Web Service Composition: Architecture, Frameworks, and Techniques

Rajesh Karunamurthy

A Thesis

In the Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy at

Concordia University

Montreal, Quebec, Canada

January 2009

Your file Votre référence
ISBN: 978-0-494-63400-4
Our file Notre référence
ISBN: 978-0-494-63400-4

ABSTRACT

Web Service Composition: Architecture, Frameworks, and Techniques

Rajesh Karunamurthy, Ph.D.

Concordia University, 2009.

OASIS defines Service Oriented Architecture (SOA) as a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. One approach to realize SOA is Web services. A Web service is a software system that has a machine processable Web Services Description Language (WSDL) interface; other systems interact with it using SOAP messages in a manner prescribed by its description. Descriptions enable Web services to be discovered, used by other Web services, and composed into new Web services. Composition is a mechanism for rapid creation of new Web services by reusing existing ones.

Web services have functional, behavioral, semantic, and non-functional characteristics. These characteristics have to be considered for composition, as they provide essential information about the services. In order to compose Web services with these characteristics, they have to be described appropriately. However, the existing techniques do not consider all these aspects together for description and composition.

This thesis proposes a business model, also referred to as architecture, a description framework, and a composition framework for Web service composition. Techniques for matching, categorizing, and assembling the composite services are also proposed as a part of the composition framework. The architecture, frameworks, and techniques describe, discover, manipulate, and compose Web services by taking into

account all their characteristics. The standard Web service business model is extended by the proposed business model to support Web service composition. In the model, based on their demand, the requested Web services are composed by the Web service composer.

In the proposed architecture, Web services are described using the description framework languages. The proposed framework combines Semantic Annotations for WSDL and XML Schema (SAWSDL) for functional and semantic description, Message Sequence Charts (MSC) for behavioral description, and a simple and new Non Functional Specification Language (NFSL) for the non-functional properties description of Web services. It uses Higher Order Logic (HOL) for formalizing and integrating the three languages.

The role of Web service composer in the architecture is realized by the composition framework. It essentially defines the architecture of the composer. In this framework, matchmaking, categorization, and assembly techniques are used to create the requested composite service. These techniques manipulate the Web services at HOL-level. The formal matchmaking technique discovers the primitive Web services by using a HOL theorem prover. The categorization and the assembly techniques manipulate the matched services and orchestrate the composite service.

The concepts of the model, frameworks, and techniques are implemented, and their working is illustrated using case studies. Prototypes of the model's components (extended registry and extended requester) and the composition framework are developed, and their performance is analyzed. Case studies to illustrate the description and the composition frameworks are also presented.

ACKOWLEDGEMENTS

My doctoral study was a memorable and a great learning experience, thanks to everyone who made it this way. Firstly, I would like to express my deep gratitude to my thesis supervisors, Dr. Ferhat Khendek and Dr. Roch Glitho. They were always inspiring, supportive, and helpful. Their comments shaped this thesis work at every stage. It was a great learning experience to work under their supervision.

I would like to thank my internal thesis committee, Dr. Anjali Agarwal, Dr. Patrice Chalin and Dr. Abdeslam En-Nouaary, for their critical and helpful comments throughout my thesis work. I would also like to thank Dr. Hanan Lutfiyya from The University of Western Ontario for the comments and for examining my thesis as external. Thanks to Dr. Rachida Dssouli for her comments that she gave during our Telecommunications Service Engineering (TSE) lab research meetings.

I am thankful to the Natural Sciences and Engineering Research Council of Canada (NSERC), and Ericsson Canada Inc. for their financial support. Many thanks to Concordia University for providing all the resources, and for the opportunity it provided me to meet so many wonderful people. I will always be a proud Concordian.

I would like to thank all my colleagues from TSE lab for their support and help. I have leant a lot from each one of them and their research work. I am thankful to all my friends for all the memorable times I spent with them.

My family in Chennai, India and in Montreal, Canada is the source of energy for my every step in life. Words are not enough to thank everyone, and my life is incomplete without them.

Table of Contents

List of Figures

List of Tables

List of Acronyms

ACP – Algebra of Communicating Processes

CCS – Calculus of Communicating Systems

CPXe – Common Picture eXchange Environment

CSP – Communicating Sequential Processes

DSD – DIANE Service Description

FSM – Finite State Machine

HOL – Higher Order Logic

LOTOS – Language of Temporal Ordering Specifications

LTS – Labelled Transition System

MSC – Message Sequence Charts

MMS – Multimedia Messaging Service

NFSL – Non Functional Specification Language

OASIS – Organization for the Advancement of Structured Information Standards

OWL – Ontology Web Language

OWL-S – Ontology Web Language for Services

ITU – International Telecommunications Union

SAWSDL – Semantic Annotations for WSDL and XML Schema

SOA – Service Oriented Architecture

SWSO/L – Semantic Web Services Ontology/Language

TINA – Telecommunications Information Networking Architecture

UDDI – Universal Description Discovery and Integration

UML – Unified Modeling Language

USDL – Universal Service-semantics Description Language

WS – Web Service

WS-BPEL – Web Services Business Process Execution Language

WS-CDL – Web Services Choreography Description Language

WSCI – Web Service Choreography Interface

WSCL – Web Service Conversation Language

WSDL – Web Services Description Language

WSDL-S – Web Services Description Language with Semantics

WSLA – Web Service Level Agreements

WSMO/L – Web Services Modeling Ontology/Language

WSOL – Web Service Offering Language

W3C – World Wide Web Consortium

XML – Extensible Markup Language

Chapter 1

Introduction

1.1. Web Services and their Composition

Service Oriented Architecture (SOA) and Web services have gained widespread adaptation and usage as the next generation distributed computing and software development paradigm. Their popularity is credited to the numerous advantages they offer such as loose coupling, interoperability, coarse granularity, and platform neutrality. According to Organization for the Advancement of Structured Information Standards (OASIS), SOA is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains [1]. One approach to realize SOA is Web services.

A Web service (WS) is a software system designed to support interoperable machine-to-machine or application-to-application interactions over networks [2]. They have a WSDL interface [3], which can be published and found from Universal Description Discovery and Integration (UDDI) [4], and used by other entities using SOAP protocol [5]. The standard Web services architecture [2] proposed by W3C has three entities. *Web service requester* uses the Web services offered by the *Web service providers. Web service registry* allows these entities to publish and find the Web services.

Web services have functional, non-functional, behavioral, and semantic characteristics. The functionality of Web services is described using interfaces with input

and output parameters. The quality of Web services like performance is described by the non-functional specification usually given as cost, response time, availability, security, reliability, and reputation. The behavior states, how to interact with the Web services, in terms of sequence of input/output interactions, for instance. Semantics describe the meaning of services, and ontologies are usually used for the semantic description. The descriptions of Web services expose these different aspects of Web services that enable them to be published, found, and used by other Web services. They are also the key in composing the Web services into new ones.

Composition is a mechanism by which new services can be created by reusing available services. An example of a composed telecommunication Web service is a dating service. The semantics of this dating service is to create a call between "near-by" "willing" users with matching profiles; the service also sends pictures of partners using Multimedia Messaging Service (MMS). A presence Web service is used to determine the willingness of users. The user's location information is obtained from a location Web service. A MMS Web service is used to send the pictures using multimedia messaging. The actual call between the users is created by using a call control Web service.

1.2. Motivations for the Thesis

SOA and its main realization technology, Web services, are considered the most promising paradigms for distributed computing at present. Composition is the cornerstone in SOA (and Web services) for creating new services [6]. It is one of the important benefits of Web services, as it not only allows rapid creation of new Web services, but it also reuses existing services for such a service creation.

Web service composition should be handled and supported at the architectural level. The standard Web services architecture should enable and have the capabilities for performing composition of Web services. However, the standard Web services architecture does not support composition explicitly. By explicitly, we mean that it is not clear where composition is carried out, who performs it, and by what means.

In [7], it is identified that, the composed service should have syntactic, behavioral, and semantic conformance with the component services that make the service, and it should also be QOS-aware. Basically, it states that the composition should consider Web services having the functional, behavioral, semantic, and non-functional characteristics. In order to compose Web services having these four characteristics, they have to be described appropriately.

The different aspects of Web services have to be considered because they provide different information about the service, which influence the selection and composition of these services into composite ones. The functional aspect specifies what the service does. The semantic characteristic helps in understanding the meaning of the service offered. The details like how to interact with the service is provided by the service behavior. The quality of the service and the constraints in using the service is found in their non-functional characteristics.

Composing Web services by taking into account their conformance from functional, non-functional, behavioral, and semantic aspects and also describing all these aspects together is a challenging issue. The challenge comes from the fact that these aspects have to be used together consistently for description and manipulation of services by these techniques.

1.3. Objectives of the Thesis

In the standard Web services architecture, it is not possible for a requester to get services that are not available in the registry, which may be composed by providers in a demand-driven manner. Therefore, the first objective of the thesis is to extend the standard Web services architecture to support Web service composition. To perform Web service composition, new entities and interactions have to be defined and developed.

The second objective is to compose Web services by taking into account their functional, behavioral, semantic, and non-functional aspects together. It makes the composition problem much harder, because all these aspects have to be considered for matching, selection, and composition of services. In order to compose Web services with these characteristics the Web services have to be described suitably with these aspects. Therefore, the third objective of this thesis is to describe Web services with these four characteristics together.

Techniques for matching, selecting, and assembling the component services with these characteristics have to be developed, which is the last objective of the thesis. To summarize, the goal of this thesis is to develop architectures and techniques for composing Web services by taking into account their functional, non-functional, behavioral, and semantic characteristics.

1.4. Contributions of the Thesis

The contributions of the thesis are summarized subsequently with a reference to the publications generated from this work.

- **Critical analysis of the related architectures and techniques** – We have derived requirements for the architectures (business model) in supporting Web service composition, classified the existing architectures (business models), and critically analyzed them. We have derived requirements for the existing description techniques in supporting the four characteristics of Web services, proposed a classification of the existing techniques, and analyzed them. Similarly, we have derived requirements for the composition techniques, classified them, and analyzed them. We have also analyzed the related matchmaking techniques.
- **Architecture for Web service composition [8, 9]** – We propose an architecture (business model) for Web service composition by extending the standard Web services architecture (business model). We introduce three new roles (components) and six new interactions in the architecture. Web service composer, Web service composition registry, and third party Web service provider are the new roles. They interact with each other and with the other roles using the register, inform, get, give, put, and locate interactions, and also by using the old interactions: publish, find, and bind. The register and inform interactions are realized by extending the subscription API of UDDI V3. The get and give interactions are realized as a new API.
- **Framework for describing Web services [10]** – We propose a framework for describing Web services with functional, non-functional, behavioral, and semantic characteristics together. In this framework, SAWSDL, MSC, and NFSL are used for describing these various characteristics, and HOL is used as a semantic framework for integrating these languages. To do so, the concepts of SAWSDL and NFSL are directly formalized in HOL, and work on CSP-Prover [13] is reused for formalizing

MSC in HOL. Web services are described in our proposed architecture using the concepts and languages of this description framework.

- **Framework and techniques for composing Web services [11, 12]** – We propose a framework for composing Web services, which realizes the role of Web service composer in the architecture. It uses three techniques to compose Web services: matchmaking, categorization, and assembly. The techniques manipulate the Web services at HOL-level. The matchmaking technique discovers the primitive Web services that can make the requested composite service, taking into account all their characteristics. Primitive services are third party services that are typically used in the composition of composite services, rather than directly used by the requesters. The technique is formal, and it is based on Isabelle theorem prover [14]. The categorization technique helps the assembly technique in creating the composite service by categorizing the matched services. The assembly technique manipulates and selects the categorized services from different categories, uses BPEL to orchestrate the selected services, and selects the best-assembled service as the composite service.
- **Prototypes and case studies** – The roles and the interactions of the architecture are implemented. A prototype of the register and inform interactions as an extension to an existing UDDI is implemented. It realizes the role of extended Web service registry in the architecture. The Web service requester is implemented by extending its basic operation, and by implementing the get and give interactions. A prototype of the composition framework and its matchmaking, categorization, and assembly techniques has been developed. Moreover, performance evaluations of these

prototypes are performed. A case study to describe formally and integrate the different characteristics of a presence service is developed. To illustrate the concepts and the working of the architecture, composition framework, and the techniques, another case study to create a composite dating service is also developed. The reason for using two case studies is because the focus of the case studies is different, and it also allows explaining the concepts easily.

1.5. Thesis Organization

The rest of the thesis is organized as follows.

Chapter 2 provides the necessary background information for the thesis. Service oriented computing and architecture, the basic concepts of Web services, their different characteristics, their architecture, and concepts related to Web service description and composition are introduced in this chapter.

Chapter 3 presents the proposed architecture for Web service composition. The related architectures and models are evaluated. After that, our architecture, its roles and interactions are presented. Finally, the realization of the architecture is discussed.

The description framework is introduced in Chapter 4. Requirements for description techniques are derived. Related techniques are discussed, and evaluated based on the derived requirements. The basic concepts of the framework are then described, followed by an elaboration on the different languages of the framework. The integration related details, and the formalization of the languages concludes this chapter.

Chapter 5 presents the framework and techniques proposed for composing Web services with different characteristics. The related composition techniques, their

classification, and their analysis based on a set of derived requirements are presented first. After that, analysis of the related matchmaking techniques is presented. Subsequently, the composition framework is introduced. The matchmaking, categorization, and assembly techniques are described after that.

The prototypes and case studies are discussed in Chapter 6. The implementation of the architecture roles, the extended registry and the requester, are presented. The prototype of the composition framework is discussed after that. The concepts of the description framework are then illustrated using a presence Web service. Finally, the architecture, the composition framework, and its techniques are illustrated using a dating service.

Chapter 7 presents our conclusions, where an overview of the contributions is presented, and possible future work is discussed.

Chapter 2

Service Oriented Architecture and Web Services

2.1. Service Oriented Computing and Architecture

Service oriented computing paradigm utilizes services as the basic constructs to develop interoperable distributed applications. Services are autonomous, loosely-coupled, platform-independent entities that can be described, published, discovered, used, and composed [1, 7]. The key to this concept is the service oriented architecture. In SOA, service consumer seeks to satisfy its specific *needs* by using the *capabilities* offered as a service by the service provider.

Visibility, interaction, and effect are the three basic concepts in SOA [1]. The ability with which the service providers and service consumers are able to find each other and interact refers to the concept of visibility. Awareness, willingness, and reachability are the preconditions for visibility. Interaction is the activity of utilizing the offered capabilities to acquire some specific real world effect, which is essentially the actual result of using the service. Generally, a service consumer interacts with the service by sending and receiving messages.

The interaction with the service is made possible by the information model and the behavioral model of the service, which are part of the service description. Information model characterizes the structure and the semantics of the data and information that is exchanged with the service. The knowledge of the actions and their temporal

dependencies are characterized by the behavioral model. Service description promotes visibility by providing information that is required by the consumer to use or consider using a service. Service interface is the means by which the service description allows the capabilities of a service to be accessed. The constraints or conditions of using a service are referred as a policy, which is a part of the service description.

The key elements of SOA are identified as loose coupling, implementation neutrality, flexible configurability, persistence, coarse granularity, and teams (cooperation in problem solving) [6]. Service oriented methodology combines service discovery, selection, and engagement for software development. Service composition is the key for engineering SOA, where reuse is essential because services cut across inter-organizational boundaries. In SOA, to use services owned by others, they have to be put together (composed) appropriately to get a desired real world effect.

The SOA functionality can be divided into three planes (see Figure 2.1): service foundation, service composition, and service management [7, 15]. Semantics, non-functional properties, and quality of service characteristics cut across all the three planes. The service foundation layer, the bottommost layer, has a service oriented middleware backbone that realizes the SOA infrastructure. It connects heterogeneous components and systems, and also enables accessing services over different networks. The functionality and roles for aggregating multiple services into composite service is provided by the service composition layer. The service management layer is the top layer that manages and monitors the loosely coupled applications in the SOA. Service management spans a range of activities from installation and configuration to metrics collection and tuning.

One of the monitoring-related activities of this layer is to monitor the events and the information produced by the services and processes.

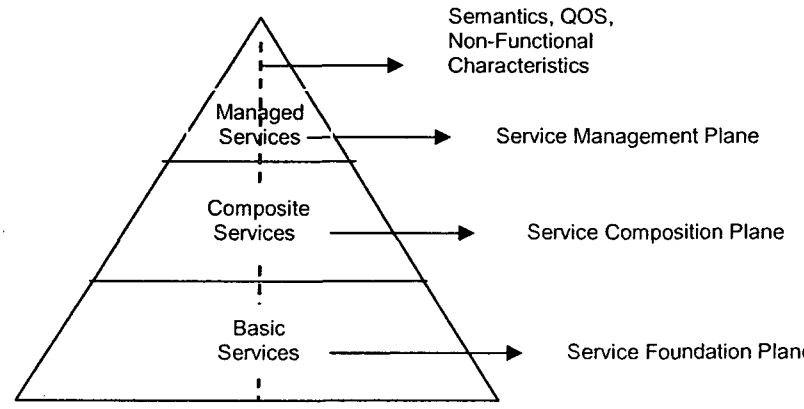


Figure 2.1 SOA: A Three Plane Perspective

2.2. Web Services

2.2.1. Basic Concepts of Web Services

Web service is a software system that has an interface described in a machine-processable format, particularly WSDL [3]. Other systems interact with the Web service in a manner prescribed by its description using SOAP [5] messages, typically conveyed through HTTP in conjunction with other Web-related standards [2]. Web services enable loose coupling of services, and it is based on open standards, which also makes it interoperable across heterogeneous domains. WSDL, SOAP, and UDDI [4] are the three core standards of Web services. SOAP is an XML-based protocol for communication between the Web services. WSDL is the XML-based language for describing the Web services. UDDI is the registry standard, where Web services can be published and found by other entities. More information on Web services and its standards can be found in [16] and [17].

2.2.2. Different Characteristics of Web Services

In this subsection, we introduce the different characteristics of Web services using a ring tone selling Web service as example.

2.2.2.1. Functional Aspects of Services

In the Web services domain, the functionality is described using interfaces with input and output parameters, and possibly preconditions and post conditions. For example, a ring tone selling Web service X, can have an interface (method) Y, which takes credit card number, credit card expiration date, ring tone id, and mobile number as input and produces a (un) success message as output. The service may have preconditions such as the credit card has enough credit and ring tone exists in the online store. The effects can be that after successful execution the credit card is charged and the ring tone is sent to the mobile number. The functional characteristics of the ring tone selling Web service is shown in Figure 2.2. The main feature of this service is selling ring tones with these functional parameters, but it can also allow other functionalities like login and searching.

Figure 2.2 Functional Characteristics of a Web Service

2.2.2.2. Non-Functional Aspects of Services

In the Web services domain, the most commonly used non-functional parameters are cost, response time (performance), availability, security, reliability, and reputation. Basically, the quality related details of the Web services is given by the non-functional aspects. For example the non-functional parameters of the ring tone selling service X can be cost of 2

USD, response time of 2 seconds, availability of 99.9%, and security to be high. Figure 2.3 shows the non-functional characteristics of the ring tone selling Web service.

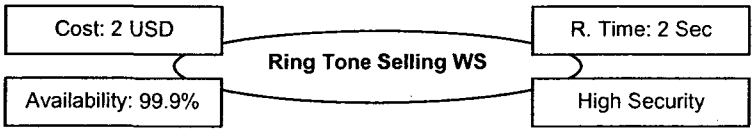


Figure 2.3. Non-Functional Characteristics of a Web Service

2.2.2.3. Behavioral Aspects of Services

The behavior states how to interact with the Web service. It is possible that the service allows entities to interact only in some specific order with the service. For example, the ring tone selling Web service X can have three methods allowing the three functionalities of login, searching, and buying, but it may not allow to search or buy without logging-in. The behavioral characteristics of the ring tone selling Web service is shown in Figure 2.4. This might not be the behavior that all the consumers are looking for, as some of them might be interested in just searching and knowing details about the ring tones without logging-in, and these set of consumers might not be interested in using this particular service.

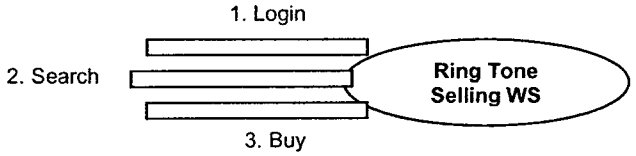


Figure 2.4. Behavioral Characteristics of a Web Service

2.2.2.4. Semantic Aspects of Services

Semantics describe the meaning of services. Ontologies are mainly used in the Web services domain for describing the semantics. Ontology is a type of data model that describes different concepts and their relationships in a specific domain. For example,

from the simple credit card ontology shown in Figure 2.5, it can be interpreted that visa card number is a credit card number because visa card is a type of credit card, and so is master card. It should be noted that these semantics are different from the semantics used in the formal methods community, where semantics refers mainly to the formal meaning of the behavioral expressions and data.

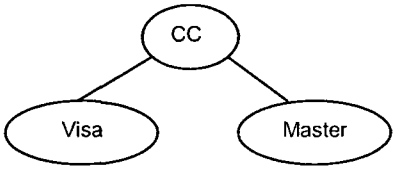


Figure 2.5. Simple Credit Card Ontology

We consider these four characteristics independently because Web services can have any of these aspects separately or they can have a different mix of these characteristics. For example, some Web services can have only functional characteristics, or some other Web services can have functional and semantic characteristics together, or further some Web services can have non-functional aspects along with the functional and semantic aspects, or they can also have all the four characteristics simultaneously. The different characteristics provide various key details about the Web services in discovering and using (composing) them.

