and associated with the service interface, defines the constraints on the valid contents of these message exchanges. The specification has defined different operations to encapsulate set of message exchange patterns such as *GetResourcePropertyDocument, GetResourceProperty, GetMultipleResourceProperties, PutResourcePropertyDocument, QueryResourceProperties and SetResourceProperties.*

WS-ResourceProperties also defines the Notification TopicExpressions and TopicNamespaceelements [63] that are used to express the organization of the WS-Resource property element value change notifications. By understanding the relationship between TopicExpressions and resource properties, and examining the set of TopicExpressions supported by the NotificationProducer Web service, the service requestor can determine which of the resource properties are able to participate in the value-change notification pattern.

2.3.2.2 WS-ResourceLifetime

The WS-ResourceLifetime [64] (WS-RL) specification defines the life time of resources. The lifetime resource is a period between resource instantiation and destruction. WS-RL standardizes the means by which a WS-Resource can be destroyed immediately or at a scheduled time. The specification also defines the means by which the lifetime of a WS-Resource can be monitored. The scheduled destruction of the WS-Resource means that a resource may be destroyed after a certain period of time. In a distributed computing environment, a client may become disconnected from the service provider's endpoint and therefore may be unable or unwilling to cause the immediate destruction of the WS-Resource. This specification defines the means by which any client of a WS-Resource may establish and extend the scheduled termination time of a WS-Resource. If that time expires, the WS-Resource may self-destruct without the need for an explicit destroy request message from a client. Periodically extending the termination time of a WS-Resource can serve to extend its lifetime.

WS-ResourceLifetime defines a standard message exchange by which a service requestor can establish and renew a scheduled termination time for the WS-Resource, and defines the circumstances under which a service requestor can determine that this termination time has elapsed. To support the standard message exchange pattern WS-ResourceLifetime declares various methods such as *Destroy, CurrentTime, TerminationTime and SetTerminationTime*. A WS-Resource may support the optional pattern of notifying interested parties when it is destroyed. A WS-Resource may choose to implement *TerminationNotification* notification pattern by using the WS-BaseNotification [65].

2.3.2.3 WS-ServiceGroup

The WS-ServiceGroup [66] provides a description of a general-purpose WS-Resource which aggregates information from multiple WS-Resources or Web Services for domain specific purposes. The aggregated information can be used as a directory in which the descriptive abstracts of the individual WS-Resources and Web Services can be queried to identify useful entries. Membership in the group is in constrained way; which can be controlled through policies. Controlled membership enables requestors to form meaningful queries against the contents of the WS-ServiceGroup. The constraints for membership are expressed by intention using a classification mechanism. Further, the members of each intention must share a common set of information over which queries can be expressed. It describes the method for retrieval of resource properties belonging to a specific service group.

WS-ServiceGroup itself is a stateful Web Service that is a collection of other Web Services or WS-Resources and the information that pertains to them. The model for membership of a ServiceGroup is an entry resource property of the Service Group. Specification itself doesn't address the means of membership in the ServiceGroup and it can be either through ServiceGroupRegisteration or through any other means. Details of each member in the ServiceGroup are in the form of WS-ResourceProperties; which wraps the EndpointReference and the contents of the member.

WS-ServiceGroup specification describes different components and the message exchange patterns for its smooth functioning such as *WS-ResourceProperty "Entry", ServiceGroupEPR, MemberEPR, Content, WS-ResourceProperty, "MembershipContentRule", MemberInterfaces, ContentElements, ServiceGroupRegisteration and Notification Message Exchange*.

2.3.2.4 Characteristics of ServiceGroup

Each ServiceGroup has the following characteristics:

- When a WS-Resource corresponding to ServiceGroup is destroyed, all of the ServiceGroupEntry's, modeling the membership of the ServiceGroup, are destroyed. Destruction of the WS-Resource results in the empty ServiceGroup.
- Once a ServiceGroup is destroyed, corresponding WS-Resources that represent the ServiceGroup membership are also destroyed.
- A member can belong to several ServiceGroups; by implementing multiple portTypes or managing multiple WS-Resources.
- The member of a ServiceGroup can implement message exchanges from various interfaces or portTypes.
- Service Group can have multiple membership criteria's to categorize members within the group and a member can belong to the same ServiceGroup more than once fulfilling different membership requirements..
- When a member WS-Resource is destroyed, the ServiceGroup destroys the corresponding entry WS-Resource that represents the membership of that WS-Resource in the ServiceGroup.
- The grouping and membership aspects of a ServiceGroup are only manifest in the linkage between a ServiceGroup and a ServiceGroupEntry. Accordingly, a ServiceGroupEntry in

isolation has no semantic meaning; as contents of the ServiceGroupEntry only represents the membership requirements rather than the WS-ResourceProperties managed by the member.

2.3.2.5 WS-BaseFaults

A typical designer of a Web services application often uses interfaces defined by others. Managing faults in such an application is more difficult when each interface uses a different convention for representing common information in fault messages. Support for problem determination and fault management can be enhanced by specifying Web services fault messages in a common way. When the information available in faults from various interfaces is consistent, it is easier for requestors to understand faults. It is also more likely that common tooling can be created to assist in the handling of faults. WS-BaseFaults [67] defines an XML Schema type for a base fault, along with rules for how this fault type is used by Web services. It standardizes the way, in which errors are reported by defining a standard base fault type. It defines a standard base fault type, which, will be utilized for reporting errors and procedure for use of this fault type inside WSDL. WS-BaseFault defines various elements in an XML Schema such as *Timestamp, OriginatorReference, ErrorCode, Description, FaultCause and {any}.*

2.3.3 WS-Notification

The Publish-subscribed, Event-driven and Notification-based interaction patterns are commonly used patterns for inter-object communications. Publish-subscribe is known from Message Oriented Middleware time. Different vendors are supporting this pattern in the technologies; such as RMI and CORBA. Due to stateless nature of Web Services, we don’t have notation for Notifications; which limits the applicability of Web Services for complicated application development. WSRF defines conventions for managing 'state' so that applications can reliably share changing information, discover, inspect and interact with stateful resources in standard and interoperable way. WSRF brings Notification-based interaction pattern in Web Services domain. WS-Notification (WSN) [42] is built on WSRF as a set of three separate specifications (WS-BaseNotification, WS-BrokeredNotification, and WS-Topics) to support event and notification.

2.3.3.1 WS-BaseNotification

The WS-BaseNotification [65] is the base specification on which all the other specifications in the family of WSN depend. It defines NotificationProducer and NotificationConsumer. This specification defines many different roles and any single entity can fulfill the criteria of different roles. NotificationProducer is also managing subscription request i.e. SubscriptionManager and a NotificationConsumer itself is subscribing for the notification i.e. Subscriber. This specification includes standard message exchanges to be implemented by service providers that wish to act in these roles, along with operational requirements expected of them. In the Notification pattern a Web service, or other entity, disseminates information to a set of other Web services, without having to have prior knowledge of these other Web services. Latest WS-BaseNotification specification supports ‘Pull’ based notification for resource constrained devices; but yet none of the WSRF framework supports those recommendations and is not discussed.

2.3.3.1.1 NotificationProducer

A NotificationProducer is an entity, which monitors the state of different resources and detects various types of events. Whenever there is any change in the state of a resource or occurrence of any new event; which may qualify for certain actions the NotificationProducer notifies the relevant entities. These entities may be only interested in the changes or may initiate the series of events to accommodate changes.

The NotificationProducer must support any appropriate mechanism that lets a potential Subscriber to discover which resources and events are monitored by a NotificationProducer. These resources and events monitored by the NotificationProducer are called Topics (discussed in section 2.3.3.2). For this purpose, each NotificationProducer must support following resource properties in its resource other than any custom resource properties. Out of these resource properties, the ‘TopicSet’ is the collection of topics supported by the NotificationProducer expressed as a single XML element as described in [63].

```
<xsd:element ref="wsnt:TopicExpression" minOccurs="0"
maxOccurs="unbounded" />
<xsd:element ref="wsnt:FixedTopicSet" minOccurs="0" maxOccurs="1" />
<xsd:element ref="wsnt:TopicExpressionDialect" minOccurs="0"
maxOccurs="unbounded" />
<xsd:element ref="wstop:TopicSet" minOccurs="0" maxOccurs="1" />
```

To support notification message exchange pattern, NotificationProducer must implement different operations such as *Subscribe, GetCurrentMessage, Renew, Unsubscribe, PauseSubscription and ResumeSubscription*. These operations are related to the subscription for the notification and provide the standard format for message exchange. These methods heavily rely on Topics declared in the TopicSet.

2.3.3.1.2 NotificationConsumer

NotificationConsumer may have interest in the elements monitored by the NotificationProducer. A NotificationConsumer can subscribe to receive notifications directly from a NotificationProducer, supporting only direct and point to point notifications. A NotificationConsumer discovers the NotificationProducer and browses the Topics for subscription (i.e. sending Subscribe request or invoking Subscribe operation). A NotificationConsumer must implement the call back methods to receive the notification. Normally the NotificationConsumer implements Notify call back operation; which is invoked by the NotificationProducer. The NotificationConsumer must support one of the two or both NotificationMessage format; or at least should be in position to handle the form of Notification it has requested for the given Subscription.

2.3.3.1.3 NotificationMessage

The NotificationProducer must also clarify the supported formats for the notification messages. When NotificationProducer has a notification to distribute, it matches the notification against the subscription list and issues the notification to the subscriber which is registered for the notification of such event. WS-Notification allows a NotificationProducer to send a NotificationMessage to a NotificationConsumer in one of the two ways:

1. The NotificationProducer may simply send the raw NotificationMessage (i.e. the application-specific content) to the NotificationConsumer.
2. The NotificationProducer may send the NotificationMessage data using the Notify message, which means wrapping the application-specific content in the Notify element along with additional information i.e. the source of notification, time, last value etc.

When a Subscriber sends a Subscribe request message, it indicates which form of Notification is required (the raw NotificationMessage or the Notify Message). The NotificationProducer must observe this Subscription parameter and use the form that has been requested.

2.3.3.2 WS-Topics

WS-Topics [63] defines a mechanism to organize and categorize items of interest for subscription knows as 'topics'. WS-Topics defines an XML model for describing metadata associated with topics. It defines the three topic expression dialects that can be used as subscription expressions to subscribe request messages. The specification aims at categorizing topics in different categories under which the topics are clubbed.

2.3.3.2.1 Topics

The WS-Notification specifications allow the use of Topics as a way to organize and categorize a set of Notifications messages that relate to a particular type of information. The Topics mechanism provides a convenient means by which subscribers can reason about Notifications of interest. A Topic is the concept used to categorize Notifications and their related Notification schemas. These Topics are used as part of the matching process that determines which (if any) subscribing NotificationConsumers should receive a Notification. When Topic generates a Notification, a NotificationPublisher can associate it with one or more Topics.

2.3.3.2.2 Topic Namespaces

The set of Topics associated with a given XML Namespace is termed a Topic Namespace. Any XML Namespace has the potential to scope a collection of Topics. A Topic Namespace is just an abstract set of Topic definitions. Each Topic in a Topic Namespace can have zero or more child Topics, and a child Topic can itself contain further child Topics. A Topic without a parent is termed a root Topic. A particular root Topic and all its descendents form a hierarchy (termed a Topic Tree).

2.3.3.2.3 Topic Expression Dialects

Topics are referred to by TopicExpressions. TopicExpression is a query mechanism to reach one particular Topic in the Topic Tree. WS-Notification specifications provide an extensibility mechanism to allow vendors to define their own topic expression dialects. The different default topic expression dialects supported by different Application Server.

2.3.3.2.4 Topic Set

The Topic Set is a collection of Topics supported by a NotificationProducer. Topics from a single Topic Namespace can be referenced in the Topic Sets of many different NotificationProducers. Moreover the Topic Set of a NotificationProducer may contain Topics from several different Topic Namespaces. A NotificationProducer can support zero or more Topics, and these can come from multiple Topic Namespaces. A NotificationProducer can support an entire Topic Tree or just a subset of the Topics in a Topic Tree. The set of Topics supported by the NotificationProducer may change over time.

2.3.3.3 WS-BrokeredNotification

The WS-BrokeredNotification [68] specification defines the Web services interface for the NotificationBroker. A NotificationBroker is an intermediary between message Publishers and message Subscribers. A NotificationBroker decouples NotificationProducers and NotificationConsumers and can provide advanced messaging features such as demand-based publishing and load-balancing. A NotificationBroker also allows publication of messages from entities that are not themselves service providers. The NotificationBroker interface includes standard message exchanges to be implemented by NotificationBroker service providers along with operational requirements expected of service providers and requestors that participate in brokered notifications. A NotificationBroker is capable of subscribing to notifications, either on behalf of a NotificationConsumers, or for the purpose of messaging management. It is capable of disseminating notifications on behalf of Publishers to NotificationConsumers. A NotificationBroker aggregates NotificationProducer, NotificationConsumer and RegisterPublisher interfaces.

2.3.3.4 Notification Pattern

Notification pattern has roots in the message exchange pattern of message-oriented middleware. Publish-subscribe pattern introduces service provider and service requestors where message is exchanged to send notification. Publisher allows subscriber to choose and register topics either directly or thru broker service. When an event occurs on a given topic, publisher broadcasts new information to all the subscribers of topic either directly or thru broker. WSRF and WSN support publish-subscribe, event-driven and notification interaction pattern in Web Services domain. The event-driven and notification pattern are used to develop inter-service communication. It leverages commonly used design patterns and can be extended to fulfill the application needs.

2.3.3.4.1 Client as Notification Consumer

In this type of interaction pattern, client works as a Notification Consumer and processes the notification messages. It is responsible to inter-relate dependent WS-Resource instances. It notifies any change in the subscribed resource instance state and updates instances of other related Resources through corresponding Instance Service(s). Client application exposes "notify" operation to receive asynchronous notification messages; and implements complicated logic of associating different Resources instances together, whereas Enterprise Application is simple and independent of notification.

2.3.3.4.2 Service as Notification Consumer

At application level, services managing different Resource instances are associated together. Due to inter-dependency of WS-Resources these managing services have interest in the state of other Resource instances. Thus handling the notifications at service level is more appropriate without involving client applications. This is the situation where automatic and quick actions are required. The client does not manage actions and it does not have a role in the decision-making. Actions required are related to the main functionality of the application and not with a specific client. In this approach client applications are independent of notification processing.

2.3.3.4.3 Resource as Notification Consumer

The two notification approaches discussed above (Sections 4.1 and 4.2) have their own limitations and benefits. A third notification model can provide the best of both approaches with an even cleaner design. Applications are still easy to manage and maintain and WS-Resource instances can have one-to-one associations. In this approach WS-Resource itself is a notification consumer, yet may also act as a producer. Each instance of the WS-Resource can subscribe to ‘state’ changes of specific WS-Resource instances whilst broadcasting notification messages related to its own ‘state’. Overall this mechanism gives tighter control on the business logic without interference from the client side.

Implementing a WS-Resource as a notification consumer or a consumer-producer can result in large numbers of messages which can overload the Subscription Manager Service, thus affecting the overall performance of the application. The more inter-related instances, the worse the problem becomes. This model should be applied with caution and WS-Resources serving as notification consumers should obey the following guidelines:

There are a controlled number of instances of each WS-Resource at any given time. Each WS-Resource has limited dependency on other WS-Resources and is not involved in complicated association linkage. At least one of each WS-Resource instance is available all the time, Brokered Notification is required for persistent WS-Resources; Producer-consumer WS-Resources should be avoided if possible to avoid cyclic notification chains.

2.3.4 WS-DistributedManagement

Web Services Distributed Management (WSDM) [43] is model for managing distributed services. WSDM is the remit to create a model for web services management that allows interoperability between multiple environments. The management of the Virtual Organizations (VO) and the resources available within the VO is a complicated process. The Grid Environment consists of variety of resources and applications/services provided by various vendors, with varying compatibility levels. A variety of management systems already co-exist to be able to manage the breadth of resources but their effectiveness is undermined due to lack of interoperability, compliance to standards, acceptance from industry and compatibility.

WSDM is the first step to solving the management integration problem of Web Services. The WSDM standard uses Web Services as a platform to provide all of the essential functionality that supports the distributed management platform. The purpose of extending the existing standards is to utilize and reuse much of the technology, experience and tools which also contribute in the platform independence. WSDM relies directly on other standards and implementations such as WSRF, WSN and WSA. The WSDM standard is made up of two different standards Web Services Distributed Management: Management Using Web Services (MUWS) and Web Services Distributed Management: Management of Web Services (MOWS) specifications.

2.3.4.1 Management Using Web Services (MUWS)

The Management Using Web Services (MUWS) [69] specification defines how any IT resource can use Web services technologies for exposing manageability interfaces. MUWS standard provides the necessary structure for advertising the service, its capabilities and the required information to manage the resource. MUWS is mostly used for the discovery and selection of the suitable resources within the Grid environment.

2.3.4.2 Management of Web Services (MOWS)

The Management of Web Services (MOWS) [70] specification builds on MUWS and specifically addresses how a Web service resource can be managed. The MOWS component of WSDM provides the methods and interface specific to resource/service in the form of Web Services for its remote management. It is MOWS that is used more for the management of business processes; integration and collaboration of different business process in the form of workflows.

2.3.5 WS-ReliableMessaging

Web services is loosely coupled environment so there is no immediate way to know whether message is send successfully or whether message has failed while transmitting or whether receive messages are in sequence. WS-ReliableMessaging (WS-RM) [71] standard is designed to provide reliable messaging between services consumer and provider. It guarantees the successful delivery, proper sequence of the messages, prevents duplication of messages and reports failure of delivery of the messages. It improves the quality of services of SOAP message transmission by using delivery and fault report mechanism.

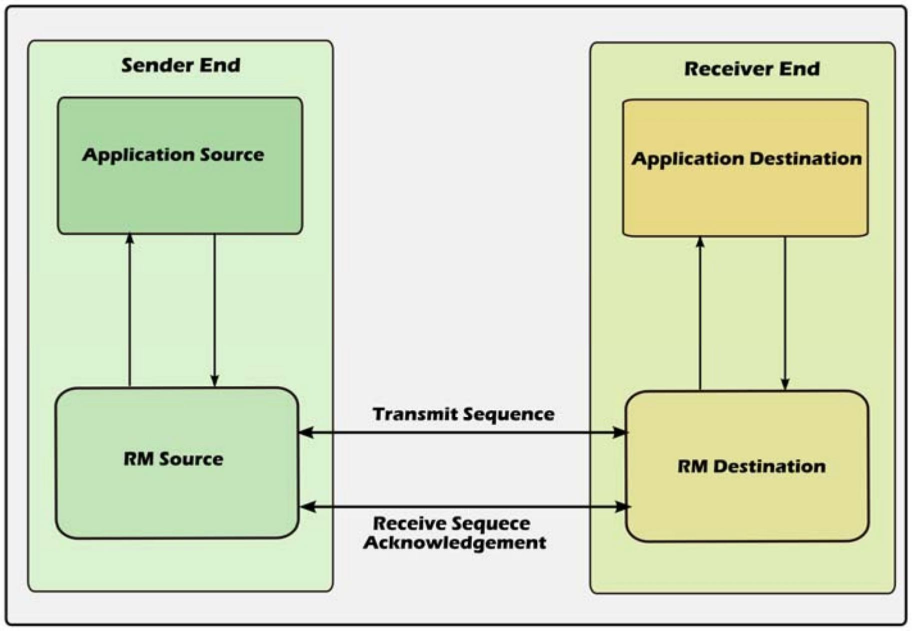


Figure 10: Reliable Messaging Interaction

WS-RM defines four abstract components: Application Source, RM Source, Application Destination and RM Destination. Application Source and RM Source are at the sending endpoint. Application Source is the service that sends messages to the RM Source. RM source is responsible for the transmission of the message and processing of SequenceAcknowledgement messages received from RM Destination. Application Destination and RM Destination are at the receiving endpoint. RM Destination receives message from RM Source and delivers to the Application Destination for processing. WS-RM uses sequence to maintain the order of transmitted messages. Each Message in the sequence is assigned a unique message number to identify the position of the message. The receiving endpoint issues a SequenceAcknowledgement for all the messages that have been received within a Sequence. RM source does not have to wait till the last message acknowledgement; it can send request acknowledgement (AckRequested) to RM Destination any time. In case of failure RM Destination will send fault information as SequenceFault to RM Source. WS-RM is supporting At-Most-Once, At-Least-Once, Exactly-Once and In-Order delivery assurances.

WS-RM depends on other WS specifications such as WS-policy and WS-Addressing. WS-Addressing allows identification of services using service EPR and also allowed retransmission using same message ID for the delivery assurances. Policy assertion defined by WS-Policy and

WS-PolicyAttachment specification that aimed at enabling endpoints that participate in the WS-RM to specify the functional or non-functional requirements.

2.3.6 WS-Policy

Every business process is governed by a set of rules and constraints. Business process contains services with certain characteristics, properties, limitations and non-functional requirements. WSDL document only describes the functional requirements of services. There are number of factors that both service provider and service consumer should consider before they interact with each other. A service consumer may require invocation of Web services with security, reliability information etc. These non-functional requirements are important to describe the properties of Web services. There is no way of expressing such information except via documentation or direct inquiries etc. For satisfying these types of requirements of a service consumer, policy is used in Web service selection phase. Once the Web services which satisfies the functional requirements of a consumer have been found or identified, they will be compared with information extracted from the policy to find out whether they satisfy the non-functional requirements of a consumer or not. Services which do not satisfy the non-functional requirements will be rejected. Only those services which satisfy functional and non-functional requirements will be selected. Thus, the use of policy during Web service selection phase helps service consumers to get services based on specified functional and non-functional requirements. One can describe these requirements using WS-Policy [72] specification.

WS-Policy framework is defined as a set of three specifications (WS-Policy, WS-Attachments and WS-PolicyAssertions) to describe properties, rule and non-functional requirements of services. These specifications allow specifying the policies of business process and capability, limitations, characteristics and general requirements of services involved in the process using the grammar and policy assertions. This policy can attach to WSDL and UDDI using WS-PolicyAttachement. WS-Policy framework is associated with other specification such as WS-ReliableMessaging and WS-Security etc. to achieve the desired functionalities.

WS-Policy depends on other WS specifications such as WS-Security and WS-ReliableMessaging. WS-Security policy assertion checks whether the header is encrypted or not. WS-SecurityPolicy assertion requires Basic256Rsa15 algorithm suite which uses X509 Version

3 token. WS-RM puts limits on InActivityTimeout, BaseRetransmissionInterval, AcknowledgementInterval and ExponentialBackoff. WS-Coordination related assertion specifies that the coordination context expires in 3000 milliseconds.

2.4 Semantic Web

The Semantic Web is an extension of the current web in which information has well-defined meaning, where data has abstract representation and is linked to provide more effective discovery, automation, integration, and reuse across various applications [14]. The current web is using HTML to specify how to display information on web page for humans to read. Information on the web is searched by the humans, interpreted by the humans, consumed and queried by the humans. In other words, Information is not machine readable and machine understandable. Current web is flooded with the information so searching, extracting, querying, accessing, maintaining and viewing of information is affected [73]. The vision behind Semantic web is to provide knowledge-based web with more meaningful, machine processable information, automated tools and services for accessing, searching, querying, viewing and maintaining information. The realization of Semantic web also helps in B2B and B2C interactions by providing data interoperability and helps to retrieve information from multiple sources. It also provides support for automated negotiation, auctioning, policy and contract management.

The Semantic Web is built upon XML based standards, languages and tools. The Semantic web protocol stack follows layered approach, each language using the features of the underneath layers for extending the capabilities [73]. This is illustrated in the following Semantic Web protocol stack [74].

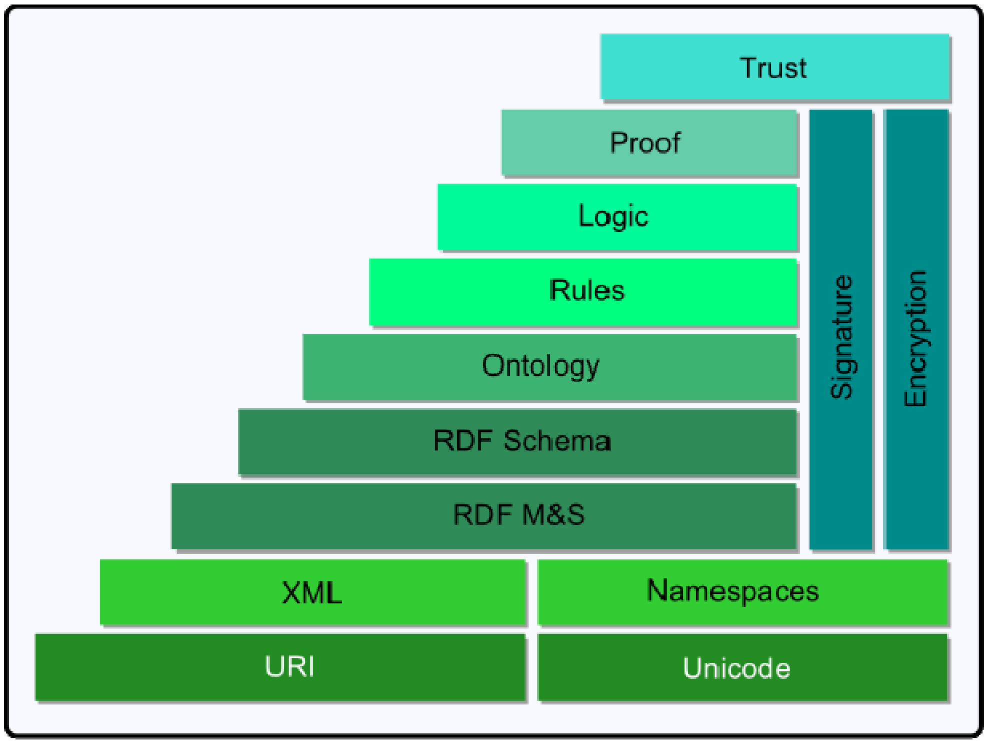


Figure 11: Semantic Web Protocol Stack

The Semantic web uses metadata to provide meaning of the contents. Metadata is data about data. There are three types of metadata syntactic, structural, and semantic [75]. Metadata add more semantic – meaningful information about the contents. In other words, Semantic web transforms structural information to semantic information. Many standards have evolved to provide semantic information or to capture the meaning of information and for the realization of Semantic web. W3C has developed XML-based RDF and OWL languages for abstract representation of data, with collaboration from a large number of researchers and practioner. XML allows to create own tags. Tags provide an elemental syntax to add content structure to the documents. It provides no information about meaning of structure or no semantics of the content. XML Schema is restricting the structure and content of XML documents. A rule language can be used to query and filter ontology, to infer new knowledge and to make decisions. The rules layer provides simple logic capability. Logic layer provides more advanced logic features. The logic layer enhances the ontology language for writing application-specific declarative knowledge and allows formal logic proofs to be shared. The Proof layer allows representation of proof in the Web language, proof validation and actual deductive process. Finally, through the use of digital signatures, robust proofs and other knowledge, a trust layer will emerge for application-to-application trust. "Web of trust" is used to indicate the third and final web in Semantic web vision. We will discuss some of the major Semantic web standards such as RDF, OWL and Semantic Web Services standards like OWL-S and WSDL-S which transform Web services to Semantic Web Services and provide automated discovery, composition and execution support.

2.4.1 Resource Description Framework

XML is extensible markup language for describing tree structures using our own well-defined tags. XML defining the structure of the data and providing syntactic information of data to make it interchangeable but provides no meaningful information about the data. As XML is not sufficient to fulfill the goal of Semantic web, W3C proposed Resource Description Framework (RDF) [76] standards for representing metadata, for adding semantic and for capturing knowledge. RDF is based on XML and URI; it uses XML syntax for adding meta-information and URI for unique identification of resources. RDF document is very simple, contains information about resources in terms of collection of statements. Each statement is expressed as a (subject, predicate, object) triplets. Subject is a resource that can be referred by URI. Predicate is a property to define binary relationships between subject and object. Object represent the data type, which is replaced by the value of the resource. Using this RDF triplet one can make statements about statements.

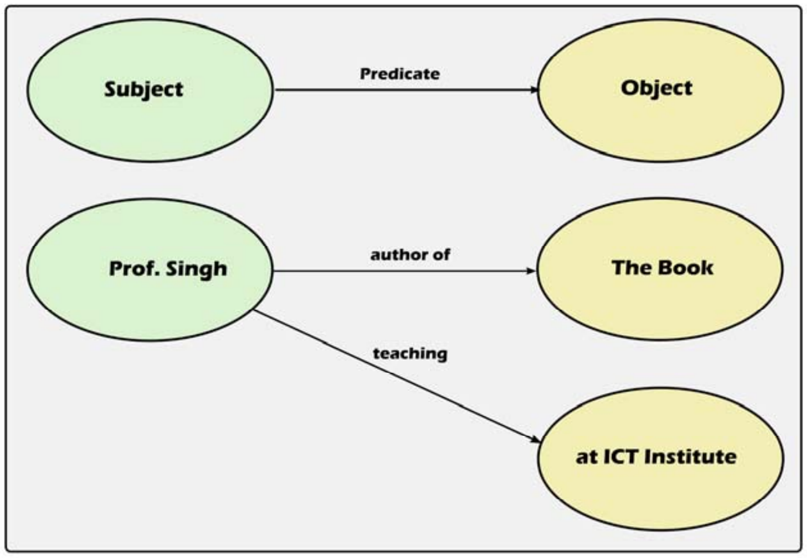


Figure 12: RDF triplets

RDF describes the resources without any assumption about any domain. To define own terminology RDF-Schema has been proposed as a primitive ontology language. It defines domain vocabulary, properties, range and relationships between objects. It specifies classes, property, subclass and subproperty relationships. Many domain modeling features are missing in RDFS such as

- Disjoint classes
- Cardinality restrictions
- Building of new classes using boolean operators (union, intersection and complement)
- Range restriction on specific classes only
- Properties such as transitive, unique and inverse

Therefore we need better expressive ontology language to address above mentioned requirements.

2.4.2 Web Ontology Language

Ontology provides conceptual representation of information for making it machine interpretable. Many definitions exist, but most common definition of ontology is: “An ontology is a formal explicit specification of a shared conceptualization for a domain of interest” [77]. Ontology languages follow formal approach to make domain specific concepts and relations machine processable. Ontology is an evolving research area both in academia and industry. Many ontology languages have been developed to provide different facilities. The most recent development is W3C Semantic Web group, Web Ontology Language (OWL) [78]. OWL is developed on top of the RDF and RDFS and derived form DAML+OIL (DARPA Agent Markup Language + Ontology Inference Layer). OWL uses description logic to define classes and properties to support boolean operators, constraints and characteristic properties. OWL is available in the form of following three versions with different expressiveness to fulfill the different requirements of the user. The selection is based on expressiveness and reasoning support.

- OWL Lite: It is easy to implement and easy to understand for the user but it has limited expressive power. It provides a class hierarchy and limited constraints. It supports cardinality constraints but does not support negation and union operators.
- OWL DL: It is based on Description Logics and It is more expressive then OWL Lite. It provides efficient reasoning support and maximum expressive power, but it is not fully compatible with RDF.
- OWL Full: It is most expressive language. It provides all modeling primitives but no guarantees about reasoning support and decidability.

2.5 Semantic Web Services

Unfortunately, today's Web Services standards are not machine understandable and does not support automated composition. Existing technologies provide syntactic descriptions and making it difficult for a requester to interpret. Semantic Web Services (SWS) provides semantic description of the capabilities of Web services. It supports automated composition, discovery, invocation and execution of services. SWS depends on the further development of Web Services and Semantic Web technologies. Semantic Web provides interoperability to Web Services by adding more knowledge and meaningful information. To realize the SWS various research initiatives are taking place in academia and industry. However, over the last few years, a number of SWS standards have been developed, including OWL-S [25], WSMO [41] and WSDL-S [26]. The SUPER (Semantics Utilized for Process Management within and between Enterprises) project is an EU-funded Integrated project. SUPER project aims to integrate SWS with BPM. Main purpose of the SUPER project is to integrate semantic web services (WSMO) with BPM for better modeling and management of business processes. It raises BPM from IT level to the business level and targeting to handle complex business processes [79].

2.5.1 OWL-S

OWL-S is an agent-oriented approach, where an agent needs semantic description of services and its capabilities. OWL-S extends the OWL capability to provide ontology for the Web Services. OWL-S is OWL ontology for Web Services, for providing automated service discovery, invocation, composition, execution monitoring and interoperability. The ontology of services provides three main upper ontologies: ServiceProfile, ServiceModel and ServiceGrounding.

2.5.1.1 Features of OWL-S

- ServiceProfile describes what a service is doing. It describes service capabilities and functional properties in terms of IOPEs (input, output, preconditions and effects).
- ServiceModel describes how a service is working. It provides control flow and data flow to enable automated composition and execution of services.
- ServiceGrounding describes how a service can be accessed. It provides details about communication protocols, message description and information related to binding of the services.