2.2.3. Standard Web Services Architecture

World Wide Web Consortium (W3C) has proposed a standard Web services architecture [2]. The architecture is shown in Figure 2.6. The three components in the Web services architecture are: Web service requester, Web service provider, and Web service registry. The Web service requester is an entity that wishes to use the Web services that are owned or offered by the Web service provider. The Web service registry puts the providers and

the requesters in contact. The components use agents to communicate with each other, and use *publish*, *find*, and *bind* interactions. The *publish* interaction is used by the providers to publish Web services in the registry. Requesters use the *find* interaction to discover the Web services from the registry. The *bind* interaction is used by requesters to access the Web services offered by the providers.

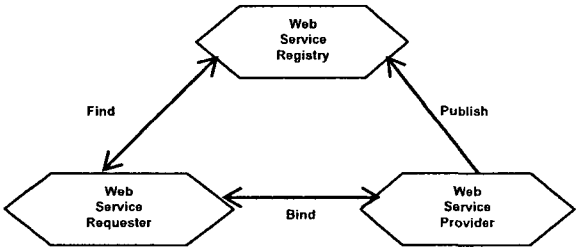


Figure 2.6. The Standard Web Services Architecture

We present an illustrative scenario to explain the components and their interactions in the Web services architecture. Suppose that there are three service providers: A, B, and C, where A provides a ring tone selling Web service. We assume that B provides a conferencing Web service, and C provides a dating Web service. All three service providers will use the publish interaction to publish their services in D, which we presume to be a service registry. The Web services architecture assumes that the providers and the requesters have the necessary information to interact with the registry. Now, if a service requester E wants a ring tone selling Web service it will use the find interaction to search D for the service. Consequently, the service requester E will discover the ring tone selling service provider A. Using the information obtained from registry D, the requester E will bind to the service offered by A.

A business model describes different parties involved in service provisioning and their relationships. Each party (also called as business role) has a set of responsibilities in

providing the services. Therefore, a business model can be considered as architecture, and vice versa. The business models are generally used as a starting point for standardization. The standard Web services architecture can also be considered as a business model as it clearly defines different parties and their interactions in Web services provisioning. Telecommunication and Web service business models have been proposed and are currently in use. The telecom business models and Web service business models that extend this standard business model for specific purposes are presented in the next chapter.

2.3. Describing and Composing Web Services

The description of Web services is important as the features of the Web services are only those that are exposed through their descriptions. Web services have to be described appropriately to be found, used by other Web services, and composed into composite Web services. WSDL is the *de-facto* standard description language in the Web services domain. WSDL mainly describes the functional aspects of Web services. SAWSDL [18], an extension for WSDL, allows annotating semantic information with WSDL by providing mechanisms to refer the semantic concepts in WSDL components. Other than WSDL there are also many languages that have been developed or reused for describing Web services. For example, semantic Web service languages like Ontology Web Language for Services (OWL-S) [19], Web Service Description Language with Semantics (WSDL-S) [20] have been developed, which uses ontologies as the basis for the service description. Moreover, formal techniques like Finite State Machines (FSM)

[21], UML models and diagrams (state charts and activity diagrams) [22], and Message Sequence Charts (MSC) [23] have been reused for describing Web services.

Web service composition is a technique by which new services can be created by reusing available services. A vast number of services can be created by composing services from different domains. Advanced telecom applications (services) can be composed by using basic telecom services like call control service, presence service, messaging service, sensor service, and location service. For example, an advanced conferencing service can be created by reusing and composing a call control WS, a presence WS, a sensor WS, a document WS, a printing WS, and a shipping WS. The semantic of the service is to create a conference between a group of participants using call control WS, when the participants are available and physically present in their office, which is identified by the presence and the sensor WS in that order. After the conference is ended, the minutes of the meeting is written using a document WS, which is printed and shipped to all the participants using the printing and shipping WS respectively.

Web service composition can be classified based on many parameters. For instance, based on when the composition occurs they can be classified as static and dynamic composition. In static composition, the composition is designed and the basic services to be composed are chosen at design time. The services to be composed are selected at runtime in the case of dynamic composition, where the composition logic is also created at runtime. Orchestration and choreography provides different views of the composition. Orchestration shows the interaction of a set of services from a single service viewpoint, while choreography presents the interaction and collaboration of services from a global perspective.

Web Services Business Process Execution Language (WS-BPEL) or BPEL for short, [24] is the most widely used solution for carrying out the static Web service composition as orchestrations, while Web Services Choreography Description Language (WS-CDL) [25] is popular for creating choreographies. Dynamic composition uses automatic and semi-automatic techniques based on artificial intelligence, workflows, and formal methods to solve the composition problem. For instance, SWORD [26] represents the Web services internally as rules, and it uses a rule-based expert system to automatically decide if a desired composite service can be composed from a list of available services. Many other solutions have been proposed for both static and dynamic composition. Reference [27], [28], and [29] presents some of the solutions for Web service composition.

In order to compose a Web service, the right primitive services that make the composite service have to be discovered. A matchmaking technique enables discovering these primitive Web services, and it is the core technique in any composition method. In the standard Web services architecture, UDDI provides the capabilities for matching the Web services. This keyword-based matching mechanism of UDDI mainly allows discovering services with functional aspects, which are generally described using WSDL. Many other mechanisms for discovering Web services with different characteristics have been proposed. For instance, [30] allows discovering Web services with semantic characteristics along with the functionalities, and the work has been implemented as an extension to UDDI in [31].

Chapter 3

A Business Model for Web Services Composition

3.1. Critical Review of Existing Business Models

In this section, the existing business models are analyzed with respect to a set of requirements we derive. The requirements are presented in the first subsection, followed by the existing business models. In the last subsection, the analysis is discussed.

3.1.1. The Derived Requirements

- **Static composition support** - The first requirement is that the business model should support static Web service composition. Static composition will give service providers the opportunity to create (design) composed services 'offline' and provide them for the requesters.

- **Dynamic composition support** - The next requirement is that dynamic Web service composition should also be supported. This gives a demand-based service provisioning aspect to Web services. Both static and dynamic compositions should be supported because they are functionally different and have different goals.

- **Allows requesting for services not found in the registry** - The third requirement stipulates that the business model should allow the requester to ask for services, describing the type of service they need, which they did not find in the registry. We assume that the service the requester needs has no pre-stored descriptions in the

registry, nor they have been published in the registry earlier. This requirement is necessary because service providers cannot create and store descriptions of all the potential services that the requesters might require in the registry. Some requested services can be created by composition, and only when they are requested.

- **Shows who does the composition and where** - Because Web service composition is an essential activity, clearly showing which role performs the composition and where the actual composition is done in the business model is the fourth requirement. Currently, for instance, some assume that the composition is done by the requesters, and others assume it to be the responsibility of the providers or the registry.
- **Allow new players** - The final requirement is that, since Web service composition has the potential for new business opportunities, the business model should allow extensions to accommodate new players. For example, there can be some entities that do not have the capability to compose services, but they should be able to contribute by providing (primitive) services for composition.

3.1.2. Existing Business Models

3.1.2.1. Telecommunication Business Models

Telecommunications Information Networking Architecture (TINA) [32] and Parlay [33] are the two main business models in the telecommunication domain. TINA is a set of specifications, developed between telecommunication and information technology industries, for defining a common architecture, and also for provisioning telecom and information services [34]. TINA uses the business model as a starting point for other specifications by defining the roles and the interfaces. A simplified business model

without the interactions of the business roles is shown in Figure 3.1. It defines five business roles: consumer, retailer, broker, third party service provider, and communications provider. The interactions that take place between the roles are called reference points, which consist of a set of interfaces.

The consumer is the service user (end-user) or an entity that has an agreement for the service usage (subscriber). It should be noted that the end-user is not necessarily the subscriber. For example, the enterprise can be the subscriber, and the employees the end-users. The retailer is the entity that provides the services, and it also has an agreement with subscriber for service usage. The broker provides information to find other parties and services.

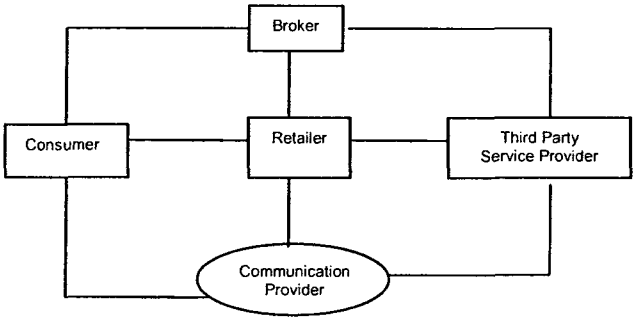


Figure 3.1. The TINA Business Model

A third party service provider supports retailers or other third parties with services. It has a business agreement with the retailer and no direct agreement with the subscriber. A specialized role is a special category of an existing business role, and it has a specific set of responsibilities that are relatively different from the responsibilities of its generalized role. In TINA, content provider is a specialized role. Content provider is a specialization of the third party service provider business role that is exclusively focused on contents generation (e.g., movie production companies). A communication provider owns or

manages the network. An entity can be in two different but related business roles at the same time. For more details, refer to [32].

Parlay is a set of open, technology-independent API's for accessing telecommunication capabilities and it simplifies the development of telecom-based applications [35]. The Parlay business model, inspired by the TINA business model, treats services as Service Capability Features (SCF). Parlay's business model is shown in Figure 3.2. It describes three business roles: client application, enterprise operator, and framework operator.

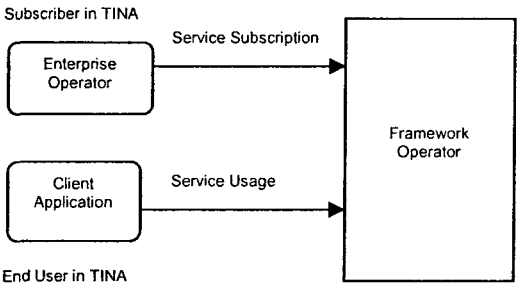


Figure 3.2. The Parlay Business Model

The client application consumes or uses Parlay's services (SCFs). In Parlay, functionalities offered by service capabilities such as call control and presence are called SCFs. The SCFs can be accessed using standard APIs. The enterprise operator is an entity that subscribes to the Parlay services by having a business agreement for service usage with the framework operator. The initial contact point for service discovery is the framework operator, which also handles subscriptions.

Third Generation Partnership Project (3GPP) uses the Parlay business model for providing service capability features for applications, where it is called Open Service Access (OSA) architecture [36]. In 3GPP, concepts like service capabilities and SCFs are

used as defined by Parlay. It can be noted that, in Figure 3.2 for each business role in Parlay, the corresponding business role in TINA is provided. Reference [33] provides more information on the Parlay business model.

3.1.2.2. Web Service Business Models

The standard Web services architecture [2, 37], which can be considered as the business model, was presented in the last chapter. The standard service oriented architecture [38] is similar to the standard Web services architecture [37], and this architecture can also be considered as a business model. The standard business model in the Web services and SOA domain has been extended for specific purposes.

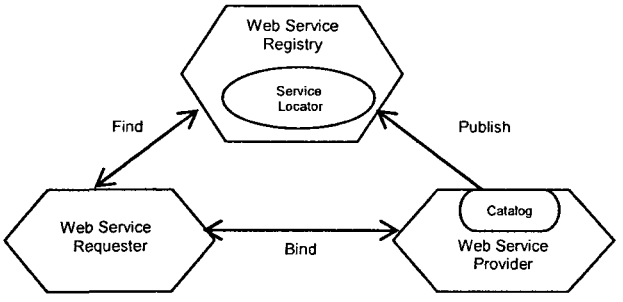


Figure 3.3. The CPXe Business Model

Common Picture eXchange environment (CPXe) [39] extends the standard Web services business model to provide and search for low-level (fine granular) information that cannot be provided by service registries like UDDI. Information like 'where to get a t-shirt printed with my vacation picture that is one kilometer from my home, and open at some specific hours' cannot be offered by UDDI, because its data structures does not support this type of fine-granular information.

CPXe is an initiative by the digital photography industry that leverages the Web services paradigm for the automation of manipulating, printing, and sharing digital

images. The business model of CPXe is shown in Figure 3.3. The business model describes two new entities: the service locator and the catalog. The service locator interacts with the registry (UDDI) and the catalogs to find specific services. The catalogs provide a standard way to obtain detailed information about products and services for locators and requesters. The locators and catalogs can be accessed using standard APIs.

The basic SOA model has been extended, for handling advanced functionalities like composition and management, which is called as xSOA [15, 38, and 40]. A simplified xSOA business model is shown in Figure 3.4. The role of service provider and service client (requester) are the same as standard Web services business model. For conceptual simplicity, it is assumed that providers or client can act as service registries (brokers). The model introduces three new entities: service aggregator, service operator, and market-maker.

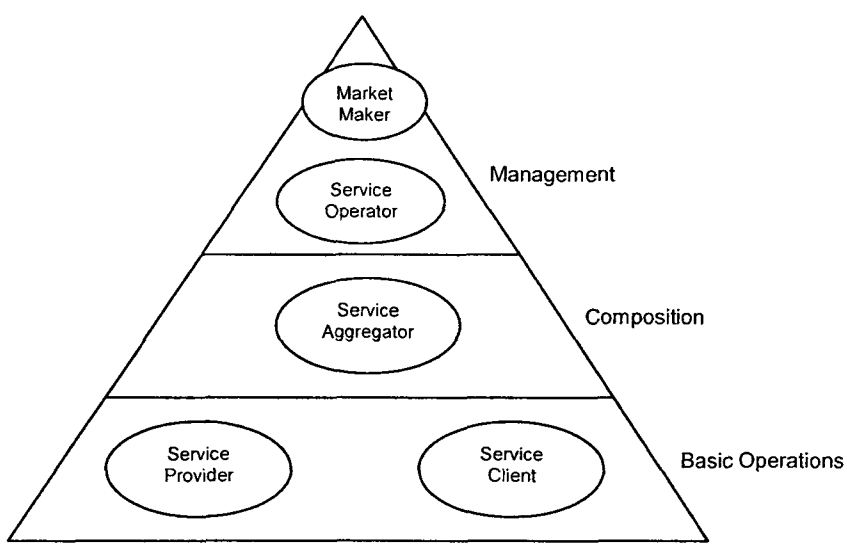


Figure 3.4. The xSOA Business Model

The service aggregator is responsible for aggregating/grouping services from different service providers into value added composite services. The important functions of aggregator are coordination, monitoring, and conformance. They can be considered as a

special type of Web service provider; however it can also act as service registry. Already composed services can be reused for creating further composite services.

Service operator is responsible for service (operation) management functionality, which aims at managing the service platform, deployment of services and applications, and monitoring the correctness and functionality of the aggregated services. It should be noted that either the service aggregator or the service requester can act as service operator. Market-maker supports the market management functionality and is responsible for the creation and maintenance of open service marketplace, which brings the suppliers and vendors together. For more details on this model the reader can refer to [38] and [40]

3.1.3. Analysis of the Business Models

Table 3.1. Comparative Analysis of Existing Business Models

Requirement	TINA	Parlay	WS and SOA	CPXe	xSOA
Static composition support	No	No	Yes	Yes	Yes
Dynamic composition support	No	No	No	No	No
Allows requests for services not found in the registry	No	No	No	No	No
Shows who does the composition and where	No	No	No	No	Yes
Allows new players	Yes	Yes	Yes	Yes	Yes

The business models are analyzed in Table 3.1 with respect to the five proposed requirements. It is clear from the table that telecommunication business models (TINA and Parlay) satisfy only one requirement. Although TINA uses the concept of composition, it does not support providing composed services statically or dynamically in

the business model, as it does not realize the potential of composition at the business model level. For the same reason, it does not show clearly who does the composition and where it is done. TINA also does not allow the requesting of services that are not found in the registry. On the other hand, Parlay does not support the concepts of composition and registry at all, so it does not support the first four requirements.

The standard Web services and SOA business models do not support dynamic service composition. However, it should be noted that service providers can offer statically composed services to requesters using the standard models. They neither allow non-existing services in the registry to be sought by requesters, nor do they show clearly who does the composition and where it is done. As CPXe does not introduce new entities or functionalities for handling composition, it clearly satisfies the same two requirements as the standard Web services model.

On the other hand, as xSOA introduces service aggregators to handle composition, so it satisfies the fourth requirement. However, it does not support the second requirement as it does not (explicitly) support dynamic composition. Moreover, it does not allow requesters to ask for services that cannot be found, so the third requirement is also not supported. To conclude, the current business models do not support all our requirements together.

3.2. The Proposed Business Model

In this section, the proposed business model along with the business roles are discussed first, and then the interactions of the business roles are presented.

3.2.1. The Business Roles

The proposed model [8, 9] is an extension of the standard Web services business model. The model is demand-driven, where services are composed based on their demand from the requesters. This TINA-inspired business model introduces three new roles and six new interactions. TINA's sound business model concepts, such as third party service provider and role specializations, are applicable to the telecommunication and information technology industries, specifically in the Web services domain.

We propose two specialized roles and one new role. The specialized roles are Web Service Composer and Web Service Composition Registry, and the new role is Third Party Web Service Provider. The proposed business model is shown in Figure 3.5, where the 3 new business roles and the 6 new interactions are shown using italics and dotted lines. It should be noted that, in Figure 3.5 the specialized roles are contained within their generalized roles.

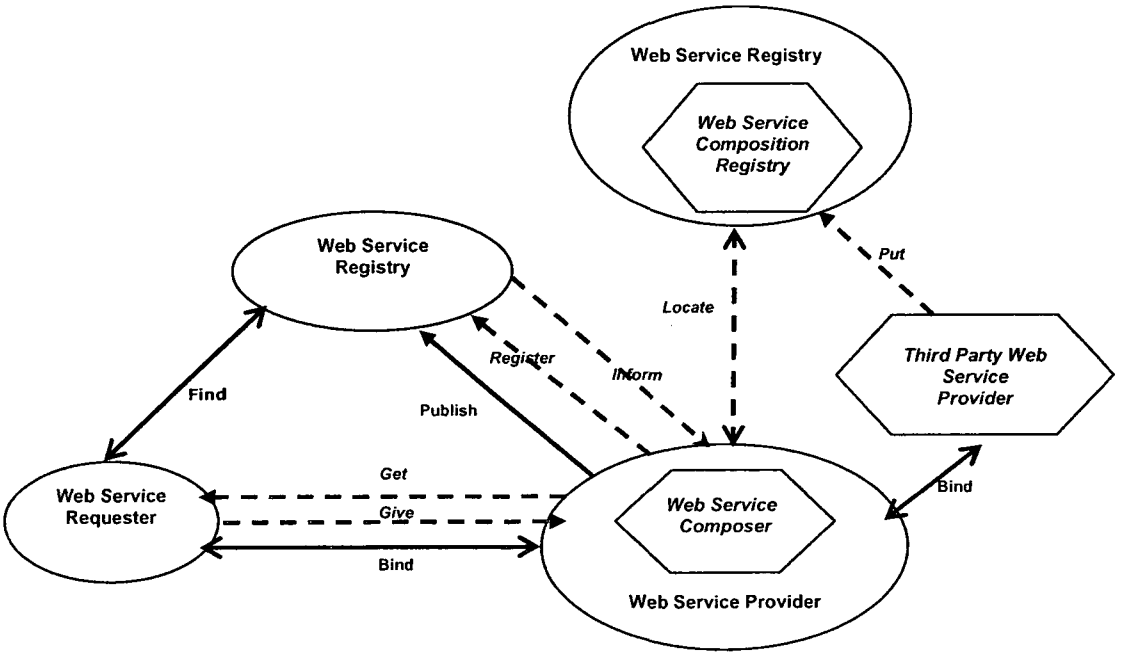


Figure 3.5. The Proposed Business Model for Web Service Composition

A particular specialization of the Web service provider business role is that of Web service composer, which is exclusively focused on composing services, both statically and dynamically. Composers do not necessarily own any services. It should be noted that this role is similar to the role of 'content provider' in the TINA business model.

We propose a new business role called the third party service provider, whose main aim is to support the composer business role with Web services that can be used to compose complex services. This role is similar to and inspired by the role of the 'third party service provider' in the TINA business model. In our model, the primitive services are designed, developed, and deployed by third parties.

Generally, third party services are primitive services that are typically used in the composition of composite services, rather than directly used by the requesters. However, third parties can also provide requester-usable services for composition. For example, a presence service can be provided as a third party service, which can also be used by requesters. It should be noted that any service that could be built with two or more primitive services can qualify as a composite service. For example, a presence-based call forwarding service, that can be composed of call forwarding and a presence service, can be called a composite service.

The main difference between the Web service provider and the third party service provider is that the third party service provider does not provide Web services for the requesters. Moreover, it cannot have any business agreements with the requester business role. However, third parties do provide Web services for composers and can have business agreements with them.

We also propose a specialization of the Web service registry business role as Web service composition registry, which is focused on allowing third parties to publish their Web services and on assisting composers to discover the published Web services. Similar to all business models, an entity can play several roles at the same time. For example, a third party service provider can also play the role of Web service provider.

3.2.2. Interactions among the Business Roles

The six new proposed interactions are: Register, Inform, Get, Give, Put, and Locate. We also reuse some of the interactions proposed in the standard Web services business model, without changing the semantics, to support some of the new business roles.

3.2.2.1. The Register and Inform Interactions

The main motivation behind the 'register' and 'inform' interactions is to alert the composer about the possibility of performing service composition. The composer uses the *register* interaction to tell the registry that it is a composer and that it would like to be informed about services that requesters are unable to find in the registry. This will give the composer an opportunity to create the services the requesters are looking for 'on-the-fly.' To provide notification to the composer about missing services that requesters are seeking but that are not present in the registry, the *inform* interaction is defined between the registry and the composer business role.

3.2.2.2. The Get and Give Interactions

The 'get' and 'give' interactions are used to transfer composition-specific information from the requester to the composer, in order to compose the exact services that the requester needs. After the composer receives notification about the missing services

sought by the requester(s) using the *inform* interaction, the composer has only limited information about the service the requester needs. This information is not sufficient to compose services, so the composer uses the *get* interaction to query the requester for the complete information needed to perform the composition. The requester then uses the *give* interaction to specify the details of the composition request.

3.2.2.3. The Put and Locate Interactions

The main motivation behind the 'put' and 'locate' interactions is to enable third party services to be published and found. Third party service provider publishes its Web services using the *put* interaction in the composition registry. Composers find the services of third party service providers from the composition registry using the *locate* interaction. It should be noted that these interactions have been developed to keep the semantics of *publish* and *find* interactions unchanged, although the new interactions are inspired by and similar to the old ones.

3.2.2.4. Reused Interactions from the Standard Business Model

The composer business role interacts with the registry and the requester business roles with standard *publish* and *bind* interactions, like the Web service provider. Composers bind to the third party service providers using the *bind* interaction, which is the same *bind* interaction (with the same semantics) used by the requester to interact with the provider.

3.2.2.5. Demand-Driven Composition Using the Interactions

The scenario of demand-driven Web service composition in the business model is explained using interactions. We assume that the requester wants a presence-based call forwarding service that is not in the registry. We discuss the set of steps that allows the requester to receive the composed service. We assume that one third party service

provider publishes a call forwarding service, and another third party publishes a presence service in the Web service composition registry using the *put* interaction. We also assume that the composer uses the *register* interaction to register with the registry.

When the requester tries to find the presence-based call forwarding service from the registry, it will get a response 'no available services', but the requester can provide an endpoint where the composer(s) can contact it. Subsequently, the registry will use the *inform* interaction to notify the composer about the requester's endpoint and its need for the presence-based call forwarding service. At this stage, the composer only knows that a requester needs a presence-based call forwarding service, but this information is not sufficient to compose. Therefore, the composer uses the *get* interaction to communicate with the requester, which provides the complete service request using *give* interaction.

The composer now knows the exact specification of the presence-based call forwarding service, so the composer uses its techniques to compose it by using the third party services that can be found using the *locate* interaction. Finally, the composed service is provided to the requester.

3.3. Realization of the Business Model

3.3.1. Interactions Realization

In this section, the realization of the business model interactions is discussed.

3.3.1.1. Realization of Register and Inform Interactions

The register and inform interactions are realized by extending the subscription API of UDDI V3. UDDI Version 3 (V3) [4] is the latest version of UDDI with some significant

new features like subscription using subscription API. Generally, the subscription API is used for monitoring and for notification of events that occur in the UDDI. The API supports both asynchronous notification and synchronous change tracking. Subscribers use the subscription API to receive notification about changes, additions, and deletions to the UDDI data structures. This API needs to be extended, because the composers must be notified about services that requesters need, but are not available in UDDI.

We propose three new methods to the subscription API for realizing the register and inform interactions. Two methods support asynchronous communication and the other supports synchronous mode of communication. The methods are quite straightforward, and they are inspired from the subscription API methods.

1. subscribe_seekedServices – To subscribe, unsubscribe, and resubscribe for the data that the requesters are seeking and that are not found in the UDDI. It realizes the register interaction in asynchronous communication mode. The expiration value parameter of the method will determine whether the subscription request is for a new subscription, to renew (modify) an old subscription, or to delete an existing subscription. The method may return a structure with subscription-related values, depending on which values are changed from the requested values. If all the requested values are accepted, then an empty message is returned as the response.

2. notify_seekedServices – To notify composers with information, such as services that requesters are unable to find in the registry. It realizes the inform interaction in asynchronous communication mode. The notifications are provided using the template specified in the *subscribe_seekedServices* method. A successful notification will return an empty message.

3. get_seekedServices - To synchronously retrieve the services that the requesters are looking for and which are not found in the UDDI. This method realizes both register and inform interactions in synchronous communication mode. The message will return a structure that will have the information required by the composer to communicate with the requesters, along with the details of the requester queries that did not retrieve a result.

We also propose to extend the *find_service* method of the Inquiry API that is used by the requesters to find services from the UDDI. We extend its business logic to update the database with service requests that do not produce any results. We propose adding a new argument 'requesterAddress' to the *find_service* method, which will be used by requesters to specify an endpoint where potential composers can communicate with them. Similarly, we add a new attribute 'note' to the response returned to the requester for the 'unavailable-service' query. This attribute will provide information about the composers that may contact the requesters to help them to receive the requested service.

3.3.1.2. Realization of Get and Give Interactions

The Get and Give interactions are realized as a new API, called as Get-Give API. This API is used for all composition-related communication between the requesters and the composers. It has two methods, and they support asynchronous communication.

1. get_compositionRequest – To request for the information related to the composition request and to provide details about the composer. This method realizes the get interaction, and has three parameters. The id parameter identifies each request and it is used for synchronizing the get and its respective give message. The address of the composer where it can be contacted is provided in the address parameter. The details

about the composer are provided in the third parameter, which enables the requester to decide if the composition request should be given to the composer.

2. give_compositionRequest - To provide the detailed composition request to the composer. This method realizes the give interaction, and has two parameters. The first parameter is used for identification, and it synchronizes the give message with its corresponding get message. The other parameter provides the details about the composition request that is needed for creating the composite service, which would satisfy the requester's need.

It should be noted that the address of the requester is available to the composer, which is informed by the registry. This information is provided to the registry by the requester in the extended *find_service* method (presented in the last subsection).

3.3.2. Roles Realization

In the standard Web services architecture, the registry is realized as UDDI. On the other hand, there is no particular realization scheme for the requester and the provider. However, in order to access the UDDI the requester should support Inquiry API, and should use SOAP for communicating with the UDDI and the providers. Similarly, the provider should support the Publish API of UDDI, and should also use SOAP for communication.

In the proposed architecture, we realize the registry as an extended UDDI that supports the register and inform interactions in the form of extended subscription API. Moreover, the extended requester realizes the extended *find_service* method of the Inquiry API and the Get-Give API. The composer is realized as composition framework, which is discussed in Chapter 5.

Chapter 4

A Framework for Describing Web Services with Different Characteristics

4.1. Critical Review of Existing Description Techniques

In this section, we initially derive the requirements for analysis. We then classify the existing description techniques, briefly present the techniques, and critically evaluate them with respect to the set of derived requirements.

4.1.1. The Derived Requirements

Descriptions enable Web services to be discovered and composed, as mentioned before. In order to compose Web services with functional, behavioral, non-functional, and semantic characteristics, these aspects should be appropriately described. Consequently, support for the different aspects is the first four requirements.

- The technique should describe functional aspects of the services
- The technique should describe behavioral aspects of the services
- The technique should describe non-functional aspects of the services
- The technique should describe domain semantics for functionalities

The other five requirements are

- The technique should not use domain semantics for non-functionalities
- The technique should allow describing services by requesters and third parties

- The technique should try to reuse existing methods, wherever possible
- The technique should align with existing Web services standards
- The technique should be simple and easy to use

The fifth requirement is important in order to balance between expressiveness and computation, because although semantics allows good expressivity it is computationally demanding. The last four requirements are generic requirements.

4.1.2 Description Techniques: State of the Art

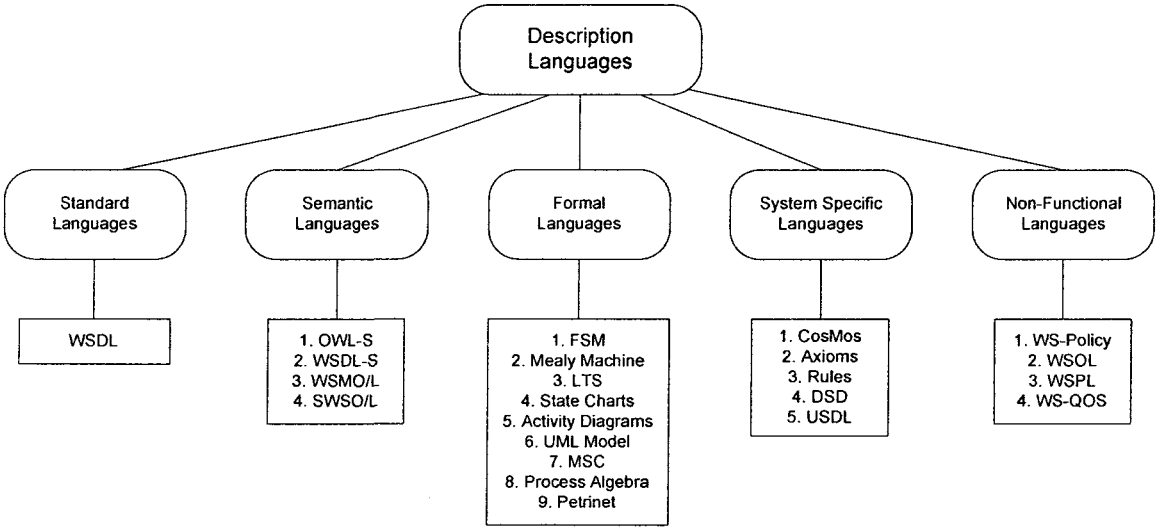


Figure 4.1. The Proposed Classification of the Web Service Description Techniques

We propose a classification to comparatively analyze the different description techniques/languages as shown in Figure 4.1. The classification is based on the core characteristics of the languages and on the standardization perspective. The main motivation for the classification is to understand the languages and also to aid in analysis. The description languages are classified into five types: standard languages, semantic

languages, formal languages, system specific languages, and non-functional languages, which are subsequently presented.

4.1.2.1. Standard Languages

Standard languages are developed by Web services standards developing bodies like W3C and OASIS, and specifically for describing Web services. The only language in this category is WSDL [3], which is currently the *de-facto* standard description language in the Web services domain. WSDL describes Web services abstractly as a set of endpoints that has a set of operations described using XML schema types, which is concretely bounded using network protocols and message formats. It supports semantic description by using SAWSDL [18], which defines a set of extensions to describe semantics with WSDL.

4.1.2.2. Semantic Languages

The semantic-based languages fall in the second category, which mostly use ontologies for describing the domain semantics. W3C has a Semantic Web Services Interest Group (SWSIG) [41], which provides an open forum for discussion on semantic-based Web service description languages. The semantic Web service languages in our classification that are also submitted to W3C's SWSIG are OWL-S [19], WSDL-S [20], Web Services Modeling Ontology/Language (WSMO/L) [42, 43], and Semantic Web Service Ontology/Language (SWSO/L) [44, 45].

OWL-S is an ontology of services that has three parts: service profile, service process, and service grounding. Service profile describes what the service does, and it is used for advertising and discovery purposes. Service process describes how the service works, and it gives description of service operations using various constructs. Service

grounding describes how to access the service, and it provides information on interaction with services. WSDL-S allows annotating semantic information with the WSDL documents by defining a set of WSDL extension elements and attributes. It allows specifying preconditions, effects, and semantic concept referencing of operations, input messages, and output messages.

WSMO provides a conceptual model for the semantic markup for Web services and it is made up of four elements: ontologies, Web services, goals, and mediators. WSML is a language that provides the formal syntax and semantics for the WSMO and is based on different logic formalisms like description logic, first-order logic, and logic programming. SWSO/L is different but complimentary to WSMO/L; SWSO/L enables description of process orchestration using first order logic ontology, while WSMO/L focuses on Web services choreography description using guarded transition rules. SWSO provides a conceptual model for Web services description, and is expressed in first order logic ontology form and in rules ontology form. SWSL allows specification of SWSO and individual Web services, and it consists of first-order logic language and rules-based language.

4.1.2.3. Formal Languages

The next category is formal languages. The list includes Finite State Machines (FSM) [46], Mealy Machines (MM) [47], Colombo Framework's [48] that combines FSM and MM, Labelled Transition Systems (LTS) [49], Unified Modeling Language (UML) diagrams like state charts [50] and activity diagrams [51], UML models [52], Message Sequence Charts (MSC) [53], Process Algebra [54, 55, 56], and Petri Net [57].

Berardi *et al* [46] developed a formal framework for theoretical investigation of service composition, where the services are represented as FSM and the user requirements are also captured using a FSM. FSM is an abstract machine with a finite number of states and transitions. Fu *et al* developed techniques for validation of asynchronously communicating Web services, where the Web services are represented using mealy machines [47]. A mealy machine is a FSM with input and output, where the output depends on the input and the current state. The Colombo framework [48] combines four aspects of Web services: message passing based on MM, behavior of Web services using FSM, a world (database) schema representing the 'real world', and atomic processes inspired from OWL-S. Colombo proposes novel techniques for synthesizing service compositions described using these different aspects.

Pathak *et al* represents Web services and composes them as LTS [49], where the service client and the developer model the Web services as UML state charts (and BPEL), and the system automatically maps/translates them to LTS. Generally, LTS is an abstract machine with a set of states and transitions between them and states need not be necessarily finite. However, the LTS used here has a finite number of states and transitions that are annotated with actions or guards (preconditions for actions).

The Self-Serv system [50] provides middleware and tool for composition and peer-to-peer orchestration of Web services, where the composite services are represented using UML state charts. Medjahed *et al* developed a novel approach for automatic composition of Web services that uses a concept of community to cluster Web services, where each community is defined as an instance of community ontology [51]. The system uses composition specification language, which is an extension of the activity diagrams

for composition request specification. An activity diagram represents a workflow of a system and is a type of UML diagram.

Jaeger and Gronmo proposed a model driven (semantic) Web service composition system [52] by creating the services as UML models, where they use four phases (modeling, discovery, selection, and deployment) for carrying out the composition. The approach by Foster *et al* uses MSC for designing the Web service composition, and then the MSC specifications are synthesized into state transition system [53]. The implementation model is developed using BPEL and mapped into a finite state process.

Process algebra's are generally used to describe formally concurrent and communicating systems. Process algebra's like Communicating Sequential Processes (CSP) [58], Calculus of Communicating Systems (CCS), Algebra of Communicating Processes (ACP), Language of Temporal Ordering Specification (LOTOS) [59], and Pi-calculus exist. Using process algebra for describing Web services have been proposed in [54], [55], and [56], and specifically [55] and [56] maps BPEL into LOTOS and then manipulates LOTOS. It is also important to mention about usage of process algebra for BPEL semantics, although here it is not used for service description. In [60], Pi-calculus based semantics for WS-BPEL 2.0 [61] is provided.

Web services are modeled as Petri nets in [57] for the purpose of composition and analysis. Petri net is a directed and connected bipartite graph, where nodes represent transitions or places, and directed arcs run between transitions and places. Petri net based semantics have been proposed for BPEL and OWL-S. In [62], Petri net based semantics is proposed for BPEL; and [63] provides Petri net based semantics for DAML-S for the purpose of simulation, verification, and validation.

4.1.2.4. System Specific Languages

The fourth set of languages is specifically developed for describing Web services by some (composition) systems that include Component Service Model with Semantics (CoSMoS) [64], axioms [65], rules [26], DIANE Service Description (DSD) [66], and Universal Service-semantics Description Language (USDL) [67]. CoSMoS is developed specifically for Web service composition by Fujii and Suda, which is an abstract component model that uses graph representation to integrate functional and semantic aspects of a component [64]. CoSMoS can be used with different languages like WSDL, XML, and Resource Description Framework (RDF) as it is just an abstract representation.

Rao *et al* uses linear logic based program synthesis for tacking the Web service composition problem [65], where the services are externally represented using OWL-S and internally they are represented using linear logic axioms and proofs. Linear logic allows capturing concurrent features of Web services formally. SWORD [26] is a Web service composition system that represents the services internally as rules, which take a set of inputs and produce a set of outputs. In SWORD, the individual services are defined initially using inputs and outputs as an entity-relationship based 'world model'.

DSD [66] is a service description language based on its own lightweight ontology. The basis for the ontology is the standard object orientation along with four new elements: operational elements, aggregating elements, selecting elements, and rating elements. In DSD, services are mainly described by their real-world effects, which are expressed with operational elements. Aggregating elements capture a set of similar effects. The requester chooses the effects that are applicable in the particular context using selecting elements. Rating elements are to model the preferences of the requesters.

USDL [67] is a service description language that attempts to capture the semantics of Web services in a universal manner, and it relies on a universal ontology. USDL is complimentary to OWL-S as it specifies the semantics of atomic services, which is not specified as a part of OWL-S. Similar to WSDL, USDL describes services using the messages and portType, and these concepts are mapped to the concepts in the ontology. It should be noted that extra semantics are provided in terms of real world effects.

4.1.2.5. Non-Functional Languages

Non-functional languages mainly describe the non-functional aspects, and the constraints in using Web services. They are the last type of languages, which consists of languages like WS-Policy [68], Web Service Level Agreement (WSLA) Language [69], Web Service Offering Language (WSOL) [70], and WS-QOS [71]. WS-Policy 1.5, a W3C recommendation, is a framework for describing policies of entities in a Web service based system [68]. It provides a model, necessary syntax, and basic constructs for the description of a wide range of requirements, capabilities, and constraints.

WSLA language [69] is based on XML, which defines the agreement between service providers and consumers. It mainly describes the obligations of a service provider in performing a service based on the agreement. The SLA also provides details about the measures to be taken when the service does not fulfill the agreement. Moreover, the language also allows for the management and the monitoring of services by third parties.

WSOL is a XML-based language that allows specifying constraints, management-related statements, and classes of service of Web services [70]. Class of service of a Web service means a distinct variation in the service and its QOS. Service offering is a formal description of a class of service, and it defines various constraints of functional or non-

functional type. WS-QOS [71] comprehensively supports QOS integration in Web services, allows selecting services based on QOS requirements, and also enables real-time QOS monitoring of services by providing an instant QOS feedback. It introduces a new entity called as Web service broker in the standard architecture, and the client interacts with the broker to get services with the required QOS. It also defines a XML QOS schema that can be used by the service providers and the consumers to define the QOS requirements and offers, and also provides WS-QOS ontology for defining new parameters.

4.1.2.6. Composition Languages

It is also important to mention the composition languages, which do not describe the features of a single service but a (combined) set of services. The composition languages developed by the Web service community are WS-BPEL [24, 61], WS-CDL [25], Web Service Choreography Interface (WSCI) [72], and Web Service Conversation Language (WSCL) [73]. WS-BPEL is a XML-based orchestration language that provides formal specification of business processes and business interaction protocols [61]. It describes the behavior of abstract and executable processes using different constructs. It extends the Web service interaction model to support business transactions.

WS-CDL [24] is a XML-based choreography language, which from a global perspective describes peer-to-peer collaboration between participants by defining their complementary and common observable behavior. WSCI is a XML based interface description language that allows specifying the flow of messages between different Web services [72]. WSCL [73] allows describing business level conversation of Web services and can be used along with other languages like WSDL.

4.1.3 Analysis of Existing Description Techniques

Table 4.1. Comparative Analysis of the Description Techniques

Requirement	WSDL	Semantic Languages	Formal Languages	System Specific Languages	Non-Functional Languages
Allows functional aspects of services	Yes	Yes	Yes	Yes	No, partially by WS-Policy
Allows non functional aspects of services	Yes	No, generally does not support	No for FSM, MM, & Colombo	Yes	Yes, every language supports
Allows behavioral aspects of services	No	Yes except WSDL-S	Yes	No, generally no behavior	No
Semantic aspects for functionalities	No	Yes	No, none of them uses	No	No
No-semantics for non functionalities	Yes	No, tries using domain semantics	Yes	Yes	No except for WS-QoS
Allows description by consumers & 3rd parties	Yes	Yes	Yes	Yes	Yes
Reuse existing methods	No, developed from scratch	No, except WSDL-S	Yes, all are well known methods	Yes, where ever possible	No, developed from scratch

Align to WS standards	Yes	No, except WSDL-S	No	No	No, except for WS-Policy
Simple & easy to use	Yes	Partially yes, else needs special tools	No for FSM, MM, & Colombo	No	No, needs special tools

The comparison of the different description techniques is presented in Table 4.1. It is clear from the table that WSDL supports only functional and non-functional aspects of the services; it does not support behavior description, functional semantics, and non-functional semantics. However, SAWSDL supports semantic description. It can be used by requesters and third parties alike as it is simple and easy to use, and it is also the basic description standard. However, it is a mechanism developed from scratch so it does not reuse any method.

From the table it is apparent that except non-functional languages all the others support functional description of services; the reason is that these languages are specifically developed for non-functional (and constraint) description of services. WS-Policy is the only language that can be used for functional description, but special extensions are needed. FSM, MM and Colombo do not support non-functional description of services, and so are the semantic languages. The behavior aspects are mostly supported by semantic languages (except WSDL-S) and formal languages, but it is not supported by system specific languages. Semantic aspects are supported only by semantic languages, but generally it also tries to use semantics for non-functionalities. Similarly, WS-QOS also supports semantics only for non-functional description.