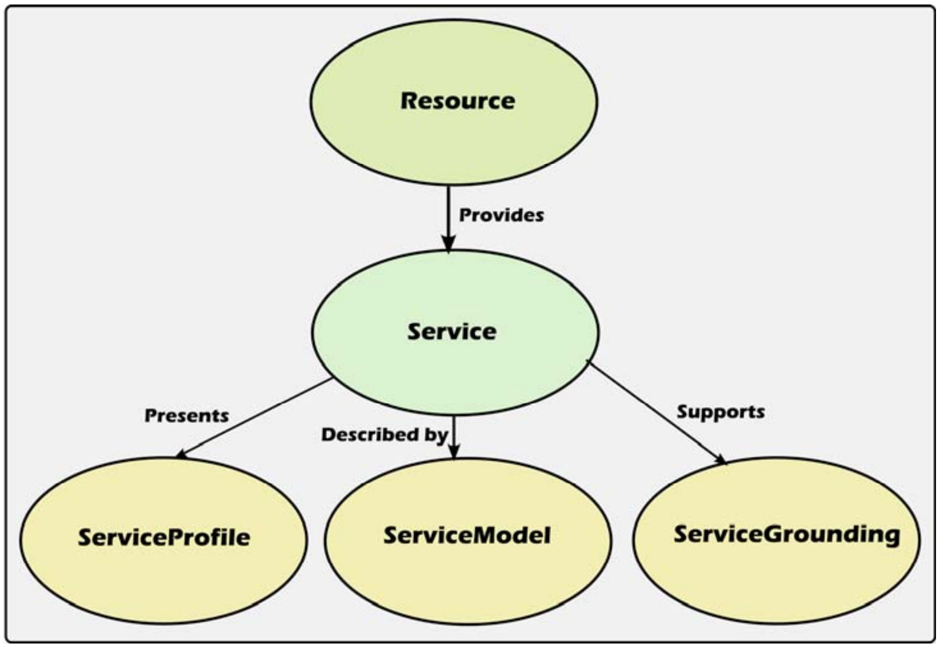


Figure 13: Ontology of Services

2.5.2 WSDL-S and SAWSDL

Various standards have been proposed for the Semantic annotation of web services. Annotation of Web Services with enhanced service description is a prerequisite for automation in discovery, invocation, binding or composition of the developed Web Services. Among important standards OWL-S, WSMO and the Internet Reasoning Server (IRS II) introduce specific solutions for respective philosophy, the METEOR-S [80] project resolves the issues by leveraging advantage of semantics with the existing standards. It provides complete life cycle of Semantic Web Processes including development, annotation, discovery, composition and orchestration. For semantic annotation, METEOR-S project proposed WADL-S as an extension of existing WSDL to provide semantic annotation of various service elements and developed Web Service Annotation Framework (MWSAF) [80]. MWSAF matches the terms used in the WSDL with the concepts given in the ontology and generates the annotated WSDL file. In 2006, W3C created a charter for the Semantic Annotation of Web Services (SAWSDL), which used WSDL-S as a primary input. W3C declared Semantic Annotations for WSDL and XML schema as a recommendation in 2007 [81]. SAWSDL introduces new extension attributes for WSDL and XML schema such as modelReference and SchemaMapping attributes. In domain model ModelReference links operations to functional descriptions and SchemaMapping provides mapping between schema type and semantic model [81].

2.6 Grid Computing

Grid computing is a distributed computing paradigm to provide effective utilization of heterogeneous networked resources in a dynamic, loosely coupled manner for large-scale, resource-intensive, distributed computation intensive applications. It provides workload management, virtualization, dynamic allocation of resources and centralized management. Grid provides an environment for building geographically and organizationally dispersed high-performance, data-intensive, scientific and engineering applications. Grid computing comprises of tools, middleware and services to provide access to a wide variety of distributed resources, to support virtualization, management and remote monitoring of computing infrastructure. Grid computing environment uses open standards and frameworks to design OGSA and OGSI. Grid computing adapts Web services as a key component to achieve integration and interoperability. OGSA aligns Grid with Web services and defines interfaces of Grid Services for developing interesting applications. More recently with the replacement of OGSI with WSRF, grid becomes suitable for enterprise applications, multimedia, ubiquitous and autonomic computing.

2.6.1 Open Grid Service Architecture

The Open Grid Service Architecture (OGSA) [82] is a layered architecture and built upon existing infrastructure framework to provide core platform services. The OGSA is developed by the Global Grid Forum (GGF) organization. It defines open standard, architecture, and services for grid-based applications. The role of OGSA is to provide distributed resource management, policy, logging, authentication and authorization to access resources, Scheduling and monitoring of resources, discovery, allocation and management of available resources. OGSA is based on Web services and open standard to provide heterogeneity and integration support. One of the most important requirements of OGSA is stateful services. Initially OGSI was responsible for delivering stateful services for OGSA. In January 2003 Globusworld replaced OGSI with a new set of Web services standards, Web Services Resource Framework. WSRF provides required support for stateful services and infrastructure for OGSA.

2.6.2 Open Grid Service Infrastructure

The Open Grid Service Infrastructure (OGSI) [83] is a grid software infrastructure layer for OGSA. It was published by GGF. OGSI extends Web services to accommodate state in grid computing resources. It defines mechanism for creating, managing, and exchanging information among Grid services. According to OGSI, a Grid service is a Web service that conforms to common interfaces and behaviour. With the development of WSRF to integrate state into the Web services, OGSI is now obsolete.

2.6.3 Comparison between OGSI and WSRF

OGSI is replaced by WSRF for the development of stateful Web services. There are a number of reasons summarized in the table below, which explains why WSRF is now the preferred standard for development of stateful Web services [61].

OGSI	WSRF
OGSI talks about instance of Web services. It creates one instance for each user. This is not a best practice for Web service development. Fundamentally Web	WSRF doesn't talk about instances of Web services. It talks about instances of resources. It creates multiple instances of resources for a single Web service. The

services is stateless and does not have instance.	Web service maps the request to the respective resource.
It aggressively violated xml schema with the help of xsd:any elements. It created confusion for end user, as to what kind of parameters are to be passed in the function. Also, use of xsd:any creates problems with some existing XML, Web services toolkits and standards.	It removes the use of xsd:any type. It makes mandatory to write a separate function for every kind of value the function expects. Hence, the problem of poor binding is also resolved.
The OGSI specification was not modularized. It didn't classify entities according to the requirements. Everything was put together in one big specification.	WSRF specification is highly modularized. There are various categories to allow them to be picked up according to the requirement.
It created a tight coupling between the Web service instance and the resource. So that end user will actually interact with a Web service.	It has one instance of a Web service, and numerous instances of the resource. Hence, the Web service exists independent of the resource which the client interacts.
It uses WSDL 2.0 specification constructs in the GWSDL. WSDL 2.0 is a draft specification and not yet approved by OASIS. Hence, the initial goal of convergence between Web services and grid services was affected.	It utilizes approved WSDL 1.1 specification constructs. Hence, it allows the convergence of stateless Web services and stateful Web services (Grid Services). Also, WSDL1.1 is compatible with the existing tools and parsers.

Table 1: OGSI vs. WSRF

2.7 Web Services Composition Standards

To achieve the vision of Web services such as integration and composition of scattered services, we need a process modeling language, which allows orchestration of a business process. Web services composition is a very important aspect to realize enterprise integration and business process. The research work to accomplish Web services composition can be mainly divided into two categories. One approach is Workflow based which contains a set of atomic services with control and data flow. Various standards have been proposed for workflow-based composition using formal languages such as Petri Nets, Pi-Calculus, Finite state machine and logical programming language [84]. Petri-nets based standards like BPML is proposed by Business Process Modeling Initiative (BPMI) and Yet Another Workflow Language (YAWL) is proposed by W. Aalst [85]. Pi-calculus based standards like Web Services Choreography Description Language (WS-CDL) proposed by W3C. Object Management Group (OMG) proposed Model-Driven Business Process Management (BPM). Among these standards BPEL is the most popular and widely accepted standard. BPEL combines the feature of both Petri-nets based IBM's Web Services Flow Language (WSFL) and Pi-calculus based Microsoft's XLANG [84]. Even though BPEL is considered to be the most comprehensive compared to other specifications, but it can support only eleven out of eighteen workflow patterns [86]. These approaches are not able to describe business process with continuous changes in state and event. Core Web services standards are stateless and don't have notation for event and notification. All these existing standards and approaches are not considering both event and state while modeling and composition of a business process. In this thesis we have proposed a formal approach for modeling and composition of a business process using Event Calculus and Rules.

Second approach is AI planning and Semantics based. Due to the limitations of above-mentioned standards for automation and lack of support for semantics, various semantic and AI planning based standards and methods have been proposed. Amongst standards for semantic Web services, OWL-S is the most widely accepted standard to describe the semantics of a Web Services. Various standards and methods are developed by research and academic groups for static and dynamic composition of services and for automated composition. A survey of these methods is discussed by S. Dustdar et. al. and J. Rao et. al. in [15, 17] along with the issues related to transaction, coordination, execution and monitoring of service composition. A classification of issues and approaches related to automate service composition is discussed by U. Kuster et. al. in [16].

2.7.1 Business Process Execution Language for Web Services

Business Process Execution Language for Web Services (BPEL) [21] is an XML based language for composition and execution of a business process. BPEL is develop by IBM, Microsoft and BEA and submitted to OASIS for standardization. BPEL is built on existing Web services standards to provide a business process extension. BPEL allows a single point interface to a composite Web service. It defines the behaviour of a business process, control the flow and invocation of services. BPEL process consists of two files, WSDL file and BPEL files. BPEL has defined various tags such as process, operations, variable, fault, events, partner link, correlation etc. These tags help in the composition of complex business process, selection of partners at run time and dynamic invocation of service [21, 87, 88]. Some of the key tags are explained below.

- process: A business process contains one or more tags.
- partnerLink: It describes the actual Web service with which, the BPEL process interacts. Only after the creation of a partner link, methods of Web services can be utilized by BPEL process. Partner link is like a bridge, which, helps in joining two disjoint ends, and once they are joined they can exploit features of each others.
- invoke: It provides a way to invoke Web services.
- receive: Receives input from the client and starts the business process.
- reply: In case of synchronous business process, it returns the result back to the client immediately, then role of sending the response back is performed by reply
- assign: It uses to manipulate the value of the involved variables. It allows values to be assigned to different variables.
- sequence: It allows invocation of a set of activities in a particular way.
- variables: It is typed data structure to store different kinds of values associated with a business process.

A Business process (WSDL and BPEL files) deploys on a BPEL execution engine. BPEL engine manages the execution of the process and interaction of services.

Various extensions of BPEL are also available to provide additional functionalities. WS-BPEL extension for people is designed to support human interaction and automated business process. WS-BPEL extension for sub-processes is defined to reuse sub-processes with the same or among different BPEL processes.

2.7.1.1 Features of BPEL

BPEL is accepted by major industry vendors due to the benefits offered by it. Following are the benefits of BPEL.

- It controls the flow of a process without altering the code of any service and modification of a WSDL file.
- It allows invoking of services based on the events.
- It provides notification based on the events. It supports various notification delivery modes such as delivery on mobile devices, fax etc.
- If in case a new partner is to be added, then it can be done without any hassles, as it requires no alteration in the code of any of the Web services. Only entity requiring alteration is the BPEL process, which is external to the entire system.
- It provides facilities similar to rollback. They allow changes made due to failed transactions to be rolled back, if in case some error occurs during the transaction processing.
- Normal Web services are stateless. Hence, it is difficult for different client instance to be differentiated from each other. BPEL allows creation of a new instance of a process for every client. Hence, state of data, for every client can be maintained using the instance of a BPEL process.
- Business processes are long running. Hence, many times, reply arrives after a long time when message was sent. BPEL process uses the concept of correlation to know the

instance at which it has to deliver the message. It matches the instance using some key vital properties from the message, which has been delivered.

Chapter 3
Event-driven Service-Oriented Architecture

Event-driven Service-oriented Architecture (EDSOA) is a type of SOA where services have the capability to react to events. Events can be in the form of beginning or end of execution of services, changes in the value of critical variables or exceptions generated at the run time. An event is a change in the state of a service, start or end of an activity. Event is a primary entity in Event-driven Architecture (EDA) and it depends on time and location. Business processes are event-driven and requires a set of tasks to be executed to fulfill the goal. It requires composition and execution of services as per the events. Events are responsible to change the state of a process. We can model the business process based on event and state. EDSOA defines event-driven business process model by representing process as conceptual process flows. Process is represented as a set of services and completion of each service as an event which is responsible to trigger next service. Each service taking part in the execution of a process is a stateful service and supports event and notification. To monitor the event on services, properties and life cycle of services should be defined. Property describes the parameter and state and it is responsible for state transition of a process. Lifecycle is responsible to define the start, end and termination time of a service, which is required to maintain and control the execution of a process.

In this chapter the life cycle of EDSOA is presented in section 3.1. In section 3.2 EDSOA is proposed. Section 3.3 briefly discusses the Agro-produce marketing business process use case scenario. We have considered Agro-produce marketing as a proof of concept and prototype implementation to demonstrate the proposed approach. Section 3.5 shows the realization of proposed architecture based on the use case scenario.

3.1 Life Cycle of Event-driven Service-oriented Architecture

Business processes are represented as composition of services. Business processes are dynamic in nature due to changes in policies, rules, partners and events. There is a need for a formal framework and event-driven service-oriented architecture to model, compose and execute the dynamic business processes.

Event-driven business process developed using EDSOA follows the following EDSOA development lifecycle.

Model: The model phase includes analysis and design of business processes, services, events and messages. In model phase, actual services are modeled which can perform single business function and take part in composition. Ontology should be defined to provide common domain vocabulary, knowledge and business service templates, which we can leverage during the Model phase.

Compose: In the composition phase, one needs to create or reuse existing services and combine it to create composite services. Business rules, business policy, and business metadata should be defined at composition phase. It will describe the control and data flow, mapping input, output of services, which are required for generating dynamic composition schema.

Deploy: The deploy phase involves activities like creating environment for deployment, registering endpoints, deploying the business services, provisioning business services to organizations and subscribers. Components and services are to be deployed in an integrated way so they can participate and get executed in a proper sequence at the time of execution of a composite business process.

Execution and Monitoring: The execution and monitoring phase involves managing, entitlements and subscriptions of the business services to organizations and users. Event and state provides business context visibility for composite services. For example, you can determine how a business service is utilized on various channels for different consumers, as well as determining how different endpoints are performing for a specific business context along with which services are in execution and in which state.

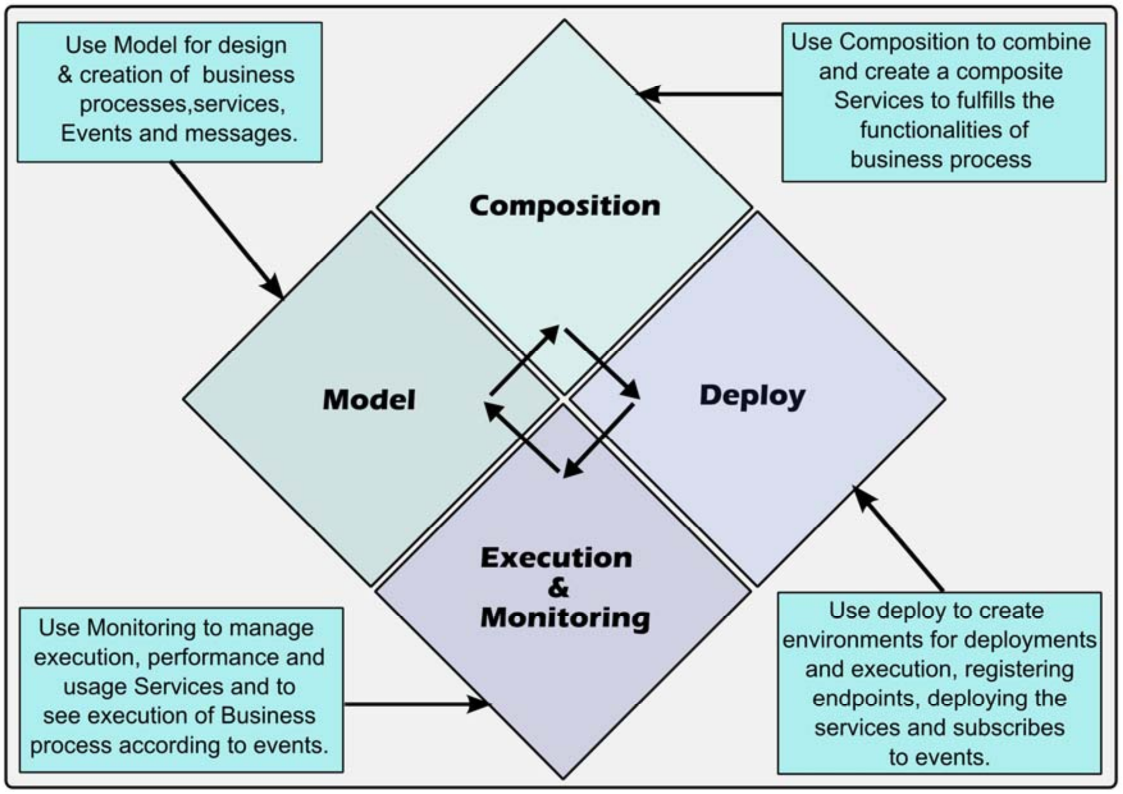

Use Model for design & creation of business processes,services, Events and messages.
Use Composition to combine and create a composite Services to fulfills the functionalities of business process
Composition
Model
Deploy
Execution & Monitoring
Use Monitoring to manage execution, performance and usage Services and to see execution of Business process according to events.
Use deploy to create environments for deployments and execution, registering endpoints, deploying the services and subscribes to events.

Figure 14: Event-driven Service-oriented Architecture Lifecycle

3.2 Proposed Event-driven Service-oriented Architecture

Event-driven Service-oriented architecture is proposed to provide state, transaction management, notification, execution and monitoring, and scalability. The role of the targeted system is to provide dynamic composition of a business process with interoperable integration of scattered services and resources. Our architecture is domain and tool independent. The proposal is mainly dependent upon existing grid computing, Semantic web and WS-* standards and specifications to achieve compliance. Our architecture presented in figure-15, contains various components to decompose the parts of an enterprise application based on the principles of SOA.

The architecture shown in figure-15 comprises of seven components:

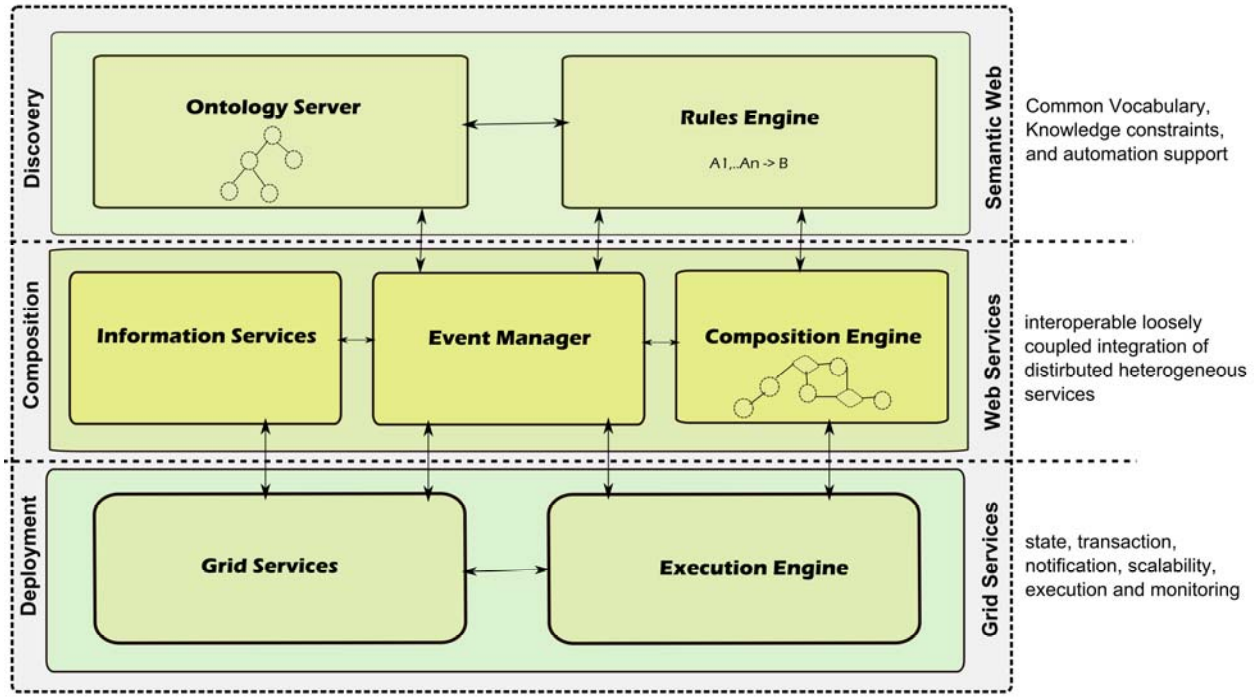

Discovery
Ontology Server
Rules Engine
A1,..An -> B
Semantic Web
Common Vocabulary, Knowledge constraints, and automation support
Composition
Information Services
Event Manager
Composition Engine
Web Services
interoperable loosely coupled integration of distirbuted heterogeneous services
Deployment
Grid Services
Execution Engine
Grid Services
state, transaction, notification, scalability, execution and monitoring

Figure 15: Proposed Architecture

3.2.1 Information Providers

In this experiment, Web Services acts as data service providers, maintained and published by respective organizations. These services provide access to different data sources with different mechanisms such as JDBC connection, stored procedure etc. It also provides access to existing applications or legacy applications, by using wrapper and exposing them as services. In this way they are allowing reuse of existing applications.

3.2.2 Event Manager

This component is the main controlling component of the architecture. Event manager is designed to communicate dynamically with the distributed heterogeneous services and other components such as ontology server, rules engine and composition engine. It is configured to furnish precise and event specific composition. It helps in the selection of services for composition. When any event is received by event manager, it will check the precondition of services, required input, and output parameters. Event manager uses backward chain algorithm for verifying above mentioned parameters. It works in coordination with the composition engine to generate the composition schema at runtime.

3.2.3 Composition Engine

Composition engine controls and manages the composition of services and allows orchestration and automated execution of a business process. It contains the composition logic and uses business rules to manage the flow of a business process. It works with the event manager and uses backward chain algorithm for checking the precondition, input and output of services. Forward chain algorithm is used with ECA rules to get the next sequence of services. According to events, composition engine generates composition schema at runtime based on events. The composition engine will be based on BPEL and will be executed by execution engine.

3.2.4 Ontology Server

Ontology server contains two levels of ontology, one provides the knowledge about application domain and another provides the knowledge about business processes and services involved in an organization. Ontology represents knowledge about the domain in terms of semantic graph, where concepts are represented as a node of the graph and relationships are represented as arch among the nodes.

To enable composition of business process, one of the important challenges is the utilization of uniform terms across the process. We have to cover all the terms, relations among them and the logical expression of the legal provision of business process. Functions and rules can be defined to capture the behavior, properties, and constraint of the concepts [89].

3.2.5 Rules Engine

Rules enhance the functionality of application to be developed based on the principle of Semantic web. It provides behavioural knowledge, constraints expression, and reaction to events. It defines flow and constraints of a business process and controls the behavior of a business process. It is also used to model the process, to define the policy, to handle the exception and to define the plan of a process [90]. There are four categories of rules, but I have considered mainly ECA rules and inferences rules. Rules engine contains ECA rules, which are triggered based on occurrence of an event. To achieve interoperability, ontology server will verify events and conditions, and corresponding to that an action will be taken. An action contains a set of tasks to be executed and required to compose the BPEL schema [46]. Inference rules are of if-then type and used to represent facts and required conditions.

3.2.6 Execution Engine

Execution and management of distributed and heterogeneous services is more complex than traditional enterprise applications. Execution engine utilizes grid middleware services and working as an integrated part of grid environment. Schema generated by composition engine will be executed at centralized execution engine. It is supported by grid middleware services for monitoring, event management, automation, orchestration, security, performance management, transaction management and notification services. It will take care of execution and monitoring of a business process. Any exception or final output will be forwarded to the event manager; from there it will be sent to the user. For the execution of BPEL schema in a grid environment two types of grid services are developed: business services and grid services.

3.2.7 Business Services

Business services implement the core business functionality and participate in the composition and execution of a business process. These services are modeled at the business process design time. It represents the decomposed business logic of a process and performs individual business function and combines to fulfill the requirement of a process. These services are crucial services and require support for notification, state and transaction management. These services need to be implemented as grid services and exposed as stateful resources by following WSRF specifications [45]. Notification is achieved by following WSN specifications. These services are part of grid environment to get the benefits of grid like security, execution, monitoring, resource sharing and scalability. These services are interacting with other external services to get the required information and knowledge for the execution of a business process.

3.2.8 Grid Services

These services provide resource management, virtualization capability, security, scalability, management, performance etc. It is deployed to manage the execution and control the behavior of grid. Inbuilt grid services like, service correlation, service isolation, self-discovery, virtualization, dynamic provisioning, resource pooling and persistent storage are utilized to fulfill the functionalities of middleware [53]. It offers deployment infrastructure services to manage hardware, software, and resources.

3.3 Motivating Scenario

As a proof of concept, I have taken an example of Agro-Produce Marketing Process in India. The Model Act [91] is expected to bring reforms in the Agro-produce Marketing Process. As indicated in the Model Act, a typical trade can span across the markets located at various places. I have considered a trading scenario to sell agro-produce across the markets. A seller (or farmer) joins the market place and expresses his intention to start a trade of agricultural produce. An authorized market functionary carries out measurement and grading of the produce. Seller needs to pay transportation fees in case vehicle is used to transport an agro-produce. Price of the produce can be set by tender bid, auction or any other transparent system. In case of direct sale, a seller is exempted to pay market fee or commission; whereas in case of indirect sale the market fee is imposed on the seller for using different utility services. Only license holders are allowed to carry out trade in the market area. If they fail to pay fees to the Agro-Produce Marketing Committee (APMC) or fail to pay the agreed-upon price of the purchased good, the APMC may cancel the license of the trader. If the trade is carried out by license holders in a manner explained above, the bill will be issued and the transaction will be recorded in the APMC database. Figure-16 represents a simplified workflow capturing few aspects of a typical trade. A framework is also evaluated by applying to another use case scenario of agricultural recommendation system [47, 52].

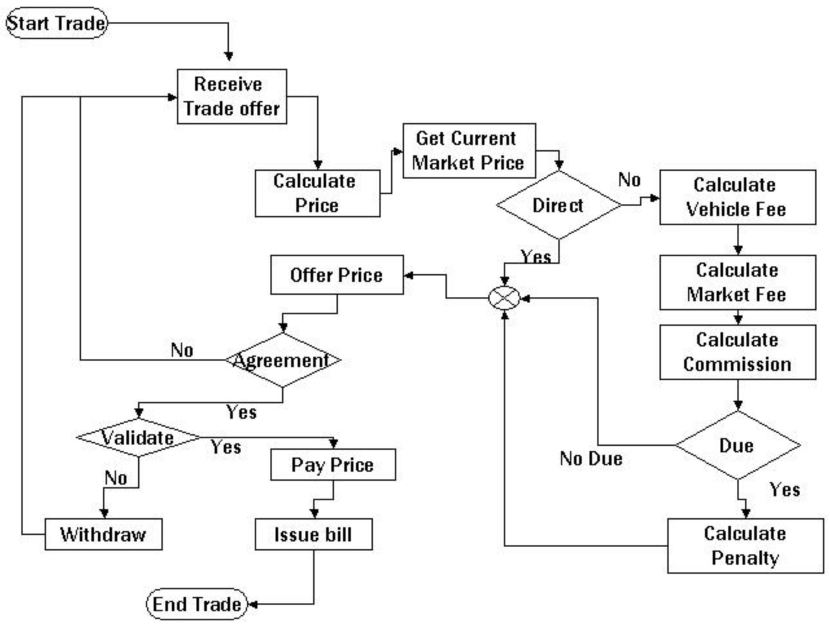


Figure 16: Workflow of Marketing of Agro-produce

3.4 Realization of Event-driven Service-oriented Architecture

The business logic adopted for this experiment follows the proposed legal framework for marketing of agricultural produce. Business processes like product discovery, negotiation, agreement, validation, payment, and transaction management are covered for implementation. We have applied these convergence and architecture on Agricultural Marketing Process and with this approach, we are expected to achieve following objectives:

- Enabling Direct and Indirect Marketing for farmers to sell agro-produce.
- Enabling effective product discovery for buyers.
- Enabling simultaneous composition and negotiations at different markets.
- Implementation of transparent business process and transaction management for APMCs.
- Providing effective implementation framework and architecture to realize the proposed legislation for the government.
- Enabling decision support for traders by providing access to services mapping to actual business processes.

Here events are generating from different sources such as trader, buyer, seller and different APMCs. Events generally reflect a change in an agricultural marketing business process and also trigger other event. Events are fine grained and represent data, message, location or context. For e.g. farmer located in the city of Visnagar wants to sell Mango in Ahmedabad APMC. EDSOA captures these diverse events dynamically from multiple sources. It associates this information using declarative rules and taking corresponding action based on conditions. EDSOA reduces the complexity using dynamic modeling and composition when information is coming in real-time and in parallel from diverse sources. SOA can address reliable sequential process, as it does not allow multiple events to execute an action and to attack above requirements with traditional service architecture. EDSOA and SOA can work together to achieve dynamic modeling, composition and execution of business processes by capturing event and correlating them using rules.

Different components of EDSOA are described below:

3.4.1 Information Providers

Agro-market and Agro-business processes are collection of services of APMC, like receive trade offer, calculate price, get current market price, market mediation, price calculation, offer price, pay price, issue bill and so on [49]. Thus, this component is planned as a collection of web-enabled Agribusiness services that can serve as information and service providers. These services help in storing and retrieving information from APMC database and external applications.

3.4.2 Event Manager

Event manager maintains the farmer subscription by retrieving his preference from the APMC database. When a farmer put forward any buy or sell request, event manager will treat as event and start the composition process. It will interact with ontology server, rule engine, composition engine, and based on the farmer marketing policy and rules it will generate the composition schema. It also generates recommendation to the farmer by maintaining subscription, preferences and with event correlation [47, 52].

3.4.3 Composition Engine

When farmer starts the Agro-produce marketing process, event manager interacts with composition engine and orchestrates the services. Composition engine follows the Agro marketing business rules and starts the backward and forward chain algorithms to manage the sequence of services. It generates the Agro marketing business process composition schema as per the event generated during the execution of a business process.

3.4.4 Ontology Server

Ontology service contains the agriculture domain ontology. Agricultural ontology provides common vocabulary about agricultural and marketing of agricultural produce. AGROVOC [92] is used as a base vocabulary to build Agricultural ontology. We have tried to cover all the terms, relations among them and the logical expression of the legal provision. The ontology serves as a building block to an agriculture information system. It answers the queries of farmers, helps to make decisions about the crop production, helps to generate the recommendation and helps in discovery, selection and composition of agro marketing services.

3.4.5 Rules Engine

Rules engine is the rules repository, which contains information in the form of rules. Rules are added by the agricultural experts. Agricultural Experts are agriculture scientists and APMC authorities, who add agriculture related and marketing related rules. An example of a rule is “If sale is direct, then seller is exempted to pay market fee”. This rule will be triggered against corresponding event like “A seller enters the market place” or “A farmer starts the trade”.

3.4.6 Execution Engine

Execution and management of distributed and heterogeneous services is more complex than traditional enterprise applications. Execution engine utilizes the grid middleware services and working as an integrated part of grid environment. Schema generated by composition engine will be executed by centralized execution engine. It is supported by grid middleware services for monitoring, event management, automation, orchestration, security, performance management, transaction management and notification services. It will take care of execution and monitoring of a business process. Any exception or final output will be forwarded to the event manager; from there it will be sent to the user.

3.4.7 Business Services

In Agro Marketing business process, core business functionalities are implemented as business services such as TradeOfferGridService, PayPriceService, NetCostComputationService. These services are developed by following WSRF, WSN and grid computing standards. Thus, they have support for state, notification, resource sharing and execution monitoring. These services work with Agricultural information provider services to get and store information about farmers and crops.

3.4.8 Grid Services

Various grid components and APIs are used to develop services which provide security, data management, resource management, virtualization and execution monitoring facilities. Grid Security Infrastructure (GSI) and Certification Authority (CA) are used to control the access of services and resources. OGSA-DAI (Data Access and Integration) are used to access and integrate datasets in a grid environment. Grid Resource Allocation and Management (GRAM) helped in execution management. Various other services are also used to achieve different functionalities such as index service is used to aggregate the resources.

The proposal for Event-driven Service-oriented Architecture, its lifecycle and realization of architecture based on agro-produce marketing use case scenario is discussed in this chapter. In the rest of the thesis, phases of EDSOA lifecycle are discussed in subsequent chapters. Chapter 4 discusses modeling and composition of event-driven business process. Chapter 5 discusses research experiments based on the development of dynamic business process with state, notification, service grouping and policy support. Chapter 6 describes the execution of direct and indirect agro-produce marketing process with notification support.

Chapter 4

Event-driven Service Modeling and Composition

The existing WS standards are not supporting event-driven business process. Business processes need to be composed and modified at runtime. Because of the limitation of current WS standards there is a gap between SOA and EDSOA for modeling and composition of event-driven process. In this chapter, we propose event based modeling and dynamic composition approach.

This chapter is organized as follows. Section 4.1 provides Event-driven composition modeling using Event Calculus and section 4.2 discusses dynamic composition schema generation using ECA rules. Section 4.3 describes the development and deployment of ontology to model both Agricultural and Agro Marketing domain.

4.1 Event Driven Composition Model

Modeling typically involves abstract representation of a business process. Designer and architect collect relevant information about the process and transforms into a model. They represent the model using formal language or graphical symbols. Due to the formalization, model supports simulation, automated processing, testing and visualization. There are many formal approaches and standards available to model and compose Web services as discussed in section 2.7. Existing approaches and standards are not able to describe business process with continuous changes in state and event. In this thesis Event Calculus based formal approach is proposed to model a business process using event.