All the languages can generally be used by requesters and third parties. Formal languages and system specific languages uses mostly well known existing methods, but the others are mostly developed from scratch. All the languages do not align with Web service standards, but semantic languages are getting significance and some of them might eventually become W3C standards. Almost all the languages are not simple and easy to use, mostly special tools are needed to work with them. The analysis concludes that none of the existing description languages support all our derived requirements. It is particularly interesting to note that none of the mechanisms allow functional, non-functional, behavioral, and semantic service characteristics to be described together.

4.2. The Proposed Description Framework

In order to propose a novel description technique that allows describing all the different Web service characteristics, there are two options: revolutionary approach, and evolutionary approach. In revolutionary approach, a new technique is developed from scratch. On the other hand, evolutionary approach builds a new technique by reusing existing languages/techniques. We have chosen the evolutionary approach.

The basic idea behind our proposal is that instead of extending a language that provides description of all the aspects together, we propose a framework that uses different languages that provide these descriptions separately and integrate them together for service description. The main reason behind this idea is to exploit the powerfulness and expressiveness of different languages, and to overcome the shortcomings of each individual language. For example, SAWSDL is a powerful language for functional and semantic description of Web services, but it cannot describe behaviors.

Moreover, a single language cannot be extended to support the four Web service characteristics together, as this will make the language complex to develop, understand, and use. Integration is the key to reason and manipulate the different aspects together, because it makes the manipulations easier in a single domain. In addition, existing tools/techniques in the semantic domain can be used for such manipulations.

The concept of using and integrating different languages for describing different aspects of complex systems exists in the formal methods literature. For example, Object-Z [74] and CSP have been used and integrated in [75] for describing concurrent systems, where Object-Z is used for describing the complex data structures and CSP is used for modeling the interactions between the processes. Similarly, in [76], CSP, Object-Z, and duration calculus [77] are combined for the specification of process, data, and time.

The author in [78] identifies three cases of language integration: data with data (example: Z [79] and B [80]), data with process (example: Object-Z and CSP), and process with process (example: LOTOS [59]). However, the concept of using different languages for describing different aspects of Web services, and subsequently integrating them together in a common semantic domain for manipulation purposes, has not been explored in Web services domain so far.

4.3. Using Different Languages for Different Characteristics

4.3.1. Using SAWSDL for Functional and Semantic Description

The basic issue is which languages/mechanisms can be combined for describing the different Web service characteristics. Choosing a semantic description language as a part

of the framework is a good alternative as it handles both the functional and semantic aspects. We propose to use SAWSDL [18] for describing the functional and semantic characteristics of Web services. We chose SAWSDL because it is a W3C standard, simple, lightweight, evolutionary, and semantic representation language agnostic. SAWSDL allows annotating semantic information with WSDL documents by providing mechanisms to refer the semantic concepts in WSDL components. The semantic information present in the semantic models is machine interpretable information that models knowledge in some domain. Ontology is an example of the semantic model, but other models can also be used. It is important to note that as SAWSDL is based on and extension of WSDL, using SAWSDL in our framework means also indirectly using WSDL as a part of our framework.

SAWSDL, like WSDL, has abstract and concrete description parts. The abstract part specifies the data types, messages, and operations, while the concrete part specifies the binding, endpoint, and service elements. In the case of our proposal, the Web service requester specifies the functionalities (and semantics) that are required by using only the abstract part of the SAWSDL file. On the other hand, the Web service providers (and third parties) must specify the functionalities (and semantics) using both the abstract part and the implementation (concrete) part of the SAWSDL file. We assume that the requesters and the providers (third parties) share common ontologies for service description. For more information on SAWSDL, refer to [18].

4.3.2. Using MSC for Behavioral Description

For describing the behavior of Web services, formal languages are good candidates. There exist a wide variety of formal notations for behavior description. Therefore, we

define a certain number of constraints or requirements a candidate language should satisfy. First, the language should be completely formal. Second, it should be simple and easy-to-use. Third, the language should be scenario-based. The third constraint is necessary because the behavior description is used by both the requesters and the providers, moreover, scenario-based descriptions helps in easier understanding.

Message sequence charts [23] satisfy all the requirements, so we propose to use MSCs for describing the behavior of Web services. MSC is an International Telecommunication Union Telecommunication standardization sector (ITU-T) standardized formal language used for describing interactions between entities in a system. It is similar to sequence diagrams of UML [22]. MSCs are made of bMSC and HMSC. The bMSC usually describes only partial behaviors or a few scenarios of a system. HMSC is a directed graph that is composed of bMSCs and/or other HMSCs in recursive manner using operators. MSC is a powerful language with constructs for timers, loops, conditions, optional and exceptional system behaviors, and more. Reference [23] gives more information on MSC. The requesters use MSC for specifying their interaction sequences with the required (composite) Web services. Similarly, the providers (and third parties) use MSC to describe their Web services behavioral interactions with other services and requesters.

4.3.3. Using NFSL for Non-Functional Description

We have six requirements to be satisfied by the non-functional description language. The first requirement is that it should not use domain semantics (ontologies) for description. This requirement stems from Section 4.1.1, in order to balance between expressiveness and computation. The next requirement is that the language should be XML-based, so

that it is extensible and aligns with other Web service standards. The third requirement is that the language should address specifically the basic non-functional parameters of Web services: performance, cost, availability, security, reliability, and reputation.

The next requirement is that the language should be 'off-the-shelf,' so that it is directly usable without any need for modifications. The fifth requirement is that the language should allow 'per Web service' non-functional parameter specification, and not 'per Web service operation' non-functional specification. This will enable easier specification of the non-functional properties. The last requirement is that the language should be simple and easy-to-use. The analysis of existing NF languages is presented in Table 4.2.

Table 4.2. Comparative Analysis of the Non-Functional Description Languages

Requirement	WS-Policy [68]	WSLA [69]	WSOL[70]	WS-QOS [71]
No ontology usage	Yes	Yes	Yes	No
XML-based	Yes	Yes	Yes	No
Specifically address basic NF parameters	No	No	No	Yes
Off-the-shelf	No	No	No	No
Per WS NF description	Yes	Yes	Yes	Yes
Simple and easy-to-use	Yes	Yes	No	No

It is clear from the table that none of the existing languages satisfies all our requirements. Therefore, we propose to use a novel language for describing the non-functional aspects of Web services. We call it the Non Functional Specification Language (NFSL). It is a

simple XML-based language. The non functional parameters covered in the language are cost, response time (performance), availability, security, reliability, and reputation. It can be easily extended later, if needed.

In NFSL, cost, response time, and availability are expressed in numeric values with metrics. The cost of the service is specified using United States Dollar (USD). Response time is specified as milliseconds (ms). Percentage (%) is used for specifying the availability. Security, reliability, and reputation are specified in terms of four scales: high, medium, low, and none. The semantics of the different scales of security and reliability are defined based on the technologies supported. For instance, high security in NFSL means when both WS-Security [81] and SAML [82] are supported. The security is interpreted as medium when only WS-Security is used, and support for only SAML means low security. When both are not supported, then the security is none. The semantics of the four scales of reputation is based on the (user) rating of the service. It should be noted that the metrics and the semantics of the scales can be changed (by the user). For example, the cost metric can be changed to Canadian Dollars or Euros.

```
<? Xml version="1.0" encoding="utf-8"?>
<NFSL>
<WSName>                </WSName>
<Cost>                  </Cost>
<ResponseTime>          </ResponseTime>
<Availability>          </Availability>
<Security>              </Security>
<Reliability>           </Reliability>
<Reputation>            </Reputation>
</NFSL>
```

Figure 4.2. Template of the Proposed Non Functional Specification Language

Figure 4.2 shows the XML template used for specifying the NFSL specification of the services. The tags <NFSL> and </NFSL> marks the beginning and end of the definitions.

The name of the Web service (<WSName> tag) is used to identify its non-functional description. It should be noted this name is same as the name used in the SAWSDL description that identifies the particular service. The different non-functional values are specified in-between their appropriate tabs. For example, <Cost> 2 USD </Cost> specifies that the particular Web service has a cost of 2 USD.

4.4. Integrating the Different Languages

In this section, the integration approach is discussed in the first subsection, and then the mapping of MSC concepts to HOL is presented. The third subsection describes the formalization of SAWSDL in HOL. The formalization of NFSL in HOL is discussed in the last subsection.

4.4.1. The Integration Approach

After selecting three languages for describing the four aspects of Web services, the next issue is which formalism can be used for integrating the languages and how? The languages need to be integrated because the Web services described using these languages have to be manipulated for matching and other purposes with all the characteristics. The author in [78] discusses some issues and proposes solutions for the integration of description techniques/languages. The author further shows that syntax and semantic level compatibility are the main issues in such integrations. The semantic basis can be viewed as a common compatibility level for the different techniques, where the reasoning can be done.

A similar approach is followed in our case, where a common semantic basis is used to reason about the Web services. Moreover, [78] also points out Finite State

Machines, First Order Logic, and Higher Order Logic [83] as possible candidates for such integration. These formalisms are the possible integration domains in our case also. FSM is not a good candidate because it cannot describe data in SAWSDL and NFSL well as it describes the behavior. We believe that MSC and the XML-based languages, SAWSDL and NFSL, can have common semantics in the logic domain. However, FOL is not as expressive as HOL, and specifically, it would not be very suitable in expressing complex MSC behaviors. On the other hand, HOL provides the flexibility and the necessary expressiveness for expressing all the complex behavior and data. Therefore, we use HOL as the semantic domain for integration as shown in Figure 4.3. HOL allows variables to range over functions and predicates. There are different kinds of HOL depending on the type system they provide for the use of functions and predicates.

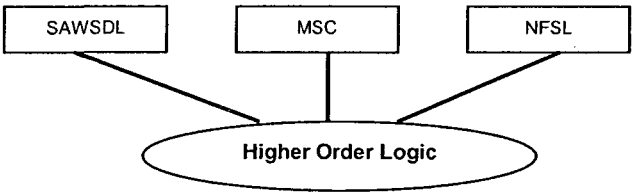


Figure 4.3. Conceptual Integration

The subsequent issue to be handled is that how can MSC, SAWSDL, and NFSL be formalized in HOL. We provide HOL semantics to MSC using a two-step mapping. In the first step, MSC is given process algebra semantics, which is then mapped into HOL. MSC has process algebra semantics as specified by ITU-T in the Z.120 Annex specifications [84]. Several proposals exist for providing HOL semantics to process algebra's like CSP. We use the work done by CSP-Prover [13] for formalizing CSP in

Isabelle/HOL. By combining the current MSC semantics and the CSP-Prover work, MSC is formalized in HOL.

Isabelle [85] is a generic theorem prover for implementing logic formalisms. Isabelle/HOL [14] is a specialized version of Isabelle for higher order logic. Theories are the building blocks of Isabelle, which is a named collection of types, terms, formulas, and theorems. New concepts are introduced and proved in Isabelle using theories. SAWSDL and NFSL are directly formalized in Isabelle/HOL. By using these formalizations we describe Web services using HOL theories by importing the theories from the HOL formalization for MSC, SAWSDL, and NFSL. The concrete integration of SAWSDL, NFSL, and MSC is presented in Figure 4.4. The Web services derived in this manner have functional, semantic, non-functional, and behavioral characteristics.

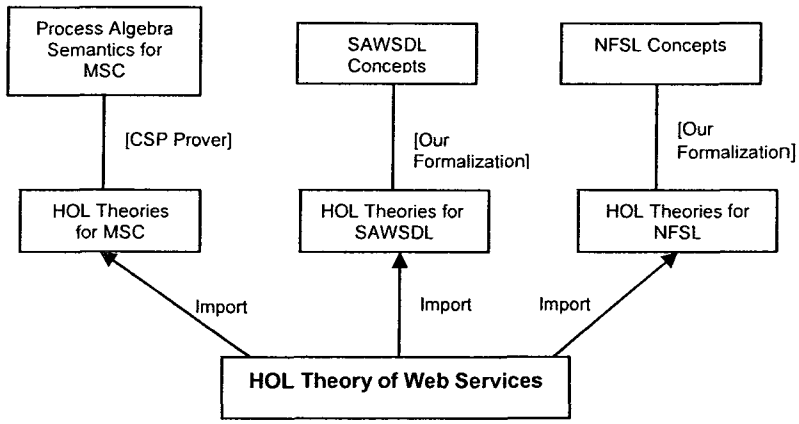


Figure 4.4. Concrete Integration

4.4.2. Formalizing MSC Concepts in Isabelle/HOL

MSC has textual and graphical syntax. The textual syntax is generally used by the users and tools for communicating the MSCs. The textual syntax of MSC considered in [84] is event-oriented. The set of events that a MSC allows determining what it means. MSC has

different types of events. However, generally events used in the context of Web services are message events. The relation between an input and an output is called a message (event) in MSC. This message can be split into two events, message input and message output, and it is mapped to message input and output events in the process algebra domain. The message sending entities (instance) name, message receiving entities (instance) name, abstract representation of the gate using which the message is sent, and the message name are the information that represents the message in process algebra. In Isabelle/HOL domain the message is mapped to an action, which is identified by the action name.

The main concepts of MSC that can be directly mapped to process algebra and Isabelle/HOL (using CSP-Prover concepts) are presented in Table 4.3. A co-region is an unordered set of events, which are defined on same instance. The horizontal composition of its events is the semantics of the co-region. The semantic of a co-region is basically the parallel composition of the events in that co-region. The parallel composition operator in Isabelle/HOL is captured using the synchronous parallel operator. The encoding of the synchronous parallel operator is defined using the stable failure semantics of CSP with the Isabelle syntax by CSP-Prover. The reader can refer to the details of the encoding of this operator and others in [86].

Table 4.3. Mapping MSC concepts with Process Algebra and Isabelle/HOL Concepts

MSC Concept	Process Algebra Concept	Isabelle/HOL Concept / Symbol
Message (event)	Message output/input events (Out/In)	<a> (Action name)

Co-region	Parallel composition operator (||)	|| (Synchronous parallel)
Sequential composition operator (seq)	Sequential composition operator (.)	;; (Sequential composition)
Parallel composition operator (par)	Parallel composition operator (||)	|| (Synchronous parallel)
Alternative (composition) [choice] operator (alt)	Alternative composition operator (+)	[+] (External choice)
MSC	Process	$

MSC defines three basic composition operators: seq, par, and alt. These concepts are used in MSC by inline expressions, MSC references, and HMSCs. The notion of vertical, horizontal, and alternative compositions is used for describing the semantics of seq, par, and alt operators respectively. These concepts are mapped to the respective sequential, parallel, and alternative composition operators in process algebra. The sequential and alternative composition operators in Isabelle/HOL are captured by sequential composition and external choice operators. Similar to the synchronous parallel operator, stable failure semantics of these operators are encoded in Isabelle/HOL by CSP-Prover.

Moreover, the vertical composition of MSC events is equivalent to the sequential composition operator, where the event orders in instances is maintained. Therefore, two message events that are vertically composed (that is which follow each other) can be combined using the sequential composition operator in process algebra, and mapped to sequential composition operator in Isabelle/HOL. In MSC documents, MSC References are used to refer other MSCs and it is identified by its name. MSC references also refer to

MSC expressions using the operators. MSCs that are used in these expressions are generally identified by its name, which is mapped to the concept of processes in process algebra and it is encoded using the symbol $ in Isabelle/HOL by CSP-Prover.

4.4.3. Formalizing SAWSDL Concepts in Isabelle/HOL

WSDL and SAWSDL are data-oriented languages as they are basically developed for describing Web services. There are two versions of WSDL – WSDL 1.1 [3] and WSDL 2.0 [87]. SAWSDL can be used as an extension to both the versions of WSDL. We use WSDL 1.1 for our formalization. The main reason for this choice is that although WSDL 2.0 is the latest W3C recommendation it is not widely adopted yet. On the other hand, WSDL 1.1 is the widely used language for Web service description at present. However, most of the concepts in WSDL 1.1 are similar with little modifications in WSDL 2.0, and the formalization can be easily extended to be used with WSDL 2.0.

The main concepts of WSDL 1.1 are types, message, portType, binding, port, and service, where the first three concepts describe the abstract part of the Web services, and the other three concepts describe the Web services concretely. 'Messages' abstractly define the transmitted data using the data type definitions provided by the 'types'. A set of abstract operations is a 'portType', where operations refer to messages exchanged in some predefined formats. 'Binding' specifies protocol and data format details for the messages and operations, and 'port' specifies an address for the binding. A set of related ports is aggregated as 'service'.

The concepts specific to SAWSDL are model reference, (lifting and lowering) schema mapping, and attribute extensions. 'Model reference' associates concepts in some

semantic model (generally ontology) and WSDL concepts. The structural differences and the mapping between the schema elements and its corresponding semantic model concepts are handled by (lifting and lowering) 'schema mapping'. 'Attribute extension' is used for extending the attributes where the element extension is allowed but not attribute extension. It is used only in WSDL 1.1, mainly for using model references in 'operations'.

Isabelle provides different data structures like sets, lists, and records for describing data. We use the record type for formalizing the WSDL 1.1 and SAWSDL concepts. Records are extensible in Isabelle. New records can be defined by extending the existing records. The concepts of WSDL1.1/SAWSDL are defined by using one record per concept. By using only the abstract concepts of WSDL 1.1, abstract descriptions can be easily defined. The whole formalization was specified in a single Isabelle theory. The syntactic correctness of the formalization was checked with the Isabelle (proof checking) tool.

The formalization of WSDL 1.1 concept 'portType' as a record in Isabelle/HOL is shown below. The keyword *'record'* identifies the record definition followed by the record name *'portType'* with three attributes, which is separated from the record name by a '=' symbol. The attribute names are separated from its type with a symbol '*::*'. For instance, *'modelReference'* is an attribute of this record and of type *'listOfAnyURI'*, which is specified as a new type in the theory. The *'porttypeOperation'* attribute is specified as a list of operations shown as '*operation list*', where *'list'* is a pre-defined type constructor in Isabelle, and *'operation'* is defined later.

```
types listOfAnyURI = string

record portType =
```

```
porttypeName :: string
porttypeOperation :: "operation list"
modelReference :: listOfAnyURI
```

The 'operation' concept is formalized below, where it is defined using its name, type, input/output/fault name and type, and extension. It uses the formalized concept of 'message', which is subsequently presented. The operation can be any of the four types defined using the *'opType'*: request-response, solicit-response, one-way, and notification.

```
datatype opType = requestResponse | solicitResponse | oneway |
notification
record operation =
operationName :: string
operationType :: opType
operationInputName :: string
operationInputType :: message
operationOutputName :: string
operationOutputType :: message
operationFaultName :: string
operationFaultType :: message
operationExtension :: string
```

The concept of 'message' is formalized using the following record, which uses a formalized concept part, which is not shown here.

```
record message =
messageName :: string
messagePart :: part
```

The concept of 'attribute extension' that is specific to SAWSDL is specified in the following record description. As mentioned before, this extension element is added mainly for specifying model references in operations as captured by the *'modelReference'* attribute, but some extra details can also be specified with the *'otherDetails'* attribute of the record.

```
record attrExtensions =

modelReference :: listOfAnyURI
otherDetails :: string
```

All the other concepts of WSDL 1.1 and SAWSDL are also formalized similar to the examples shown above. Finally, the formalization of the 'SAWSDL' definition is shown below. It used the previously formalized concepts of types, message, porttype, binding, and service. It should be noted that *'partEle'* captures the definition of types, which can be of type simple or complex. As SAWSDL can have a list of types and messages, they are defined as lists.

```
record SAWSDL =

wsdlName :: string
wsdlTypes ::  "partEle list"
wsdlMessagez :: "message list"
wsdlPorttype :: porttype
wsdlBinding:: binding
wsdlService :: service
```

It should be noted that as with any programming language, the same concepts can be formalized in numerous other ways in Isabelle/HOL.

4.4.4. Formalizing NFSL Concepts in Isabelle/HOL

Formalization of NFSL in Isabelle/HOL is simple and straightforward as the number of concepts and their complexity is small. The concepts are defined using a single record data structure. However, the definitions can be extended based on the extensions to the language itself. The formalization of NFSL is specified below. The scale of *'high'* or *'medium'* or *'low'* or *'none'* is specified for the last three parameters, using the new data type *'valueScale'*. We use the metrics of USD for *'cost'*, millisecond for *'responseTime'*, and percentage for *'availability'*, but they are not explicitly defined below. The type of natural numbers is specified as *'nat'* in Isabelle.

```
datatype valueScale = high | medium | low | none
record NFSofWS =
serviceName :: string
cost :: nat
responseTime :: nat
availability :: nat
security :: valueScale
reliability ::valueScale
reputation ::valueScale
```

Chapter 5

A Framework for Web Service Composition

5.1. Critical Review of Existing Composition Techniques

In this section, we derive requirements for the composition technique; classify the existing techniques, describe and critically evaluate them.

5.1.1 The Derived Requirements

It is clear that functional, non-functional, behavioral, and semantic characteristics should be taken into account during composition. The first and second requirement states that functional and non-functional aspects should be considered by the composition technique. The third and fourth requirement deals with the behavior and the semantic service characteristics consideration by the composition technique.

The fifth requirement we place on the composition technique is that it must use services described using existing standards or known mechanisms for service description. This is because of the fact that many languages for Web service description have already been proposed, and when the composition technique uses services described using these techniques there will potentially be more services that can be used for composition.

The next requirement is scalability. Scalability is an important factor in composition because of two reasons. First, there may be a huge number of services from which the most suitable services has to be selected. Second, there can be a large number

of unexpected requests that the technique might need to handle. The seventh requirement is that the technique should manage the composition process and should coordinate between the services. The eighth requirement is that the technique should not only be proposed but should also be implemented.

5.1.2 Composition Techniques: State of the Art

We propose a classification of the Web service composition techniques that have been proposed in the literature in Figure 5.1. The classification will ease the understanding of the techniques and also will allow easier evaluation with respect to the derived requirements. The classification has three major levels based on different parameters like when and how the composition is carried out, and using which techniques. Based on 'when' the composition takes place we initially classify the composition schemes as static and dynamic composition. As specified before, static composition is design time composition, and dynamic composition is runtime composition.

The next level of classification is based on 'how' the composition is carried out. In the case of static composition it has three sub-groups: manual, semi-automatic, and automatic, while dynamic does not have any manual composition method. Manual composition is completely performed by a human, semi-automatic composition is carried out with human assistance, and automatic composition takes place without any human involvement. The next level of classification is done based on 'what or which' methods are used for composition. We do not use this classification level for the static composition as there are very less techniques in this category. For the same reason, we also do not use this classification for semi-automatic group of dynamic composition.

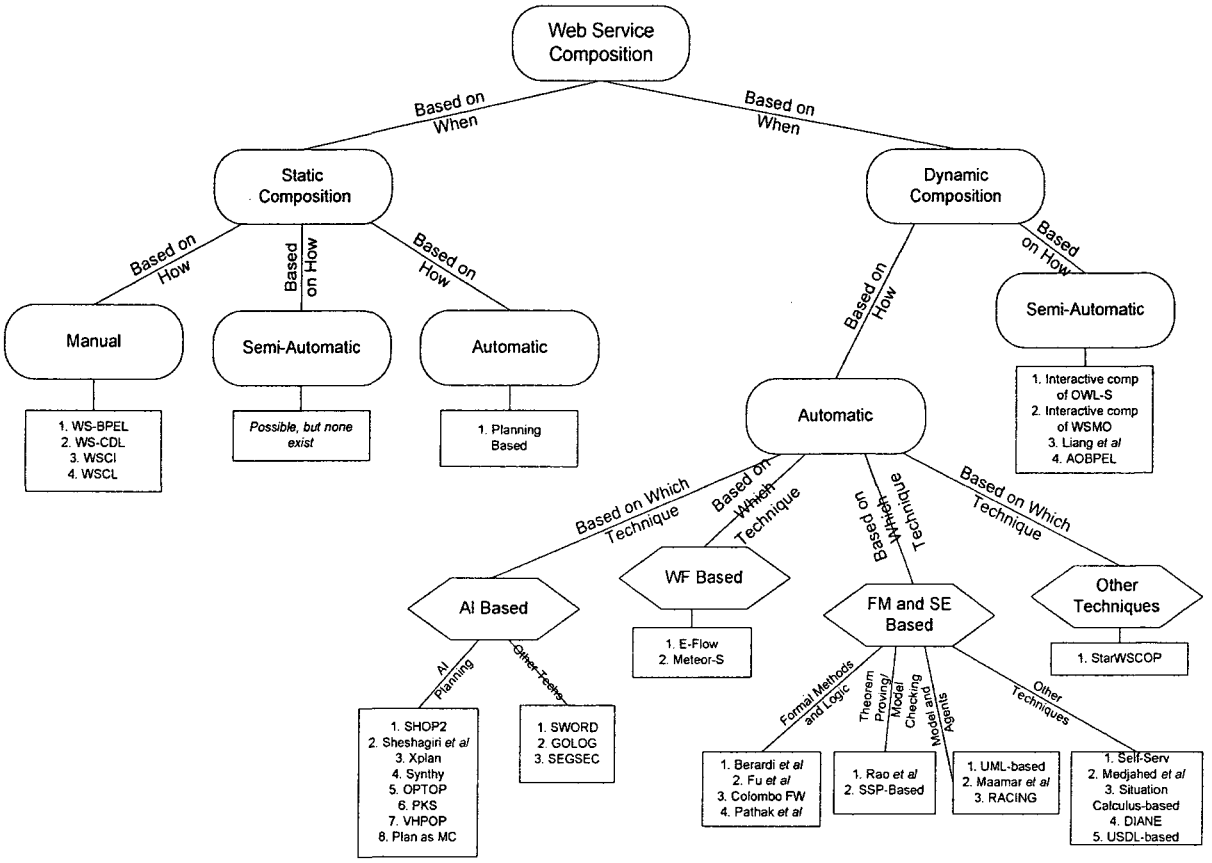


Figure 5.1. The Proposed Classification of the Web Service Composition Techniques

5.1.2.1. Static Composition Techniques

In the manual group of static composition category, the composition is designed with the help of Web service composition languages like WS-BPEL, WS-CDL, WSCI, and WSCL and their tools by designers. On the other hand, currently no methods exist to perform semi-automated static composition. Vukovic and Robinson develop context-aware applications automatically by proactive composition using AI-based planning techniques. In their paper [88] they compare two hand-coded planners Simple Hierarchical Ordered Planner 2 (SHOP2) and TLPlan for Web service composition using three technical requirements. This method is a kind of automatic proactive composition.

5.1.2.2. Semi-Automatic and Dynamic Composition Techniques

In the dynamic composition category, manual composition is not possible, while semi-automatic composition has four techniques. The semi-automatic composition techniques are interactive composition of OWL-S [89], interactive composition of WSMO [90], work by Liang *et al* [91], and Aspect Oriented BPEL (AOBPEL) [92].

A prototype for semi-automatic composition of services described with OWL-S is developed in [89]. Here, the semantically matching services are presented to the user (with domain knowledge) at each step of composition and the possibilities are filtered out based on the user selection. The prototype has two basic components, a user interface that is used to communicate with the human assistant, and an inference engine. The engine is basically an OWL reasoner that does the matching on functional properties and filtering on non functional attributes. A tool is developed in [90] that help in human assisted composition of WSMO-based semantic Web services. The tool guides the human in a step-by-step composition by recommending and selecting goals, mediators, and control flow operators, which are the basic building blocks of WSMO.

Liang *et al* [91] uses a semi-automatic approach for composite Web service description, discovery, and invocation by introducing an intelligent service registry. It assists the requestors by interacting with them through a user interface to get their service requirements. The requirements are captured interactively with service dependency graphs that is formally represented as And-Or graph. An And-Or search algorithm is used to construct composite service template that satisfy the requestors requirement. AOBPEL [92] tries to bring dynamicity and flexibility to BPEL using aspect oriented programming, which addresses the modularization of cross cutting concerns (separation of concerns). It

extends BPEL with an aspect-oriented extension, where aspects (units of modularity) can be plugged or unplugged into the composition process at runtime.

5.1.2.3. Automatic and Dynamic Composition Techniques

The automatic composition techniques in the dynamic composition group are further divided into four groups based on which specific methods are used for composition. This classification level has artificial intelligence (AI)-based techniques, workflow (WF)-based techniques, formal methods and software engineering (FM & SE)-based techniques, and generic or other techniques. As the name implies AI-based techniques group uses artificial intelligence methodologies like planning, rule-based systems, and automated reasoning for Web services composition. WF-based techniques category uses workflow concepts, principles, and methodologies for automatic composition. FM & SE group uses math based techniques and software engineering concepts for composition. We created a generic (other) group mainly to place all other automatic composition methods that does not fit in the three categories.

5.1.2.4. Artificial Intelligence Based Techniques

In the AI-based category, most of the automatic composition techniques are based on automated planning. Therefore, we further classified them as planning-based methods and generic methods, which include all methods that do not use planning for composition. Planning is concerned with realization of strategies by constructing sequence of actions to achieve some goals. Several planners have been used for Web service composition.

SHOP2 [93], a domain-independent Hierarchical Task Network (HTN) planner, which creates plan by task decomposition is used for composing Web services described using OWL-S (DAML-S). The implemented system converts OWL-S process models

into SHOP2 domains using some algorithms; similarly the OWL-S composition problem is encoded as SHOP2 planning problem. After the planning problem is solved, the plan produced by SHOP2 is converted into OWL-S format, which can be directly executed. Sheshagiri *et al* [94] solves the composition problem of OWL-S described services with a planner that uses a simple backward-chaining algorithm. The planner first converts the OWL-S service model to Verb-Subject-object (VSO) triplets, which is converted into a set of facts that form the planning operator in the next step, and then the planner is initialized by the user with a goal. The planner finds a service that satisfies this goal and includes it in the plan. It then tries to satisfy the unsatisfied goals, which are basically the inputs and the preconditions of the newly included service. This step is repeated until all goals are satisfied or when the planner fails to find any operators that satisfies the goals.

XPlan [95], a hybrid planner that extends action-based Fast-Forward planner with a HTN planning and re-planning component, is used for OWL-S service composition by the semantic Web service composition system OWLS-XPlan. This system has an OWLS2PDDL converter to convert the OWL-S service descriptions and OWL ontologies to corresponding Planning Domain Description language (PDDL) problem and domain descriptions. This is then used by XPlan to solve the composition problem by generating the plan in PDDL. In [96] a composite service creation environment based on end-to-end composition of Web services with a two-phase composition methodology is proposed. The phases are logical and physical composition phases, and it is implemented as a prototype called Synthy. The logical composition is carried out using limited contingency planning, where the abstract plan is developed based on service types. The physical composition subsequently concretizes the

executable plan by selecting appropriate Web service instances. The logical composer takes the functional requirement of the service specification as input, provides abstract BPEL workflow as output by functional composition of service types. The physical composer uses the abstract workflow along with the nonfunctional requirement to select the concrete service instances to produce the deployable workflow.

Optop (Opt-based total-order planner) is an extended Unpop planner to handle Web service problems. Unpop planner is a kind of estimated-regression planner that uses heuristics estimator, which is got by backward chaining in a relaxed problem space, to do state-space search. The Optop planner is used in [97] to solve small Web service problems; however this can also be used for solving the composition problem. The problem of automatic Web service composition is handled by Matrinez *et al* using knowledge-based planning system called Planning with Knowledge and Sensing (PKS) [98]. PKS is derived from generalization of Stanford Research Institute Planning System (STRIPS). The ability of PKS in generating parameterized conditional plans (with runtime variables) in the presence of incomplete knowledge and sensing is the main motivation for using it to solve the composition problem.

Peer solves the Web service composition problem by combining a modified Versatile Heuristic Partial Order Planner (VHPOP) with a re-planning algorithm [99]. A set of links that must be avoided by the planner called as avoid-links is the new addition made to VHPOP. The work illustrates how the feedback got from the failed plan execution can be used to avoid useless plans in the later planning attempts. Planning by model checking allows for planning under uncertainty, under partial observability, and with extended goals. This is used for Web service composition problem by Traverso and

Pistore [100]. The OWL-S process models of the available services are encoded into state transition systems which along with the composition goals are taken as inputs by the planning system that uses planning as model checking and generates automata that can be converted as executable BPEL processes.

In SWORD [26], a rule-based expert system is used for composition. If the composition is possible then a composition plan is created, and a persistent representation of the plan is generated after the developers' request. Mcllarith and Son adapt and extend logic programming language Golog [101], which is based on situational calculus, to address the problem of automated Web service composition by providing high-level generic procedures and customizable constraints. The extensions are implemented as a modification to an existing ConGolog interpreter. This interpreter has been integrated to the semantic Web architecture, which includes different service-related ontologies and an agent broker for communication with Web services.

Semantic graph based Service Composition (SeGSeC) [102] is a semantic-based service composition technique, which uses CoSMoS for service representation but gets the user's request as a natural language statement. It generates the execution path (workflow) of the service by performing semantic matching using a reasoner. SeGSeC uses Component Runtime Environment (CoRE) middleware for service discovery with the discovery interface, and also for service execution by means of the access interface.

5.1.2.5. Work Flow Based Techniques

The next category of dynamic-automatic composition techniques uses workflow concepts and principles that include eFlow [103] and Meteor-S [104]. In eFlow system the composite e-services are modeled as process schema that composes other basic and

composite services, and the process schema is enacted by a service process engine. It supports adaptive service process using concepts like dynamic service discovery and generic nodes. eFlow also allows dynamic service process modifications at a single process level (ad-hoc change) or at group level (bulk change). Managing End to End OpeRations for SWS (Meteor-S) project aims to create a comprehensive framework for Web process composition and also tries to apply semantics to Web processes. It approaches the service composition problem as a constraint satisfaction problem. The Meteor-S system allows abstract representation of the functionality required from the service, which is used by the discovery engine to find appropriate services with the help of constraint analyzer. The analyzer estimates, analyzes, and optimizes the dynamic service selection, and finally the abstract process is bound to an optimal set of services to generate an executable process.

5.1.2.6. Formal Methods and Software Engineering Based Techniques

The formal methods and software engineering category can be further divided into four sub-groups based on the specific method applied for solving the automatic composition problem. The subgroups are: (pure) formal methods based, theorem proving and model checking based, models and agents based, and other generic methods.

In the (pure) FM based methods, Berardi *et al* tackles the problem of automatic service composition by reformulating the problem in terms of satisfiability of a suitable formula in Deterministic Proportional Dynamic Logic (DPDL), where the e-services are described using FSM [105]. The tableau algorithms of Description Logic (DL) along with the optimized DL-based systems can be used to check the existence of e-service compositions, because of the correspondence between the PDL and the DL. Fu *et al* uses

a notion of conversation [47], which is a sequence of messages observed and tracked by a global watcher as they occur. Linear Temporal Logic (LTL) is extended to specify desired properties on the conversation and verified. It is proved that LTL verification for an arbitrary Web service composition is not decidable, when the Web services communicate asynchronously. To avoid the complexity, a synchronizability analysis is proposed for Web service composition.

Colombo [106] exploits and extends techniques based on propositional dynamic logic for composite service synthesis by constructing a mediator that will realize the 'goal service'. It also uses another technique to reason over finite universe of domain values rather than infinite universe to perform service synthesis. MoSCoe [107] proposes a framework for composite service creation by iterative reformulation of the functional specification of the goal service incrementally. It accepts an abstract specification of the goal service from the user in LTS form, where it uses its algorithms to 'realize' the composite service. If the composition fails, the user reformulates the goal, and this process is repeated until a feasible composition is created or when the user aborts. It also enables adaptation of the composite service by generating alternative specification on-the-fly, and also enables context-specific substitutability of the component Web services.

Rao *et al* uses Linear Logic (LL) based theorem proving for automatic and semantic Web service composition, which comes under the theorem proving and model checking subgroup of dynamic composition [65]. Linear logic enables us to define the attributes of Web services formally, while the process calculus is used to represent the process model of the composite service formally. The process calculus is attached to LL inference rules in the style of type theory, consequently the process model can be

generated from the proofs. Lammermann [108] developed a dynamic service composition with logic-based program synthesis by extending Structural Synthesis of Programs (SSP), which is an approach to deductive synthesis of functional programs from specifications. Extended Structural Synthesis of Programs (ESSP) extends the logical language of SSP with disjunction, falsity, and restrictive quantifiers. The intuitionistic proposition logic is used for solving the composition problem.

In the model and agent based category, Gronmo and Jaeger uses semantic Web service languages within the model-driven methodology for composition of Web services using a four-phase methodology [52]. The first phase involves modeling the composite service by the service developer using ontological and QOS concepts, and the second phase involves discovery based on matchmaking of semantic descriptions. The third and fourth phase deals with the selection of services based on QOS and deployment respectively.

Maamar *et al* [109] presents an agent-based and context oriented approach for Web service composition. In this work, composite-service-agents are associated with composite services, master-service-agents associated with Web services, and service-agents are associated with service instances. The different agents are aware of the context of their respective services and engage in conversation with their peers to agree on Web services that will involve in the composition process based on several factors (for e.g., availability). Ermolayev *et al* [110] presents a framework for dynamic agent-enabled Web service composition based on the understanding that dynamic coalition of software agents collaboratively performing tasks for service requesters can compose and mediate Web services. A middle agent layer is introduced between the service requester and

service provider layer that enables capability assessment, credibility assessment, and negotiation.

Self-Serv [50] provides a scalable middleware for declarative composition of Web services in a heterogeneous and dynamic environment. It uses peer-to-peer distributed orchestration with the help of coordinators that is attached to all services involved in composition. Moreover, it uses a concept that provides a flexible composition of large number of services called as service community, which is basically a container of substitutable services. At runtime, the community selects the best usable service based on some constraints. Medjahed *et al* [51] provides automatic composition of services on semantic Web by proposing an ontology based framework. It uses composability model and composability rules at both syntactic and semantic level. The approach consists of four conceptually separate phases. In the specification phase, the high-level description of the desired composition is specified. In the next phase, matchmaking is performed using rules. After that, service selection takes place based on quality of composition parameters. Finally, the service is automatically generated in the generation phase.

In [63], Web services described in DAML-S is provided situational calculus based semantics, which is then given Petri net based execution semantics. The Petri net encoded Web services are then analyzed and composed. DIANE [111] proposes an approach to Web service composition, where the composition is integrated with service matchmaking. It deals with a particular class of service requests with multiple connected effects. DIANE basically builds an automatic matcher that composes services, provides fine-grained ranking among different offers, and invokes the best offer. The basis of the approach is to initially check for plug-in matches, where available services are checked to

find if they can satisfy a part of the request. Finally, services are found that match unsatisfied parts of the request using 'single-effect' services.

In [112], Web services composition is performed using constraint matching, where the list of candidate services is narrowed down in multiple steps using constraints. The set of composable services are found using its matching engine. The sequence of execution of these services is determined based on the pre and post conditions of the individual services, using some filtering techniques. The matching engine uses constraint logic programming at its core.

5.1.2.7. Other Techniques

In the generic category of dynamic composition group, Star Web Services Composition Platform (StarWSCoP) [113] is the only system, which is a platform for dynamic Web service composition. The architecture of StarWSCoP consists of an intelligent system to decompose user's requirements, service discovery engine to discover services from service registry, and composition engine to handle compositions. The composition engine deals with events sent by event monitor, and keeps composite service trace information in service execution information library. It uses wrapper for interoperability, and QOS estimator to estimate real time QOS metric of composite Web service.

5.1.3 Analysis of Existing Composition Techniques

Table 5.1. Comparative Analysis of the Composition Techniques

Requirement	**AI Based Techniques**	**WF Based Techniques**	**FM & SE Based Techniques**	**Semi-Auto and StarWSCoP**	**Static Composition**

Uses functional aspects of WS	Yes	Yes	Yes	Yes	Yes
Uses non functional aspects of WS	Yes	Yes	Yes, except (pure) FM	Yes	No
Uses Behavioral aspects of WS	No behavior aspects	No behavior aspects	No behavior aspects, except (pure) FM	No behavior aspects	Yes - Manual No - Auto
Uses semantic functionalities of WS	Yes	E-Flow - No Meteor-S -yes	No, except model-based	No - Liang & StarWSCoP, Others Yes	WSMO/L and SWSO/L – Yes
Scalability	Yes	Yes	Yes	Yes	Yes
Uses Known DT	Mostly yes	Yes	No, mostly uses new techs	Semi-Auto, Yes	Yes
Manage & Coordinate	No	Yes	Yes	Semi-Auto, Yes	Yes
Implementation	Yes	Yes	Yes, except agent/model	Semi-Auto, Yes	Yes

The comparison of the composition techniques based on the set of derived requirements is presented in Table 5.1. It is clear from the table that all the dynamic composition techniques support functional and non-functional aspects of the services except the (pure) FM-based techniques, where methods of Berardi *et al* and Fu *et al* does not consider the non-functional service aspects. The behavior aspect is supported only by the (pure) FM-based techniques in the FM & SE category of automatic composition, all the other

mechanisms does not support the behavior aspects of services. AI-based techniques, Meteor-S, model-based, DIANE, USDL-based and semi-automatic composition of OWL-S and WSMO are the methods that supports domain semantics, the other mechanisms does not support it.

All the dynamic mechanisms are generally scalable. It should be noted that StarWSCoP does not support requirements 6, 7, and 8 as it just an initial proposal and not much work is done. Except the model and agent based techniques every other technique has been implemented. Similarly, they along with other methods in their FM and SE based category do not use known description techniques; however all other methods supports some known description technique. It can also be noted that only AI-based techniques does not support managing and coordination of the Web services after composition, while the other methods support it. Berardi *et al* and Colombo also requires the client to specify a detailed FSM to carry out the composition, which is not practical and straightforward from requester's viewpoint.

All the static composition methods support functional aspects of services, and none supports non-functional aspects of services. The manual methods support behavioral aspects, while the planning-based automatic technique does not support it. Except for WSMO/L and SWSO/L, others do not support semantic aspects. All the methods in this category support the last four requirements. From the analysis, it is clear that none of the existing methods supports all the requirements for the composition technique. Therefore, a novel composition technique/system has to be developed that will satisfy the set of derived requirements.

5.1.4. Techniques Related to Composition

5.1.4.1. Web Services Adaptation

As the Web services environment is dynamic, the components that make the composite services can change frequently. To address this problem, many solutions for adapting compositions dynamically at runtime have been proposed. We briefly discuss some of these solutions here. Generally, most of these approaches aim at adapting BPEL processes, as it is the most widely used composition language.