4.1.1 Event Calculus

This section discusses the use of Event calculus to model event driven composition. Kowalski and Sergot [93] introduced event calculus as a logic language based on first order predicate calculus for reasoning of event and change to represent dynamic behaviour of a system. Event calculus is a formal language to provide framework to model the event-driven system in terms of event and fluent. It is based on a set of entities to represent event, properties and time period for which properties are held. A set of predicates about event and fluent are defined as below:

- *happens(e, t)* : Event e occurs at some time t that is within the time range t1 and t2.
- *initiates(e, f, t)* : Fluent f is initiated after the event e at time t.
- *terminates(e, f, t)* : Fluent f is terminated after the event e occurs at time t.
- *holds_at(f, t)* : Fluent f holds at time t.
- *clipped(t1, f, t2)* : Fluent f is terminated between t1 and t2 by the event e.
- *declipped(t1, f, t2)* : Fluent f is initiated between t1 and t2 by the event e.

Using these sets of predicates we can define an event-driven services and we can model event-driven business process workflow, which changes its state based on events [94]. Different activities of agricultural market trading process are modeled using event calculus. Execution sequence of services can be represented as under:

$$happens(start(Calculate\_currentmktprice), t) \longleftarrow happens(end(Calculate\_price), t) \quad (1)$$

Rule(1) states that at the end of the Calculate_price service Calculate_currentmktprice service is to be executed.

Selection Activities (XOR-split) where one of the services is selected among two or more alternatives services based on some condition can be modeled as below:

$$happens(start(Calulate\_vehiclefee), t) \longleftarrow happens(end(Calculate\_currentmktprice), t), \\ not\ holds\_at(direct, t)$$

$$happens(start\ (Offer\_price), t) \longleftarrow happens(end(Calculate\_currentmktprice), t), \\ holds\_at(direct, t) \quad (2)$$

Rule(2) states that end of service Calculate_currentmktprice starts Calculate_vehiclefee or Offer_price service based on trade condition (direct or indirect).

In XOR-join condition, if any one of the input activities is completed its execution, the activity at the join can start its execution. This condition can be modeled as below:

happens(start(Offer_price),t) $\longleftarrow$ *happens(end(Calculate_currentmktprice), t),*
holds_at(direct, t), end_same_or_after(end(Calulate_commission), t), not holds_at(due, t),
end_same_or_after(Calculate_penalty, t)

happens(start(Offer_price),t) $\longleftarrow$ *happens(end(Calulate_commission), t),*
not holds_at(due, t), end_after(Calculate_currentmktprice, t),
holds_at(direct, t) end_same_or_after(Calculate_penalty, t)

happens(start(Offer_price),t) $\longleftarrow$ *happens(end(Calculate_penalty), t),*
end_after(Calculate_currentmktprice, t), holds_at(direct, t), (3)
end_after(end(Calulate_commission), t), not holds_at(due, t)

Rule(3) states that service Offer_price starts when any one of the services like Calculate_currentmktprice with direct trade, Calulate_commission without due or Calculate_penalty service ends at the same time or after.

Similarly when we want to execute services or group of services multiple times, we can use Iteration conditions to describe the iteration of activities as follow.

happens(start(Re*ceive_tradeoffer), t)* $\longleftarrow$ *happens(end(start_trade),t)*

happens(start(Re*ceive_tradeoffer), t)* $\longleftarrow$ *happens(end(Offer_price),t),*
not holds_at(Agrrement, t)
happens(start(Validate), t) $\longleftarrow$ *happens(end(Offer_price),t), holds_at(Agreement, t)* (4)

Rule(4) states that the iteration of Receive_tradeoffer will start when new trade will start, or agreement is cancelled, or trade is withdrawn.

Using these predicates we can also model the parallel activities like AND-join and AND-split. In AND-join condition, a service can start its execution when all the preceding services finish their execution.

$$\begin{aligned} happens(start(service\_aj), t) \longleftarrow\ & happens(end(service\_a1),t1) \\ & happens(end(service\_a2),t2) \\ & happens(end(service\_an),tn) \\ & t = \max(t1,t2,...,tn) \end{aligned}$$

In case of AND-split condition, when a service finishes its execution, other multiple services start their execution in parallel.

$$\begin{aligned} & happens(start(service\_a1), t) \longleftarrow happens(end(service\_ai),t) \\ & happens(start(service\_a2), t) \longleftarrow happens(end(service\_ai),t) \\ & \ldots \\ & happens(start(service\_an), t) \longleftarrow happens(end(service\_ai),t) \end{aligned}$$

Various BPEL tags which we have discussed in section 2.7.1 can be represented by using Event calculus. Tags such as assign, invoke, receive, reply, sequence, flow, pick, while, switch, empty, wait and throw can be transformed to event calculus for formal verification and analysis of business process. Event calculus modeling is required for checking and analysis of business process consistency and execution [95].

4.2 Event-Condition-Action (ECA) Rules for Composition

In the previous section we have presented an event-driven Web services composition model for modeling of a business process. In this section we discuss dynamic Web services composition generation using ECA rules. Events and rules have been used in active database systems [96, 97]. Active database executes a corresponding action when event occurs and condition was true. Initialy Dayal et al. [98] have used ECA rules for workflow. Casati et al. [99-101] proposed a classification of rules and rule-based exception handling in workflow. ECA rules also used for automatic control, composition and execution of workflow [102-104].

The basic elements of ECA rules are Event, Condition and Action.

```
RULE<RuleName>[(<parameter>,…)]
WHEN<Event Expression>
IF<Condition 1> THEN <Action 1>
…
IF<Condition n> THEN <Action n>
ENDRULE<RuleName>
```

When an event occurs, the condition will be evaluated and action will be taken if the condition is true. So there will be a state change in a business process from state S_i to S_j. Business process contains a set of services and each service has precondition, input and output. While taking any action, composition engine will refer precondition, input and produce the output. Formal definition of ECA-rules-based composition is defined as under:

Definition 1 (ECA rule): An ECA rule R is defined as a tuple (E,C,A)
E is a finite set of events
C is a finite set of conditions
A is a finite set of activities
Where E is an event to trigger the rule, C is a condition to be satisfied to execute the action A. Event can be atomic or composite, atomic event (AE) is a single event and detected directly.

$$\mathrm{AE} \subseteq \{ \mathrm{e} \mid \forall \mathrm{a} \in \mathrm{A}, \mathrm{e} \in \mathrm{initialize(a), start(a), end(a), overtime(a), abortion(a), error(a)} \}$$

Composite event (CE) is a composition of atomic events through operators. Two operators AND and OR are defined. AND operator means both events have to occur and OR operator means at least one event should occur.

Condition is a boolean function of object(s) to represent constraints or relationships among object(s). Action is some kind of a function or an activity that is executed by condition evaluator. ECA rule performs various kinds of activities such as start activity, end activity and gives sequence of activities.

Definition 2 (ECA rule-based composition): An ECA rule-based composition is defined as a six-tuple (R, PRC, IP, OP, D, S)
R is a finite set of rules, $R \subseteq E * C * A$
PRC is a finite set of precondition
IP is a finite set of input
OP is a finite set of output
D is a finite set of data flow used in composition
S is a set of services
Precondition and input define the required parameters, required to execute Web services operation. Output defines the output parameters, which are generated after the execution of Web

services operation. Data flow defines the flow of data from one service to another service and S gives the set of services or sequence of services as shown in following code snippet.

```
<DataFlow>
<task>
<source value="sendrequest"> </source>
<target value="calculate_currentmktprice"> </target>
<variable value="crop"> </variable>
<variable value="grade"> </variable>
</task>
<task>
<source value="calculate_current_mktprice"> </source>
<target value="calculate_commision"> </target>
<variable value="crop"> </variable>
<variable value="grade"> </variable>
</task>
```

Code Snippet 1: DataFlow.xml

While taking any action, composition engine will refer precondition, input and produce the output. Backward chain rules are used to capture the precondition, input and output. Forward chain rules are used to capture the next action sequence [104, 105]. Composition rules are mainly based on pre and post conditions. Generally, proving theorems such as condition1 $\Rightarrow$ condition2 is a NP-complete problem, it raises issue related to compatibility and it is difficult to resolve. We are not considering it in our work. In the following section, we describe backward chain and forward chain algorithms for Web services composition.

4.2.1 Event-driven Composition Algorithm

In this section, we describe backward chain and forward chain algorithms for web services composition.

4.2.1.1 Backward Chain Algorithm

```
input : BackwardChain Rules (BCR)
output    :  A={A1,  A2,  A3,....  An}  where  A1,  A2...  An  are  the
activities to be performed
begin
    Task Pool TP = Φ      //empty initially
    for each rule r in BCR
    {
        if (r.pre-condition ∉ TP)
```

```
          add r.pre-condition to TP
     }
     A=TP
end
```

A backward chain rule specifies various pre-conditions (i.e. list of tasks that should be completed before a given task is initiated) for various tasks involved in the schema. Backward chain rules are specified in an xml file called Backward.xml. Backward chain algorithm starts with the target task that needs to be performed and looks for its pre-conditions until the pre-conditions coincide with the initial task. The following code snippet shows a snapshot of Backward.xml, which shows the precondition of CalculatePrice service.

```
<BackwardChain>
……………
<task>
<name value="calculatePrice"></name>
<constraint value="sendRequest::isAccepted=='yes'"></constraint>
<pre-condition value="complete_task(calculate_currentmktprice)">
</pre-condition>
<pre-condition value="complete_task(calculate_commision)">
</pre-condition>
<pre-condition value="complete_task(calculate_vehiclefee)">
</pre-condition>
<pre-condition value="complete_task(calculate_mktfee)">
</pre-condition>
</task> ………
```

Code Snippet 2: Backward.xml

4.2.1.2 Forward Chain Algorithm-

```
Input : ForwardChain Rules – FCR, A – set of activities that need to
be performed, initial task I
Output : P[n] where  Pi ∈ A and P is the process schema
begin
     Task Pool P =   Φ    //empty initially
     Add I to P
     temp=I
     index_of_task = Search(A,temp)
     //Search will return -1 if temp is not present in A
     while (index_of_task ! = - 1)
     {
          forward_index = Search(FCR,P[index_of_task].taskname)
          add FCR[forward_index].action to P
```

```
            temp = FCR[forward_index].action
            index_of_task = Search(A,temp)
      }
end
```

A forward rule is the form of an ECA rule where completion of a task is an event to execute the given action based on some condition. Forward chain rules are specified in an xml file called *Forward.xml*. The input of the algorithm is the list of tasks to be performed A (that was already generated by the BackwardChaining algorithm), the initial task I and the set of forward chain rules. We initially start with I and search for it in A. The search will return -1 if I is not present in A otherwise it will return the index value. Using index value we will look for a forward chain rule that specifies the action to be taken on I's completion. Now we update P to FCR.action and continue until we cover all the tasks specified in A. The following code snippet shows a snapshot of Forward.xml, which shows ECA rules:

```
<ForwardChain>
…………………..
<task>
<event value="TaskEvent::complete_task(sendRequest)"> </event>
<condition value="sendRequest::isAccepted=='yes">
</condition>
<action value="execute_task(calculate_currentmktprice)"> </action>
</task>…..
```

Code Snippet 3: Forward.xml

4.2.2 Generation of Dynamic Composition Schema

For the demonstration we have implemented *BackwardChaining* and *ForwardChaining* algorithms to achieve dynamic composition and to generate a dynamic BPEL schema at run time. Figure-17 and 18 shows the class diagram and interaction among the various classes required to generate the composition schema. These classes are part of the event manager and the composition engine. Whenever any event occurs, event manager queries the ontology server to correlate the event and pass on the event to the composition engine. Composition engine follows the backward and forward chain algorithm and checks the rules for precondition, input and output of services and triggers the ECA rules to generate the composition plan that will be forwarded to execution engine. Execution engine will execute the services according to the composition plan.

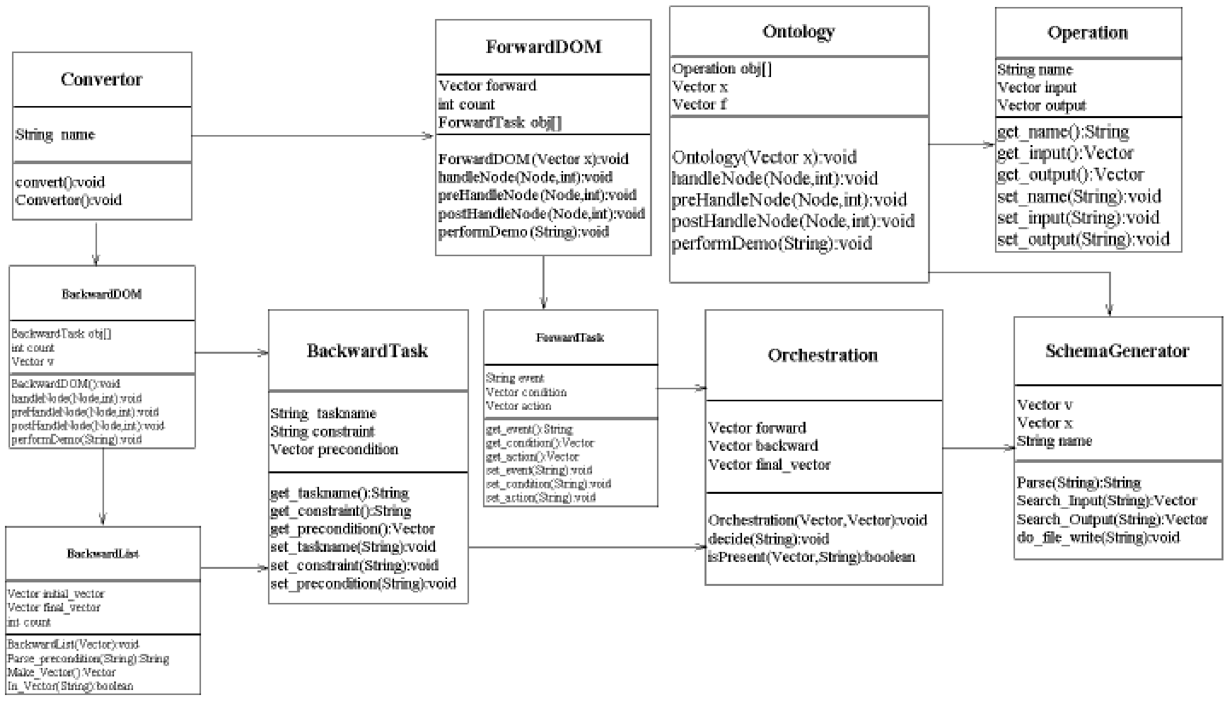


Figure 17: Class diagram for schema generation

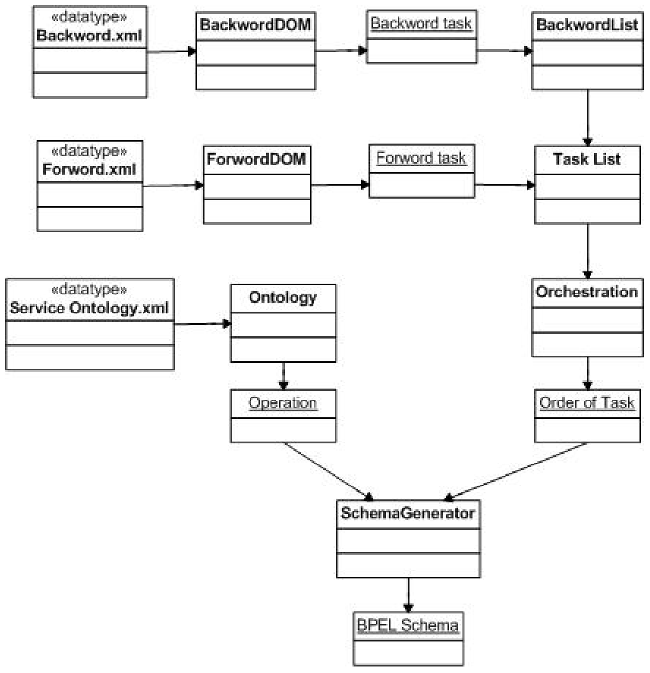


Figure 18: Class Interaction Diagram for schema generation

A brief outline and description of various classes are given below:

RuleML_Impl – It generates an xml file from a given RuleML that specifies the various backward and forward chain rules. Note that we can ignore the use of this class, if the rules are already written in standard xml format as specified by our prototype. The use of this class makes our prototype compatible with standards like RuleML.

BackwardDOM – It parses the Backward.xml file and converts each of the backward chain rules into objects of class BackwardTask whose attributes are namely taskname, constraint and pre-conditions. A vector containing these objects is then passed on to the BackwardList class.

BackwardList – It applies the BackwardChaining algorithm to the given vector and creates a new vector containing the tasks to be performed and passes this vector to the Orchestration class.

ForwardDOM – It parses the Forward.xml file and converts each of the forward chain rules into objects of class ForwardTask whose attributes are namely event, condition and actions. A vector containing these objects passed on to the Orchestration.java class.

Orchestration – It takes a vector from the BackwardList class and a vector from the ForwardDOM class as input. It implements the ForwardChaining algorithm for both of these vectors and generates a new vector that contains various tasks to be performed in order. This new vector is then passed to the SchemaGenerator class.

Ontology - It parses the Service-Ontology.xml file and converts each of the backward chain rules into objects of class Operation, whose attributes are namely taskname, inputvariables, and outputvariables. A vector containing these objects is then passed to the SchemaGenerator class.

SchemaGenerator – It takes vectors from Orchestration class and Ontology class as input. It uses both these vectors to generate a BPEL schema at run-time. Last code snippet of this section shows generated BPEL file.

Driver – It contains the main function of the prototype.

The final output of these classes is a BPEL file containing the dynamically generated process schema based on the backward and forward chain rules. The heart of the BPEL file is the sequence activity that contains the above mentioned order. The following code snippet shows the generated BPEL schema. The BPEL file is in accordance with the BPEL syntax specified by the Oracle BPEL execution engine [106].

```
<sequence>
<receive partnerLink = "Farmer" portType = "tns:APMC Trading"
operation = "sendRequest"
variable = "FarmerName"
variable = "CropName"
variable = "Grade"
variable = "MandiName"
variable = "Price"
variable = "Mode"
</receive>
<invoke name = "calculate_currentmktprice" partnerlink = "Trader"
portType = "task:TaskManager" operation = "calculate_currentmktprice"
inputVariable = "CropName"
inputVariable = "Grade"
inputVariable = "MandiName"
inputVariable = "Price"
outputVariable = "CurrentPrice" /> …………… </sequence>
```

Code Snippet 4: Generated BPEL schema

4.3 Ontology to model real world

There are many tools available to build ontology; some tools are equipped with the facility of validating the ontology and the reasoning capability to infer new facts from the concepts. Some of the known tools for developing ontology include Protégé, OilEd, KAON, OntoEdit and OntoStudio. We have used Protégé tool [107] for the creation of ontology in our experiment. We have used the following approach for the development of ontology.

Development of Agricultural ontology

Each concept is identified as a set of individual class and defined under the set owl:Thing. In the same fashion, other concepts of the agricultural domain are identified appropriately and each one is added as a subclass. Different types of classes like primitive class, class with restriction etc are defined. This kind of restriction can be covered in OWL representation as follows:

```
  <owl:Class rdf:ID = "Fenugreek">
      <rdfs:subClassOf>
        <owl:Class rdf:ID = "Spices"/>
      </rdfs:subClassOf>
    </owl:Class>
    <owl:Class rdf:about = "#Spices">
```

```
      <rdfs:subClassOf>
        <owl:Class rdf:ID = "Crop"/>
      </rdfs:subClassOf>
    </owl:Class>
    <owl:Class rdf:ID = "Cucumber">
      <rdfs:subClassOf>
        <owl:Class rdf:ID = "Vegetable"/>
      </rdfs:subClassOf>
    </owl:Class>
    <owl:Class rdf:ID = "Oats">
      <rdfs:subClassOf>
        <owl:Class rdf:ID = "Cereal"/>
      </rdfs:subClassOf>
    </owl:Class>
    <owl:Class rdf:ID = "Marketing_Concepts">
      <rdfs:subClassOf>
        <owl:Class rdf:ID = "Agriculture_Market"/>
      </rdfs:subClassOf>
    </owl:Class>
```

Code Snippet 5: Agricultural ontology

After the identification of each concept, the relationship and properties such as inverse, transitive, symmetrical or functional relationships are identified. After that we have defined the restrictions with concepts like Universal Existential Quantifiers, cardinality restrictions on the value etc. Once concepts and all the relationships are defined with appropriate restrictions, instances of all the classes are created.

Development of Agro Marketing business process ontology

Marketing of agro-produce processes are also defined in terms of ontology. We have identified the services, which are involved in the business process and created process ontology to specify common vocabulary. The process ontology defines the basic concepts and terminologies used by all the participants of an agricultural market. It consists of a set of synonyms and a set of service classes that specify the attributes and operations, relationship and property of a service. Apart from the functional properties of a service, a service class also defines non-functional properties of the service such as response time, execution time and cost.

The code snippet of the Process Ontology is shown below:

```
<ontology NAME = "APMC Trading" VERSION = "1.0">
<domain> eTrade </domain>
<domainSynonym> eAgriBusiness </domainSynonym>
```

```
<domainSynonym> Mandi </domainSynonym>
<domainSynonym> Trading </domainSynonym>
<domainSynonym> InternetBuying </domainSynonym>
<domainSynonym> APMCTrading </domainSynonym>
<superDomain> ROOT </superDomain>

<serviceclass NAME = "SellCropservice" SUPERCLASS-OF =
"DirectSellingservice,IndirectSellingservice">
<serviceDescription> This enables a farmer to sell his producer to a
consumer or a trader </serviceDescription>
<attribute NAME = "serviceProvider" TYPE = "String"> </attribute>
<attribute NAME = "url" TYPE = "String"> </attribute>

<operation>
<method NAME = "calculatePrice"> </method>
<inputData NAME = "CropName" TYPE = "String"> </inputData>
<inputData NAME = "Grade" TYPE = "String"> </inputData>
<inputData NAME = "MandiName" TYPE = "String"> </inputData>
<inputData NAME = "Price" TYPE = "Float"> </inputData>
<outputData NAME = "AmtPayable" TYPE = "Float"> </outputData>
</operation>
```

Code Snippet 6: agro-produce marketing ontology

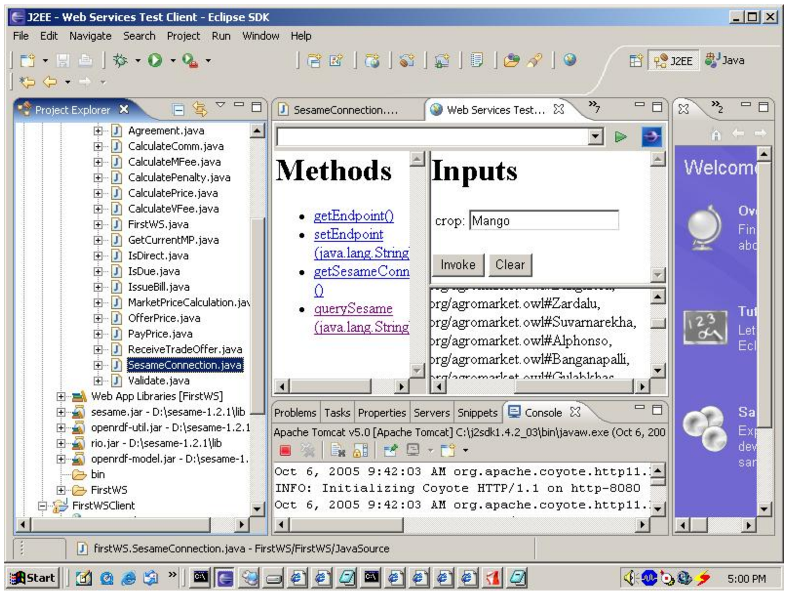
4.4 Annotation of Web Services

The agricultural domain ontologies developed in the previous section are used for the annotation of the web services. Annotation of Web Services enhances service description, discovery and composition of Web services.

• The semantic Web service representation will make it possible to annotate various Web services deployed at different markets. Hence, the concentration will be on semantically enriching the developed Web services to enable appropriate integration.

• The Business Process Modeler will utilize annotated Web services to design complete business process according to the provisions of the Model act.

The ontology repository is deployed to host the agricultural ontology and to access it programmatically by various services. We have used Sesame [108] tool to host ontology repository. It supports standard ontology formats, standard query languages and has the capability of persistent storage in popular database products. Web services is developed to query and retrieve the ontology from the ontology repository. As shown in the following figure, it takes the input string from a farmer to create a valid SeRQL query expression with required credentials and retrieves the response from the ontology.


J2EE - Web Services Test Client - Eclipse SDK
File Edit Navigate Search Project Run Window Help
Project Explorer
Agreement.java
CalculateComm.java
CalculateMFee.java
CalculatePenalty.java
CalculatePrice.java
CalculateVFee.java
FirstWS.java
GetCurrentMP.java
IsDirect.java
IsDue.java
IssueBill.java
MarketPriceCalculation.java
OfferPrice.java
PayPrice.java
ReceiveTradeOffer.java
SesameConnection.java
Validate.java
Web App Libraries [FirstWS]
sesame.jar - D:\sesame-1.2.1\lib
openrdf-util.jar - D:\sesame-1.2.1
rio.jar - D:\sesame-1.2.1\lib
openrdf-model.jar - D:\sesame-1.
bin
FirstWS
FirstWSClient
SesameConnection....
Web Services Test...
Methods
getEndpoint()
setEndpoint (java.lang.String
getSesameConn ()
querySesame (java.lang.String
Inputs
Invoke
Clear
Problems Tasks Properties Servers Snippets Console
firstWS.SesameConnection.java - FirstWS/FirstWS/JavaSource
Start
5:00 PM

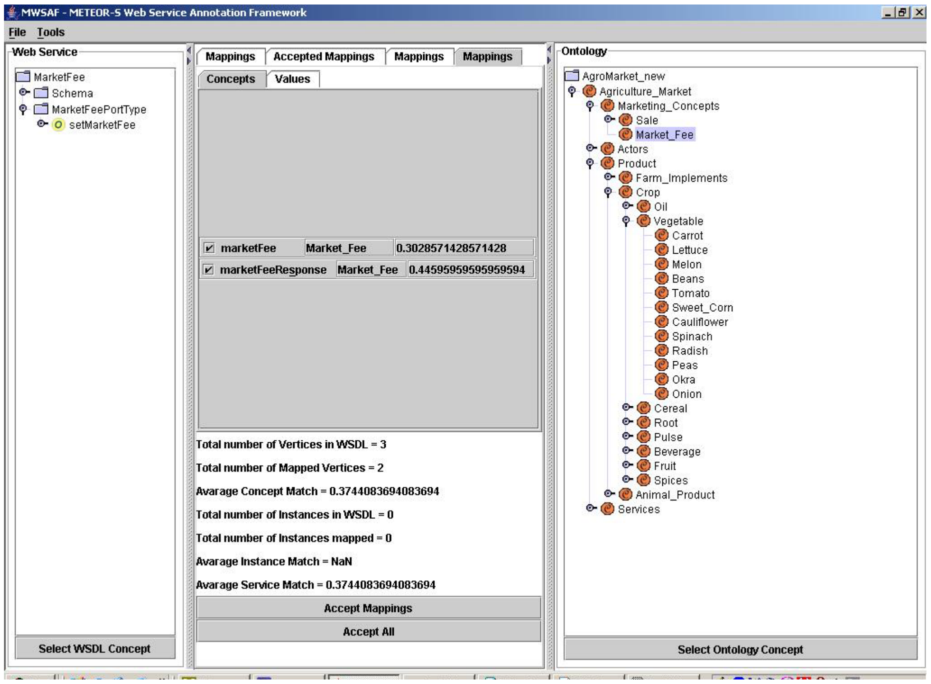
crop: Mango

```
org/agromarket.owl#Zardalu,
org/agromarket.owl#Suvarnarekha,
org/agromarket.owl#Alphonso,
org/agromarket.owl#Banganapalli,
```

```
Apache Tomcat v5.0 [Apache Tomcat] C:\j2sdk1.4.2_03\bin\javaw.exe (Oct 6, 200
Oct 6, 2005 9:42:03 AM org.apache.coyote.http11.
INFO: Initializing Coyote HTTP/1.1 on http-8080
Oct 6, 2005 9:42:03 AM org.apache.coyote.http11.
```

Figure 19: Web Service Client to query the ontology repository

For the annotation of Web services, we have used METEOR-S WSDL-S standard [109] and Web Service Annotation Framework (MWSAF) [80].



✓	marketFee	Market_Fee	0.3028571428571428
✓	marketFeeResponse	Market_Fee	0.44595959595959594

```
Total number of Vertices in WSDL = 3
Total number of Mapped Vertices = 2
Avarage Concept Match = 0.3744083694083694
Total number of Instances in WSDL = 0
Total number of Instances mapped = 0
Avarage Instance Match = NaN
Avarage Service Match = 0.3744083694083694
```

Figure 20: METEOR-S Web Service Annotation Framework

MWSAF matches the terms used in the WSDL with the concepts given in the ontology and generates the annotated WSDL file. Annotated WSDL file code snippet is shown below:

```
<message name="MarketFeeInputMessage">
     <part LSDISExt:onto-concept="AgroMarket_new:Market_Fee"
     element="tns:marketFee" name="parameters"/>
</message>
<message name="MarketFeeOutputMessage">
     <part LSDISExt:onto-concept="AgroMarket_new:Market_Fee"
     element="tns:marketFeeResponse" name="parameters"/>
</message>
```

Code Snippet 7: Annotated WSDL

In order to execute event-driven business process we have to implement stateful Web services with notification and middleware support for execution monitoring of business process. In the next chapter we will discuss the development of dynamic business process. We will also discuss how we can achieve service grouping, group notification and policy-driven business process.

Chapter 5

Research Experiments

The proposed architecture for Event-driven Service-oriented Architecture, discussed in chapter 3, is realized in this chapter with the help of research experiments. In this chapter, prototype implementation of the Grid based event-driven business process is presented to illustrate the key concepts and ideas of proposed approach. Different built-in features of the WSRF family specifications naturally map the demanding requirements of grid business services and are the first choice for these research experiments.

Grid Business Process (GBS) to capture the requirements of a dynamic business process is proposed and discussed here. The grid business process is very dynamic in nature, requires built-in support for persistence of the state, notification, monitoring and management. In section 1.3 issues and challenges of existing standards are discussed. Recently different specifications built on the top of Web service standards originated from the grid paradigm to address these issues. These emerging specifications are discussed in chapter 2. Basic understanding of these specifications is essential for designing of robust collaborative enterprise applications. Sections 5.1 and 5.2 discuss architecture and different workflows of grid business process. Section 5.3 discusses implementation of the various services for direct trade and section 5.4 discusses indirect trading services. Section 5.5 presents related research experiments to achieve automated negotiation, recommendation, service grouping, group notification and policy-based business process. Result and discussion of the deployment and execution of the grid business process is provided in the next chapter.

5.1 Architecture for Grid Business Process

To demonstrate the grid business process, agro-product marketing process has been developed. In any typical Indian Agro-market, wholesale spot markets and derivative markets are the emerging hubs of the agricultural marketing business. Agro trade in these markets is heavily influenced by the local socio-economic and cultural characteristics. This leads to variations in the crop prices of the same crop in different markets. The producers have little choice to search for the best available price and are forced to sell their products in a local market. The prohibitive transport and storage costs can also play a pivotal role apart from the urgency to sell perishable products. Buyers and wholesalers experience difficulties in purchasing the desired quality of products at competitive prices. A typical trade in a Grid Business Process can span across different markets located at various places. Privately owned markets, food processing and other related industries are allowed to trade directly with the farmers. Trading in such a competitive markets will be more complex than the one in the conventional trading scenario, constrained to a single geographical market.

This section discusses architecture for grid business process. Use case diagram of agricultural trade workflow is shown in the figure-21. It has been observed that business services might be implemented and hosted by a separate organization depending upon the specialization of the organization. If development is done using service orientation, it is likely that such business services are implemented and exposed as Web services. There is a possibility that other organizations can be enabled to utilize these loosely coupled services to meet their requirements. From the service provider's point of view, as these services can be joined together to constitute a complete business process, it has become essential to make provisions for efficient integration with heterogeneous client environments.