In [114], a framework is proposed for adapting Web service compositions using three core components: Distributed Registry (DIRE), Service Composition Execution Environment (SCENE), and Dynamic Monitoring (Dynamo). DIRE allows cooperation among heterogeneous registries by using publish-subscribe middleware for the service publication. SCENE enables development and execution of self-configurable compositions by extending BPEL with policies and constraints, and provides a runtime environment for executing such compositions. Runtime monitoring of these BPEL-like processes is carried out by Dynamo. Vienna Dynamic Adaptation and Monitoring Environment (VieDAME) [115] system proposes a non-intrusive adaptation of BPEL processes based on monitoring of the QOS attributes by replacing the partner services of BPEL processes at runtime. It uses the aspect oriented programming for its working. It works by intercepting the SOAP messages and exchanging the existing partner services with other services which are syntactically or semantically equivalent.

An aspect oriented framework is proposed in [116] for service adaptation. It identifies adaptation as a cross-cutting concern, and separation of business logic from the adaptation logic is proposed. The framework consists of, taxonomy of service

mismatches, a repository of aspect-based templates, and a tool. The taxonomy has a list of different mismatches that occurs at the functional (interface) or behavioral (protocol) level between two services. The aspect-based templates handle the different mismatches automatically using the aspects oriented programming concepts. The template instantiation and its execution are performed with the tool.

5.1.4.2. Verification and Validation of Composition

Works on verification and validation of Web service compositions, particularly BPEL-based compositions, have been proposed. In [53], a model based approach for verifying Web service compositions is proposed, where the composite Web services are modeled with MSCs in the specification stage, and BPEL is used for the composite service creation in the implementation stage. Both these models are translated into finite state processes and the trace equivalence between them is verified using Labeled Transition System Analyzer.

SPIN model checker is used in [117] to verify the compositions expressed as BPEL processes. The BPEL processes are initially converted to guarded automata model, which are then converted to Promela processes, and the Promela specifications are verified using SPIN model checker. A validation framework for BPEL processes at both design time and run time is proposed in [118], where the properties for validation are expressed using their novel asserting language for BPEL process interactions (ALBERT), ALBERT is a temporal logic language. The BPEL processes are annotated with ALBERT assertions, which are verified at design time using a model checker. The run time validation is performed by the monitoring framework Dynamo [114], which is embedded in a BPEL engine.

5.2. Analysis of the Related Matchmaking Techniques

Matchmaking is the core technique for any composition system. Different composition systems handle matchmaking in different methods. In this section, we discuss and analyze some of the significant matchmaking techniques that have been proposed. The techniques are analyzed based on their support for and manipulation of the functional, behavioral, non-functional, and semantic characteristics of Web services together.

UDDI allows matching and discovering Web services, which at present with the V3 standard [4] supports basically keyword and string based matching of functionalities. It does not allow matching the non-functional, semantic, and behavioral aspects of Web services. However, many extensions to UDDI have been proposed for supporting the other Web service characteristics. In [119], UDDI and its data structures are extended to handle non-functional based matching along with its basic functional discovery.

In [120], functional matching of WSDL specifications is performed based on the (traditional) information retrieval and structural matching methods. In this method, semantics that are specified using ontologies are not used for matching. However, WordNet, a large lexical database of English, is used for semantic-like matching. If only the natural-language description of the needed service is available, then vector-space based information retrieval method is used for matching, which is further enabled used WordNet based discovery. Structural matching of WSDL components (data types, messages, and operations) or WordNet based semantic structural matching is performed, when a partial specification of the needed service in WSDL form is available. It can be noted that this method does not perform non-functional or behavioral matching.

Similar to the work in [120], WordNet-based lexical (semantic) similarity, WSDL based interface and data type similarity, and QOS based similarity is checked in [121]. However, it does not consider and check the behaviors. A WSDL matching method is proposed in [122], which uses lexical matching and structural matching of WSDL concepts. It does not support non-functional and behavioral matching. Functional matching of WSDL concepts (operations and data types) is performed in [123] and [124]; they do not consider other matches.

Functional and semantic characteristics of the requested and the advertised Web services are matched in [30]. It describes Web services using DAML-S. The inputs and outputs of the request and the advertisements that are described using the service profile are checked for semantic matching. Matching of behavioral and non-functional aspects of Web services are not used here. This work is used in [31] for extending UDDI with semantic matching capabilities. The work in [30] is further extended in [125] by extending the inquiry API of UDDI for requesters to specify the capabilities they require from the service. It also allows automatic composition in extended UDDI using planning-based techniques.

The authors in [126] consider functional, non-functional, and semantic characteristics of Web services for matching. An agent-based framework and QOS ontology is used for dynamic Web service selection. In this work, functional aspect of Web services are described using WSDL. The non-functional characteristics are described using WS-Policy, which uses the concepts from their proposed QOS ontology. Initially, functional matching is performed with the WSDL services interfaces. After that, non-functional and semantic matching is done with the list of functionally matched

services. It should be noted that behavioral description and matching of Web services is not considered by this work.

The functional, semantic, and behavioral characteristics of Web services are used for matching Web services in [127]. It describes Web services with OWL-S. The service process that describes the Web services behavior is represented using Petri nets. The functional and the semantic description of the services described as inputs and outputs in the service profile are used in the first step of matching. The behavior is then matched using the Petri net representation of the service processes. However, in this technique, non-functional characteristics are not used for Web services matching.

Similar to the last work, in [128], a modal logic based framework is used for matching and composing Web services with functional, semantic and behavioral characteristics. The Web services are described using an agent based language based on modal logic, which is extended to describe the behavior in terms of communication interactions. Matching is done by reasoning about interactions in the logic domain. Non-functional matching is ignored in this work also.

Web services are represented using labelled transition system, where their compatibility (matching) is defined using three notions [129]. The first notion is that when two Web services have opposite behaviors they are compatible. The second notion of compatibility is based on unspecified reception, and the third is based on deadlock freeness. In this work, it is assumed that the matched services are semantically (and functionally) compatible. The non-functional compatibility is not considered.

It is clear from the discussions that the above mentioned techniques for Web services matchmaking do not match services with all the characteristics together. It is also

important to note that the composition techniques presented in the last section also matches services at some point. It is apparent that none of those techniques also support matching Web services with all their characteristics together. To the best of our knowledge, no matchmaking technique exists that considers all the characteristics together for discovering Web services.

5.3. The Proposed Composition Framework

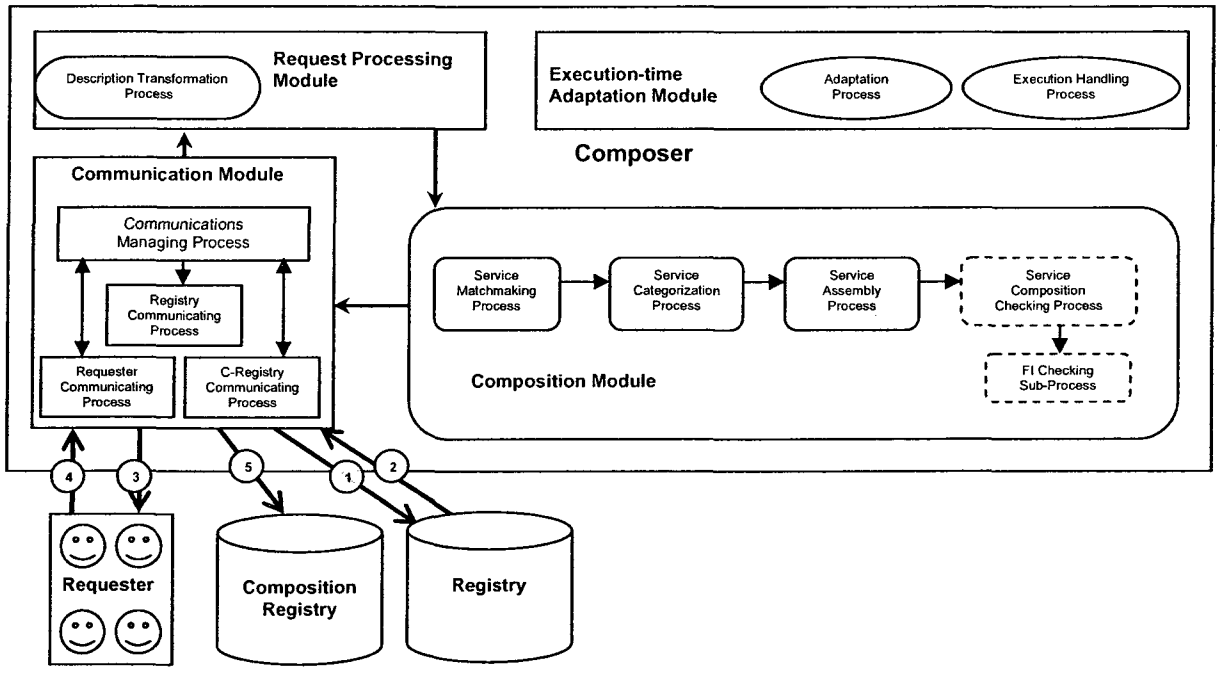


Figure 5.2. Web Service Composition Framework

The composition framework is shown in Figure 5.2. The framework externally communicates with the registry using the register and inform interactions (numbered 1 and 2 in Figure 5.2), with the requester using the get and give interactions (numbered 3 and 4 in Figure 5.2), and with the composition registry using the locate interaction (numbered 5 in Figure 5.2). The framework is functionally divided into four modules.

They are communication module, request processing module, composition module, and execution-time adaptation module.

The communication module is the interface of the framework for all external communications. It consists of three processes for communicating with the three external entities, and an entity to manage all the communications. As the names suggest, the registry communicating process (RCP), the requester communicating process (REQCP), and the c-registry communicating process (CRCP) are used for all communications with the registry, the requester, and the c-registry respectively. The communications managing process (CMP) manages all the external communications using the other processes.

The RCP realizes register, inform, and publish interactions from the framework's perspective. The REQCP realizes get and give interactions for obtaining composition-specific information from the requester to compose the correct services it wants. The locate interaction is realized by the CRCP. The CMP internally communicates with RCP (which is notified using the inform interaction) for getting information about the requester, and using this information CMP invokes the REQCP. The CMP allows the other modules to use CRCP for communicating with the c-registry.

The request processing module manipulates the incoming composition request and routes it to the composition module. It uses the description transformation process (DTP) to convert the incoming composition request to a format that can be processed by the composition module. The composition request is directed to the request processing module by the communication module, specifically the REQCP. As mentioned before, the composition framework uses the concepts of the description framework presented in

the last subsection. The DTP basically maps and integrates the request in SAWSDL, MSC, and NFSL descriptions to HOL description for internal manipulations.

The composition module performs the actual composition using three core processes: service matchmaking process (SMP), service categorization process (SCP), and service assembly process (SAP). These processes manipulate the Web services at HOL-level. The SMP aims at checking the available services to find if they fully or partially match the requested service. The matched services are categorized based on their level of functional, behavioral, and semantic match by the SCP. The SAP assembles the categorized services by manipulating them, and selects the best-assembled service based on non-functional matching.

The composed service is optionally verified using the service composition checking process (SCCP) to find if the composed service satisfies the request. SCCP uses another optional sub-process for checking the possible feature interactions of the component services, called as feature interaction checking sub-process. Feature interactions are unintended interactions that occur when different Web services work together to accomplish some task, where a Web service modifies the working of another Web service. More information on Web service feature interactions can be found in [130]. It can be noted from Figure 5.2 that the optional processes are represented using dashed boxes.

The execution-time adaptation module is used during the execution of the composed service. It is not used during the creation of the composed service. It is made up of adaptation process and execution handling process. The adaptation process helps in adjusting the execution environment for the composed service to work flawlessly. The

composed service is handled appropriately during execution in order to perform its task by the execution handling process.

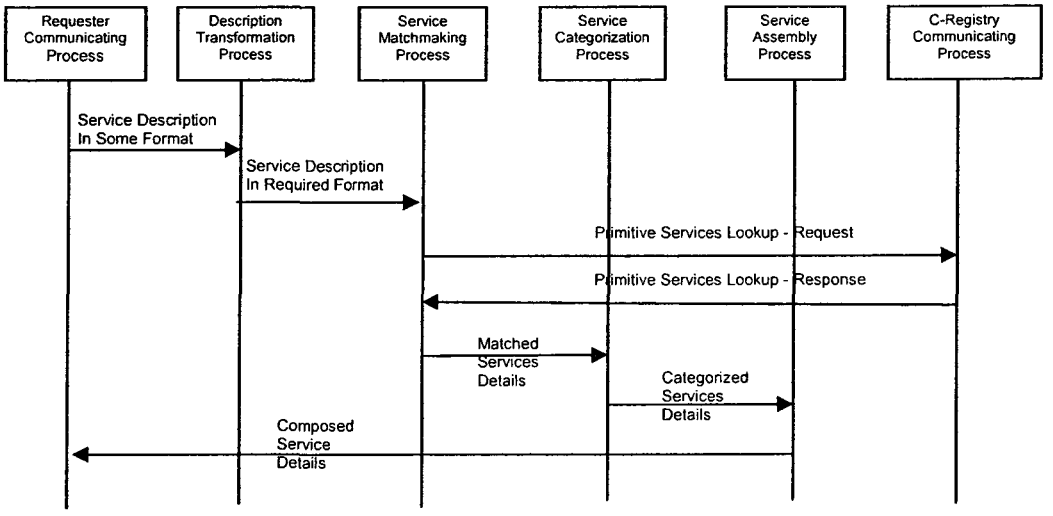


Figure 5.3. Interactions of the Different Processes during Dynamic Composition

The interaction between the different processes at the time of dynamic composition is illustrated in Figure 5. 3. The scenario assumes that no third party service exists that can directly satisfy the composition request, and also the composed Web service satisfies the requirement of the requester from all perspectives. We also assume that CMP allows the other processes to access CRCP seamlessly. The external communications of the framework is not shown in the Figure. The service description received by REQCP in some acceptable format (SAWSDL, MSC, and NFSL) is converted by DTP, using its technique, to (HOL) format required for performing composition. The request is then directed to SMP. The service matchmaking process uses its technique to match the request with all the available third party primitive service's using CRCP. The information about the matched services is given to SCP, which categorizes them using its technique. The SAP receives the details of the categorized services, which is used by its technique to

select and assemble the composite service, and select the best-assembled service. The composed service details are then given to REQCP, which is disseminated to the requester.

5.4. Service Matchmaking Technique

In this section, the matchmaking technique, which discovers the third party primitive services, is discussed in detail. The basic concepts and principles that we use for matchmaking is presented in the first subsection. The second section describes formally matching Web services using Isabelle/HOL. The matchmaking procedures are discussed in the next two subsections, where the procedure for finding fully matched Web services is presented in the third subsection. We conclude this section by discussing the procedure for finding partially matched services.

5.4.1. Basic Concepts and Principles in Service Matchmaking

In our matchmaking technique, Web services are discovered with functional, behavioral, non-functional, and semantic characteristics. The basic idea of the matchmaking technique is to manipulate Web services with all their four characteristics at higher order logic level using existing HOL theorem provers like Isabelle. Matching Web services that have all these characteristics together is possible in our case because of the description framework, which integrates the different Web service aspects in higher order logics. The matchmaking technique uses Isabelle theorem prover for behavioral, functional, and non-functional matching. The semantic matching is performed using a description logic reasoner. It should be noted that semantic reasoning and matching can also be done with

Isabelle, when Web Ontology Language (OWL) concepts are formalized in HOL and ontologies are developed using these formalized OWL concepts.

The basic principle used by the matchmaking technique is that if a third party service exists that fully matches the requested composite service then it is orchestrated as a BPEL process and directly provided to the requester (and no further processing is performed). Otherwise, a composition is carried out using partially matched services. Fully matched Web services are third party Web services that completely match the requested service from functional, behavioral, non-functional, and semantic viewpoints. Partially matched Web services are third party Web services that partly (partially) match with the requested service.

A Web service is considered as a match to another Web service when it provides 'equivalent' or 'more' features than the other service. Obviously, if the service offers fewer features than the other service, it is not considered as a match. The concepts of equivalence and refinement (preorder) from the Isabelle perspective, and the usage of Isabelle theorem proving for matching is discussed in detail in the next subsection.

There are mainly four types of semantic matching in the literature, among which exact, plug-in, and subsumes are considered as matches by the matchmaking technique. The last match is 'fail', where the concepts do not match at all and this match is obviously not considered. The *exact* match is a type of match where the concept of the requested service and the concept of the matched service is equivalent. In the *plug-in* match, the concept of the requested service is a sub concept of the matched service. The concept of the requested service is a super concept of the matched service in the *subsume* match.

The exact match is the best and the most preferred match, but sometimes it may not be available. In this case, the plug-in match is the next preferred match. However, sometimes it is possible to only have subsume match, which although is a weak match, offers some level of concepts matching. In our matchmaking technique, we allow all three levels to give more flexibility for the users in matching. Nevertheless, for example, if some user only wants to consider exact matches of semantic concepts, then the composition framework and this technique can be configured to only accept this particular matching.

The service matchmaking technique uses two procedures, one for finding fully matched services, and the other procedure for finding partially matched services.

5.4.2 Using Isabelle/HOL for Formal Matching

CSP-Prover concepts are used for matching the Web services in Isabelle. It should be recalled that the high-level description of Web services in SAWSDL, MSC, and NFSL format has been translated into HOL. CSP-Prover provides a deep encoding of CSP in Isabelle. The syntax and semantics of CSP is encoded in Isabelle using the logic HOL-Complex. CSP-Prover implements the stable-failures model as the denotational semantics of CSP. Denotational semantics allows creating *denotations* that are mathematical objects, which describes the meaning of the expressions in the formalized language. CSP-Prover defines the stable failures model in [86].

Given a set of communications Σ, the domain of the stable failures model F_Σ is a set of pairs (T, F) satisfying the following healthiness conditions, where $T \subseteq \Sigma^{*t}$ and $F \subseteq \Sigma^{*t} \times P(\Sigma^t)$ ($\Sigma^t := \Sigma \cup \{t\}$, $\Sigma^{*t} := \Sigma^* \cup \{t \frown \langle t \rangle \mid t \in \Sigma^*\}$, and t means termination).

T1: *T is non-empty and prefix closed,*

T2: $(t, X) \in F \Rightarrow t \in T$,

T3: $t \hat{} \langle t \rangle \in T \Rightarrow (t \hat{} \langle t \rangle, X) \in F$,

F2: $(t, X) \in F \wedge Y \subseteq X \Rightarrow (t, Y) \in F$,

F3: $(t, X) \in F \wedge (\forall a \in Y.\ t \hat{} \langle a \rangle \sim \in T) \Rightarrow (t, X \cup Y) \in F$,

F4: $t \hat{} \langle t \rangle \in T \Rightarrow (t, \Sigma) \in F$.

The labels from ***T1*** *to* ***F4*** *of the healthiness conditions are the same as ones used in [131]. Condition F2 states that a process can refuse a subset of X, when it can refuse the set X. Condition F3 states that if a process can refuse the set of events X in some state, then the same state must also refuse any set of events Y, which the process can never perform after s. The last condition F4 states that a process can refuse to do anything but terminate, if the process can terminate. In CSP-Prover, the set of traces satisfying* ***T1*** *is denoted by* T_{Σ}*, which is exactly the domain of the traces model.*

A *trace* of the behavior of a process is a finite sequence of symbols recording the events in which the process has engaged up to some moment in time [58]. A *failure* is a pair (s, X), where $s \in traces\ (P)$ and $X \in refusals\ (P/s)$ (P/*s* represents process *P after* the trace *s*), and *failures* (*P*) is the set of all *P*'s failures [13]. A *refusal set* is a set of events that a process does not accept, and *refusals* (*P*) are the set of *P*'s initial refusals [13]. Failures (P) are a set of all failures of P taking into account all the traces of P. Definitions of Traces (P) and Failures (P) as given in [131] is shown below.

$$Traces\ (P) = \{s \in \Sigma^{*} \mid \exists Q.P \Rightarrow^{s} Q\}$$

$$Failures\ (P) = \{(s, X) \mid \exists Q.P \Rightarrow^{s} Q \wedge Q\ ref\ X\}$$

$$\cup\{(s^\frown\langle t\rangle, X) \mid \exists Q.P \Rightarrow^{s\langle t\rangle} Q\}$$

CSP-Prover defines process equivalence $=F$ and process refinement $<=F$ over the stable failures model in the same way it is defined in [131], which is shown below.

$$P =F Q \Leftrightarrow traces\ (P) = traces\ (Q) \wedge failures\ (P) = failures\ (Q),$$

$$P <=F Q \Leftrightarrow traces\ (P) \supseteq traces\ (Q) \wedge failures\ (P) \supseteq failures\ (Q).$$

A process (P) is equivalent to other (Q), only when traces (P) = traces (Q), and failures (P) = failures (Q), meaning when the traces and the failures of the two processes are exactly same. A process (P) refines other (Q), if and only if traces (P) $\supseteq$ traces (Q), and failures (P) $\supseteq$ failures (Q). Basically, a process P refines other process Q, when its traces and failures are a superset of the traces and failures of Q. It should be noted that, when Q refines P and P refines Q, then P and Q are equivalent. Generally, equivalence relation is reflexive, transitive, and symmetric, while refinement (preorder) relation is reflexive and transitive, but not symmetric.

In essence, CSP-Prover allows verifying process equivalences and process refinement. The equivalence between a third party service and the requested service is captured by the stable failure equivalence between the two services. When a third party service that is matched with the requested service offers more features, this service essentially simulates the requested service, which can be captured using the stable failure refinement (preorder) between the services.

In our matchmaking procedures, the equivalence between the requested service and the third party service is checked first. If they are equivalent then the refinement is not checked as they already refine each other. Otherwise, the refinement between the two

Web services is checked, if this also fails then the two Web services does not match. An 'equivalent' third party Web service is always preferable because it is exactly what is needed. On the other hand, a third party service that 'refines' a requested service may have some extra behaviors that could have interferences when combined with other third party services to realize a composite service.

We provide a simple example at HOL-level to show how services are compared and matched based on these relations. For example, if the behavior of a third party Web service is message1 ;; message2 where ;; is sequential composition operator and the messages are described using SAWSDL. If the requested Web service behavior is messageOne ;; messageTwo, then they are compared for equivalence by checking message1 ;; message2 =F messageOne ;; messageTwo. The equivalence, if true, can be proved by using data definition unfolding, CSP laws, and semi-automatic tactics of CSP-Prover in Isabelle. Similar to this, to prove refinement between Web services, the checking of WS_thirdParty <=F WS_requested is done, where WS_thirdParty is the behavior of the third party service and WS_requested is the behavior of the requested service.

The main issue in using Isabelle theorem proving for Web services matching is automation. Generally, theorem provers like Isabelle are semi-automatic, so automatic matching is difficult. However, for proving equivalences with Isabelle in our case, a set of basic techniques are repeatedly applied. Using data definition unfolding and simplification on the behavioral expression, by applying the functional definitions of services, the equivalence is proved. Therefore, it may be possible to develop procedures/ tools that can interact with Isabelle based on a set of preconfigured rules, which can

check the equivalences. However, this has to be further explored. On the other hand, proving refinements is not straightforward, and this cannot be automated. It is important to note that, as in the case of semantic matching, the user can configure the technique for only equivalence matching in Isabelle, which can enable such automation.

5.4.3. Matchmaking Procedure for Fully Matched Web Services

The procedure for checking for fully matched Web services is presented in Figure 5.4. It is important to note that in the case of Figure 5.4 only one service is looked for that can directly satisfy the requested service. We assume that the third party services are available in HOL format. A list of these third party services that are checked by the matchmaking procedure are accessed from the composition registry using the existing keyword-based discovery mechanism of UDDI.

In this procedure, a service is initially selected from the composition registry using c-registry communicating process. The selected service is checked to find if it satisfies the requested service from the functional and behavioral viewpoints. This checking is performed using Isabelle theorem prover [14]. The selected service is considered to satisfy the requested service if it either has an equivalence relation or preorder relation with the requested service (concepts related to these were discussed in the last section). If the selected service matches from the functional and behavioral perspectives then the next checking is performed, where the selected service is checked for semantic matching using a description logic reasoner like Pellet [132, 133]. All the relevant semantic concepts of the selected service should be checked with the concepts of the requested service to find if there is an exact or plug-in or subsume match.

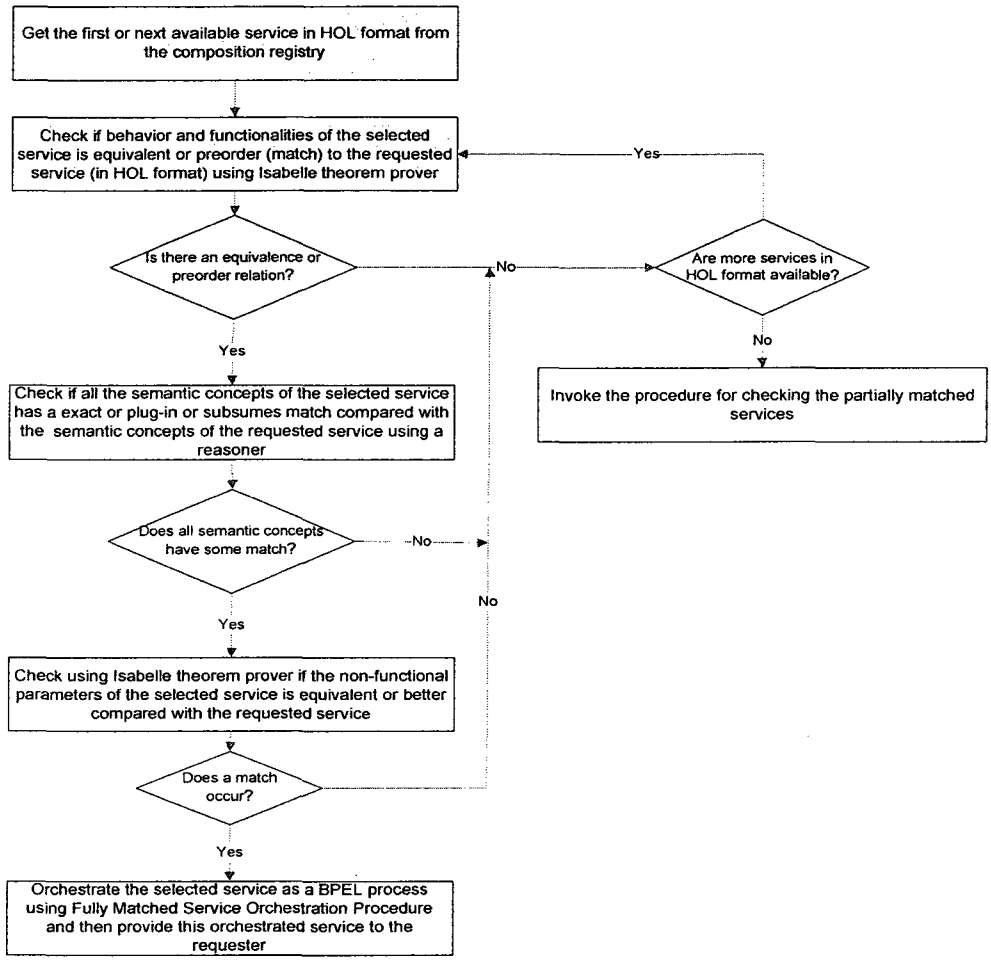


Figure 5.4. The Matchmaking Procedure for Fully Matched Web Services

If the necessary semantic concepts of the selected service match with the concepts of the requested service, then the selected service is checked for non-functional matching with the requested service. This matching is again done using the Isabelle theorem prover using the formalized NFSL concepts. It is simple to check if the selected service has equivalent or better non-functional values than the requested service using Isabelle. If the selected service also satisfies the non-functional requirements, then it is orchestrated as a BPEL process using the fully matched service orchestration procedure. Subsequently, the BPEL process is given to the requester as the composed service, and no further services are checked. If a match does not occur in any of the three matching then the next

available service is selected and checked until a match occurs, or until all the available services are exhausted. If none of the available services can fully match the requested service, then they have to be checked for partial matching.

BPEL supports data handling concepts like variables, statements, and also activities, which performs the actual logic of the processes. Activities can be basic activities like invoke, receive and reply, assign, throw, wait, empty or structured activities like sequence, while, switch, pick, and flow. Reference [134] gives tutorial level information on BPEL. The main behavioral operators that are used in our description are sequential composition, parallel composition, and the choice operators. BPEL constructs of sequence, flow, and pick are used for realizing these operator behaviors in the orchestrated service.

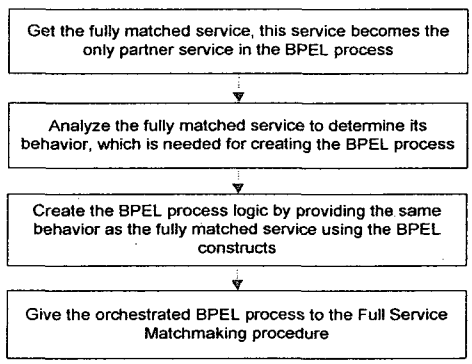


Figure 5.5. Fully Matched Service Orchestration Procedure

The fully matched service orchestration procedure is shown in Figure 5.5. In this procedure, the fully matched service, which becomes the (only) partner service for the BPEL process is initially got. Then it is analyzed to find its exact behavior, which is used in the next step to create the logic of the BPEL process that is same as the fully matched service. The process logic creation usually involves invoking the right Web service operations of the fully matched service using 'invoke' activity, and then providing the

overall logic using the 'sequence/ pick/ flow' activities. The procedure finally gives the BPEL process to the full matching procedure.

5.4.4. Matchmaking Procedure for Partially Matched Web Services

The procedure for finding the partially matched Web services is illustrated in Figure 5.6. Initially, the requested service description in HOL format is analyzed and decomposed into components. The decomposition is done based on the behavioral operators. After the decomposition, the first third party service is selected from the composition registry using c-registry communicating process, and its behavior and functional matching to the first component of the requested service is checked using Isabelle theorem prover. This checking is similar to the functional and behavioral checking done in the matchmaking procedure for fully matched services, except the fact that the matching is done with a 'part' (component) of the request.

The checking shows if there is an equivalence or preorder relation between the selected third party service and the selected component of the requested service. If the checking satisfies, then the selected third party service is checked for semantic matching. This is performed with the selected component of the requested service using a description logic reasoner like Pellet. This matching is again similar to semantic matching of concepts in the matchmaking procedure for fully matched services.

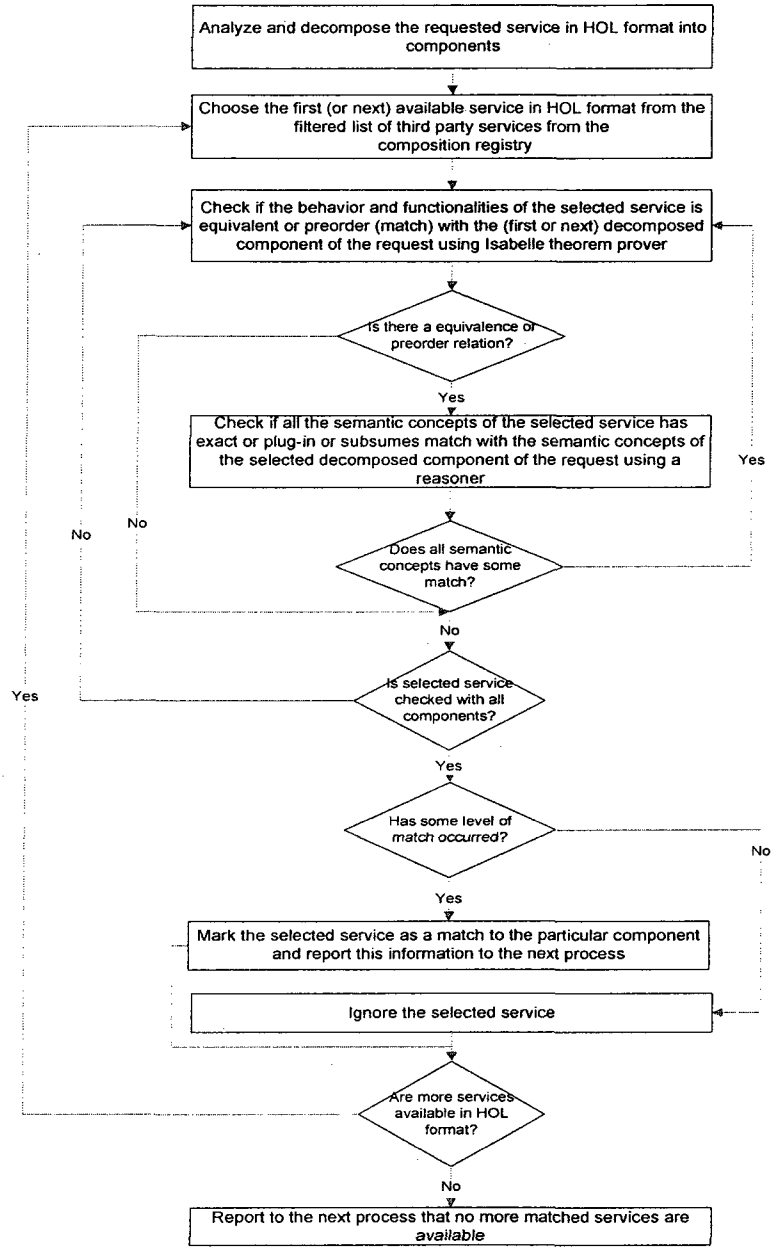


Figure 5.6. The Matchmaking Procedure for Partially Matched Web Services

If all the required semantic concepts match, then the behavioral, functional, and semantic match is performed with the next component, and this is repeated until a mismatch occurs. A decision is then taken to see if the service is checked with all the components. If it is not checked, then the service is checked with other components. After the selected service is checked with all components, it is then checked to decide if some level of

match has occurred. If yes, then the selected service is marked as a partial match to the requested service. This information is then reported to next module (service categorization process), and the next service, if available, is selected for matching. If the selected service does not partially match with the requested service, it is simply ignored, and the next third party service from the composition registry is chosen for matching.

When all the available services are checked, subsequently, this is reported to service categorization process. It could be noted from this procedure that non-functional parameters are not matched in the partial matchmaking procedure. This is because of the fact that this matching is done after the partially matched services are selected and assembled into a composite service. The service assembly technique determines the best-assembled service using non-functional matching.

5.5. Service Categorization Technique

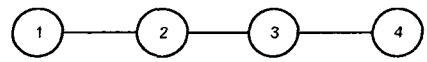
The service categorization technique categorizes the matched services to be used for composition. In this section, the basic concepts and principles of this technique are presented first, followed by a discussion on categories generation and categorization of matched Web services.

5.5.1. Basic Concepts and Principles in Service Categorization

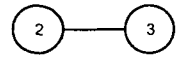
Service categorization uses a core concept called *categories* for categorizing the matched services. Categories are basically organizational elements used for classifying and arranging the matched services. The categories are dynamic, meaning the number of

categories and their addressing vary for every composition, and it is generated based on the requested service. The categories are accessed uniquely by their addresses.

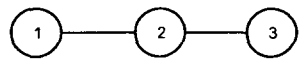
The addresses for the categories are required because the categories should be accessible uniquely for both the categorization technique to categorize the matched services, and also for the assembly technique to select and assemble the categorized services for composition. The addressing system has two-level addresses. The addressing is based on the level and sequence of the behavioral and functional matches. For explaining the concepts like level and sequence of match we use the graph-like representation of the Web service as shown below. In the graph, the nodes (numbered 1, 2, 3, and 4) represent the 'components' (and their functionalities) and the edges represent the behavioral operators. The graph below represents a requested service with four components connected by three behavioral operators.



The level of match means the number of components of the request that is matched by the third party service. The sequence of match means starting from which component till which component is matched by the third party service. For example, there can be a third party service which matches for the 2nd and 3rd component (functionally), and also uses the same behavioral operators between them as above. This partially matched service can also be represented using the graph-like structure. The level of match for this partially matched service is 2, and sequence of match for this service is also 2.



Similarly, the following representation of the service can be interpreted as a partially matched service, with a level of 3 and a sequence of 1. It should be noted that there cannot be partially matched service that satisfies only the 1st and 3rd component without satisfying the 2nd component because the matchmaking technique will not allow this.



The first-level address of the categories (represented as 'f') is based on the level of the behavioral and functional match of the matched service compared to the requested service. The second-level address of the categories (represented as 's') is based on the sequence of the matched service with respect to the requested service. The two level addresses are represented using square brackets as f[s].

The level and sequence of the behavioral and functional matching determines the placement of the matched services in different categories by basically finding the addresses of its category. The semantic level matching of the services, represented by semantic rank, determines the ranking of the services within each category. It should be noted that the details about matching are provided by the matchmaking technique. Semantic rank of the service is calculated based on the degree of match of the different semantic concepts of the matched service compared to the requested service. Exact, plug-in, and subsume matches are considered for semantic rank. It should be noted that these matching concepts are discussed in the last section.

5.5.2. Categorization of Partially Matched Web Services

The categories are generated using the procedure shown in Figure 5.7. This procedure is

invoked once for every composition request by the categorization procedure. The first and basic step in the categories generation procedure is finding the *component number*, which is generated based on the (virtual) decomposition of the Web service request into components. The decomposition is done using the behavioral operators. The number of components generated from the decomposition of the request is called the component number denoted by 'n'. For example, assume a Web service is requested with three behavioral operators that connect four components. This service is decomposed into four components based on the operators, and so the component number is 4. After finding the component number, the number of categories is generated to be equal to the sum of 2 to n. When n is 4, the number of categories generated is 9 (4+3+2).

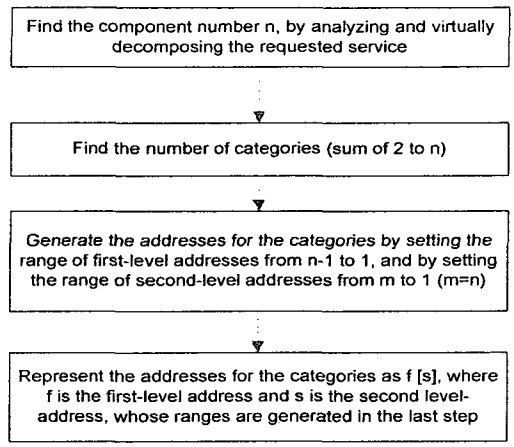


Figure 5.7. Categories Generation Procedure

The next step is to generate the addresses for the categories. The range of the first level address is from n-1 to 1; where n-1 is given to the services with next-best match compared to the full matches, and 1 to the least matching services. The range of the second level addresses is from m to 1, where m is the always the same value as n. Here, the value m is given to the last sequence(s) and the initial sequence(s) gets 1. The motivation for using different indexes (m and n) in the two levels is for easier reasoning

and understanding; actually both the values are equal. In the last step, the two level addresses are represented as f[s]. The addresses of the categories when n = 3 is 1[1], 1[2], 1[3], 2[1], and 2[2].

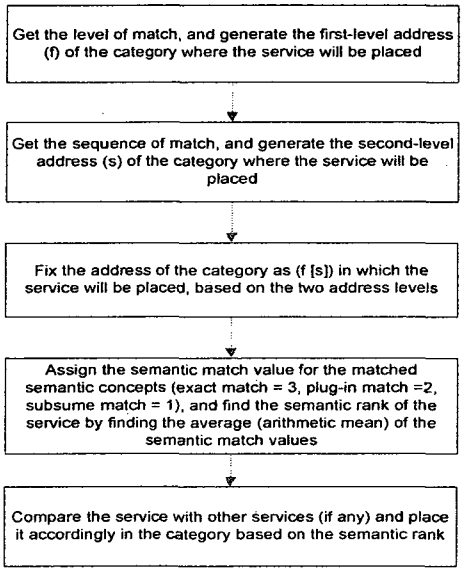


Figure 5.8. Service Categorization Procedure

The procedure for categorizing the matched services is shown in Figure 5.8. The first step is to find the address of the category in which the matched services will be placed. It is important to recall that information about the level and sequence of the behavioral and functional match. The details on semantic-level matching of the concepts are provided to this procedure by the matchmaking procedure. Based on level of behavioral and functional match the first level address (f) of the category in which this service is placed is decided. For instance, if the level of match of the matched service is 3, meaning the third party service matches three (distinct) sets of behaviors and functions with the requested service; then this service is placed in one of the categories whose first level address is 3. The second level address of the matched service (s) is determined by the sequence of the behavioral and functional match. For example, if the same matched

service sequentially matches the 1st three sequences, then its second level address is 1. Therefore, the address of this matched service is 3[1], and the service is placed in the category with this address. After finding the two address levels, the category address for the matched service is then fixed as f [s].

In the next step, the semantic rank is calculated by assigning the semantic match value for the individual semantic concepts. Every degree of match has its own match value. The exact match has the highest value of 3, the plug-in match has a value of 2, and 1 is the value of the subsume match. Semantic rank is the average of the semantic match values of all the individual semantic concepts. The semantic rank is generated after getting the degree of match of all the semantic concepts. Subsequently, the semantic rank of the matched service is compared with semantic rank of all the existing services in the category, to decide on its ranking within the category. If the matched service is the first (and only) one in the category, it is obviously listed as the first service.

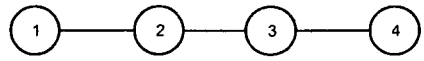
5.6. Service Assembly Technique

The service assembly technique uses the categorized services for selecting and assembling the composite service. The basic concepts and principles of the technique are explained in the first subsection. The subsection after that presents the technique. In the last subsection, non-functionalities based best-assembled service selection is discussed.

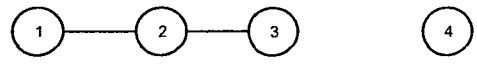
5.6.1. Basic Concepts and Principles in Service Assembly

In service assembly, mainly the categories are manipulated, which is a container of matched services, to assemble the compositions. We explain the concepts and principles of service

assembly using a simple example. Let us assume the requested service has 4 components joint with 3 behavioral operators represented with a graph-like structure below.



The request can be satisfied by a finite number of matching service combinations. For example, the request can be satisfied by assembling 2 services, where the first service matches for the first three components (with their behavioral operators) and the second service matches for the last component. We refer to these kinds of service combinations as combinable matched services. The categories in which these two combinable matched services are available are called as *combinable categories*. The combinable categories are represented using curly brackets. For example, the combinable categories that contains the graph-like services shown below can be represented as {3[1], 1[4]}.



In the same way, the requested service can also be satisfied by assembling 2 other matched services, where the first service matches for the first two components (with the behavioral operator). The second service matches for the last two components (with the behavioral operator), as shown below using the graph-like structure.