5.1.1 Components of Grid Business Process

We are following the event-driven service-oriented architecture to provide state, notification, execution monitoring and scalability. The role of the targeted system is to support EDSOA lifecycle. The proposal is mainly dependent upon various WS-* specifications, specifically set of WSRF and WS-Notification (WSN) specifications. The business logic adopted from the marketing of agro-produce use case scenario. The distributed architecture implemented in the use case comprises of various components and resources; the role of these resources and components in the application is discussed below:

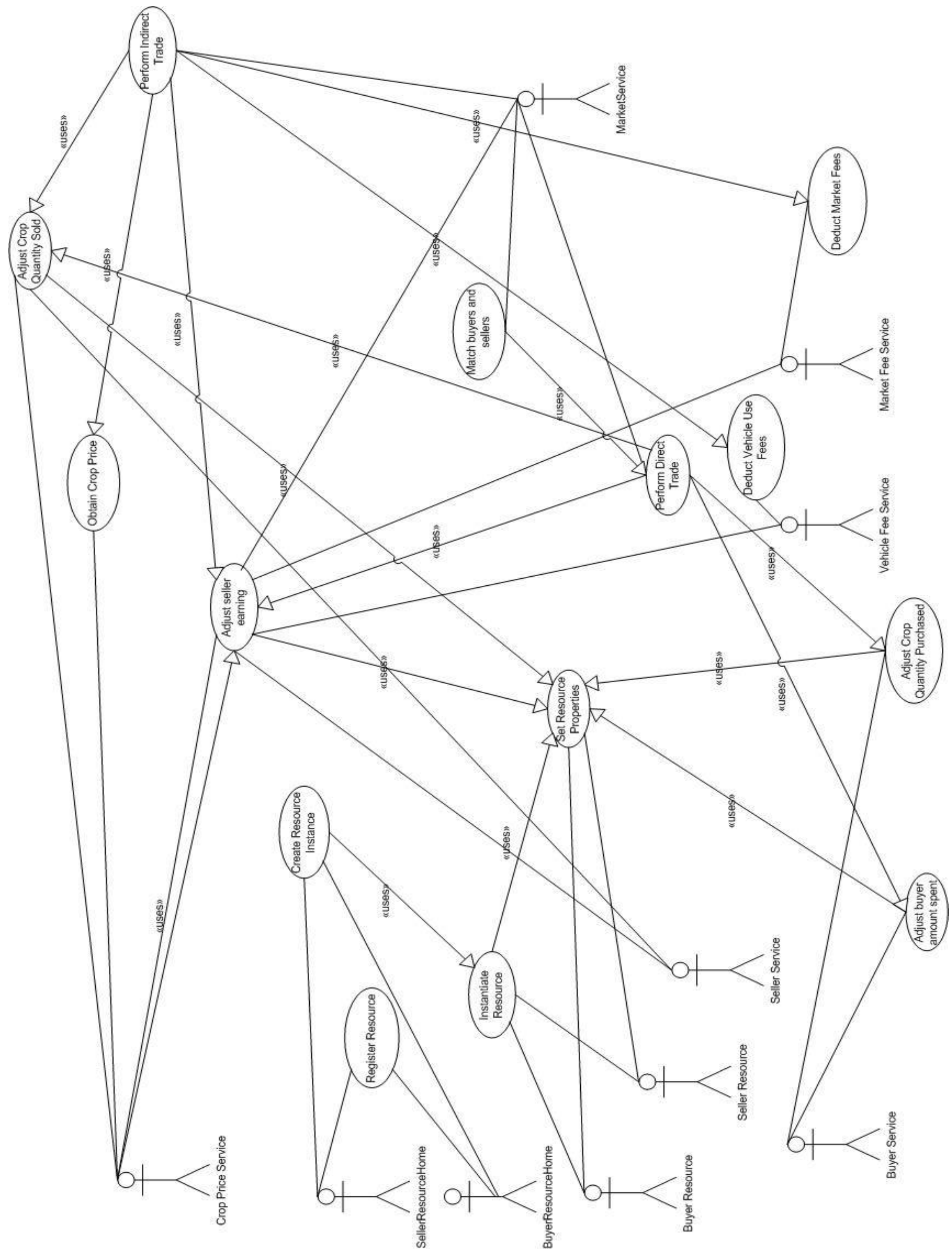


Figure 21: Use Case Diagram of Trading Process of Agricultural Marketing System

5.1.2 Grid Manager

The Grid Manager Service is implementation of the Factory/Instance Collection Pattern (Section 5.3.6) which itself is extension of the Implied Resource Pattern commonly used in WSRF. It initiates different type of resources such as Seller, Buyer, Crop and Market according to the client request. Different grid nodes can host these resources. The Grid Manager orchestrates different components of the system in predefined manner for smooth working of the whole application. The Grid Manager is the first point of the contact with the system.

5.1.3 Grid Nodes

The Grid Nodes are hosting environment, which can host different vanilla or stateful Web services. The services hosted on different nodes are used in various combinations to capture the requirement of the application. Each initialized WS-Resource has Endpoint Reference (EPR) attached to it, which identifies the resource itself and the location of URL of the managing Instance service. Clients use the corresponding EPR to interact with the specific resource.

5.1.4 Grid Registry

The Grid Registry is the special service, which keeps track of various resources initialized by the Grid Manager during the course of application. The seller or buyer can query the Grid Registry to search appropriate resources and can update the resource related to it. Grid Registry is based on the WS-ServiceGroup specification, which aggregates information related to different resources for search and query.

5.1.5 Grid Client

Grid Client is an external application, which either requests for initialization of the resource/s or interacts with the existing resource/s. The client has compatibility with WSA specification to work with WS-Resources and corresponding services through their EPR. Manipulation of state-full resources requires WSRF API available on the client side to create new resource, query, destroy or modify existing resources etc. Seller and Buyer fall in the category of the Grid Client. In this case study, the Grid Client has three flavors, one for the buyers, one for the sellers and one for the administrator, which instantiates the market instance for the trade.

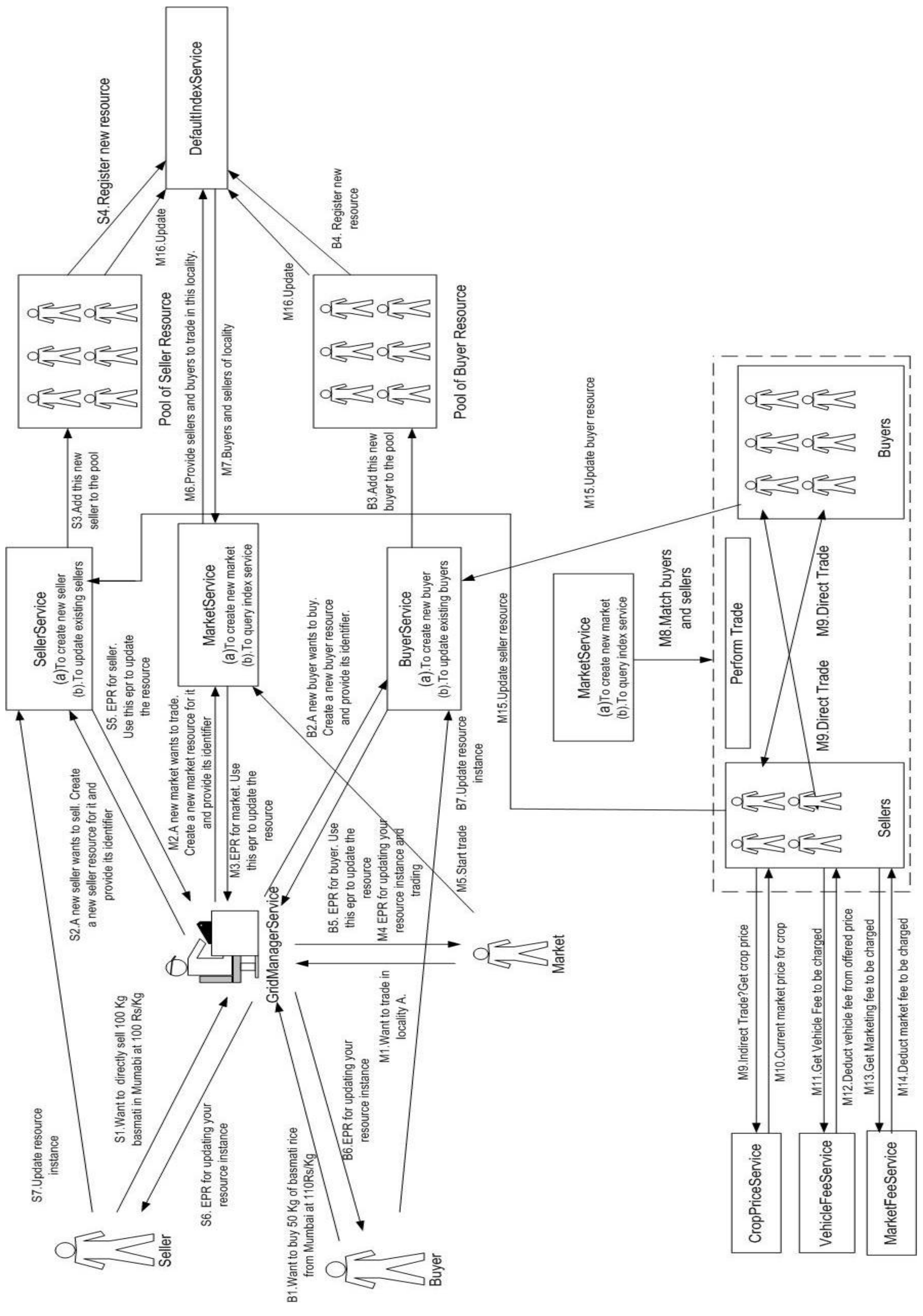

DefaultIndexService
S4.Register new resource
M16.Update
B4. Register new resource
Pool of Seller Resource
Pool of Buyer Resource
M6.Provide sellers and buyers to trade in this locality.
M7.Buyers and sellers of locality
S3.Add this new seller to the pool
B3.Add this new buyer to the pool
M15.Update buyer resource
SellerService
(a)To create new seller
(b).To update existing sellers
MarketService
(a)To create new market
(b).To query index service
BuyerService
(a).To create new buyer
(b).To update existing buyers
M15.Update seller resource
M8.Match buyers and sellers
Perform Trade
M9.Direct Trade
M9.Direct Trade
Buyers
Sellers
S5. EPR for seller. Use this epr to update the resource
S2.A new seller wants to sell. Create a new seller resource for it and provide its identifier
M2.A new market wants to trade. Create a new market resource for it and provide its identifier
M3.EPR for market. Use this epr to update the resource
B2.A new buyer wants to buy. Create a new buyer resource and provide its identifier.
B5. EPR for buyer. Use this epr to update the resource
M4 EPR for updating your resource instance and trading
B7.Update resource instance
M5.Start trade
GridManagerService
Market
M1.Want to trade in locality A.
S7.Update resource instance
S1.Want to directly sell 100 Kg basmati in Mumbai at 100 Rs/Kg
S6. EPR for updating your resource instance
B1.Want to buy 50 Kg of basmati rice from Mumbai at 110Rs/Kg
B6.EPR for updating your resource instance
Seller
Buyer
M9.Indirect Trade?Get crop price
M10.Current market price for crop
M11.Get Vehicle Fee to be charged
M12.Deduct vehicle fee from offered price
M13.Get Marketing fee to be charged
M14.Deduct market fee to be charged
CropPriceService
VehicleFeeService
MarketFeeService

Figure 22: Architecture for Grid Business Process

5.2 Interactions among Components

In the architecture, various WS-Resources are involved and which are initialized during the different stages of the business process. Following WS-Resources are involved in the agro-produce marketing scenario:

- Seller
- Buyer
- Market

At any time there can be various instances of similar WS-Resource instantiated during different phases of the business process lifecycle. For example, at any time there can be various buyers and sellers in a single market and similarly the same buyer or seller may be participating in different transactions initiated in different markets. In the following sub-sections, steps involved in the instantiation of different WS-Resources and interactions among different components involved in the instantiation are illustrated through sequence diagram.

5.2.1 Instantiation and Interaction with the Seller entity

The first interaction of a seller with the system results in the creation of the new seller WS-Resource. This resource captures all the details of the specific seller and is used for future interactions with that seller. The Figure-23 shows the sequence of events, which allows registration of the buyer with intention to purchase the required produce in the market.

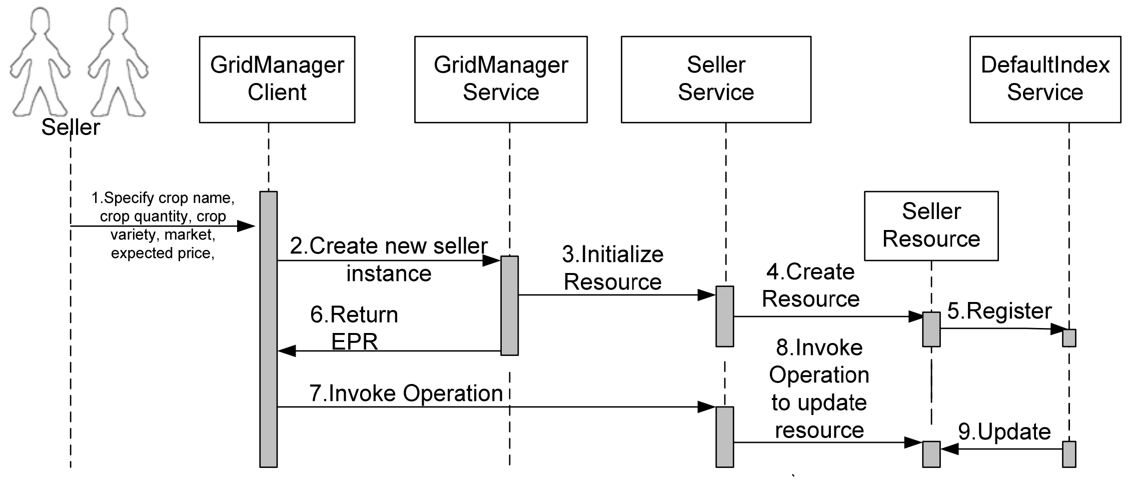


Figure 23: Seller Service Sequence Diagram

1 The trading for a seller begins with the expression of intent to sell the available crops by supplying crop name, crop quantity, crop variety, expected price/kg of crop and market city for trade by using the GridManagerClient.

2 The GridManagerClient sends a request to GridManagerService, which performs look up for instance service of a Seller to create an instance of SellerResource.

3 Newly created Seller resource, registers with the DefaultIndexService (i.e. Grid Registry) exposed by container to register the details of the newly created seller resource.

4 The GridManager returns back the EPR of a newly created SellerResource to the GridManagerClient.

5 The GridManagerClient, utilizes the EPR to invoke different operations to update the newly created resource.

5.2.2 Instantiation and Interaction with the Buyer entity

Similarly the first interaction of the buyer with the system results in the creation of the new buyer WS-Resource. This resource captures all the details of a specific buyer and is used for the future interactions with that seller. The Figure-24 shows the sequence of events, which allows registration of a buyer with intention to participate in the trade.

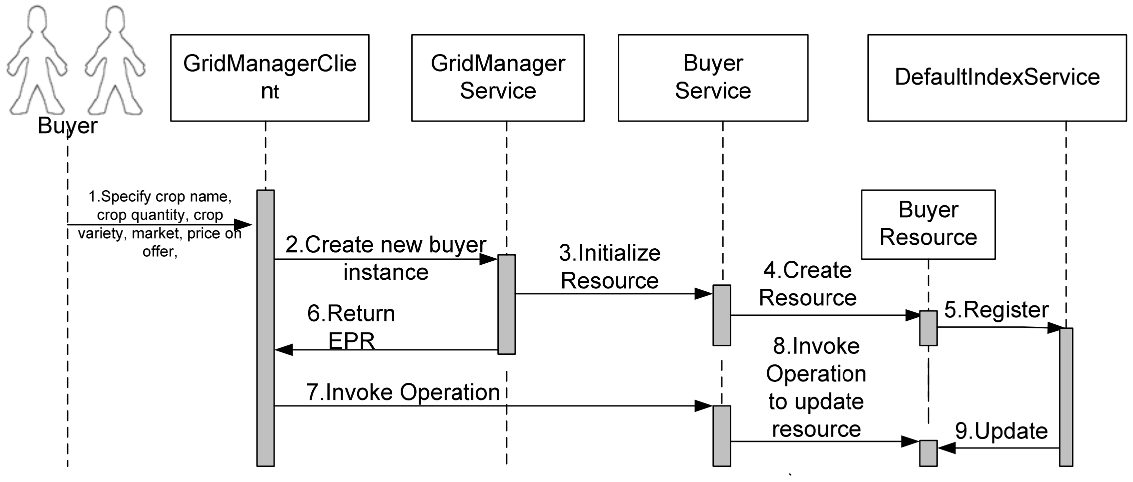


Figure 24: Buyer Service Sequence Diagram

1 The trading for a buyer begins with the expression of intent to buy the available crops by supplying crop name, crop quantity, crop variety, offered price/kg of crop and market city for trade by using the GridManagerClient.

2 The GridManagerClient sends a request to GridManagerService, which looks up the instance service of a Buyer to create an instance of BuyerResource.

3 Newly created Buyer resource, registers with the DefaultIndexService (i.e. Grid Registry) exposed by Globus container to register the details of the newly created buyer resource

4 The GridManager returns back the EPR of a newly created BuyerResource to the GridManagerClient.

5 The EPR is utilized by the GridManagerClient to invoke different operations to update the newly created resource.

5.2.3 Market Service for Direct Trading

In Direct Trading the system matches resources for different sellers and buyers which are trading at the same location. Once any such match is found then it means there is at least one buyer interested in the produce of a seller. To simulate the successful trade, the resource properties of each seller and the buyer are updated. Below is the sequence diagram of events involved in direct trading.

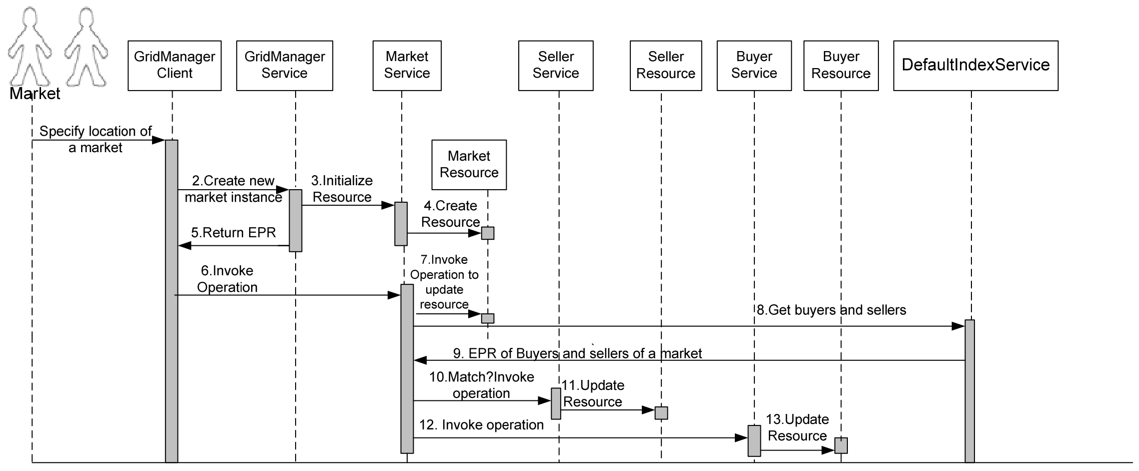


Figure 25: Sequence Diagram of Market Service for Direct Trading

1 The first step for successful trade is the instantiation of the MarketResource by passing the required location. The market only starts its operation once MarketResource is instantiated through GridManagerClient.

2 The GridManagerClient invokes the GridManagerService to create a new instance of Market Resource. The GridManagerService triggers the creation of a new MarketResource using the Market instance service and returns the EPR to the GridManagerClient.

3 The newly created EPR is utilized to invoke operations to start trade.

4 The market service queries the DefaultIndexService (i.e. Grid Registry) by supplying the location to obtain the information about sellers and buyers registered with preferences to trade at a particular location for which this market service instance is running.

5 The DefaultIndexService returns the EPR of the Buyer and Seller resources interested in trading at a given location.

6 The MarketService compares for the resource properties of the buyer and seller resources by utilizing retrieved EPR’s. If the trading mode of a seller is direct and if its resource properties match with that required by a buyer, trading is performed by invoking operations on the seller and buyer resources to adjust the crop quantities and earning by a seller, crop quantity and amount spent for a buyer.

5.2.4 Market Service for Indirect Trading

The sequence of events for indirect trade is quite similar to the direct trade. The initial matching of the seller and the buyer resources in the market is followed by invoking operations for the services specific to indirect trading and calculation of the final price after deducting appropriate charges. Below is the sequence diagram for the Indirect Trading:

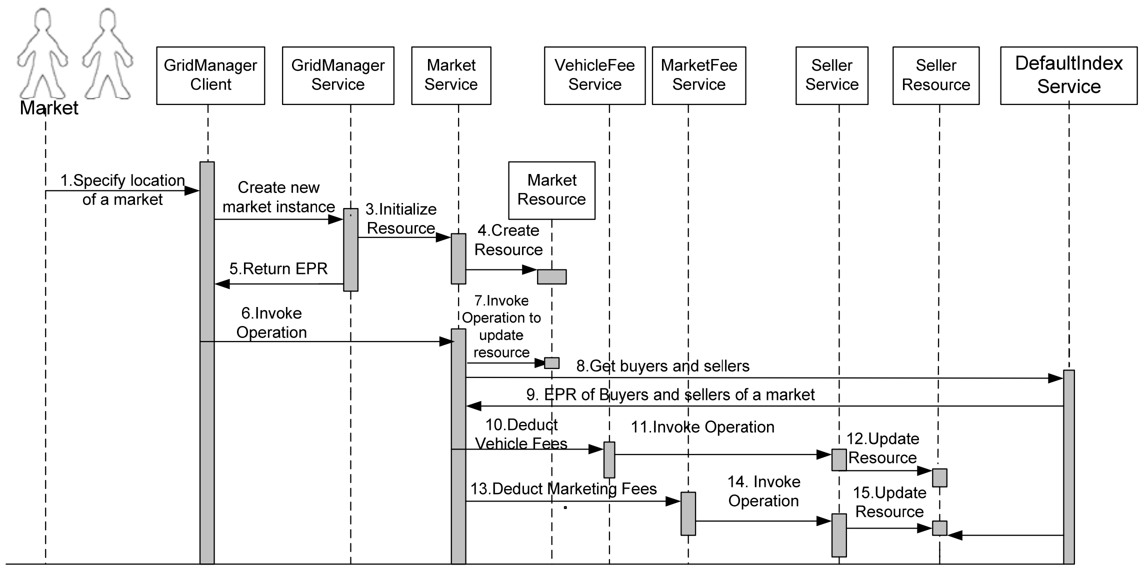


Figure 26: Sequence Diagram of Market Service for Indirect Trading

- The market starts its operation by expressing its intent to trade. Market uses the GridManagerClient and informs about the location at which it wants to trade.
- The GridManagerClient invokes the GridManagerService to create a new instance of Market Resource. The GridManagerService triggers the creation of a new MarketResource using the Market instance service and returns the EPR to the GridManagerClient.
- The newly created EPR is utilized to invoke operations to start trade.
- The MarketService queries the DefaultIndexService (i.e. Grid Registry) by supplying the location to obtain the information about sellers and buyers registered with preferences to trade at a particular location for which this market service instance is running.
- The DefaultIndexService returns the EPR of Buyer and Seller resources interested in trading at a given location.
- The MarketService compares the resource properties of the seller and buyers trading at the same location. On any successful match, the trade preference of the seller is queried for indirect trading. The indirect trading is simulated by deducting transportation and marketing fees from the expected price.

- As a result of successful transaction the seller resource is updated to reflect the new values i.e. crop still available for sell and total earnings.
- Similarly the buyer resource is updated to reflect new quantity of crop required and amount spent in a single year.

5.3 Services for Direct Trading

In the section 5.1 different components required in any grid applications are discussed. In the agro-marketing use case apart from these components, different stake holders are also involved such as buyer, seller, market etc. These components and stake holders are implemented either as stateful Web services or simple stateless Web services. The list of different developed Web services with their role and responsibilities is given below:

Seller Service: The Seller Service implements the business logic necessary for the seller. This service exposes different operations related to the seller resource such as instantiation of seller resources, monitoring and modification of different properties for seller resources. These operations are either executed manually by the seller or invoked automatically by other components during the course of the application as discussed in the different sequence diagrams in section 5.2.

Buyer Service: The Buyer Service is the implementation of buyer stake holder. It instantiates the resource for a buyer, and provides different operations on the buyer resources. Instantiation of a buyer resource results in the registration of a Buyer with the system for future interactions. The service instantiates a new resource instance for the buyer after receiving the preferences. The service also registers the preferences with the DefaultIndexService and provides operations to be invoked on the resource instance.

Market Service: The Market Service acts as an intermediary between buyers and sellers. The role of the Market Service is like a broker service among different stake holders. It queries the Grid Registry to obtain the list of sellers and buyers trading in the same market and continuously monitors the trading preferences of buyers and sellers to perform trade. Market Service matches the seller and buyer preferences. After successful match it instantiates the new trade and transaction. Initiation of the trade triggers sequence of events which results in the updating of the buyer and the seller resources involved in the trade and notification to interested parties.

GridManager Service: The GridManager Service acts as a common factory service to instantiate different resources involved in the research experiment. The GridManager Service is developed according to the Factory/Instance Collection Pattern and Master/Slave Pattern [55]. A GridManager Service instantiates different resource through the specific service, for example seller resource is instantiated through Seller Service.

5.3.1 Grid Registry

The role of the Grid Registry is to monitor all the resource instantiated during the life cycle of an application; irrespective of its nature and type. GT4 provides the DefaultIndexService according to the specifications of WS-ServiceGroup discussed in the section 2.3.2. The Index Service provides the required functionalities of the Grid Registry; which aggregates the information related to all available resources at one place. It can be built on top of the other Index Services in the hierarchical manner. Each child Index Service leads to better management and organization of resources by mapping specific type of resources. Each grid node container has one Index Service supporting all the operations required by WS-ServiceGroup specification. By default the resources instantiated through resource homes are not registered with the Index Service. The resource home explicitly registers newly created resource instance with the Index Service; which aggregates all the resources at a single point. The registration process is done through ServiceGroupRegistrationClient. Code of the resource home is to register the resource with the Index Service. It is given in the appendix.

The resources have tight control on the amount of information they want to advertise in the Index Service. As well as, how frequently the Index Service queries their state to update itself and the polling interval. The Index Service is a distributed registry; which keeps on updating itself regularly after specific intervals. Any change in the state of the resource or its deletion results in the notification of the Index Service; which adjusts itself accordingly. This information is provided in the form of configuration file '*registration.xml*' for each resource. Configuration registration.xml file is given in the appendix.

5.3.2 GridManagerService

The "GridManager" service is an interface provided to the Client application to instantiate resources related to different stateful Web services. The GridManager instantiates appropriate resource according to the client preference.

5.3.2.1 WSDL Document for GridManagerService

The development of the Web service must be according to the corresponding WSDL document and it is part of contract of the user of the service. WSDL document of the GridManager Web service is given in appendix. There are few important things to note about this WSDL file; which may differ from any normal WSDL document.

- This WSDL file is using a data type declared externally i.e. wsa:EndpointReference. The WSDL needs to import any externally declared data type used within the document, so that parsing tools can create appropriate stubs and proxies. The imported data types are referenced with the fully qualified name; which requires declaring the appropriate namespace in the <wsdl: definition> element as an attribute.
- The WSDL document for the GridManager Service exposes only one operation createResource which takes single String as a parameter wrapped in the data type createResource. Based on the value of the string parameter appropriate resource is instantiated and the corresponding EPR is returned to the client for future interactions.
- The GridManagerPortType is a collection of all the operations exposed by the GridManager service.

5.3.2.2 Implementation of GridManagerService

The WSDL document is a contract between the service and the client. Development of the service strictly follows this contract and it has one-to-one mapping of all functionality advertised through the WSDL. The GridManager implements the only operation, exposed by the WSDL document. We can have any number of utility operations to support the mandatory operation. The GridManager createResource() method instantiates the appropriate resource and returns the corresponding EPR. The main points related to the development are discussed below with reference to relevant implementation. Implementation file is given in appendix.

1 The createResource operation retrieves the name of the service from the client request for which the resource has to be instantiated. The serviceName is a private attribute of type string in the CreateResource class.

2 The getInstanceResourceHome is a private utility method to obtain the home of an appropriate resource. Each resource is instantiated through its corresponding home.

3 The object of ResourceHome is utilized to invoke create method of resource home object. The create() method creates a new resource instance and returns its unique identifier or the resource key. The properties related to resource key like the type of key, name of the key etc are defined in the JNDI file of the instance as discussed in section 5.3.5.

4 Once the create method returns a key, then the key is used to construct the EPR of the WS-Resource.

EndPointReference=URL of the instance Service + Unique Resource Identifier.

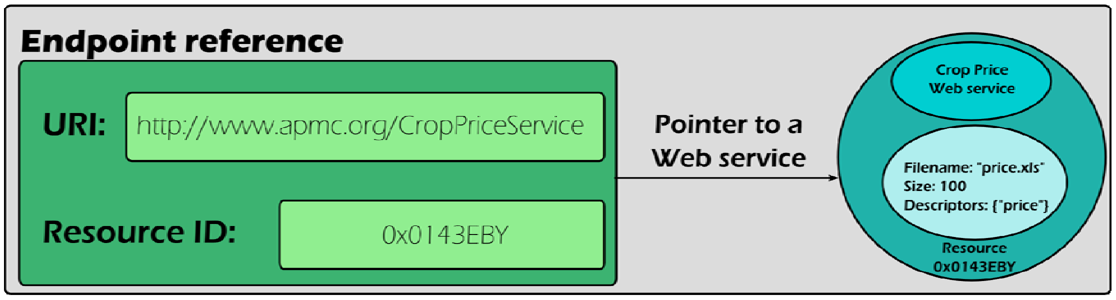


Figure 27: Endpoint reference with WS-Resource

5 The URL of the instance service is created by adding the name of the instance service to the base URL of the container. The factory service obtains the name of the appropriate instance service from the “wsdd” (Web service Deployment Descriptor) file.

```
String instanceURI = baseURL.toString() + instanceService;
```

6 The “wsdd” file of GridManager service has an entry corresponding to the instanceService,

```
<parameter name="buyerInstance"
value="sws/examples/BuyerService"/>
```

7 Once URL and resource key are available, the remaining step is to create the EPR and return it back to the client using the generated stub classes from the WSDL file.

```
epr = AddressingUtils.createEndpointReference(instanceURI, key);
CreateResourceResponse response = new CreateResourceResponse();
response.setEndpointReference(epr);
return response;
```

This utility method getResourceInstanceMethod() retrieves the information from the JNDI file, associated to locate the appropriate resource home.

5.3.2.3 Resource Home and Resource

The GridManager service doesn’t manage any resource. It instantiates different resources managed by other services. Thus, there is no specific resource and resource home attached with the GridManager service.

5.3.3 BuyerService

Once the resource associated with the BuyerService is instantiated and the corresponding EPR is returned to the GridManagerClient. The GridManagerClient invokes different operations of the instance service referenced in the EPR; which may result in modification and update of the resource properties of the resource associated with the instance service.

5.3.3.1 WSDL Document for the BuyerService

Various operations are exposed by the buyer service; which are invoked during the transaction such as setting the crop name, the crop quantity, crop variety, market etc. These entities related to the buyer are exposed as Resource Properties of the Buyer resource and are declared in the Buyer.wsdl file. Buyer.wsdl file is given in appendix.

```
<xsd:element name="BuyerCropName" type="xsd:string"/>
<xsd:element name="BuyerCropQuantity" type="xsd:float"/>
<xsd:element name="BuyerCropVariety" type="xsd:string"/>
<xsd:element name="BuyerMarket" type="xsd:string"/>
<xsd:element name="BuyerOfferedPrice" type="xsd:float"/>
<xsd:element name="BuyerAmountSpent" type="xsd:float"/>

<xsd:element name="BuyerResourceProperties">
<xsd:complexType>
<xsd:sequence>
<xsd:element ref="tns:BuyerCropName" minOccurs="1" maxOccurs="1"/>
```

```
<xsd:element ref="tns:BuyerCropQuantity" minOccurs="1" maxOccurs="1"/>
<xsd:element ref="tns:BuyerCropVariety" minOccurs="1" maxOccurs="1"/>
<xsd:element ref="tns:BuyerMarket" minOccurs="1" maxOccurs="1"/>
<xsd:element ref="tns:BuyerOfferedPrice" minOccurs="1" maxOccurs="1"/>
<xsd:element ref="tns:BuyerAmountSpent" minOccurs="1" maxOccurs="1"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
```

Code Snippet 8: Resource Property declaration in Buyer.wsdl

The above mentioned code snippet shows how different resource properties of a buyer resource are declared in the WSDL document. The individual elements are declared as string, float etc. and then they are referenced in the complex element BuyerResourceProperty. The resource properties are declared according to document/literal style; which is mandatory according to WS-Resource Framework specification version 1.0.

BuyerResourceProperty element is coupled with the Web service through “ResourceProperties” attribute of <wsdl:portType> element. The “ResourceProperties” attribute of the <wsdl:portType> element is the extension of the WSDL 1.1 specification is only meant for the WSRF compliant containers and clients. Following code snippet demonstrates the association of a portType with a ResourceProperty.

```
<portType name="BuyerPortType"
wsrp:ResourceProperties="tns:BuyerResourceProperties">
```

5.3.3.2 Implementation of BuyerResource

The resource declared in the WSDL document is implemented as a separate class. It is important, that the Factory service can instantiate the resource and the Instance service can locate and update the resource properties of the corresponding resource. The container locates any particular resource through its unique ID and its qualified name as declared in the WSDL document. The namespaces in the Web services are always error prone and these are normally declared in a separate interface for reusability and maintenance purposes.

Resource properties and resources are referenced by their fully qualified name. The fully qualified name includes the namespace to which the resource property and resource belongs and its local name.

The namespace and local name of the resource and resource properties in the interface must be in accordance with the fully qualified name of the resource and resource properties in the WSDL document. It is possible to hard code these qualified names in the Web service and resource implementation class which can lead to unnecessary typo errors. However, use of separate interface is an elegant approach since this allows all declaration in a single place. In case of any change, only interface needs update, which considerably minimizes the possibilities of errors.

Each resource is implemented as a separate class and the name of the resource class is declared in the JNDI configuration file. The resource class is very simple class only instantiating different private resource properties and providing corresponding accessor methods.

```
public class BuyerResource implements Resource, ResourceIdentifier,
           ResourceProperties, TopicListAccessor {
private String buyerCropName;
public Object initialize() {
      this.key = uuidGen.nextUUID();
      this.propSet = new SimpleResourcePropertySet(
                 BuyerConstants.BUYERRESOURCE_PROPERTIES);
      try {
      buyerCropNameRP=new
      SimpleResourceProperty(BuyerConstants.RP_BUYERCROPNAME);
      setCropName("NULL");
      buyerCropNameRP.add(buyerCropName);
      this.propSet.add(buyerCropNameRP);
….    } catch (Exception e) { …..}  …..}}
```

Code Snippet 9: BuyerResource class

The above code snippet shows the declaration of a resource property set (i.e. the resource property set wraps all resource properties related to a single resource) and a particular resource property the “buyerCropNameRP”. The resource property set maps the resource declaration in the WSDL document and thus shares the same fully qualified name.

The buyerCropNameRP is declared as an object of a SimpleResourceProperty type. The SimpleResourceProperty eases the efforts to manage, monitor and update individual resource properties. The objects of SimpleResourceProperty can also be advertised as WS-Notification topics. The container monitors the states of topics and generates notification messages on any state change. We will see WS-Notification in a subsequent section.

The SimpleResourceProperty has two constructors; one takes the qualified name of the resource property and other takes the object of ResourcePropertyMetaData. The ResourcePropertyMetaData is an interface and SimpleResourcePropertyMetaData is its concrete implementation. The ResourcePropertyMetaData provides developer tight control on the cardinality of different resource properties within the resource. The utility interface *BuyerConstants* explained earlier is utilized to initialize resource property set and different member resource properties.

As mentioned earlier the resource property set is the wrapper around different resource properties, it easy to transform these objects in the corresponding XML document using API. This XML document is part of SOAP messages exchanged between a client and the service. In this development, different resource properties are initialized without any default values. It is possible to assign the default values directly to the resource properties but the more flexible approach is to declare private variables for each resource property and these variables are used to manage the state of resource properties. Each private variable has corresponding accessor methods which are indirectly used to modify and query values of resource properties at run time. For example, the resource property BuyerCropName declared in the WSDL has corresponding utility variable buyerCropName of type String.

```
private String buyerCropName;
public void setCropName(String cropName) {
this.buyerCropName = cropName; }
```

Code Snippet 10: Accessor method for setting resource property

The buyerCropNameRP; which is the actual resource property is passed the utility variable buyerCropName. Below is the code to assign the value to the resource property buyerCropNameRP through the utility variable buyerCropName.

```
buyerCropNameRP.add(buyerCropName);
```

Once the resource properties are initialized properly they are added to the resource property set. The same process has to be followed for initialization of all other resource properties, which collectively represent the state of the resource.

```
this.propSet.add(buyerCropNameRP);
```

5.3.3.3 Implementation of BuyerResourceHome

The development of BuyerResourceHome is the same as discussed earlier, therefore the same details are not repeated here. The main functionality of the BuyerResourceHome is outlined below.

- It creates a new instance of BuyerResource and passing the EPR to the GridManager
- It provides the querying and searching functionality of the resource based on the resource key
- It registers the newly created resources in the DefaultIndexService of Globus Toolkit 4.

5.3.3.4 Implementation of BuyerService

The BuyerService provides various operations to manage and alter the state of already created buyer resources through the EPR. The BuyerService is the interface between buyer resources and entities interested in its state i.e. client, GridManager Service, Market Service etc. The BuyerService is a stateful Web service exposing various business operations and few operations related to the management of the resources as specified in the WSRF specification. The BuyerService does not implement WSRF specific operations as they are provided by the WSRF container; but it has to provide the signature of WSRF specific methods supported by the service in the WSDL file. Any service deployed in the WSRF container is not stateful service unless it manages any resource and supports few of the WSRF specific operations. The stateful service does not need to support all of the WSRF specific operations and it can only support subset of operations appropriate for its business logic. The development details of the BuyerService are pretty trivial, and most of the logical details are related to the deployment descriptor and the WSDL document. The “wsdd” file which is discussed earlier in the section 8.3 is specific to the Axis framework. The service informs the WSRF container of its dependency on the WSRF specific operations; which are implemented by the container through the parameter “providers”. The parameter “providers” take space separated list of Java classes implementing various WSRF specific operations. These classes are already registered with the WSRF container and the container knows which class implements which operation. Below is the fragment from the “wsdd” highlighting the use of parameter “providers”:

```
<parameter    name="providers"    value="GetRPProvider    GetMRPProvider
DestroyProvider"/>
```

The GT4 provides extension of the <wsdl:portType> in the form of attribute 'wsdlpp'. The 'wsdlpp' stands for 'WSDL pre processor'; which is precise way to add the details of the WSRF specific operations in the WSDL supported by the service. Below is the portion of the WSDL document from the BuyerService showing the use of 'wsdlpp'.

```
<portType name="BuyerPortType"
     wsdlpp:extends="wsrpw:GetResourceProperty
wsrpw:GetMultipleResourceProperties"
       wsrp:ResourceProperties="tns:BuyerResourceProperties">
```

Code Snippet 11: Usage of WSDL pre processor (wsdlpp)

This extension property is a utility feature only provided by GT4, for the convenience. In the final flattened file generated by GT4, the actual signature (i.e. messages and elements) related to wsrpw:GetResourceProperty will be inserted in the WSDL. It can be understood as a copy paste operation of picking elements from one WSDL file and placing them in another WSDL. The use of 'wsdlpp' restricts the reuse of the WSDL document across different implementations of WSRF so it should be used with care. The actual signature of the 'wsrpw:GetResourceProperty' is shown below:

```
<operation name="GetResourceProperty">
     <input name="GetResourcePropertyRequest"
     message="wsrpw:GetResourcePropertyRequest"
     wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
     ResourceProperties/GetResourceProperty"/>
     <output name="GetResourcePropertyResponse"
     message="wsrpw:GetResourcePropertyResponse"
     wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
     ResourceProperties/GetResourcePropertyResponse"/>
     <fault name="InvalidResourcePropertyQNameFault"
     message="wsrpw:InvalidResourcePropertyQNameFault"/>
     <fault name="ResourceUnknownFault"
     message="wsrpw:ResourceUnknownFault"/>
</operation>
```

Code Snippet 12: Signature of the wsrpw:GetResourceProperty

The GetResourceProperty and GetMultipleResourceProperties allow the clients to query and retrieve the current values of the resource properties of the resources, for details read section

5.3.4. These portTypes are also utilized by the Index Service to regularly query the resources for updated information regarding their state.

The business operations implemented by the BuyerService are pretty trivial. Each method looks up the resource referenced in the EPR through the resource home and then after locating the resource it invokes the appropriate action on it.

The utility methods provide more elegant solution to update the resource properties. These utility methods can validate the values before assigning to the resource properties. This validation can be of various natures such as the appropriate range for numerical values, decimal rounding for floating points, and regular expressions for the String values.

BuyerProperties is a complex data type, which is declared in the WSDL document of the BuyerService. A wrapper class is generated for this data type when the tool “wsdl2Java” is used. The object of this class is used within the actual development of the research experiment. It is used by the client application rather than working with raw XML documents through DOM or SAX API.

5.3.4 SellerService

The SellerService manages and monitors the resources related to the seller and its development is very similar to the BuyerService. Hence, the explanation of the SellerService is not repeated here. The complete WSDL document and the SellerService are included in appendix.

5.3.5 MarketService

The MarketService is an intermediate between the BuyerService and the SellerService. The MarketService provides the required platform to carry out any trade. The MarketService orchestrate different services in the pre-set manners for successful transactions, updating the involved resources, notifying the interested parties.

5.3.5.1 WSDL Document for the MarketService

The WSDL document of the MarketService is similar to any other WSDL document, with few business operations, subset of WSRF specific operations and single resource. The WSDL document for the MarketService needs no further explanation and is not repeated here. The resource, resource home and utility interface are developed according to the Implied Resource Pattern.

5.3.5.2 Implementation of MarketResource and MarketResourceHome

The MarketResourceHome has exactly the same business logic and nearly the same operations as discussed in the BuyerService in section 5.4.3. The MarketResource is instantiating the resource property set and different resource property elements on the same line as discussed in the BuyerResource. These classes are provided in appendix.

5.3.5.3 Implementation of MarketService

The MarketService implements few business operations to manage resource but the most important methods in the MarketService are the fireXpathQuery() and storeEntries() method. In this section these two methods are discussed. Familiarity of these methods will help to understand the working of IndexService i.e. Grid Registry provided by the GT4.

5.3.5.3.1 The fireXpathQuery() Method

The fireXpathQuery() method is used to construct and fire appropriate XPath query, to query the IndexService (i.e. Grid Registry). The main purpose of this operation is to match trading preferences of buyers and sellers registered in the same market.

The fireXpathQuery() operation is a complicated method and good understanding is important for the understanding of the role played by the MarketService. Below are the main points related to the fireXpathQuery() operation.

1 Firstly the fireXpathQuery() operation retrieves the URL of the IndexService. This URL is used to create the EPR of the IndexService. This newly created EPR is used to invoke different operations on the IndexService.

2 The fireXpathQuery() invokes different operations on the IndexService through Axis generated utility classes, i.e. WSResourcePropertiesServiceAddressingLocator and QueryResourceProperties_PortType, which serialize and de-serialize SOAP request and response messages before transmitting.

3 The next step is to create an appropriate XPath query. However, the version of XPath query being used is to be specified as there are two different versions of XPath. The XPath version used for the query should match the XPath supported by the WS-ServiceGroup. This purpose is achieved by using the dialect.

4 The XPath query is used to create an appropriate request message. This newly created request message is passed to the queryResourceProperties(..) operation of the QueryResourceProperties_PortType class. The QueryResourceProperties_PortType is the local stub for the remote Web service.

5 The queryResourceProperties(..) returns the result of the query wrapped in the QueryResourcePropertiesResponse object.

5.3.5.3.2 The storeEntries() Method

After receiving the response from the queryResourceProperties(..) in the fireXpathQuery, the next step is to get the entries matching the query criteria. Remember that the DefaultIndexService of GT4, is actually an Aggregator Service i.e. its function is to aggregate several resource properties together according to the WS-ServiceGroup specifications. All entries are retrieved from the response by calling queryResponse.get_any() method; which returns an array of MessageElement type.

```
MessageElement[] buyerEntries = queryResponse.get_any();
```

The MessageElement is the Axis framework specific Java mapping of data type xsd:any. The data type xsd:any is used to pass raw XML between services particularly when the contents of the XML are dynamic and changes at run time. Hence, we need to de-serialize the MessageElement entries to invoke different operations on them and particularly to retrieve the EPR of the resources for further point-to-point communication. The business logic of retrieving the information from individual entries is implemented in the storeEntries(..) operation.

First an array of the MessageElements is de-serialized into an object of EntryType class. From the object of EntryType different operations are invoked to retrieve information related to that particular entry i.e. EPR. AggregatorContent class is used to retrieve the aggregated data within a particular “entry” object of the EntryType class. Since, the actual data i.e. resource properties are required for match making; which is the key to initiate the trade.

```
AggregatorContent content = (AggregatorContent) entry.getContent();
AggregatorData data = content.getAggregatorData();
```

The “data” object obtained, is actually an array of the information advertised by the resource within the Index Service in the form of String. Hence, each entry in the array is reference to the specific resource property. The registration.xml file is used to set the resource properties to be advertised in the IndexService.

String cropName = data.get_any()[0].getValue(); is utilized

The data.get_any()[0].getValue() retrieves the very first entry of the array which has been returned, which is of String type. Similarly, the process has to be repeated to obtain the values for other entries. Since the position of the element can change; which requires the appropriate change in the ‘index’ passed to the get_any() operation. Once all entries are retrieved they are stored in a Vector, whose elements are accessed later to mach if the requirements of any buyer which are satisfied by any seller. Operation performTrade() uses this Vector in which the buyer and seller entries are stored and then performs the trade contingent if the following conditions are being met:

- Crop required by any buyer is being offered by any seller
- Quantity is appropriate
- Crop Variety is matching
- Offered price of the buyer is higher or equal to the expected price of the seller

If these conditions are met, then the performDirectTrade0 function is invoked by passing the seller & buyer EPR, the quantity to be purchased and the amount, which will be offered to seller. Operation performDirectTrade(), is a normal function which utilizes the EPR’s of the buyer and the seller to adjust the quantity and the amount spent by buyer and amount earned by the seller. Thus, by adjusting these resource properties, simulate the successful trade and transaction.

For example if a seller has 60Kg of rice offered at 10Rs/kg and its preference matches that of the buyer who needs this 60Kg at 12Rs/Kg, the SellerCropQuantity ResourceProperty for this instance will be set to 0 and BuyerCropQuantity will be also set to 0, since the seller has sold as much as it wanted to and the buyer has purchased as much as it wanted to. However, the

SellerEarning ResourceProperty will be set to Rs 60 * 12= Rs 720 and the BuyerAmountSpent ResourceProperty will be set to Rs 60 * 12 = Rs 720.

5.4 Services for Indirect Trading

The market service also takes care of performing the trading for sellers interested in indirect trading. This is achieved with the help of simple services by invoking operations on them by passing suitable parameters. These services manipulate a resource properties associated with the EPR of the seller for whom the trade is being performed. Hence, they have functions which are invoked by passing EPR of the seller. Crop quantity, crop name, crop variety etc are also passed depending upon the provided functionality.

5.4.1 CropPriceService

CropPrice Service is used for retrieving the current market price of the crop involved in the trade. The CropPrice Service calculates the price of the crop based on different factors such as the type of the crop, season, its availability in the market and its demand etc. It returns the price to be offered to any farmer, which is interested in indirect mode of trade. For any crop name, which is provided as input, it provides the price per kg, which is offered. It does not have a resource factory with it, as the function is invoked by passing input parameters. It is based on singleton resource instance pattern. This multiplied by quantity returns the total amount for this crop variety. The service further invokes the setEarning() method of SellerService to set the earning for this crop. The code of the service is easy to understand and comprises of just one java file namely CropPriceService.java along with other additional files related to container deployment.

5.4.2 MarketFeeService

The MarketingFee Service is used for deducting any marketing fee incurred during the course of indirect trade. Similar to the transportation charges, marketing fee is deducted from the final selling price and is paid by the seller. MarketFeeService is based on similar pattern as CropPriceService. It deducts the marketing fee, which is imposed on the earning of the seller. As explained above, this also takes seller EPR as input. It utilizes this EPR to obtain the earning of the seller and deducts the market fee as applicable and invokes operation setEarning() of SellerService to set the earning for the seller after deduction of market fee.

5.4.3 VehicleFeeService

VehicleFeeService is responsible for calculating any transportation charges incurred in case of indirect trade. The transportation charges are deducted from the final selling price and are paid by the seller. VehicleFeeService imposes vehicle fee levied on the seller for using vehicle inside market area for transport of the crop. It takes seller EPR and crop quantity transported as input and then invokes setEarning() operation to set the earning for this seller after deducting the vehicle fee.

5.5 Related Research Experiments

Other related research experiments are perfomed to prove effectiveness of our proposed EDSOA. Four more research experiments are perfomed and discussed in the subsequent subsections.

5.5.1 Agro-produce Marketing Automated Negotiation

To demonstrate automated negotiation support we have modified the Agro-produce marketing scenario and developed prototype for the proof of concept [51].

A seller (farmer) or buyer comes to the market place and expresses his/her intention to start a trade of agricultural produce. System will automate the process of finding the farmers (from a buyer’s perspective), negotiating with them and establishing a deal with the farmer by giving the best offer. From a farmer’s perspective, buyers who wish to buy his agro-produce initiate transaction with him, automatically generated offer is sent back, and then negotiation and finalization of deal takes place, without any human intervention.

The major functional requirements identified are:

- Automation: Once a buyer initiates a transaction, it is carried out automatically till a deal is established or till it is terminated.

- Negotiation: In a competitive scenario negotiation plays key role when the buyer demands a price lower than the offer price indicated by a farmer. This brings negotiation in picture, which again is an automated process based on rules specified by each party.
- The best deal for the buyer: The buyer initiates transaction with all the farmers offering the product he requires. As a result, the buyer might end up with a number of favourable offers made to him.
- The transactional requirements: For the computations involved in each phase of a transaction, the transacting partners need information about each other. Partners should be able to communicate and retrieve information about each other.

Following figure represents the workflow of a typical trade. The trade starts with an intention of a trader to sell a particular agro-produce. The price is set by the auction or any other transparent system as defined in the model act. Once the agreed upon price is received, it is published for the traders.

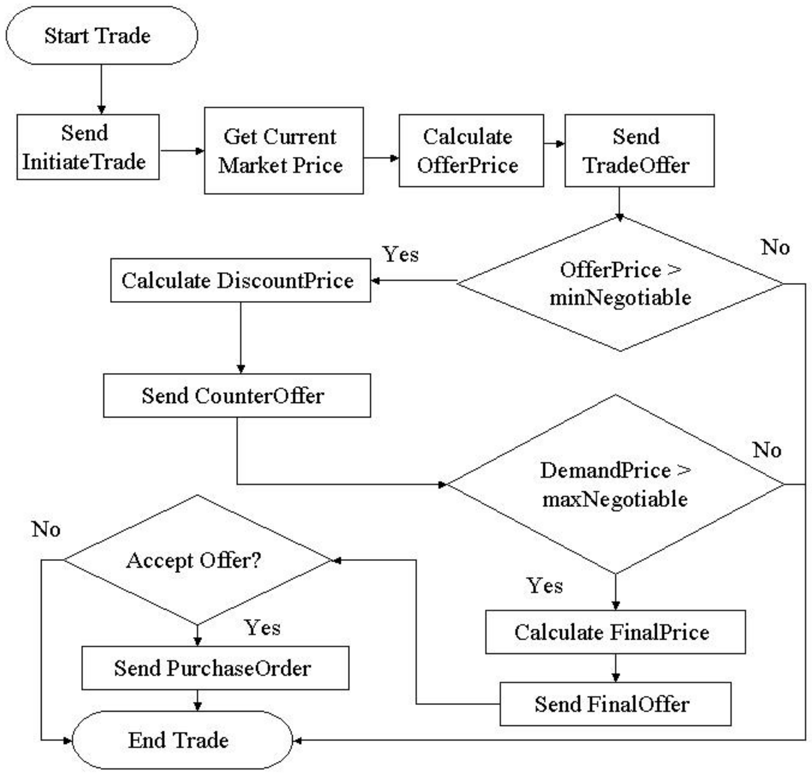


Figure 28: Flowchart explaining the trading process

5.5.1.1 Negotiation Protocol

To automate negotiation of marketing of agro-produce in a process, the following negotiation protocol is followed. The buyer first sends InitiateTrade to the farmer, which is responded by a TradeOffer. If the buyer wishes to continue the transaction, he will send a CounterOffer. The farmer after evaluating the CounterOffer sends a FinalOffer. If the buyer is interested in the FinalOffer, he will respond by sending a PurchaseOrder.

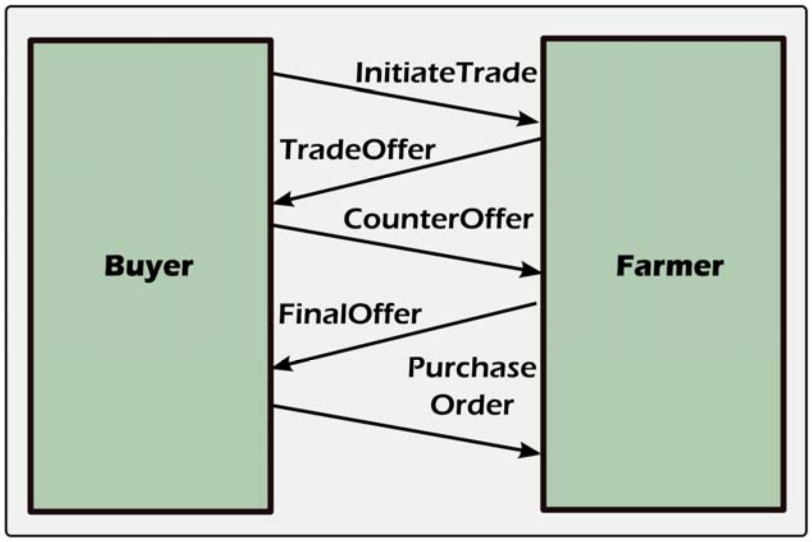


Figure 29: Negotiation Protocol

5.5.1.2 Workflow Implementation

Sequence diagram for the discussed application workflow is shown in the following figure.

When a buyer wants to buy a certain crop, he creates an object InitiateTrade that has the name of the required crop, the quantity required and the buyerID. Then he finds out (from the GridRegistry) which farmers offer the crop and sends InitiateTrade to them. Then nogotiations will take place by following negotiation rules.

Negotiation Rules can be specified in different rule languages like RuleML, Courteous Logic Programs (CLP), SWRL, etc. For the development of rules, we have used SweetDeal [110] and SweetRules [111] tool. These tools provide various facilities like create a new object that integrates the rules of the individual objects; translate objects from one rule language to another etc. It is also used to derive conclusions from rules, which results either in the generation of facts or execution of services. The rules are written in CLP as follows:

```
if
wantsToBuy(?Product) and sells(?Farmer, ?Product)
then
   sendInitiateTrade(?Farmer);
```

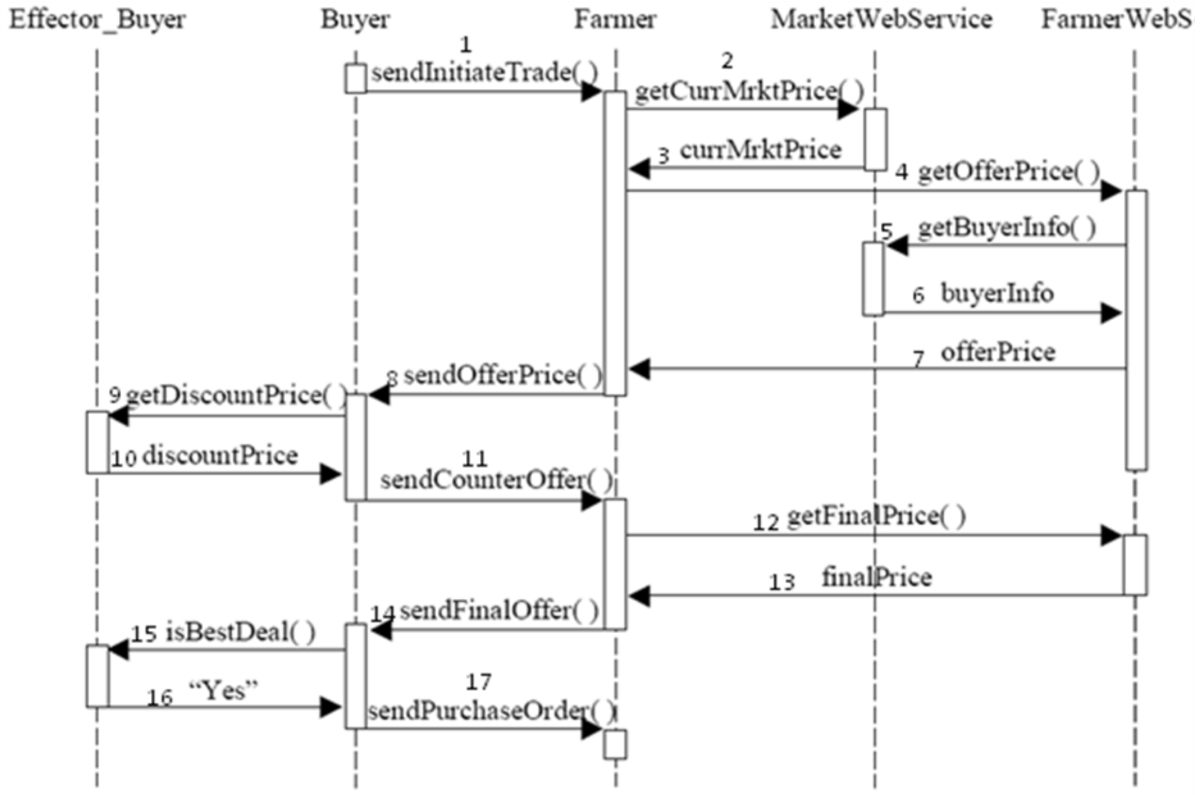


Figure 30: Sequence Diagram of a workflow

When farmers receive InitiateTrade, they calculate the price to be offered to the buyer. This involves following steps:

1. Get the current market price of a crop: by invoking the web service of a market.
2. Calculate the price to be offered: Each farmer in his own way calculates the price he wants to offer to the buyer using the current market price obtained in the previous step. The overheads, if applicable, are also calculated which are based on the location of a buyer, preference for transportation and the quantity required.

```
if
buyerID(?BuyerID)       and       requiredQtyInQuintal(?Qty)       and
currMrktPrice(?Product, ?Price) and
minPrice(?Product, ?MinPrice)
then
getOfferPrice(?BuyerID, ?Qty, ?Price, ?MinPrice);
```

(minPrice is the minimum price that the farmer wants for his crop)

Farmers then send a TradeOffer to the buyer(s) specifying the offerPrice and overheads. On receiving TradeOffer from the farmers, the buyer either sends back a CounterOffer or terminates the transaction. The buyer has a value maxNegotiable stored for the product. If the offered price is more than this value, he terminates the transaction. If not, he calculates a discountPrice (price after discount) that he will demand. The rule for calculating this price varies from buyer to buyer.

```
if
offerPrice(?Product, ?OfferPrice) and
discountPercent(?Product, ?Discount)
then
     getDiscountPrice(?OfferPrice, ?Discount, ?Product);
if
discountPrice(?Product, ?DiscountPrice) and
maxPrice(?Product, ?MaxPrice) and
isGreaterThan(?DiscountPrice, ?MaxPrice)
then
     demandPrice(?Product, ?MaxPrice);
if
discountPrice(?Product, ?DiscountPrice) and
maxPrice(?Product, ?MaxPrice) and
isGreaterThanOrEqual(?MaxPrice, ?DiscountPrice)
then
     demandPrice(?Product, ?DiscountPrice);
```

The buyer then sends a CounterOffer to each farmer who has offered a price less than maxNegotiable. This CounterOffer specifies the price the buyer wants (i.e. demandPrice).

```
if
sellerID(?Farmer) and offerPrice(?Product, ?OfferPrice) and
maxNegotiable(?Product, ?Negotiable) and
isGreaterThanOrEqual(?Negotiable, ?OfferPrice) and
demandPrice(?Product, ?DemandPrice)
then
     sendCounterOffer(?Farmer, ?Product, ?DemandPrice);
```

The processing done at the farmers' end on receiving CounterOffer is similar to the previous step. The farmers either send back a FinalOffer or terminate the transaction. Farmers have a value minNegotiable stored for the crop. If the demanded price is less than this value, the farmer may terminate the transaction or decide the initial offered price as the final price. Otherwise the final price to be offered is calculated. Farmers have a minPrice value stored for the crop, which is the minimum price they want for the crop. The final price offered is the maximum of minPrice and the price demanded by the buyer. Farmers then send a FinalOffer to the buyer (or terminate transaction) specifying the final offered price.

```
if
buyerID(?BuyerID) and demandPrice(?Product, ?DemandPrice) and
minNegotiable(?Product, ?Negotiable) and
isGreaterThanOrEqual(?DemandPrice, ?Negotiable) and minPrice(?Product,
?MinPrice) and isGreaterThanOrEqual(?DemandPrice, ?MinPrice)
then
     sendFinalOffer(?Product, ?DemandPrice, ?BuyerID);
if
buyerID(?BuyerID) and demandPrice(?Product, ?DemandPrice) and
minNegotiable(?Product, ?Negotiable) and
isGreaterThanOrEqual(?DemandPrice, ?Negotiable) and minPrice(?Product,
?MinPrice) and
isGreaterThan(?MinPrice, ?DemandPrice)
then
sendFinalOffer(?Product, ?MinPrice, ?BuyerID);
```

At this point, the buyer receives FinalOffer from a number of farmers and has to decide the best deal available. A comparison of various factors like the price offered, overheads, location of a farmer etc. is done to choose the best deal. The one chosen is responded with a PurchaseOrder, while transactions with others are terminated.

5.5.1.3 Automated Negotiation Implementation

Each partner (buyer and seller) carries the rules for negotiations for his/her own use (not to be shared with other parties). The rules contain information as follows:

- farmer-end: minPrice, minNegotiable, preferred buyer list
- buyer-end: maxPrice, maxNegotiable

sendInitiateTrade, sendTradeOffer, sendCounterOffer, sendFinalOffer, sendPurchaseOrder are objects that contain corresponding rules. When an object is to be sent, a Web service is invoked to transfer it to the desired location. Rules are written in RuleML to invoke particular Web services as shown in following code.

Code snippet is from the file getCurrMrktPrice_eff.ruleml.

```
  <_opr> <rel>getCurrMrktPrice</rel> </_opr>
  <_aproc>
     <wsproc>
     <serviceName>marketService</serviceName>
          <portName>marketIF</portName>
          <operation>
               <operationName>getCurrMrktPrice
               </operationName>
               <parameters>
               <parameter parameterName="Crop"/>
               <parameter parameterName="Quantity"/>
               <parameter parameterName="Market"/>
               </parameters>
          </operation>
          <targetNamespace href="urn:markteService"/>
          <endPointAddress
     href="http://dslab.daiict.ac.in/marketService/market"/>
     </wsproc>
           . . .  .
  </rulebase>
```

Step-by-step interaction of Web services and corresponding rules are shown in figure-33. At each step we are merging and inferring the facts with the help of SweetRules tool. After the inference of rules, we are invoking the corresponding services to execute subsequent steps.

In the last step, the buyer evaluates each available deal. getOptions has the rules to put all of them together in finalOptions object from where a final deal can be selected. The Purchase order sent by buyer1 to farmer1 is shown below:

```
  <emptyLabel>clp_Name();
  <emptyLabel>buyerID(buyer1);
  <emptyLabel>requiredProduct(rice);
  <emptyLabel>requiredQtyInQuintal(50);
  <emptyLabel>sellerID(farmer1);
  <emptyLabel>offerPrice(rice, 650);
  <emptyLabel>overheads(2700);
  demandPrice(rice, 585);
  finalPrice(rice, 585);
```

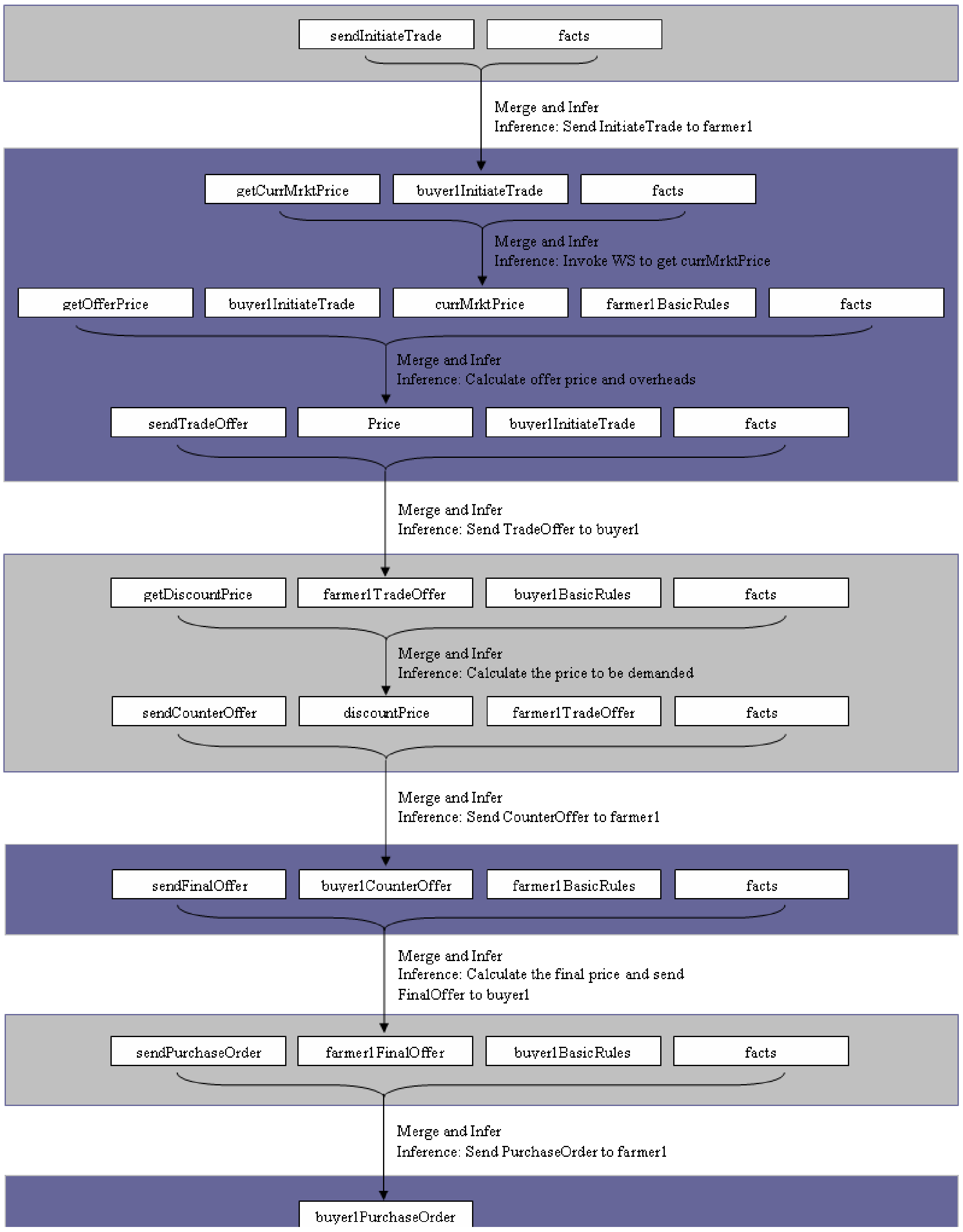


Figure 31: Step-by-Step integration of Web Services and Rules

5.5.2 Agricultural Recommendation System

Agricultural Information System (AIS) is cyclic in nature and affected by physical, biological and climatic conditions. AIS is gradually becoming available to farmers, covering nearly all aspects of the agriculture and farming related activities. It is typically accessed by kiosks, web browsers or any other special interfaces. The AIS knowledge base contains best practices, rules, recommendations and other important information provided by various experts in their respective fields. AIS also hosts vast amount of spatial and non-spatial databases with possible temporal versions. This data includes information like soil type, soil profiles, climatic conditions, crop production, distribution of natural resources, availability of water resources, and similar parameters that are vital for decision making. On the receiver's end, the targeted users have also accepted such systems for their agricultural practices. The large scale adoption of such systems is still challenged by certain unaddressed issues. This challenge is well addressed by the research community by providing means of ICT based systems. Advancement in sensor technology has made it possible to deploy sensors for real-time monitoring of critical parameters and with the advancement of ICT, web has become a huge pool for storing, publishing, searching and accessing various types of information. The main focus is shifting towards the convergence of the sensor technology, communication technology and Information technology to improve the user involvement and ultimately increase the usage of the AIS. Along with the countless benefits of these advancements there are also certain challenging problems associated with it. Finding, sharing and integration of information have raised many issues. Information has become heterogeneous and distributed across the networks. Single point access of such information has created problems related to interoperability and integration of information. Hence, a need to find useful information in an easy manner has become increasingly important. On the other hand, user requirements have increased and each user demands personalized, relevant information based on preferences, context, location as well as changes occurring among various events.

As an extension of our work, we utilize Event-driven Service-Oriented Architecture for agricultural recommendation system [47, 52]. An agricultural system typically comprises of two major categories of actors namely experts and users. Our approach considers experts and information sources as service providers, and agro-professionals as service consumers. From the perspective of the users, instead of considering users referring to AIS casually, we allow them to subscribe to ‘events’ that really meet their decision-making requirements. Thus, we tried to increase the involvement of AIS for every possible decision likely to be taken by the user. We have demonstrated the use of semantic to deliver common vocabulary and knowledge of agricultural domain, automated discovery of event sources for subscription and event correlation. The proposed architecture facilitates the seamless and meaningful information integration and delivery of personalized context and location based information.

5.5.3 Service Grouping and Group Notification in Grid Business Process

Today enterprise application performs its business operations as a set of business processes. Even in single enterprise multiple processes are running at the same time and those processes are interacting with multiple services, partners and customers as per the requirement and policy. There is a need to get aggregated information from the distributed services. We have to group the services to achieve better search, query and group notification as per the event. With our architecture we are expected to achieve following objectives:

- Aggregation of resources and services to achieve domain specific search and query.
- Delivery of Notification on selected group of services as per the event.

5.5.3.1 Service Grouping

The WS-ServiceGroup provides a description of a general-purpose WS-Resource which aggregates information from multiple WS-Resources or Web Services for domain specific purposes. The aggregated information can be used as a directory in which the descriptive abstracts of the individual WS-Resources and Web Services can be queried to identify useful entries. The membership in the group is in a constrained way, which can be controlled through policies.

Using WS-ServiceGroup Specification we create a group of Sellers. All buyers in the market are given notifications (WS-Notification) about any change in the status of a Seller, e.g. a new Seller is joining the Market or there is a change in resource property of seller (e.g. out of 100 kg. of wheat to be sold, 50 kg. is sold).

The resources have tight control on the amount of information they want to advertise in the Index Service. The Index Service is distributed registry, which keeps on updating itself regularly after specific intervals. Any change in the state of the resource or its deletion results in the notification of Index Service, which adjusts itself accordingly. This information is provided in the form configuration file ‘registration.xml’ for each resource.

We can use WS-Policy Specification for creating a rule such as only license holders are allowed to carry out any trade in the market area. If they fail to pay any fees to the APMC or fail to pay the agreed-upon price of the purchased good, the APMC may cancel the license of the trader. Thus our Market Service follows the concept of virtual organization (VO) in which some access control policy is set up that encloses some boundaries and specifies the different rights of the users within the VO.

5.5.3.2 Notification delivery to Services Group

WS-ServiceGroup itself is a stateful Web Service that is a collection of other Web Services or WS-Resources and the information that pertains to them. The model for membership of a ServiceGroup is an entry resource property of the Service Group. Details of each member in the ServiceGroup are in the form of WS-ResourceProperties, which wraps the EndpointReference, and the contents of the member. WS-ServiceGroup specification describes different components and the message exchange patterns for its smooth functioning; which are described below:

Notification Message Exchange: WS-ServiceGroup specifies the notification messages if the WS-ServiceGroup also implements the NotificationProducer interface defined in the WS-BaseNotification specification. These notification messages are generated either when new member is added in the service group or when details of the existing member are modified.

MemberShipContentRule helps in setting a member interface, whom we would like to allow as a part of our service group.

WSN based Notifications are delivered to the buyer when a new seller is added in the seller’s ServiceGroup. This requires registering the ResourceProperty whose value change is to be added to the “TopicExpressionType”. Following code snippet shows the delivery of notification.

```
public void deliver(List topicPath, EndpointReferenceType producer,
Object message) {
ResourcePropertyValueChangeNotificationType changeMessage =
((ResourcePropertyValueChangeNotificationElementType) message).
getResourcePropertyValueChangeNotification();
if (changeMessage != null)
{ ………}
```

Code Snippet 13: Notification Delivery

5.5.4 Policy-driven Grid Business Process

Composition of Business process requires dynamic discovery, selection and composition of services. Many service providers may provide identical or similar functionality. A huge set of services providing identical functionality may exist. The best available services should be discovered and selected to generate composite service. Selection is influenced by quality, non-functional requirements and evaluation of features. Every services taking part in composition have certain characteristics, properties, limitations and non-functional requirements. For discovery of services we have to consider functional and non-functional requirements of services. We also have to consider the policy associated with services while discovery and composition of services. Policy defines set of rules and condition to access the services. Business process is context-aware; it should detect event and respond to changes in the state during the execution of a process. Event can provide the context of environmental changes that occur during execution. Interoperability among services needs to be achieved when a process spans across the boundaries of multiple business organizations and domains, where vocabulary is different. Because of heterogeneity, dynamic nature and lack of knowledge among business partners, we need a mechanism to select the services based on context, content and contract.

WSDL represents functional aspects of a Web service. Service consumer selects the Web service, which matches the functional requirements. Client and service provider must agree on these functional requirements for interactions among them. A service consumer may require invocation of Web services with security, reliability information etc. These non-functional requirements are important to describe the properties of Web services. There is no way of expressing such information except via documentation or through direct inquiries etc. For satisfying these types of requirements of a service consumer, policy is used in Web service selection phase. Once the Web services which satisfies the functional requirements of a consumer have been found or identified, they will be compared with information extracted from the policy to find out whether they satisfy the non-functional requirements of a consumer or not. Only those services that satisfy functional and non-functional requirements will be selected. Thus, the use of policy during Web service selection phase helps service consumers to select services based on their requirements. One can describe these requirements using WS-Policy specification.

5.5.4.1 Architecture for Policy-driven Grid Business Process

Policy-driven grid business process is proposed to achieve dynamic services selection with context, content and contract based information. Dynamic service selection is achieved with the use of event, policy and semantic. Event is used to get the context of the system, semantic to provide vocabulary and more contextual information and policy to define rules and contract for the selection of services. Meaningful services are selected for generation of the composition schema dynamically as per the events. Components of the proposed architecture are described below:

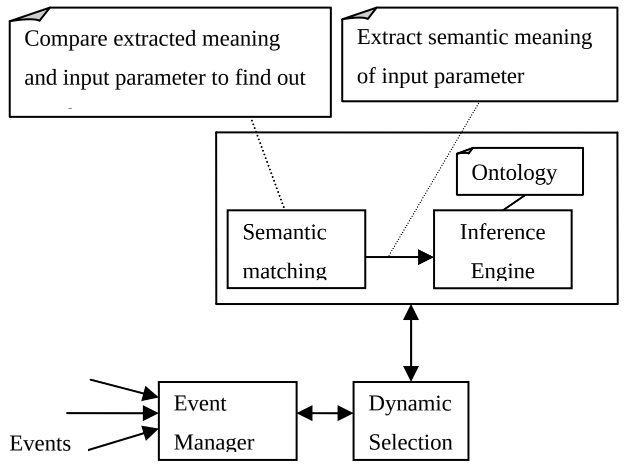

Compare extracted meaning and input parameter to find out
Extract semantic meaning of input parameter
Ontology
Semantic matching
Inference Engine
Events
Event Manager
Dynamic Selection

Figure 32: Proposed Architecture for Policy-driven Grid Business Process

Event Manager: Event Manager is designed to communicate dynamically with the distributed heterogeneous services and other components such as ontology server, rules engine and composition engine, etc. as indicated in the figure-15. It is configured to furnish precise and event specific composition. It helps in selection of services for composition. It is responsible for receiving events from services and requesting dynamic selection to select proper web service according to functional and non-functional requirements.

Dynamic Selection: It is responsible for selecting web service according to requirements specified in WSDL and WS-Policy. It contains two sub components

1. Semantic matching: It extracts data from the policy, performs matching and returns matched data to the Dynamic Selection. It exposes operations such as extraction of policy data and matching of input parameters with semantic meaning. It interacts with inference engine and provides parameters to it.

2. Inference Engine: It is responsible for extracting meaning of input parameters and uses ontology. For extracting the meaning of input parameters, it executes and resolves semantic query with the help of ontology.

5.5.4.2 Research Experiment

In this experiment, the aim was to apply semantic and policy in grid business process. The GridManagerService contains the policy and the ontology is deployed in the ontology server. Based on semantic and policy, the MarketService executes the appropriate operations on the resources of sellers and buyers registered with the Grid Registry, i.e. DefaultIndexService. The MarketService gets policy from the GridManagerService. The sequence of events is as follows:

1. The MarketService queries the DefaultIndexService to find out sellers and buyers registered with the preferences to trade at a particular location.

2. The DefaultIndexService returns EPR of Sellers and Buyers resources interested in trading at a given location.

3. The MarketService requests the GridManagerService for getting a policy file. The MarketService sends a request of extracting policy data to the semanticMatching module. The SemanticMatching returns data to the MarketService.

4. The MarketService again invokes the SemanticMatching to find out matched results. The SemanticMatching invokes the InferenceEngine to extract the meaning of data by executing semantic query and the InferenceEngine returns results of a query in response. The

SemanticMatching then compares query results with seller and buyer properties and returns results to the MarketService.

5. The MarketService invokes an operation according to the result obtained. If the trading mode of the market is direct and if the result contains matches then the trading is performed by invoking operations on seller and buyer resources to adjust the crop quantity and earning for a seller and; the crop quantity, and amount spent for a buyer. Appropriate messages will be sent to the corresponding sellers and buyers in the form of WS-Notification based notifications message.

For example, a buyer can express intention to purchase by executing corresponding buyer service. Here one buyer is ready to purchase 10 kg. of kamla (a type of wheat) at Bhopal location by offering price of Rs. 70 per kg. Execution of a buyer service will start and will wait for appropriate match identification in a market located at Bhopal. When buyers and sellers express their intentions to trade, their intensions will have been registered with the DefaultIndexService. The MarketService queries the DefaultIndexService to identify the buyers and sellers interested in the trading. It reads the policy and matches the policy data with the stored information or the constraints that must be satisfied by instances of seller and buyer services. If the constraints are satisfied then the MarketService will start a process of matching. The market service starts execution by comparing values supplied by service instances of buyers and sellers. Seller supplies wheat as crop and Buyer supplies kamla as crop. Both crop names are syntactically different but semantically equal. For semantic matching ontology is developed, which contains semantic information about crop, e.g. wheat. Based on the domain knowledge in ontology, it is resolved by the following facts.

```
Kamla is the type of wheat.
```

Hence “wheat=kamla” fact will be generated using the ontology. By using this fact, the MarketService matches appropriate sellers and buyers. If the appropriate match has been found, then the notification will be sent to both of them to inform about entering into a trade deal. Otherwise they will wait for appropriate offer to come.

In this chapter, the development of grid business process with direct and indirect agro-produce marketing scenario is discussed. Execution of grid business process is discussed in the next chapter.

Chapter 6
Results and Discussion

The final phase of the grid business process is deployment and execution of a process. Four phases of a business process (Model, Compose, Deploy and Execution) are discussed in chapter 1. In section 4.1 modeling is discussed and in section 4.4 composition of business process is given. Chapter 5 presents development and experiment details of a grid business process. Now for deployment and execution of direct and indirect agro-produce marketing process, we have to orchestrate various services as per composition schema.

This chapter is organized as follows. Section 6.1 discusses orchestration and deployment of the grid business services. Section 6.2 presents the implementation of the client to execute grid business process. Section 6.3 provides grid business process execution steps along with interactions among different components. Sections 6.4 and 6.5 discuss execution of direct and indirect trading process.

6.1 Deployment and Orchestration of Grid Business Process

For the purpose of the experiment and to achieve statefulness, notification, service grouping and scalability, we have followed factory pattern and a number of stateful Web services were created using WSRF and WSN standards. The stateful Web services provides required functionality, but still it suffers from a few drawbacks. A stateful Web services is unable to handle change in partner, change in business requirements and change in business flow. These are the major drawbacks from the point of view of enterprise applications. In order to remove these drawbacks, BPEL is used as a composition language.

The biggest hurdle with BPEL for resolving these issues is the lack of support for stateful WSRF/OGSI based Web services. For invocation of stateful Web services from BPEL process, extra information related to WS-Addressing [60] is required. We have to provide this extra information at the time invocation using header variables [112]. The header variables are present in the header of the invocation message. It helps in unique identification of the resource using the WS-Addressing attributes.

By default, header message is not present in the WSDL file of WSRF based Web services. In a grid environment, WSRF based service generates four WSDL files at the time of deployment. Each WSDL file is having a different functionality and each requires some modifications as follows:

1. ServiceInstance.wsdl: In this file, following code snippet should be inserted. It will be processed by ant for generation of grid archive (gar) file and it will be inserted in the ServiceInstance_flattened.wsdl file.

```
<!-- REQUESTS AND RESPONSES -->
     <xsd:element name="CropPriceResourceKey" type="xsd:string"/>

<message name="Header">
     <part name="MessageID" element="wsa:MessageID"/>
     <part name="To" element="wsa:To"/>
     <part name="Action" element="wsa:Action"/>
     <part name="From" element="wsa:From"/>
     <part name="CropPriceResourceKey"
     element="tns:CropPriceResourceKey"/>
</message>
```

Code Snippet 14: NetCostComputationService

2. ServiceInstance_flattened.wsdl: As mentioned above, in this file changes were done at the time of gar file generation.

3. ServiceInstance_service.wsdl: In this file address location should be modified to reflect the actual URL where the service is deployed.

```
<soap:address location=http://10.100.64.64:8080/WS-RF/services/>
```

should become

```
<soap:address location="http://10.100.64.64:8080/WS-
RF/services/croppriceservice"/>
```

4. ServiceInstance_bindings.wsdl: This file contains the header, which will be passed to any operation before invocation. Following code snippet must be inserted in every operation name found inside the WSDL file, which contains binding related information.

```
<wsdl:operation name="setCropPrice">
     <soap:operation
      soapAction="http://dcomp.daiict.org/CropPrice/CropPricePortTyp
      e/setCropPriceRequest"/>
     <wsdl:input>
           <soap:header use="literal" message="tns:Header"
           part="MessageID"/>
          <soap:header use="literal" message="tns:Header"
           part="To"/>
          <soap:header use="literal" message="tns:Header"
           part="Action"/>
           <soap:header use="literal" message="tns:Header"
           part="From"/>
          <soap:header use="literal" message="tns:Header"
           part="CropPriceResourceKey"/>
           <soap:body use="literal"/>
     </wsdl:input>
       <wsdl:output>
         <soap:body use="literal"/>
      </wsdl:output>
</wsdl:operation>
```

Code Snippet 15: NetCostComputationService

6.1.1 Orchestration of Grid Business Process

The Agro-produce marketing business process was realized using BPEL. Before execution of grid business process, various services like Buyer service, Seller service, Market service, Vehicles fees service, Crop price service etc are orchestrated using partner links. Business process is orchestrated using OraleBPEL editor. Agro-produce marketing grid business services act as partner links of the BPEL process. They provide their functionality to the process by allowing their port types to be used. Using these portTypes various operations are invoked on these Web services. Client can invoke these trading processes directly or indirectly. Interactions among various components of the architecture are shown in figure-35.

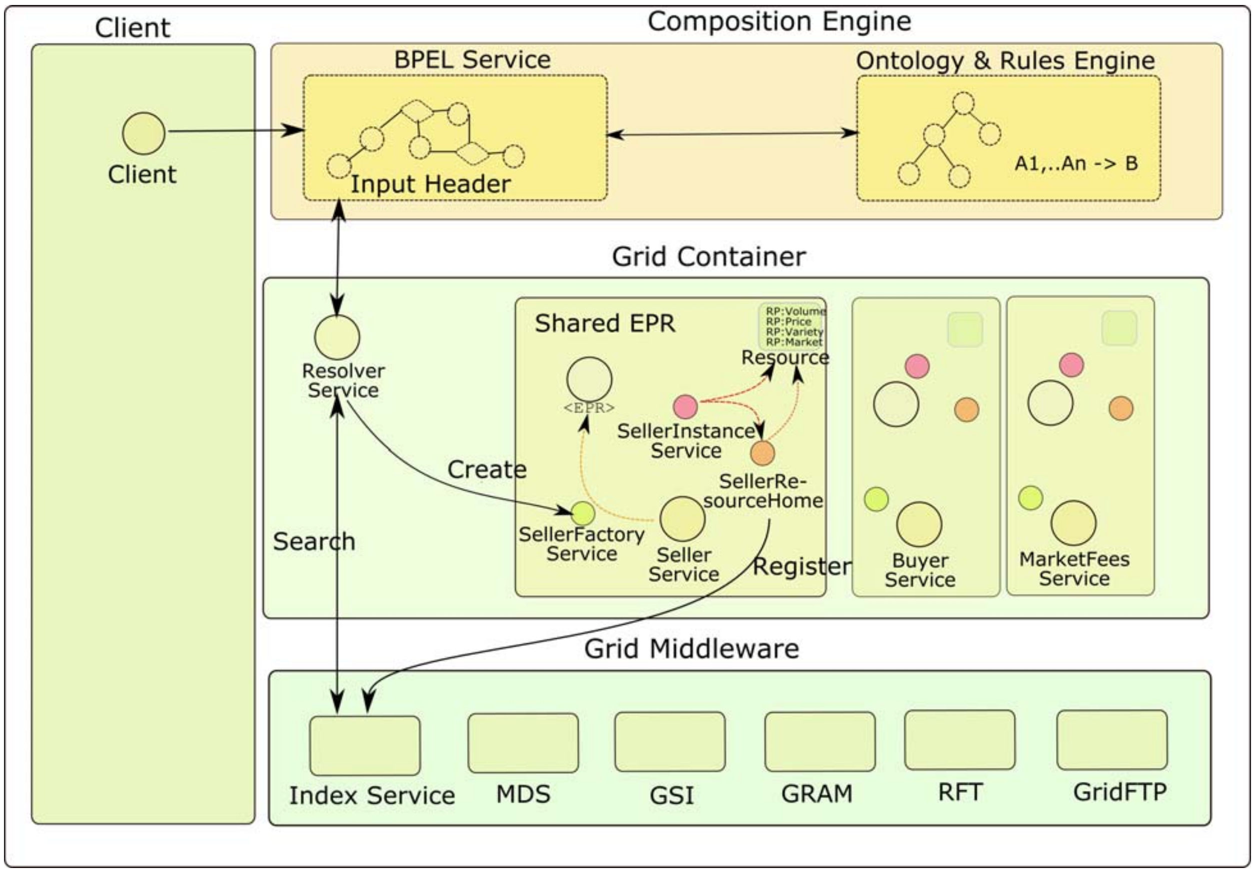


Figure 33: Component Interaction Diagram

Interactions among business services and grid services during the execution of a business process are shown in the form of sequence diagram in figure-36.

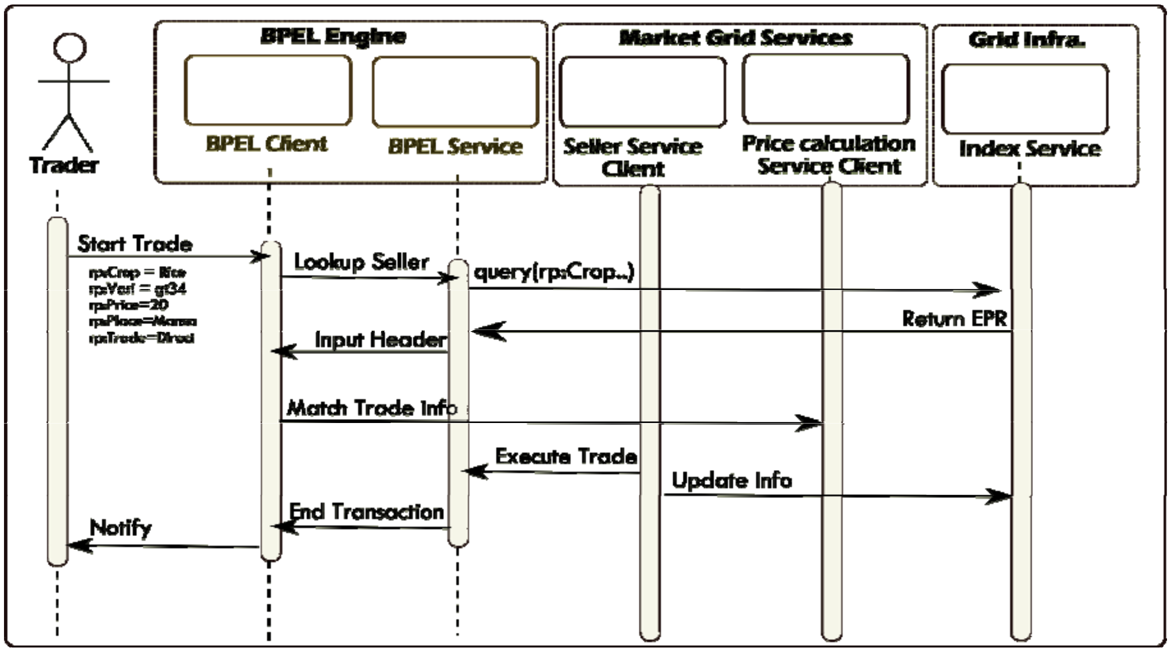


Figure 34: Interactions among services

6.2 Execution of deployed Grid Business Process

The client will start the execution of a process. The flow of the complete process is controlled by a Grid Manager. Grid Manager is responsible for invoking the services, changing the flow as and when required and passing the information from one service to another.

Execution of a business process involves following steps:

- The client starts the trade process. The receive activity receives the request of the client, and a new instance of the process is created as soon as the request is received. This instance which, has been created is specific to the particular client invocation and is not visible to other clients.
- The process invokes the factory service which is responsible for creating an instance of a resource. Invocation of factory service doesn't require WS-Addressing related header information to be passed in order to be used. The only change involved is to

modify *FactoryService_service.wsdl* file, where the full URL of the deployed factory service should be provided.

- The factory service creates an instance of the resource, which is to be instantiated. The EPR of the instantiated resource is returned back to the process. The resource key is contained as a part of the EPR, as a child element.
- Now various stateful Web services are to be invoked. Modified WSDL files are invoked one by one according to the desired functionality. The order in which, they are invoked will be dependent as to how it is described in the sequence activity. For invocation of any service, which has to act on a specific instance of the resource, the WS-Addressing related header will have to be passed to identify the particular resource instance. They will use the resource key contained as a part of the EPR to alter the state of the resource.
- Once all the services, which are to be consumed as a part of the process, are executed, then the final value of the resource property is displayed to the end user and process instance is destroyed.

Interactions among various components during the execution of grid business process are shown in figure-35. Interactions among business services and grid services during the execution of a business process are shown in the form of sequence diagram in figure-36. These interactions among business services, grid services and different components of a grid business process are discussed in the following sections with the example of direct and indirect agro-produce marketing process.

6.3 Execution of Direct Trading Process

The end user interacts with the developed grid business process and services with the help of a client program. The end user executes this client program by providing certain parameters. These parameters are different for a seller, buyer and market user. We are passing these parameters as command line arguments.

In this section the trading process is displayed by execution of various services depending on the preferences indicated by a client. Data shown in various tables is taken as input to execute various services part of a grid business process. The following table indicates the preferences of the buyers:

Buyer	Crop Name	Crop Variety	Quantity	Price Offered Rs/kg	Trading Location
1	Rice	Basmati	50 Kg	100	Mumbai
2	Wheat	Sindhi	20 Kg	80	Delhi
3	Rice	Carolina	100 Kg	90	Mumbai
4	Wheat	Kamla	10 Kg	70	Bhopal
5	Potato	Katahdin	60 kg	60	Kolkata

Table 2: Data about buyers

The data corresponding to various sellers is shown in the table as under:

Seller	Crop Name	Crop Variety	Quantity	Price Expected Rs/Kg	Trading Location	Trade Mode
1	Rice	Arboria	100 Kg	65	Mumbai	Indirect
2	Wheat	Kamla	50 Kg	60	Bhopal	Direct
3	Wheat	Dandi	70 Kg	50	Mumbai	Direct
4	Soyabean	Monetta	40 Kg	70	Delhi	Indirect
5	Corn	Indian	110 Kg	55	Amritsar	Direct
6	Potato	Katahdin	60 Kg	10	Kolkata	Indirect
7	Rice	Basmati	120 Kg	120	Mumbai	Direct
8	Onion	Nasikred	50 Kg	8	Mumbai	Direct

Table 3: Data about sellers

In the following table, locations of various markets are shown:

Location
Mumbai

Delhi
Bhopal
Calcutta
Amritsar

Table 4: Data about locations of various markets

6.3.1 Execution of Seller Grid Service

The execution of a grid business process from the Seller's perspective involves the following steps:

(a) A seller expresses his intention to offer his crop for sale by running the client program and by providing required parameter using command line arguments. These arguments are in the following order:

1. URL of the GridManager service
2. seller: This word is provided to indicate client wants to have an instance of a seller service
3. Crop Name: The crop seller wishes to sell
4. Crop Quantity: The quantity of a crop, which seller is interested in selling
5. Crop Variety: The variety of a crop
6. Trade Location: The location where the seller wishes to trade
7. Expected Price/Kg: The amount which a seller wishes to obtain per kg of a crop
8. Trade Mode: If the seller directly wants to sell to a buyer then direct mode is adopted, else indirect mode is adopted.

In the following command, an intention of a seller is shown, who is ready to sell 50 kg. of wheat at Bhopal location, and expecting minimum of Rs. 60 per kg.

```
$ java -DGLOBUS_LOCATION=$GLOBUS_LOCATION
org.sws.examples.clients.GridManager.Client
```

```
http://10.100.64.65:8080/wsrf/services/sws/examples/GridManager
seller wheat 50 kamla Bhopal 60 direct
```

(b) Using values passed by seller, an object of the SellerProperties is created.

```
sp=new
SellerProperties(cropName,cq,cropVariety,op,market,tradeMode);
```

(c) Next a reference to the GridMangerPortType is obtained to create a new instance of the Seller Resource. This is achieved by using the EPR of the GridManager service, which can be created by using its URI.

```
GridManagerPortType gridManager;
SellerPortType seller;
gridManagerEPR = new EndpointReferenceType();
gridManagerEPR.setAddress(new Address(gridManagerURI));
gridManager                                                   =
gridManagerLocator.getGridManagerPortTypePort(gridManagerEPR);
```

Code Snippet 16: EPR retrieval of GridManagerService

(d) Security related parameters are to be initialized in order to encrypt the SOAP message exchange between the Client and the GridManagerService. Here, Message Level Security is being utilized. Hence, only the body of the message is encrypted leaving the header untouched.

```
((Stub)gridManager)._setProperty(Constants.GSI_SEC_CONV,Constants.EN
CRYPTION);
((Stub) gridManager)._setProperty(Constants.AUTHORIZATION,
NoAuthorization.getInstance());
```

Code Snippet 17: Initialization of security related parameters for SOAP message encryption

(e) The GridManagerService acts as a common factory for the seller, buyer and market service. Its create method will be invoked by passing an object of the stub class CreateResource, which was discussed in the section related to GridManagerService. The constructor of CreateResource class expects a string argument namely serviceType, to identify which stake holders Resource instance has to be created and initialized. In this case the serviceType argument will have value as “seller” to indicate that a new Seller Resource has to be created.

The EPR of a Resource instance is to be returned after creating it. The EPR returned will be stored in an object of createResponse. The client needs to inform the GridManagerService about the stakeholder of a newly created resource instance namely the seller, buyer or the market.

(f) Once the EPR is returned and stored in the object createResponse, operations can be invoked on Seller Resource to modify the default values and set the trading parameters as indicated by the end user. This is done, by first obtaining a reference to the SellerService port type, setting of security related parameters to secure the conversation, and then invoking operations exposed by the SellerService namely setSellerProperties() by passing the object of SellerProperties.

```
if(serviceType.equals("seller"))
{
  sellerInstanceEPR= createResponse.getEndpointReference();
  notificationEPR=sellerInstanceEPR;
  seller = sellerInstanceLocator.getSellerPortTypePort
      (sellerInstanceEPR);
  ((Stub)seller)._setProperty
      (Constants.GSI_SEC_CONV,Constants.ENCRYPTION);
  ((Stub) seller)._setProperty
      (Constants.AUTHORIZATION, NoAuthorization.getInstance());
  . . .
}
```

Code Snippet 18: Invocation of operation setSellerProperties of SellerService

(g) The EPR of the Seller Resource is created and written in a file. EPR will be utilized by a client and it is responsible for providing the notification related information when changes take place in the value of the SellerResourceProperties; i.e the sellerStatus. After writing EPR in a file, the NotificationListenerClient is responsible for listening to notifications. It is invoked by invoking startSubscription operation by passing serviceType and filename, which contains the EPR of newly created Resource.

```
if(!serviceType.equals("market")){
  Random rand=new Random();
  int i=rand.nextInt();
  Integer number=new Integer(i);
  String fileName=number.toString()+".epr";
  Client cl=new Client();
  cl.writeEPR(notificationEPR,fileName);
  NotificationListenerClient     notificationListener     =     new
NotificationListenerClient();
  notificationListener.startSubscription(serviceType,fileName); }
} catch (Exception e){     e.printStackTrace(); } }
void  writeEPR(EndpointReferenceType  epr,String  fileName)  throws
IOException,SerializationException {
    FileWriter fw = null;
    try    {
      fw = new FileWriter(fileName);
       // write the EPR into the file
      ObjectSerializer.serialize(fw, epr, RESOURCE_ENDPOINT);
                 fw.write('\n'); }
    Finally { if(fw != null){
        try { fw.close();}catch (Exception e) {}}}}
```

Code Snippet 19: Code snippet for writing EPR

(h) The execution of seller grid service will start and will wait for the appropriate offer to come from a potential buyer. The seller grid service will receive notification, once an appropriate offer is floated by a buyer. Initially the following output will be generated:

```
END POINT STRING:<ns1:SellerServiceEndpoint
xsi:type="ns2:EndpointReferenceType"
xmlns:ns1="http://www.apmc.org/namespaces/examples/SellerService"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns2="http://schemas.xmlsoap.org/ws/2004/03/addressing">
 <ns2:Address
xsi:type="ns2:AttributedURI">http://10.100.64.65:8080/wsrf/services/
sws/examples/SellerService</ns2:Address>
 <ns2:ReferenceProperties xsi:type="ns2:ReferencePropertiesType">
  <ns1:SellerResourceKey>a1662350-1628-11db-9878-
ce21816644d9</ns1:SellerResourceKey>
 </ns2:ReferenceProperties>
 <ns2:ReferenceParameters xsi:type="ns2:ReferenceParametersType"/>
```

```
</ns1:SellerServiceEndpoint>
Waiting for notification. Ctrl-C to end.
Waiting for notification. Ctrl-C to end.
```

6.3.2 Execution of Buyer Grid Service

Similarly a buyer can express his intention to purchase crop by running client program but with a different set of parameters: Execution of a grid business process from the Buyer perspective involves following steps.

(a) Buyer has to provide following parameters for the execution of a process:

- URL of the GridManager service
- buyer: This word is provided to indicate that a client would like to have an instance of a buyer service
- Crop Name: The crop buyer wishes to buy
- Crop Quantity: The crop quantity buyer is interested in purchasing
- Crop Variety: The crop variety of the crop
- Trade Location: The location where the buyer wishes to trade
- Offered Price/Kg: The amount which a buyer would like to obtain per kg of a crop.

As shown in the following command, one buyer is ready to purchase 10 kg of wheat at Rs. 70 per kg in Bhopal market. Thus, buyer grid service will also wait for appropriate match to happen in the Bhopal market.

```
$java -DGLOBUS_LOCATION=$GLOBUS_LOCATION
org.sws.examples.clients.GridManager.Client
http://10.100.64.65:8080/wsrf/services/sws/examples/GridManager
buyer wheat 10 kamla Bhopal 70
```

(b) Using values passed by user, an object of the BuyerProperties is created.

```
buyerProperties=new
BuyerProperties(cropName,cq,cropVariety,market,op);
```

(c) Next a reference to the GridMangerPortType is obtained to create a new instance of the Buyer Resource. The explanation and the steps are as similar as discussed for Seller Perspective till step (e), where the serviceType will be ‘buyer” instead of “seller”.

(d) The EPR, which will be returned, will be of Buyer Resource, hence operations exposed by BuyerService, namely setBuyerProperties() will be invoked to replace the default initialized value of the Buyer Resource with the parameter passed by the end user. After this the steps related to writing EPR in a file and notification is same as that of SellerResource.

try{ buyer.setBuyerProperties(buyerProperties);}

Here, execution of buyer grid service will start and it will wait for appropriate match to be identified in a market located at Bhopal.

```
END POINT STRING:<ns1:BuyerServiceEndpoint
xsi:type="ns2:EndpointReferenceType"
xmlns:ns1="http://www.apmc.org/namespaces/examples/BuyerService"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns2="http://schemas.xmlsoap.org/ws/2004/03/addressing">
 <ns2:Address
xsi:type="ns2:AttributedURI">http://10.100.64.65:8080/wsrf/services/
sws/examples/BuyerService</ns2:Address>
 <ns2:ReferenceProperties xsi:type="ns2:ReferencePropertiesType">
  <ns1:BuyerResourceKey>3136b260-1629-11db-9878-
ce21816644d9</ns1:BuyerResourceKey>
 </ns2:ReferenceProperties>
 <ns2:ReferenceParameters xsi:type="ns2:ReferenceParametersType"/>
</ns1:BuyerServiceEndpoint>
Waiting for notification. Ctrl-C to end.
Waiting for notification. Ctrl-C to end.
```

6.3.3 Execution of Market Grid Service

Whenever the buyers and sellers express their intentions to trade, their resource instances are also registered with the DefaultIndexService. When MarketService starts its operations, it will query DefaultIndexService to obtain the buyers and sellers interest in trading. The client program is executed with a different set of parameters. Execution of a grid business process from the Market perspective involves the following steps.

(a) If an end user is interested in executing the deal then the market has to provide the following parameters:

- URL of the GridManager service.
- market- This word is provided to indicate that the client would like to have an instance of a market service.
- Trade location- The location of the market where the actual trade will occur.

```
$ java -DGLOBUS_LOCATION=$GLOBUS_LOCATION
org.sws.examples.clients.GridManager.Client
http://10.100.64.65:8080/wsrf/services/sws/examples/GridManager
market Bhopal
```

(b) Most of the explanations related to the execution of the deal of a client from Market's perspective remains same as Seller's or Buyer's Perspective. The only noticeable difference being that, after obtaining the EPR of the Market Resource, the operation invoked will be setMarketProperties(marketProperties);
marketPortType.setMarketProperties(marketProperties);

Once market service of a particular location starts executing, it tries to match potential buyers and sellers registered with the DefaultIndexService. If an appropriate buyer is found for a seller, approproate notification is sent to both of them to inform them about success of entering into a trade deal. Otherwise both of them will wait for a potential match of a crop.

6.3.4 Notification Listener Client

A notification client to subscribe and receive notification messages is developed.

1. The client is a notification consumer. It will receive notifications, which will be sent by the grid services, BuyerService and SellerService. The services acting as notification producer will invoke Notify operation on this client. The client runs in a daemon mode to receive the notifications at any time. Thus it acts like a server on which operations can be invoked any time. In order to attain this property NotificationConsumerManager class is utilized.

```
NotificationConsumerManager consumer = null;
consumer = NotificationConsumerManager.getInstance();
consumer.startListening();
```

2. The client starts listening to the incoming notifications. In order to identify the end point at which the notification has to be delivered by the Seller or Buyer service, a proper address has to be assigned to this client. This is achieved by creating an EPR of the client.

```
EndpointReferenceType
consumerEPR=consumer.createNotificationConsumer(this);
```

3. Next, send a request to the services to start sending the notification by using the standard Notify operation. The consumer EPR is passed to identify the end point at which the notification is to be delivered.

```
Subscribe statusRequest = new Subscribe();
statusRequest.setUseNotify(Boolean.TRUE);
statusRequest.setConsumerReference(consumerEPR);
```

4. The client needs to inform the Topic to which it wishes to subscribe. In this case the client subscribes to different resource properties depending upon the service to which it subscribes. As for example, if the client program is executed as a seller, then the Resource Properties will be SellerStatus.

```
TopicExpressionType statusExpression = new
TopicExpressionType();
statusExpression.setDialect(WSNConstants.SIMPLE_TOPIC_DIALE
CT);
if(serviceType.equals("seller")){
statusExpression.setValue(SellerConstants.RP_SELLERSTATUS);
} else {
statusExpression.setValue(BuyerConstants.RP_BUYERSTATUS); }
```

5. A reference to the notification producer is obtained for sending subscription request by obtaining reference to the NotificationProducerportType, implemented by the remote service like the BuyerService or SellerService. This is done by using extends tag in the WSDL, where the portType of these services extend the NotificationProducerPortType. The notification to be delivered is for a service based on factory pattern. Thus, EPR is to be obtained for the resource instance to be

subscribed. This is achieved by reading from a file, which is created by the client program. After reading EPR subscription a request is sent to the remote service.

```
WSBaseNotificationServiceAddressingLocator notifLocator =
new WSBaseNotificationServiceAddressingLocator();
try {
FileInputStream fis = new FileInputStream(fileName);
File file=new File(fileName);
file.deleteOnExit();
endpoint=null;
endpoint = (EndpointReferenceType)
ObjectDeserializer.deserialize(new InputSource(fis),
EndpointReferenceType.class);
if(serviceType.equals("seller")){
endpointString =
ObjectSerializer.toString(endpoint,SellerRESOURCE_ENDPOINT)
;}
else{
endpointString =
ObjectSerializer.toString(endpoint,BuyerRESOURCE_ENDPOINT);
}
System.out.println("\nEND POINT STRING:"+ endpointString);
}  catch(Exception e){e.printStackTrace();}
producerPort =
notifLocator.getNotificationProducerPort(endpoint);
```

6. For every subscription request, a unique EPR is created. This EPR can be used to subscribe, pause and terminate subscription.

```
EndpointReferenceType statusSubscriptionEPR =
producerPort.subscribe(statusRequest).getSubscriptionRefere
nce();
```

7. The client starts listening to subscriptions till the time key is pressed. Incoming notifications are handled by a different method called deliver. The deliver method uses three parameters namely the topicPath used for creating subscription, the EPR of a notification producer and message which is sent by the service.

8. Our Resource classes used ResourcePropertyTopic for notification in ResourceProperties. Hence, the notification messages are of ResourcePropertyValueChangeNotificationType. This further embeds message of

ResourcePropertyValueChangeNotificationType in itself. The object of this class contains the new resource property value.

6.3.4.1 Notification to Seller Service

After initiation of a market service for Bhopal, matching will be performed for buyer(s) and seller(s) having preference for Bhopal. Based on the match found for a seller and a buyer for wheat, following notification will be generated for a seller service:

```
A new notification has arrived
Your crop sold till now is 10.0Kg
A new notification has arrived
Your crop has been sold recently
A new notification has arrived
Your crop quantity still remaining to be sold is:40.0Kg
A new notification has arrived
Amount Offered to you per Kg is Rs70.0
A new notification has arrived
Crop Quantity Which will be purchased by a buyer is 10.0Kg
A new notification has arrived
Your Earning is Rs700.0
Waiting for notification. Ctrl-C to end.
```

6.3.4.2 Notification to a buyer service

The following notification will be generated for a buyer service:

```
A new notification has arrived
The Crop Quantity which you will purchase from a seller is 10.0Kg
A new notification has arrived
Amount Offered by you per Kg is: Rs70.0
A new notification has arrived
Amount spent by you in current purchase is Rs700.0
```

6.4 Execution of Indirect Trading Process

Similarly, in the following example, one seller is interested in selling potato of katahdin variety in Kolkata. The interactions among various services are shown in subsequent snap shots:

6.4.1 Execution of Seller Service for Indirect Trading

```
$java -DGLOBUS_LOCATION=$GLOBUS_LOCATION
org.sws.examples.clients.GridManager.Client
http://10.100.64.65:8080/wsrf/services/sws/examples/GridManager
seller potato 60 katahdin Kolkata 10 indirect
```

The execution of seller grid service will start and will wait for the appropriate offer to come from a potential buyer. Seller grid service will receive notification, once an appropriate offer is floated by a buyer. Initially, following output will be generated:

```
END POINT STRING:<ns1:SellerServiceEndpoint
xsi:type="ns2:EndpointReferenceType"
xmlns:ns1="http://www.apmc.org/namespaces/examples/SellerService"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns2="http://schemas.xmlsoap.org/ws/2004/03/addressing">
 <ns2:Address
xsi:type="ns2:AttributedURI">http://10.100.64.65:8080/wsrf/services/
sws/examples/SellerService</ns2:Address>
 <ns2:ReferenceProperties xsi:type="ns2:ReferencePropertiesType">
  <ns1:SellerResourceKey>faea5e70-162b-11db-9878-
ce21816644d9</ns1:SellerResourceKey>
 </ns2:ReferenceProperties>
 <ns2:ReferenceParameters xsi:type="ns2:ReferenceParametersType"/>
</ns1:SellerServiceEndpoint>
Waiting for notification. Ctrl-C to end.
Waiting for notification. Ctrl-C to end.
```

6.4.2 Execution of Buyer Service

```
$ java -DGLOBUS_LOCATION=$GLOBUS_LOCATION
org.sws.examples.clients.GridManager.Client
http://10.100.64.65:8080/wsrf/services/sws/examples/GridManager
buyer potato 60 katahdin Kolkata 60
```

Here the execution of the buyer grid service will start and it will wait for appropriate match to be identified.

```
END POINT STRING:<ns1:BuyerServiceEndpoint
xsi:type="ns2:EndpointReferenceType"
xmlns:ns1="http://www.apmc.org/namespaces/examples/BuyerService"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns2="http://schemas.xmlsoap.org/ws/2004/03/addressing">
```

```
 <ns2:Address
xsi:type="ns2:AttributedURI">http://10.100.64.65:8080/wsrf/services/
sws/examples/BuyerService</ns2:Address>
 <ns2:ReferenceProperties xsi:type="ns2:ReferencePropertiesType">
  <ns1:BuyerResourceKey>7be97b40-162d-11db-9878-
ce21816644d9</ns1:BuyerResourceKey>
 </ns2:ReferenceProperties>
 <ns2:ReferenceParameters xsi:type="ns2:ReferenceParametersType"/>
</ns1:BuyerServiceEndpoint>
Waiting for notification. Ctrl-C to end.
Waiting for notification. Ctrl-C to end.
```

6.4.3 Execution of Market Grid Service

```
$ java -DGLOBUS_LOCATION=$GLOBUS_LOCATION
org.sws.examples.clients.GridManager.Client
http://10.100.64.65:8080/wsrf/services/sws/examples/GridManager
market Kolkata
```

Here, market service of a particular location starts executing, it tries to match potential buyers and sellers. If an appropriate buyer is found for a seller, appropriate notification is sent to both of them to inform them about success of entering into a trade deal.

6.4.4 Notification to Seller Service

```
Waiting for notification. Ctrl-C to end.
A new notification has arrived
Crop Price for potato is Rs60.0\kg
A new notification has arrived
Total Market Price for potatois Rs3600.0
A new notification has arrived
Vehicle Fees is calculated @ two rupee per unit of crop sold
Vehicle Fees charged for 60.0 Kg of crop is Rs. 120.0
Earning after deduction of vehicle fees is Rs3480.0
A new notification has arrived
Calculating Marketing Fees
Marketing Fees will be deducted
Marketing Fees Equal To:348.0
A new notification has arrived
Final Amount Offered to you is:Rs3132.0
```

In this chapter, deployment and execution of a grid business process are discussed. The details of integration of different components and notifications occurring during execution of a process are described. The realization of proposed EDSOA and its lifecycle is demonstrated based on research experiments narrated with necessary details in chapter 5 and 6.

Chapter 7

Conclusions and Future Work

In the previous chapters, the architecture of our event-driven approach, lifecycle of Event-driven SOA is discussed starting from the model, composition, development, deployment and finally execution. In this chapter, we summarize the results and discuss future directions for this research work.

7.1 Conclusions

In this thesis, we have identified issues related to composition of a business process and discussed the requirements for event-driven composition and event-driven service-oriented architecture. We have discussed the event-driven lifecycle (model, compose, deploy and execution) of event-driven SOA. We have also discussed the need for a formal framework to model the event-driven dynamic business processes. We have demonstrated realization of event-driven Service-oriented Architecture with the prototype implementation of agro-produce marketing business process. Major contributions of this thesis are summarized below:

- **Event-driven Service-oriented Architecture [45, 47]:** Our proposal of the Event-driven Service-oriented Architecture is focused towards the convergence of existing paradigms such as Web services, Semantic web, and grid computing to achieve seamless interoperable integration of services. The ultimate goal of EDSOA is to enable organizations to seamlessly compose event-driven and dynamic business process, and to integrate them with the partners, providers and consumers at runtime. The proposed EDSOA is realized based on the prototype implementation of agro-produce marketing business process using existing standards, specifications and tools.

- **Convergence of Web services, Semantic web, and Grid computing:** We have used Web services for business process orchestration with interoperable integration of geographically dispersed and distributed heterogeneous services and information [48, 49].

- The semantic web approach is used to provide common vocabulary, domain knowledge and interoperability for business process orchestration [50]. Ontology and Rules are used to facilitate seamless, meaningful information integration and negotiation according to user context, location and requirements [46, 47, 51, 52].

- Grid computing is utilized to provide a virtualized infrastructure for integration, collaboration, monitoring and management of business processes [45, 55].

- **Event-driven Modeling and Composition [46, 54]:** The dynamic nature of a business process is modeled using event calculus. Event calculus provides formal approach to capture the event and to model a business process. We followed semantic and rules based approach for dynamic composition of services and for the generation of the composition schema.

- **Grid Business Process [55]:** To address limitations of stateless Web services and to capture requirements of a dynamic business process, we have developed Grid Business Process. Stateful services are developed using existing second generation Web services standards and specifications to show the integration of stateful services in a grid workflow [53].

- **Policy-driven Grid Business Process [57]:** We have devised policy-driven grid business process to aggregate information from multiple sources according to policy. Service grouping approach is used to aggregate services. Service grouping provides better query, search, and group notification [56]. We have used policy, event and semantic to achieve contract, context and content based dynamic service selection and composition.

- **Open Source SOA:** Emerging specifications are evaluated to capture requirements of a dynamic business process. We have provided step-by-step development procedures on relevant standards and tools. This thesis constitutes a first step towards event-driven SOA, event-driven composition, grid business process, and open-source SOA development. Effective integration of different open source standards, specifications, tools and techniques of Web services, Semantic web and Grid computing are demonstrated.

7.2 Future Work

The proposed EDSOA and its lifecycle can be enhanced to demonstrate the effectiveness and significance for any enterprise application. The future work is diverted into several paths: use of semantic, policy, service level agreement and distributed management for effective coordination of services (resources) to provide better search, query and management. There is a need to create VO based on policy; which allows service grouping and group notification, distributed service life cycle management, negotiation and dynamic services selection based on runtime environment such as context and location.

7.2.1 Policy based grid business process

The effectiveness and scalability of a VO can be enhanced by using policies. Policy associated with services or resources while discovery, selection and composition of services or resources. Policy defines a set of rules and conditions to access the resources. These policies can be anything ranging from membership criteria for a VO to nature of services provided within the VO. A policy can be defined to group the scattered services as per the needs of an enterprise to provide better search, query and management facilities to the user. WS-SG has limited built-in support for controlling various functionalities of VO through policies, which can be further coupled with Web Services Policy specification for fine grained control. The use of policy also helps during Web service selection phase to select services, which are matching specified functional and non-functional requirements of a service consumer [57]. To achieve dynamic services selection, we are using context (event), content (semantic), and contract (policy) based information.

7.2.2 Virtual Organization based Service Grouping and Group Notification

Grid allows the creation of dynamic Virtual Organization (VO) to allow discovery, access and sharing of services and resources to different users based on the WS-ServiceGroup specifications and the policy of particular VO. The VO can be the aggregation of various geographically dispersed VO's thus simulating small markets within a complex market place. WS-ServiceGroup specification is used to achieve aggregation of WS-Resources and services and for group notification.

Business processes are although immune to the state of other processes, but changes in the state of any process can trigger series of events. WSRF although doesn't specify any event or notification based protocol but do support notification of events (change of state, creation or deletion of resources) based on independent Web Services Notification (WSN) specification.

With the help of policy based VO, services grouping and notification we expect to achieve the following objectives:

- Context, contract and location based aggregation of resources and services.
- Context and location based delivery of notification, search and query.

7.2.3 Distributed Management of Grid business process

Web Services Distributed Management (WSDM) is a model for Web services management. The management of the VO and the resources available within the VO is a complicated process due to variety of resources and applications/services provided by various vendors; with varying compatibility level. WSDM is the first step for solving the management integration problem of Web Services. WSDM contains two standards for Web services distributed management: Management Using Web Services (MUWS) and Management of

Web Services (MOWS). The WSDM standard uses Web Services as a platform to provide all of the essential functionality required for the distributed management platform.

7.2.4 Negotiation and Agreement

Negotiation, agreement, and contract are the basic requirements of B2B and B2C transactions. Negotiation is required between suppliers of B2B process to form efficient supply chain and between suppliers and consumers to develop successful enterprise application. Agreement defines the protocol and message exchange pattern to negotiate the requirements, expectations and QoS. There is a need to integrate Service Level Agreement (SLA) and automated negotiation in grid business process with existing SLA standards and semantic rules support.

7.2.5 Open Source Framework and Tool

We are planning to develop open source framework, which can be use for a wide range of applications such as mobile application, embedded systems and enterprise environments. It will be based on open source standards and tools to provide SOA runtime platform.

In order to implement end-to-end EDSOA, Open source plug-in tool should be developed with single graphical user interface to manage the complete lifecycle of EDSOA, from modeling to execution monitoring. It should provide common environment for the development of the Web service, Semantic web and grid computing based application. Additionally, Open source Enterprise Service Bus (ESB) can be integrated in the proposed architecture so that one can integrate third party service based tool and services easily.

Appendix A

Implementation of the Grid Business Services

This section describes implementation, installation and configuration of a grid business process. Different commercial and open source implementation of the WSRF specifications are available. In this experiment Globus Toolkit 4.0.1 (GT4) [113] on Red Hat Linux 9 is used to develop and deploy different Web services. Globus implementation of WSRF comes with embedded container (i.e. Tomcat) and has extensive support for different level of security. GT4 is based on the Axis SOAP engine and makes maximum use of different tools and feature available in the Axis toolkit. Before discussing the implementation details of our research experiment, in the following sections establishment of grid, steps and configuration files involved in the development of stateful Web services are discussed. For developing stateful services in grid environment steps given in figure-27 are to be followed.

A.1 Establishment and Configuration of Grid

Establishment of Grid requires number of components to be configured and installed, before deployment of any Web service. A middleware namely Globus was utilized for establishment of Grid. The middleware requires the following components (which, include, not limited to) to be configured and installed before any kind of job can be submitted.

Globus Container: This is the container or the home, which hosts the various services. This is installed and configured as soon as Globus Toolkit is installed.

Security: Security is enforced by configuration of GSI (Grid Security Infrastructure). This enforces, that any entity aiming at utilizing grid, must obtain proper security certificates before any kind of facility from the grid can be harnessed. This is done by establishing a certification authority (CA) to issue certificates to any legitimate entity.

GridFTP: This is a grid specific protocol. It allows transfer of files from one grid node to another. It is very similar to FTP. GridFTP requires, that any entity which, needs transfer of files, must be a valid user. Hence, they must have proper security certificates in order to transfer files.

Reliable File Transfer (RFT): Reliable File Transfer is another utility which allows movement of data from one grid node to another. But, it differs from GridFTP in respect that it is Web service compliant i.e. it has been developed specifically according to the needs of WSRF. RFT is built upon GridFTP. Hence, configuration of GridFTP is required, before RFT is used. RFT allows user to write fire and forget kind of jobs. RFT is a web service hence, it uses SOAP message for communication. It allows multiple jobs to be queued for transfer, and allows queries to be fired on the status of transfer. RFT uses the client library of GridFTP to transfer files. Hence, it provides more utilities than GridFTP.

Postgresql: This is an open source database to be installed and configured according to the requirements of Globus. RFT uses postgresql in order to save the state of the data, which is being transferred. Hence, postgresql has to be correctly configured and installed if RFT is to be utilized

Globus Resource Allocation Manager (GRAM): Globus Resource Allocation Manager receives request from the clients for resource allocation. It allocates the required resources to the users and also controls active grid jobs. It acts like an entity responsible for managing batch jobs. Monitoring and Discovery Service (MDS) also queries GRAM to receive information regarding various resources available on grid.

Each node is a part of the grid. It individually requires the software components to be installed and configured. Grid Nodes share certification authority. CA is responsible for.

- Accepting the request for issue of security certificates
- Issuing proxy certificate as a temporary permission to submit jobs
- Checking the validity of any user who submits job for execution on grid

The Grid was established using 3 nodes of distributed computing lab of DA-IICT.

- revati.nakshatra.da-iict.org (10.100.64.64)

- pushya.nakshatra.da-iict.org (10.100.64.65)
- ashwini.nakshatra.da-iict.org (10.100.64.29)

The simpleCA was established on the node “revati”. The other nodes along with “revati”, were entrusted with the responsibility of deployment of stateful Web services using the GT4 (Globus Toolkit 4) container.

A.2 Designing WSDL Document

For the development of the stateful Web Services, first step is to design the WSDL document for the service. The WSDL document provides service interface and operations details. State information will provide as Service resource properties in <wsdl:type> section of WSDL document by using wsrp:ResourceProperties attribute of portType. The attribute ResourceProperties is the extension of the WSDL specification and is meant for WSRF compliant engines.

WSDL document is language-neutral, it means we can implement services or client in any language. But it should follow WS-Interoperability (WS-I) Basic Profile recommendations for the Document/literal style services. For the document style WSDL, each complex data types are wrapped in the element. Only elements can be referenced from other section of the WSDL documents are particularly <wsdl:message> section. WSDL of Buyer service is given in appendix for reference.

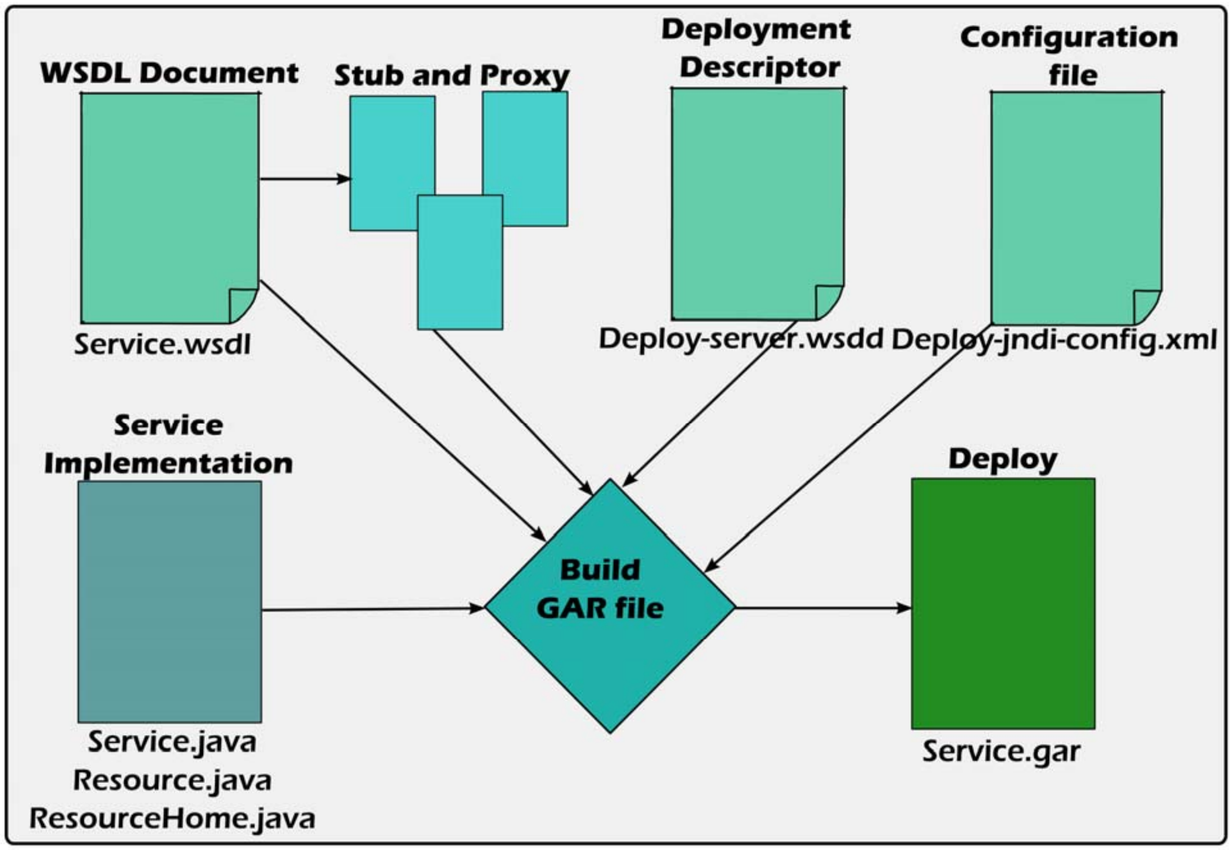


Figure 35: Grid Service Implementation Steps

A.3 Stub and Proxy Generation

The second step is to generate stubs and proxies from the WSDL document. We need to generate stub because stub will take care of SOAP invocation. There are many tools available to generate subs automatically from WSDL document.

The Axis framework provides a tool “WSDL2Java”, which parses the WSDL and generates appropriate classes compliant with the WSDL interface. The complex data types in the WSDL for Document/literal style are wrapped in elements. Elements are normally mapped to separate Java classes to shield developers from the low level details of XML. During stub and proxy generation the package structure is normally derived from the target namespace unless it is specifically mentioned through separate configuration mappings file (i.e. namespace2mapping).

A.4 Deployment Descriptor

The next step is Web services deployment, to provide access to the client service. Deployment information given in the file called deployment descriptor. The deployment descriptor provides enough information to the Axis framework to successfully deploy the Web service in Web services container. The normal information required by the Axis engine is the name of the Web service, style of the Web service, the classes implementing the business logic, the location and name of the WSDL file and number of parameters required for the successful instantiation of the service. The deployment descriptor is written in Web service Deployment Descriptor (WSDD) format. WSDD file is given in appendix for reference.

Service can access application specific parameter at run time by using either the JNDI API or the message context. GT4 specific stateful Web services uses these parameters very effectively. Below is the java code to retrieve the value of the parameter.

```
MessageContext.getCurrentContext().getService().getOption();
```

A.5 JNDI Configuration File

Deployment descriptor file contains various information of Web services but information related to resources is provided in Java Naming and Directory Interface (JNDI) file. JNDI file is responsible to retrieve Web services resources by locating resource home. According to the specifications the Web service is independent of the resource itself. It is the responsibility of the container and framework to load appropriate resource and carry out requested operations on the resource. This is one of the reasons, why deployment descriptor does not provide any information about resources coupled with the service.

JNDI is a Java API that provides interaction support and interface with a variety of directories. In GT4, directories are stored in container's memory. All the information required by the container for proper instantiation of the resource is provided in the “deploy-jndi-config.xml”. The JNDI configuration file is given in appendix.

The JNDI configuration file is basic file required by the Tomcat container to couple different resources (i.e. data objects, JDBC connection etc) within the application. For successful coupling of any resource with the service, ResourceHome is required for the instantiation of the resource which is normally called ResourceHome. ResourceHome implements the logic of the resource and the resource can only be instantiated through it. The only information specific to the WSRF implementation is the resourceKeyName and resourceKeyType. This information is used to search specific type of resource or particular resource within the same type of resources. The resourceKeyName is the qualified name and should map the qualified name of the resource as declared in the corresponding WSDL document. In the following section we discuss how the factory services instantiate the appropriate resource by retrieving the resource home.

A.6 Implied Resource Pattern

The Implied Resource pattern is a well known software design pattern to manage multiple resources. It is also known as Factory Instance Patterns. In this pattern, instead of creating instance of object directly, factory is responsible to create instance of object. In service-oriented domain when we are dealing with resources, two services (Factory service and Instance service) are responsible to manage multiple resources. A Factory service is responsible for the resource creation assigning it an identity, and creating a WS-Resource qualified endpoint reference to point to it. An Instance service is required to access and manipulate the information contained in the resources according to the business logic.

As we know, WSA is used to define the relationship between Web services and stateful resources and it is the core specification of the WS-Resource Framework. When dealing with multiple resources, the WSRF specifications recommend the use of the Implied Resource pattern. The implied resource pattern is a convention on XML, WSDL, and WS-Addressing that defines how a particular WS-Resource is referred within a Web service message context for processing. The term implied is used because the identity of the resource is implied by the context of the message and its association with an endpoint reference, not directly in the signature of the message. The use of the WSA is to identify the stateful resource to be used while processing the Web service message. The endpoint reference provides means to point to both the Web service and the stateful resource in one convenient XML element. Implied Resource Pattern can be extended in various ways to effectively capture the requirements of the application. Different possible extensions such as Factory/Instance pair pattern, Factory/Instance Collection pattern and Master/Slave pattern that we have discussed in our previous work [55].

We have followed Implied Resource Pattern for the development of grid business process. Following figure shows the interaction among client, factory service, resource home, resource and instance service [10].

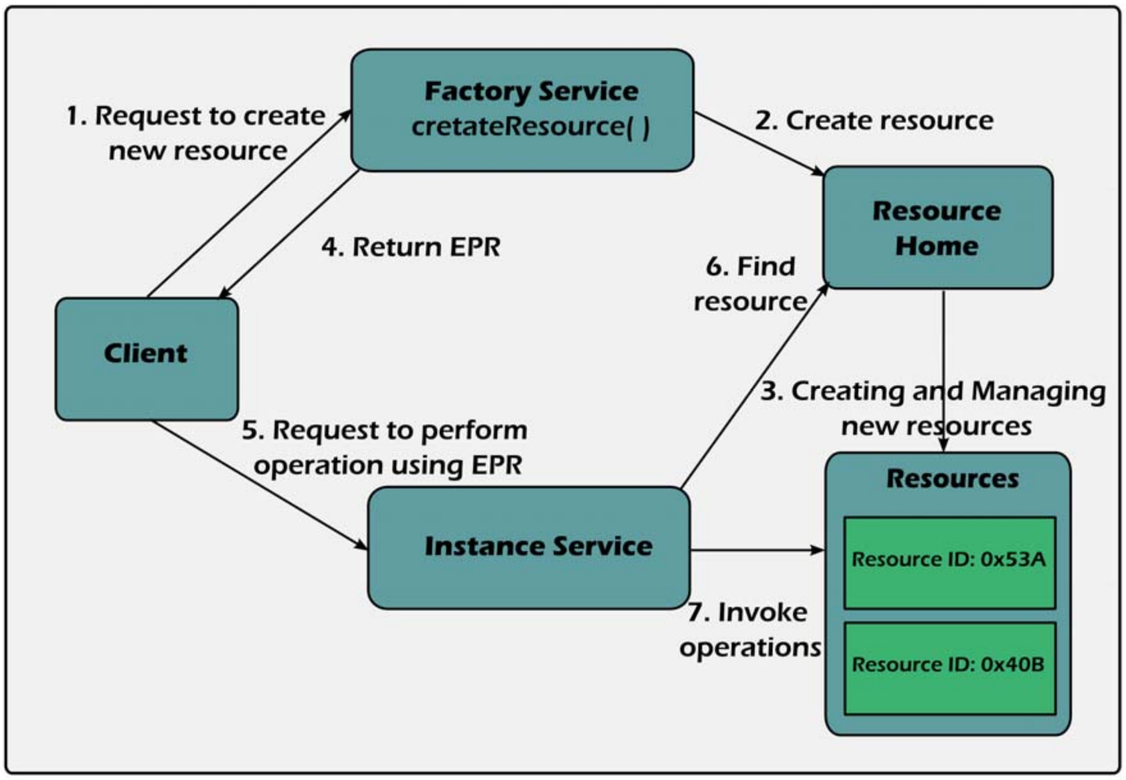


Figure 36: Interaction of Factory Instance Pattern

A.6.1 Factory Service

The role of the Factory service is to instantiate the appropriate resource through its resource home. The Factory service retrieves the information related to the resource and its resource home through the JNDI configuration file. In the JNDI configuration file discussed in the previous section, the factory service doesn’t have any resource associated with it but it does have a link to the resource home for the SellerService. The SellerFactoryService will retrieve the information about the resource, which in fact it is sharing with the SellerService, and instantiate it appropriately. After retrieving the resource home object, create() method of the resource home object will be called. The create() method instantiates the resource and returns the unique ID of the resource. This unique ID is used to create EPR, which is returned to the client.

A.6.2 Instance Service

The instance service finds the resource from resource home using EPR. It invokes the operations of resources. It also has access to state information of resources. It helps client to access multiple resources only by using EPR.

A.6.3 Resource Home

The resource home is a simple class with only one method create(); which initializes the corresponding resource. The resource home normally extends the GT4 specific ResourceHomeImpl class. The ResourceHomeImpl supper class locates the resource through the JNDI configuration file and instantiates it appropriately. The ResourceHomeImpl also provides the private hash table to store all the resources created through the custom tailored resource home. The logic of searching and updating the private hash table is automatically available in the custom resource home by extending the ResourceHomeImpl.

Other than instantiating the resource, the resource home creates the instance of the SuperResourceKey. The SuperResourceKey couples the unique ID of the resource with its fully qualified name. This fully qualified unique ID is used in the EPR. At anytime, the container can be managing many different types of resources which may have the same unique ID; it is the fully qualified name, and the unique ID of the resource, which makes any specific instance of the resource unique from other instances.

Appendix B Source Code

This appendix provides source codes, configuration files, and WSDL files to get more details.

JNDI Configuration file

Below is the sample “deploy-jndi-config.xml” file. All information required by the container for proper instantiation of the resource is provided in this file.

```
<jndiConfig xmlns="http://wsrf.globus.org/jndi/config">
    <service name="AddressBookService">
        <resource
            name="home"
type="daiict.ac.in.wsrf.addressbook.AddressBookResourceHome">
            <resourceParams>
                <parameter>
                    <name>factory</name>
                    <value>org.globus.wsrf.jndi.BeanFactory</value>
                </parameter>
                <parameter>
                    <name>resourceClass</name>

<value>daiict.ac.in.wsrf.addressbook.AddressBookResource</value>
                </parameter>
                <parameter>
                    <name>resourceKeyName</name>

<value>{http://wsrf.daiict.ac.in/addressbook}AddressBookKey</value>
                </parameter>
                <parameter>
                    <name>resourceKeyType</name>
                    <value>java.lang.Integer</value>
                </parameter>
            </resourceParams>
        </resource>
    </service>

    <!-- Factory service -->
    <service name="AddressBookFactoryService">
           <resourceLink name="AddressBookHome"
 target="java:comp/env/services/AddressBookService/home"/>
    </service>
</jndiConfig>
```

Code Snippet 20: deploy-jndi-config.xml file

Registration of resource with Index Service

Below is the resource home code snippet to register the resource with the Index Service:

```
protected void add(ResourceKey key, Resource resource) {
super.add(key, resource);
ServiceGroupRegistrationClient regClient =
ServiceGroupRegistrationClient.getContainerClient();
ResourceContext ctx;

try {
           ctx = ResourceContext.getResourceContext();
} catch (Exception e) { e.printStackTrace(); }

EndpointReferenceType epr=null;
try {
           URL baseURL = ServiceHost.getBaseURL();
           String instanceService = getInstanceServicePath();
           String instanceURI = baseURL.toString() +
           instanceService;
           epr =
      AddressingUtils.createEndpointReference(instanceURI, key);
} catch (Exception e) { e.printStackTrace(); }
// Location of the configuration file
String regPath = ContainerConfig.getGlobusLocation()
      + "/etc/org_sws_examples_services_buyer/registration.xml";

try {
           AddressBookResource buyerResource = (AddressBookResource)
           resource;
           ServiceGroupRegistrationParameters params =
           ServiceGroupRegistrationClient
                      .readParams(regPath);
           params.setRegistrantEPR(epr);
           Timer regTimer = regClient.register(params);
           addressBookResource.setRegTimer(regTimer);
} catch (Exception e) { e.printStackTrace(); } }
```

Code Snippet 21: Demonstrating registration of resource with Index Service

Registration file

Below is the sample configuration file ‘registration.xml’ which contains information of resource.

```
<ServiceGroupRegistrationParameters
xmlns:sgc="http://mds.globus.org/servicegroup/client"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
   xmlns:agg="http://mds.globus.org/aggregator/types"
   xmlns="http://mds.globus.org/servicegroup/client" >

<RefreshIntervalSecs>30</RefreshIntervalSecs>
   <Content xsi:type="agg:AggregatorContent"
xmlns:agg="http://mds.globus.org/aggregator/types">
      <agg:AggregatorConfig xsi:type="agg:AggregatorConfig">
         <agg:GetMultipleResourcePropertiesPollType
            xmlns:bs="http://www......./AddressBookService" >
            <agg:PollIntervalMillis>20000</agg:PollIntervalMillis>
<agg:ResourcePropertyNames>….</agg:ResourcePropertyNames
<!—Resource Properties specific to the AddressBookService -->
     …………………………………….
</agg:GetMultipleResourcePropertiesPollType>
      </agg:AggregatorConfig>
    <agg:AggregatorData/>
   </Content>
</ServiceGroupRegistrationParameters>
```

Code Snippet 22: registration.xml

<u>GridManagerService WSDL file</u>

The “GridManager” service instantiate resources related to different stateful Web services. Below is the WSDL document of the GridManager Web service:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="GridManagerService"
targetNamespace=
"http://www.apmc.org/namespaces/examples/GridManagerService"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://www.apmc.org/namespaces/examples/GridManagerServic
e"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```
<types>
<xsd:schema
targetNamespace=http://www.apmc.org/namespaces/examples/GridManagerS
ervice
xmlns:tns="http://www.apmc.org/namespaces/examples/GridManagerServic
e"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:import
namespace="http://schemas.xmlsoap.org/ws/2004/03/addressing"
         schemaLocation="../../ws/addressing/WS-Addressing.xsd" />

      <!-- REQUESTS AND RESPONSES -->
      <xsd:element name="createResource">
      <xsd:complexType>
           <xsd:sequence>
                <xsd:element ref="serviceName" type="xsd:string"/>
           </xsd:sequence>
      </xsd:complexType>
      </xsd:element>

      <xsd:element name="createResourceResponse">
      <xsd:complexType>
           <xsd:sequence>
                <xsd:element ref="wsa:EndpointReference"/>
           </xsd:sequence>
      </xsd:complexType>
      </xsd:element>
</xsd:schema>
</types>
<message name="CreateResourceRequest">
      <part name="request" element="tns:createResource"/>
</message>
<message name="CreateResourceResponse">
      <part name="response" element="tns:createResourceResponse"/>
</message>
<portType name="GridManagerPortType">
      <operation name="createResource">
           <input message="tns:CreateResourceRequest"/>
           <output message="tns:CreateResourceResponse"/>
      </operation>
</portType>
</definitions>
```

Code Snippet 23: WSDL file of GridManagerService

GridManagerService createResource() method

The createResource method of Grid Manager service instantiates the appropriate resource and returns the corresponding EPR. Below is the code snippet of createResource method:

```
public CreateResourceResponse createResource(CreateResource request)
                throws RemoteException {
           // Retrive the service name from the client request
           String serviceName=request.getServiceName();

           BuyerResourceHome home = null;
           SellerResourceHome sellerHome = null;
           MarketResourceHome marketHome = null;
           ResourceKey key = null;
           String homeLoc=null;
           EndpointReferenceType instanceEPR;
           try {
                if(serviceName.equals("buyer")) {
                     home = (BuyerResourceHome)
getInstanceResourceHome(serviceName);
                     key = home.create();
                }
//Similar kind of code for other services
catch (Exception e) {
                     throw new RemoteException(e.getMessage(), e);
                }
                EndpointReferenceType epr = null;
                try {
                     URL baseURL = ServiceHost.getBaseURL();
                     String instanceService="";
if(serviceName.equals("buyer")) {
instanceService=(String)MessageContext.getCurrentContext().getServic
e().getOption(
"buyerInstance");
}
// Similar kind of code for other services
                String instanceURI = baseURL.toString() +
instanceService;
                epr =
AddressingUtils.createEndpointReference(instanceURI, key);
           } catch (Exception e) {
                throw new RemoteException(e.getMessage(), e);
           }
      CreateResourceResponse response = new
CreateResourceResponse();
      response.setEndpointReference(epr);
      instanceEPR=response.getEndpointReference();
```

```
     return response;
     }
```

Code Snippet 24: createResource() method of GridManagerService

<u>Buyer WSDL file</u>

Various operations of the buyer service are exposed as a Resource Properties of the Buyer resource. Following WSDL code snippet shows different resource properties of the buyer resource:

```
<xsd:element name="BuyerCropName" type="xsd:string"/>
<xsd:element name="BuyerCropQuantity" type="xsd:float"/>
<xsd:element name="BuyerCropVariety" type="xsd:string"/>
<xsd:element name="BuyerMarket" type="xsd:string"/>
<xsd:element name="BuyerOfferedPrice" type="xsd:float"/>
<xsd:element name="BuyerAmountSpent" type="xsd:float"/>

<xsd:element name="BuyerResourceProperties">
<xsd:complexType>
     <xsd:sequence>
     <xsd:element ref="tns:BuyerCropName" minOccurs="1"
maxOccurs="1"/>
     <xsd:element ref="tns:BuyerCropQuantity" minOccurs="1"
maxOccurs="1"/>
     <xsd:element ref="tns:BuyerCropVariety" minOccurs="1"
maxOccurs="1"/>
     <xsd:element ref="tns:BuyerMarket" minOccurs="1"
maxOccurs="1"/>
     <xsd:element ref="tns:BuyerOfferedPrice" minOccurs="1"
maxOccurs="1"/>
     <xsd:element ref="tns:BuyerAmountSpent" minOccurs="1"
maxOccurs="1"/>
     </xsd:sequence>
</xsd:complexType>
</xsd:element>
```

Code Snippet 25: Resource Property declaration in Buyer.wsdl

fireXpathQuery() method

The fireXpathQuery method is used to query the IndexService to match trading preferences of buyers and sellers registered in the same market. Below is the code snippet of fireXpathQuery method:

```
QueryResourcePropertiesResponse fireXpathQuery(String type,String
marketCity) {
String
indexURI="http://10.100.64.64:8080/wsrf/services/DefaultIndexService
";
String xpathSellerQuery,xpathBuyerQuery="";
EndpointReferenceType indexEPR = new EndpointReferenceType();
try {
           indexEPR.setAddress(new Address(indexURI));
} catch (Exception e) {
e.printStackTrace();
}
// Get QueryResourceProperties portType
WSResourcePropertiesServiceAddressingLocator queryLocator;
           queryLocator = new
WSResourcePropertiesServiceAddressingLocator();
           QueryResourceProperties_PortType query = null;
           try {
                query= =
queryLocator.getQueryResourcePropertiesPort(indexEPR);
           } catch (Exception e) {
                e.printStackTrace();
           }
if(type.equals("seller")) {
xpathSellerQuery ="//*[local-name()='Entry'][./*/*/*[local-
name()='SellerMarket']/text()='"+ marketCity +"']";
} else {
xpathSellerQuery ="//*[local-name()='Entry'][./*/*/*[local-
name()='BuyerMarket']/text()='"+ marketCity +"']";
}
           // Create request to QueryResourceProperties
           QueryExpressionType queryExpr = new
QueryExpressionType();
           try {
                queryExpr.setDialect(new
     URI(WSRFConstants.XPATH_1_DIALECT));
           } catch (Exception e) { e.printStackTrace();}
           queryExpr.setValue(xpathSellerQuery);
           QueryResourceProperties_Element queryRequest = new
QueryResourceProperties_Element(
                      queryExpr);

           // Invoke QueryResourceProperties
           QueryResourcePropertiesResponse queryResponse = null;
           try {
                queryResponse =
query.queryResourceProperties(queryRequest);
           } catch (RemoteException e) {
           e.printStackTrace();
           }
           return queryResponse;
      }
```

Code Snippet 26: Operation fireXpathQuery() for querying Index Service

storeEntries() method

We need to de-serialize the MessageElement entries to invoke different operations and to retrieve the EPR of the resources for further point-to-point communication. Below is the code snippet of the storeEntries(..) method for retrieving the information from individual entries:

```
void storeEntries(QueryResourcePropertiesResponse
queryResponse,String type,
MessageElement[] entries) {
      Vector tmpVector=new Vector();
      EndpointReferenceType[] tempEPR=new
EndpointReferenceType[entries.length];
for(int m=0;m<entries.length;m=m+1) {
           tempEPR[m]=new EndpointReferenceType();
      }
      for (int i = 0; i < entries.length; i++)    {
      try {
           int j=0;
           EntryType entry = (EntryType)
           ObjectDeserializer.toObject(
           entries[i], EntryType.class);
           if (type.equals("seller")) {
                tempEPR[i]=entry.getMemberServiceEPR();
           }    else  {
                tempEPR[i]=entry.getMemberServiceEPR();
           }
           AggregatorContent content = (AggregatorContent)
           entry.getContent();
           AggregatorData data = content.getAggregatorData();
           String cropName = data.get_any()[0].getValue();
           tmpVector.addElement(cropName);
           ….
      } catch (Exception e) {…}
….
}
```

Code Snippet 27: Operation storeEntries()for de-serialization of MessageElement entries

NotificationListenerClient

Below is the code snippet of NotificationListenerClient to subscribe and receive notification messages:

```
package org.sws.examples.clients.GridManager;

public class NotificationListenerClient
    extends BaseClient
    implements NotifyCallback {

static String SellerNS =
"http://www.apmc.org/namespaces/examples/SellerService";
static QName SellerRESOURCE_ENDPOINT = new
QName(SellerNS,"SellerServiceEndpoint");

      static String BuyerNS =
"http://www.apmc.org/namespaces/examples/BuyerService";
static QName BuyerRESOURCE_ENDPOINT = new
QName(BuyerNS,"BuyerServiceEndpoint");

NotificationProducer producerPort=null;

final static WSResourcePropertiesServiceAddressingLocator locator =
      new WSResourcePropertiesServiceAddressingLocator();
public void startSubscription(String serviceType,String fileName) {

   String endpointString="";
    try {
      NotificationConsumerManager consumer = null;
      consumer = NotificationConsumerManager.getInstance();
      consumer.startListening();
      EndpointReferenceType consumerEPR
       =consumer.createNotificationConsumer(this);
      Subscribe statusRequest = new Subscribe();
      statusRequest.setUseNotify(Boolean.TRUE);
      statusRequest.setConsumerReference(consumerEPR);
      TopicExpressionType statusExpression = new
     TopicExpressionType();

statusExpression.setDialect(WSNConstants.SIMPLE_TOPIC_DIALECT);
      if(serviceType.equals("seller"))
      {

statusExpression.setValue(SellerConstants.RP_SELLERSTATUS);
      }
      else
      {

statusExpression.setValue(BuyerConstants.RP_BUYERSTATUS);
      }
      statusRequest.setTopicExpression(statusExpression);
      WSBaseNotificationServiceAddressingLocator notifLocator =
          new WSBaseNotificationServiceAddressingLocator();
                try
           {
                FileInputStream fis = new
                FileInputStream(fileName);
                File file=new File(fileName);
                file.deleteOnExit();
                endpoint=null;
     endpoint = (EndpointReferenceType)
     ObjectDeserializer.deserialize(new InputSource(fis),
                EndpointReferenceType.class);
                if(serviceType.equals("seller"))
                {
                endpointString = ObjectSerializer.toString
                (endpoint,SellerRESOURCE_ENDPOINT);
                }
                else
                {
                endpointString = ObjectSerializer.toString
                (endpoint,BuyerRESOURCE_ENDPOINT);
                }
                System.out.println("\nEND POINT STRING:"+
                endpointString);
           }
           catch(Exception e)
           {
                e.printStackTrace();
           }
producerPort = notifLocator.getNotificationProducerPort(endpoint);
EndpointReferenceType statusSubscriptionEPR =
producerPort.subscribe(statusRequest).getSubscriptionReference();
System.out.println("Waiting for notification. Ctrl-C to end.");
    }
    catch (Exception e) {
      e.printStackTrace();
    }
    while (true) {
      System.out.println("Waiting for notification. Ctrl-C to
      end.");
      try {
        Thread.sleep(30000);
      }
      catch (Exception e) {
      e.printStackTrace();
        //System.out.println("Interrupted while sleeping.");
      }
    }
  }
  public void deliver(List topicPath,
                      EndpointReferenceType producer,
                      Object message) {
    ResourcePropertyValueChangeNotificationType changeMessage =
        ( (ResourcePropertyValueChangeNotificationElementType)
message).
        getResourcePropertyValueChangeNotification();
  if (changeMessage != null) {
      String
msg=changeMessage.getNewValue().get_any()[0].getValue();
if(msg.equals("Resource Instance will be deleted since your trading
has been completed")||
msg.equals("\nYour purchasing is over!Your resource instance will be
destroyed")||msg.equals("\nYour crop is sold!Your resource instance
will be destroyed"))
      {
           System.exit(0);
      }
      System.out.println("\nA new notification has arrived");

System.out.println(changeMessage.getNewValue().get_any()[0].getValue
());     }   } }
```

Code Snippet 28: NotificationListenerClient

Bibliography

[1] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web Services: Concepts, Architecture and Applications*: Springer Verlag, 2004.

[2] C. Bussler, *B2B Integration: Concepts and Architecture*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2003.

[3] A. Sahai and S. Graupner, *Web Services in the Enterprise: Concepts, Standards, Solutions, and Management*: Springer Science, 2005.

[4] W3C, "World Wide Web Consortium, Web Services Glossary, http://www.w3.org/2002/ws/arch/2/06/wd-wsa-gloss-20020605.html," 2002.

[5] WSDL, "Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/2001/NOTE-wsdl-20010315 " 2005.

[6] SOAP, "Simple Object Access Protocol (SOAP) 1.1, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/ " 2000.

[7] UDDI, "Universal Description, Discovery and Integration, UDDI Version 3.0.2, http://uddi.org/pubs/uddi_v3.htm," 2004.

[8] P. McKee, "Grid — the 'white knight' for business?," *BT Technology Journal*, vol. 23, pp. 45--51, 2005.

[9] J. Joseph and C. Fellenstein, *Grid Computing*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2003.

[10] B. Sotomayor and L. Childers, *Globus Toolkit 4: Programming Java Services*: Morgan Kaufmann, 2005.

[11] WS-ResourceFramework, "Web Services Resource Framework, www.oasis-open.org/committees/wsrf/ " 2005.

[12] F. Leymann and K. Guntzel, *The Business Grid: Providing Transactional Business Processes via Grid Services*, 2003.

[13] L.-J. Zhang, H. Li, and H. Lam, "Toward a Business Process Grid for Utility Computing," *IT Professional*, vol. 6, pp. 64-63, 2004.

[14] H. Stuckenschmidt and F. v. Harmelen, *Information Sharing on the Semantic Web*: Springer, 2004.

[15] S. Dustdar and W. Schreiner, "A Survey on Web Services composition," *Int. Journal Web and Grid Services*, vol. 1, 2006.

[16] U. Kuster, M. Stern, and B. Konig-Ries, "A Classification of Issues and Approaches in Automatic Service Composition," *International Workshop on Engineering Service Compositions*, December 2005.

[17] J. Rao and X. Su, "A Survey of Automated Web Service Composition Methods," *International Workshop on Semantic Web Services and Web Process Composition*, 2004.

[18] S. R. Ponnekanti and A. Fox, "SWORD: A Developer Toolkit for Web Service Composition," in *The 11th International World Wide Web Conference (Web Engineering Track)* Honolulu, Hawaii, May 2002.

[19] F. Casati and M.-C. Shan, "Dynamic and adaptive composition of e-services," *The 12th international conference on advanced information systems engineering (CAiSE 00)* 2001.

[20] H. Sun, X. Wang, B. Zhou, and P. Zou, "Research and Implementation of Dynamic Web Services Composition. ," in *Advanced Parallel Processing Technologies (APPT)*: Springer Berlin / Heidelberg, 2003, pp. 457-466.

[21] T. Andrews and F. Curbera et al, "Business Process Execution Language for Web Services, Version 1.1," 2003.

[22] D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau, "Automating DAML-S Web Services Composition Using SHOP2," in *International Semantic Web Conference*, 2003.

[23] B. Medjahed, A. Bouguettaya, and A. K. Elmagarmid, "Composing web services on the semantic web," *The VLDB Journal*, 2003.

[24] D. Berardi, D. Calvanese, G. D. Giacomo, R. Hull, and M. Mecella, "Automatic Composition of Transition-based Semantic Web Services with Messaging.," *The VLDB Journal*, 2005.

[25] OWL-S, "OWL-based Web Service Ontology (OWL-S), http://www.daml.org/services/owl-s/."

[26] R. Akkiraju, J. Farrell, J. Miller, M. Nagarajan, M.-T. Schmidt, A. Sheth, and K. Verma, "Web Service Semantics - WSDL-S, W3C Member Submission 7 November 2005, http://www.w3.org/Submission/2005/SUBM-WSDL-S-20051107/ " 2005.

[27] UML, "Unified Modeling Language, Version 2.1.1, http://www.omg.org/technology/documents/modeling_spec_catalog.htm," 2007.

[28] B. Orriens, J. Yang, and M. P. Papazoglou, "Model Driven Service Composition," in *International Conference of Service-Oriented Computing*, 2003.

[29] L. Zhang, J. Chung, and H. Chang, "Model Driven Dynamic Composition of Web Services Flow for Business Process Integration," *OMG Web Services Workshop* April 22-25, 2003.

[30] D. Ardagna and B. Pernici, "Adaptive Service Composition in Flexible Processes," *IEEE Transactions on Software Engineering*, vol. Volume 33, 2007.

[31] B. Schmit and S. Dustdar, *Systematic Design of Web Service Transactions*, 2005.

[32] M. P. Papazoglou, "Web Services and Business Transactions," *World Wide Web*, vol. 6, pp. 49-91, 2003.

[33] WS-Eventing, "Web Services Eventing (WS-Eventing), http://www.w3.org/Submission/WS-Eventing/ " 2006.

[34] L. Wilkes, "The Web Services Protocol Stack, http://roadmap.cbdiforum.com/reports/protocols/index.php," 2005.

[35] H. Wang, J. Z. Huang, Y. Qu, and J. Xie, "Web services: Problems and Future Directions," *Journal of Web Semantics*, vol. 1, 2004.

[36] S. Vinoski, "WS-nonexistent standards," *Internet Computing, IEEE*, vol. 8, pp. 94-96, 2004.

[37] S. Vinoski, "More Web services notifications," *Internet Computing, IEEE,* vol. 8, pp. 90-93, 2004.
[38] K. Birman, "Can Web Services Scale Up?," *Computer,* vol. 38, pp. 107-110, 2005.
[39] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," in *Scientific American*, May 17, 2001, pp. 35-43.
[40] S. A. McIlraith, T. C. Son, and H. Zeng, "Semantic Web Services," *IEEE Intelligent Systems*, vol. 16, pp. 46-53, March 2001
[41] H. Lausen, A. Polleres, and D. Roman, "Web Service Modeling Ontology (WSMO), W3C Member Submission 3 June 2005, http://www.w3.org/Submission/2005/SUBM-WSMO-20050603/ " 2005.
[42] WS-Notification, "Web Services Notification, www.oasis-open.org/committees/wsn/ " 2005.
[43] WSDM, "OASIS Web Services Distributed Management (WSDM) TC, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm ".
[44] D. D. Roure, N. R. Jennings, and N. R. Shadbolt, "The Semantic Grid: A Future e-Science Infrastructure," in *Grid Computing: Making the Global Infrastructure a Reality*, F. Berman, G. Fox, and T. Hey, Eds.: John Wiley & Sons, Ltd, 30 May 2003.
[45] Z. Laliwala, P. Jain, and S. Chaudhary, "Semantic based Service-Oriented Grid Architecture for Business Processes," in *IEEE International Conference on Services Computing (SCC 2006), Industry Track* 2006.
[46] Z. Laliwala, R. Khosla, P. Majumdar, and S. Chaudhary, "Semantic and Rule Based Event-Driven Dynamic Web Services Composition for Automation of Business Processes," in *Third International Workshop on Semantic and Dynamic Web Processes*, 2006.
[47] Z. Laliwala, V. Sorathia, and S. Chaudhary, "Semantics and Rule Based Event-driven Services-Oriented Agricultural Recommendation System," in *International Workshop on Distributed Event-Based Systems*, 2006.
[48] S. Chaudhary, V. Sorathia, and Z. Laliwala, "Architecture of Sensor based Agricultural Information System for Effective Planning of Farm Activities," in *SCC '04: Proceedings of the 2004 IEEE International Conference on Services Computing* Washington, DC, USA: IEEE Computer Society, 2004, pp. 93-100.
[49] V. Sorathia, Z. Laliwala, and S. Chaudhary, "Towards Agricultural Marketing Reforms: Web Services Orchestration Approach," in *SCC '05: Proceedings of the 2005 IEEE International Conference on Services Computing* Washington, DC, USA: IEEE Computer Society, 2005, pp. 260-270.
[50] S. Chaudhary, Z. Laliwala, and V. Sorathia, "Building Semantic Business Services," in *The Semantic Web and its Applications*, J. Cardoso and A. Sheth, Eds.: Springer, 2006.
[51] Z. Laliwala, S. Chaudhary, C. Venus, and A. Mahajan, "Semantic and Rules Based Automated Negotiation," in *1st International Workshop on Service- and Process-Oriented Software Engineering (SOPOSE) co-located with Asia-Pacific Conference on Software Engineering (APSEC-2006)*, 2006.

[52] Z. Laliwala, V. Sorathia, and S. Chaudhary, "Semantics based Event-driven Publish/Subscribe Service-Oriented Architecture," in *SOFTPLATFORM workshop co-located with COMSWARE 2006*, 2006.

[53] Z. Laliwala, P. Jain, S. Chaudhary, and V. Sorathia, "Integrating stateful services in workflow," in *13th Asia Pacific Software Engineering Conference (APSEC)*, 2006.

[54] Z. Laliwala and S. Chaudhary, "Event-Driven Dynamic Web Services Composition: From Modeling to Implementation, 2006. ," in *Third International Conference on Innovations in Information Technology (IIT06)*, 2006.

[55] A. Akram, R. Allan, S. Chaudhary, P. Jain, and Z. Laliwala, "Grid Business Process: Case Study," in *Securing Web Services: Practical Usage of Standards and Specifications*, P. Periorellis, Ed.: Idea Group Inc, 2007.

[56] Z. Laliwala and S. Chaudhary, "Service Grouping and Group Notification in Grid Business Process," in *ACM Compute 2008* Bangalore, India, 2008.

[57] A. Desai, Z. Laliwala, and S. Chaudhary, "Policy-driven Grid Business Process," in *Workshop on Grid and Utility Computing co-located with International Conference on High Performance Computing (HiPC)*, , December 2007. .

[58] T. Erl, *Service-Oriented Architecture: Concepts, Technology, and Design* Prentice Hall PTR, 2005.

[59] Webopedia, "Webopedia: Online Dictionary for Computer and Internet Terms. http://www.webopedia.com."

[60] WS-Addressing, "Web Services Addressing, W3C Member Submission 10 August 2004, http://www.w3.org/Submission/ws-addressing/," 2004.

[61] I. Foster, K. Czajkowski, and et al, "Modeling and managing State in distributed systems: the role of OGSI and WSRF," in *Proceedings of the IEEE*, 2005.

[62] WS-ResourceProperties, "Web Services Resource Properties 1.2, http://docs.oasis-open.org/wsrf/wsrf-ws_resource_properties-1.2-spec-os.pdf," 2006.

[63] WS-Topics, "Web Services Topics 1.3, http://docs.oasis-open.org/wsn/wsn-ws_topics-1.3-spec-pr-02.pdf," 2006.

[64] WS-ResourceLifetime, "Web Services Resource Lifetime 1.2, http://docs.oasis-open.org/wsrf/wsrf-ws_resource_lifetime-1.2-spec-os.pdf," 2006.

[65] WS-BaseNotification, "Web Services Base Notification 1.3, http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-pr-03.pdf," 2006.

[66] WS-ServiceGroup, "Web Services Service Group 1.2, http://docs.oasis-open.org/wsrf/wsrf-ws_service_group-1.2-spec-os.pdf," 2006.

[67] WS-BaseFaults, "Web Services Base Faults 1.2, http://docs.oasis-open.org/wsrf/wsrf-ws_base_faults-1.2-spec-os.pdf," 2006.

[68] WS-BrokeredNotification, "Web Services Brokered Notification 1.3, http://docs.oasis-open.org/wsn/wsn-ws_brokered_notification-1.3-spec-pr-03.pdf," 2006.

[69] MUWS:, V. Bullard, and W. Vambenepe, "Web Services Distributed Management: Management Using Web Services (MUWS 1.1), http://docs.oasis-open.org/wsdm/wsdm-muws1-1.1-spec-os-01.pdf," 01 August 2006.

[70] MOWS:, K. Wilson, and I. Sedukhin, "Web Services Distributed Management: Management of Web Services (WSDM-MOWS) 1.1, http://docs.oasis-open.org/wsdm/wsdm-mows-1.1-spec-os-01.pdf," 01 August 2006.

[71] WS-ReliableMessaging, "Web Services Reliable Messaging (WS-ReliableMessaging) Version 1.1, http://docs.oasis-open.org/ws-rx/wsrm/200702/wsrm-1.1-spec-os-01.pdf," 14 June 2007.

[72] WS-Policy, "W3C working draft, Web Services Policy 1.5 - framework, http://www.w3.org/tr/2006/wd-ws-policy-20061117," 17 November 2006.

[73] G. Antoniou and F. v. Harmelen, *A Semantic Web Primer*: The MIT Press, April 2004.

[74] D. Hazaël-Massieux, "The Semantic Web and its applications at W3C, http://www.w3.org/2003/Talks/simo-semwebapp/ ", 2003.

[75] J. Cardoso and A. Sheth, "The Semantic Web and its Applications," in *Semantic Web Services, Processes and Applications*, J. Cardoso and A. Sheth, Eds.: Springer, 2006.

[76] RDF, "Resource Description Framework (RDF), http://www.w3.org/RDF/."

[77] T. Gruber, "A Translation approach to portable ontology specifications," *Knowledge Acquisition*, vol. 5, 1993.

[78] OWL, "Web Ontology Language (OWL), http://www.w3.org/2004/OWL/," 2004.

[79] M. Born, C. Drumm, I. Markovic, and I. Weber:, "SUPER - Raising Business Process Management Back to the Business Level," *ERCIM News "Special Theme on Service-Oriented Computing"*, vol. Vol. 70, pp. pp. 43 - 44, July 2007

[80] A. Patil, S. Oundhakar, and et al, "MWSAF - METEOR-S Web Service Annotation Framework.," *13th Conference on World Wide Web, New York City, USA.*, 2004.

[81] K. Verma and A. Sheth, "Semantically Annotating a Web Service," *Internet Computing*, vol. 11, pp. 83-85, March/April 2007.

[82] Ian Foster et.al, "The Open Grid Services Architecture, Version 1.0, http://www.gridforum.org/documents/GWD-I-E/GFD-I.030.pdf," 29 January 2005.

[83] OGSI, "Open Grid Services Infrastructure, version 1.0, Proposed Recommendation," *Global Grid Forum, http://www.ggf.org/documents/GFD.15.pdf*, 2003.

[84] M. Havey, *Essential Business Process Modeling*: O'Reilly, August 2005.

[85] W. Aalst and A. Hofstede, "YAWL: Yet Another Workflow Language," *Information Systems*, vol. 30(4), 2005.

[86] P. Wohed, W. Aalst, M. Dumas, and A. Hofstede, "Pattern Based Analysis of BPEL4WS," *Technical Report FIT- TR-2002-04, QUT,2002.*, 2002.

[87] D. Sherman, "BPEL: Make Your Services Flow. Composing Web Services into Business Flow," *WebServices Journal*, vol. 7, pp. 16-21, July 2003.

[88] B. Benatallah, R. M. Dijkman, M. Dumas, and Z. Maamar, "Service Composition: Concepts, Techniques, Tools and Trends," *Service-Oriented Software System Engineering: Challenges and Practices, Idea group*, 2005.

[89] S. Staab and R. Studer, *Handbook on Ontologies*: Springer, 2004.

[90] M. Singh and M. N. Huhns, *Service-Oriented Computing - Semantics, Processes, Agents*: John Wiley and sons, Ltd., 2005.

[91] Model Act, "The State Agricultural Produce Marketing (Development & Regulation Act, 2003), http://agmarknet.nic.in/reforms.htm," 2003.
[92] AGROVOC, "AGROVOC, www.fao.org/agrovoc/."
[93] R. Kowalski and M. Sergot, "A Logic-Based Calculus of Events," *New Generation Computing*, vol. Vol. 4, p. 67—95, 1986.
[94] N. K. Cicekli and Y. Yildirim, "Formalizing Workflows Using the Event Calculus," *DEXA*, 2000.
[95] Mohsen Rouached, O. Perrin, and C. Godart, "Towards Formal Verification of Web Service Composition."
[96] N. W. Paton, *Active Rules in Database Systems*: Springer-Verlag, 1998.
[97] C. W. Tan and A. Goh, "“Implementing ECA Rules in an Active Database," *Knowledge-Based Systems*, vol. vol. 12, pp. pp. 137-144, Aug. 1999.
[98] U. Dayal, M. Hsu, and R. Ladin, "Organizing Long-Running Activities with Triggers and Transactions," in *ACM SIGMOD Conference*, 1990, pp. pp. 204-214.
[99] F. Casati, S. Castano, M. Fugini, I. Mirbel, and B. Pernici, "Using Patterns to Design Rules in Workflows," *IEEE Trans. Software Eng.*,, vol. vol. 26, pp. pp. 760-785, Aug. 2000.
[100] F. Casati, S. Ceri, B. Pernici, and G. Pozzi, "Deriving Active Rules for Workflow Enactment," in *Seventh Int'l Conf. Database and Expert Systems Applications* 1996, pp. pp. 94-110.
[101] F. Casati, M. Fugini, and I. Mirbel, "An Environment for Designing Exceptions in Workflows," *Information Systems*,, vol. vol. 24, pp. pp. 255-273, May 1999.
[102] J. Bae, H. Bae, S. Kang, and Y. Kim, "Automatic Control of Workflow Processes Using ECA Rules," *IEEE Transactions on Knowledge and Data Engineering*, August 2005.
[103] L. Chien, M. Li, and J. Cao, "A Rule-Based Workflow Approach for Service Composition," *ISPA*, 2005.
[104] L. Zeng, D. Flaxer, H. Chang, and J.-J. Jeng, "PLM flow-Dynamic Business Process Composition and Execution by Rule Inference," in *Third International Workshop on Technologies for E-Services* 2002.
[105] L. Zeng, "Dynamic web services composition," University of New South Wales, 2003.
[106] OracleBPEL, "Oracle BPEL Designer , http://www.oracle.com/technology/obe/obe_as_1012/integration/bpel/install/bpel_install.htm#o."
[107] Protégé, "http://protege.stanford.edu/."
[108] Sesame, "Sesame, http://www.openrdf.org/download.jsp."
[109] R. Akkiraju and et al, "Web Service Semantics - WSDL-S Version 1.0, http://www.w3.org/Submission/WSDL-S/ " 2005.
[110] B. N. Grosof and T. C. Poon, "SweetDeal: Representing Agent Contracts with Exceptions using Semantic Web Rules, Ontologies, and Process Descriptions," *International Journal of Electronic Commerce*, 2004.

[111] SweetRules, "http://sweetrules.projects.semwebcentral.org/."

[112] M. Zager, "SOA/Web services - Business Process Orchestration with BPEL," in *SOA WORLD MAGAZINE*, *http://webservices.sys-con.com/read/155631_1.htm*, December 6, 2005

[113] Globus Toolkit (GT4), "Globus Toolkit (GT4), http://www.globus.org/toolkit/," 2005.