Similarly, there are five more combinations possible for this requested service. As these matched services are categorized by the categorization technique in different categories, basically the addresses of combinable categories in which these combinable matched

services are available has to be found. After that, all the possible service assemblies have to be generated from them.

It can be noted from the above combinations that for every combination there are some behavioral operators needed to 'simulate' the requested service behavior. For instance, in the first combination a behavioral operator is required to combine the 2nd service that matches for the fourth component with the 1st service that matches for the other three components. The assembly technique uses BPEL to orchestrate the combinable matched services, similar to how the fully matched service was orchestrated. All the partially matched services that make the composite service are used in the orchestration. BPEL constructs are used for orchestrating the categorized services, which can and cannot account for the requested operator behaviors.

Depending on how many matching services are available, many service assemblies are possible. Therefore, after all the possible assemblies are generated, the assembled services are ranked based on their non-functional matching to the requested service. The assembled service that has the best non-functional values is finally selected as the 'composite' service.

5.6.2. Assembling the Categorized Web Services

Service assembly technique uses concepts like *assembly value* and *replacement list* for manipulating the categories to create the assemblies. Assembly value is a value denoted by 'a' that helps in service assembly, and it is generated by dividing the component number 'n' by 2. In this procedure, we refer the categories with their levels, which are basically their first level addresses. For instance, categories of level 3 are categories whose first level address is 3. Replacement list is the list of all categories (that lie in the

levels less than or equal to a) which can be replaced with a combination of other lower-level categories relative to the replaceable category. For instance, when a = 2, one of the replaceable category in the replacement list is 2[1] which can be replaced with 1[1] and 1[2]. This means that a service which matches for 2 components starting from the sequence of 1st component can be replaced with services from 2 categories, where the service from 1st category matches for the 1st component and the service from 2nd category matches for the 2nd component. The replacement list is very useful in generating new assemblies from existing assemblies.

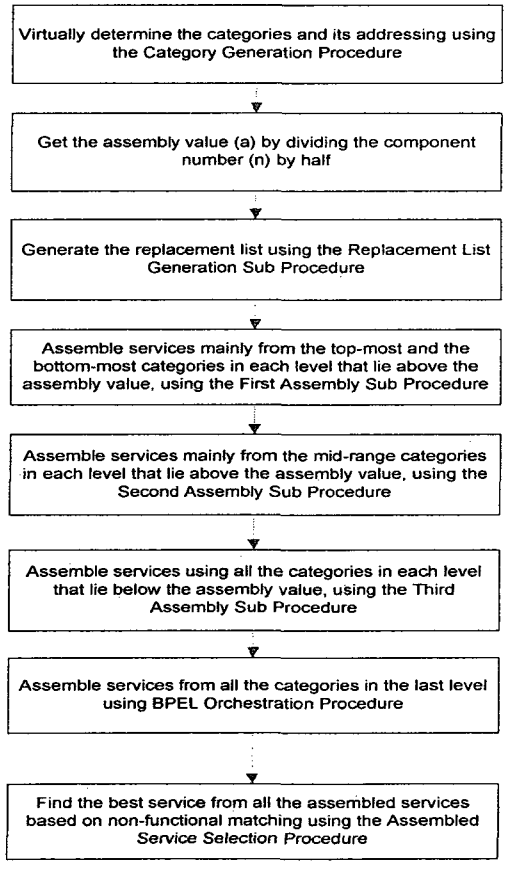


Figure 5.9. Service Assembly Procedure

The service assembly procedure is shown in Figure 5.9. This procedure uses many other procedures and sub procedures for assembling the composite services. Initially, it uses the

category generation procedure (presented in the last section) to virtually calculate the categories for internal manipulations. It should be noted that the categories and the categorized services in these categories are accessible by the assembly procedure seamlessly. Then, the assembly value is generated. The assembly value is rounded to the next number when the value of 'n' is an odd number.

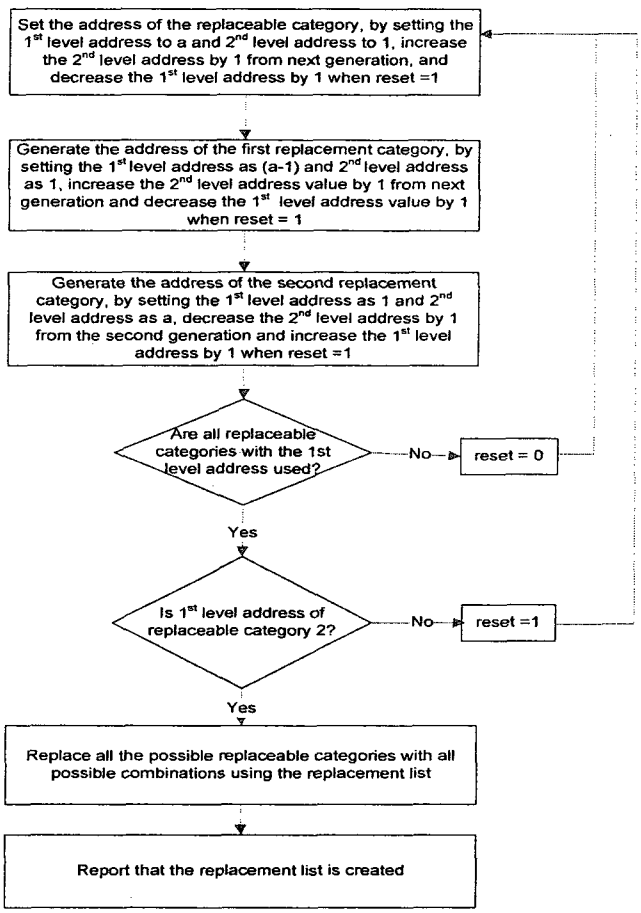


Figure 5.10. Replacement List Generation Sub Procedure

After generating the assembly value, the replacement list is generated using the replacement list generation procedure, which is shown in Figure 5.10. This procedure is simple which basically generates all the replaceable categories, similar to what is shown in an example in last paragraph. The address of the replaceable category is initially fixed, by setting the first and second level addresses to some value based on the assembly value.

In the next two steps, the address of the first and the second replacement category is generated by setting their 1st and 2nd level addresses, using the assembly value. It is then checked to see if all the replaceable categories satisfy a particular condition, if yes, then another checking is done to see if the replaceable categories 1st level address is 2. If this condition also satisfies, subsequently, all the possible replaceable categories are replaced with the possible combinations with the replacement list, and finally the replacement list is reported as successfully created. If both the previous conditions are not satisfied, then further replacement categories are generated.

Three sub procedures are used by the assembly procedure to generate assemblies from different categories. In essence, in all these assembly sub procedures, the categories that can be combined (combinable categories) to assemble the requested service are first generated. Then it is checked to find if any services are available in these combinable categories, if yes, then all the possible combinations are used for assembling with the BPEL orchestration procedure. After that, all the categories that could be possibly replaced with the replacement list are replaced. After that, again all the possible combination of the services in these categories is used for assembly generation with the BPEL orchestration procedure. It should be noted that these new generations must be unique, if any of the replacements already exist, then it is ignored. These steps are repeated till any of the categories level reaches a fixed value.

After generating the replacement list, first and second assembly sub procedures are used to generate assemblies from categories whose levels lie above the assembly value. The first assembly sub procedure is shown in Figure 5.11.

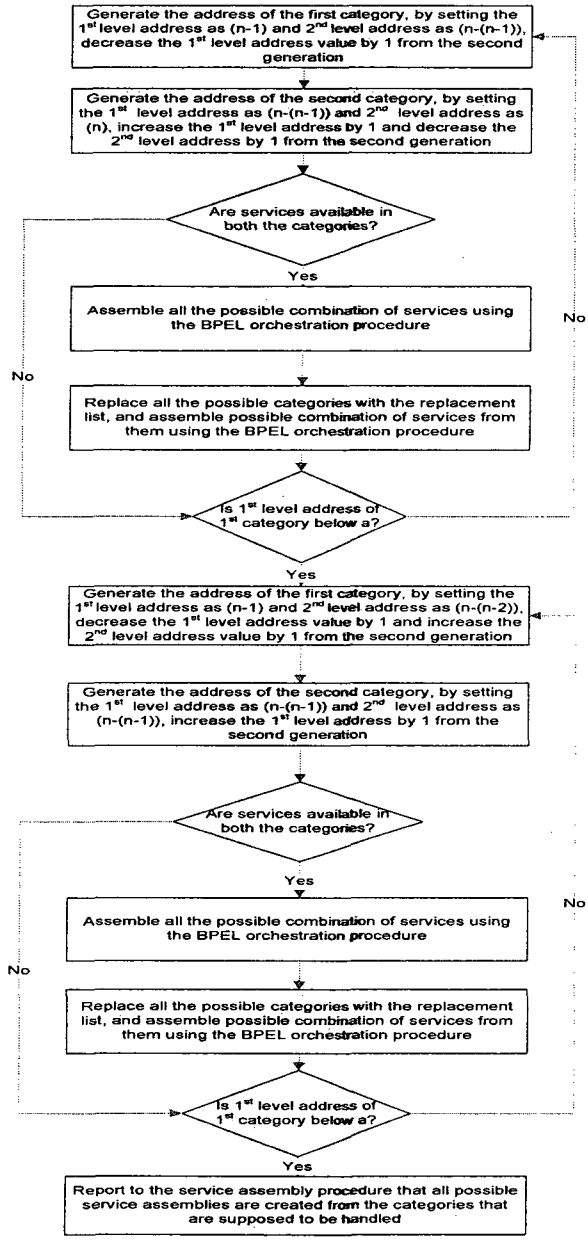


Figure 5.11. First Assembly Sub Procedure

Combinable categories and assemblies are generated from the first assembly sub procedure, by using the top and the bottom categories from each category level. For example, when n=3, using the first assembly procedure two combinable categories can be generated, which are {2[1], 1[3]} and {2[2], 1[1]}, where 2[1] is the top category in the 2^{nd} level, and 2[2] is the bottom category in the 2^{nd} level. If services are available in

these categories then all possible combinations of these services are generated, and then orchestrated using BPEL orchestration procedure. After that, using the replacement list, if new combinations are possible, then they are generated and orchestrated.

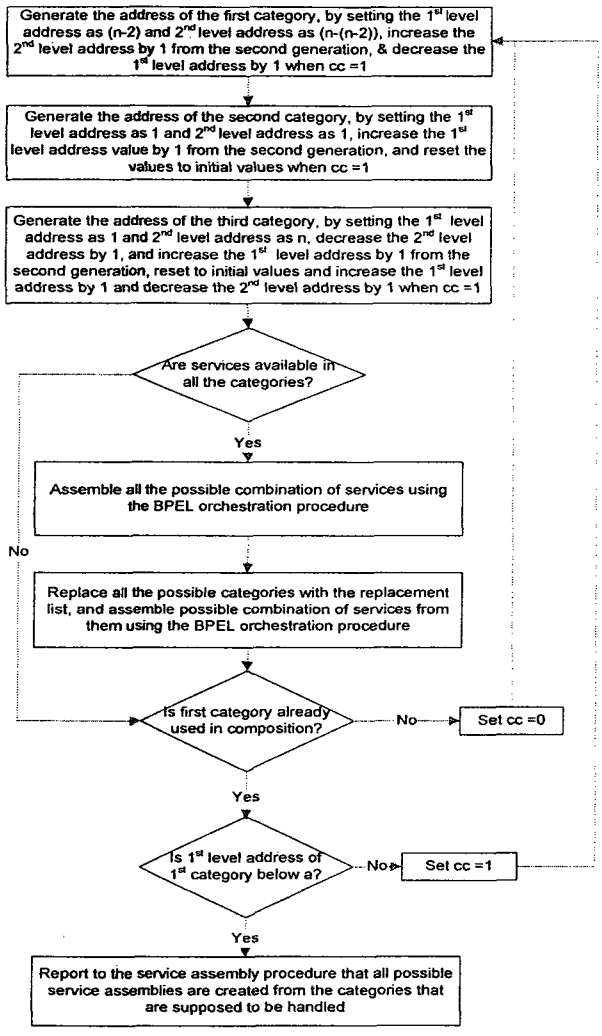


Figure 5.12. Second Assembly Sub Procedure

Second assembly sub procedure is used for generating assemblies from categories that lie in the mid-level categories of each category level. The combinable categories generated with this sub procedure uses three categories. This sub procedure is significant only when the number of components is greater than or equal to 4; otherwise, no combinable categories are generated using this procedure, as all the possible combinations are

generated by the first assembly sub procedure and from the last level categories. For example, when n=5, only one combinable category {3[2], 1[1], 1[3]} is generated using this sub procedure. It must be noted that, generally, the number of components in the requested service is 5 or less. However, theoretically, using these procedures work for any number of components.

The third assembly sub procedure is used for assembly generation from categories whose levels are below the assembly value, except the last level categories. The procedure is not presented here because the categories manipulation cannot be generalized for all the category levels. However, we present an example to show how they are manipulated and the combinable categories are generated. If services are present in these combinable categories then they are orchestrated using BPEL orchestration procedure. If further assemblies are possible with the replacement list, then they are also generated and orchestrated. It should also be noted that, assemblies are generated from this sub procedure when n >= 4. For example, when n=7, the five combinable categories generated are: {3[1], 3[4], 1[7]}, {3[2], 3[5], 1[1]}, {3[3], 2[6], 2[1]}, {2[1], 2[3], 2[5]}, and {2[1], 2[4], 1[1], 1[6]}.

An assembly can be generated from the last level categories by combining the services in all these categories together. Basically, the last-level categories has the services that matches only 'one component' of the request, meaning services in each last level category will match one component of the request individually. This assembly is then generated using BPEL orchestration Procedure. For example, when n=3, an assembly can be generated by the combinable matching services of {1[1], 1[2], 1[3]}, which are all last level

categories. When n=3, only two other combinable matching services are possible, which are generated from the first assembly sub procedure as {2[1], 1[3]} and {2[2], 1[1]}.

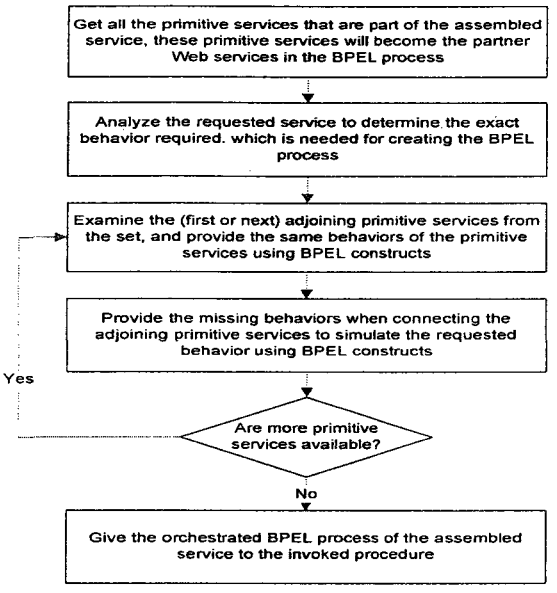


Figure 5.13. BPEL Orchestration Procedure

In the BPEL orchestration procedure (shown in Figure 5.13), all the matched services that make the assembled service are identified first. These services become the partner services for the BPEL process of the assembled service. The exact behavior of the requested service is then determined. After that, the first or next matched service is selected, and the same behavior provided by this service, if any, is provided using BPEL operators. If some behaviors are missing when joining this matched service with the next service, then they are also provided using the appropriate BPEL operators, so that the assembled service simulates the requested service. If more matched services are available, then the above two steps are repeated. Finally, the orchestrated BPEL process is provided to the procedure that invoked it. After all the possible assemblies are generated, they are ranked based on the non functional matching (presented in the next subsection), and the

best-assembled service from non-functional perspective is selected as the 'composed' service.

5.6.3. Selecting the Best-Assembled Service

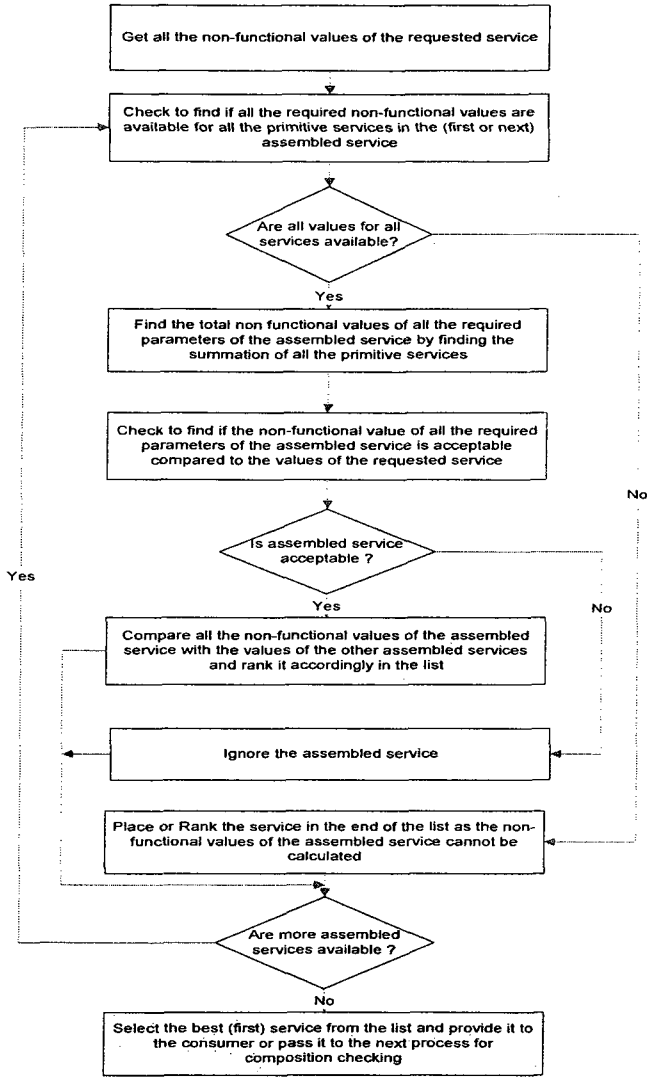


Figure 5.14. Non-Functionalities based Best-Assembled Service Selection Procedure

The best-assembled service selection procedure based on the non-functional matching is shown in Figure 5.14. Initially, all the non-functional values of the requested service are identified, and then it is checked to find if all the required non-functional values of all the primitive services of the assembled service are available. If all the values are available,

then the non-functional value of this assembled service is found by summing the numeric values for applicable parameters, and by selecting the lowest scale for other parameters. Subsequently, it is checked to find if these values are better than the non-functional values of the requested service. If the values are acceptable, then its values are compared with the non-functional values of other assembled services and then ranked in the list, else the service is just ignored. If more services are available for checking then the same steps are repeated; else the best-assembled service from the non-functional perspective is selected and given to the requester, or checked for composition using the (optional) composition checking procedure, when it is available. If, any of the values of any of the primitive services are not available then the non-functional parameter of the assembled is not generated, and this service is placed at the end of the list.

Chapter 6

Implementation and Case Studies

6.1. Implementation of the Proposed Business Model

The implementation of the business roles, the extended registry and the extended requester, are presented in this section.

6.1.1. Implementation of the Extended Web Service Registry

6.1.1.1. Prototype

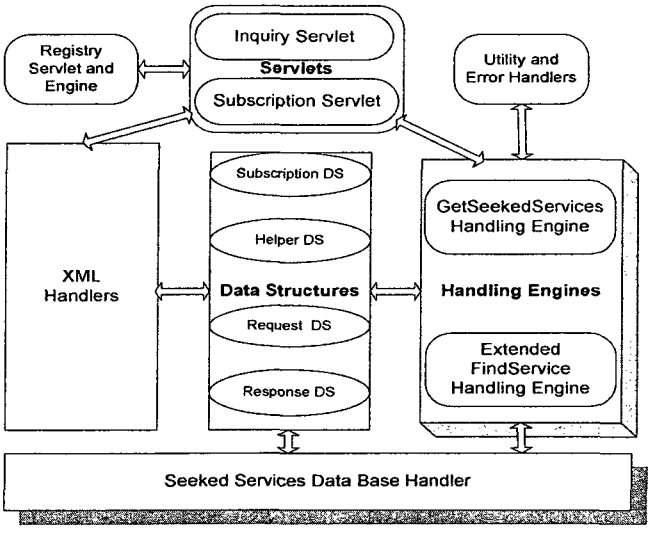


Figure 6.1. Architecture of the Extended Registry (jUDDI) Prototype

The registry prototype implements the synchronous (*get_seekedServices*) method of the extended subscription API, and the extensions to the *find_service* method. This proof-of-concept prototype is an extension of jUDDI. jUDDI [135] is an open-source Java implementation of UDDI that currently supports UDDI Version 2. However, it supports

some of the data structures and classes needed for the subscription API of UDDI V3. jUDDI is developed as a Java Web application, and it needs an external data store to manage the registry data. The reason for implementing only *get_seekedServices* method is that jUDDI only supported synchronous communication at the time of implementation.

The software architecture of the implemented extensions to jUDDI is presented in Figure 6.1. The GetSeekedServices Handling Engine handles the (*get_seekedServices*) request, implements the business logic to retrieve the information about the sought-for services using database handlers, and creates the response using the data structures. The extended *find_service* operation is processed by the Extended FindService Handling Engine, which implements the business logic to store the seeked services information. The four types of data structures are used by XML and data base handler to hold the information required to process the request and to create the response. XML handlers marshal (encode) and un-marshal (decode) the XML data. The Seeked Service Data Base Handler communicates, stores, and retrieves the seeked services information from/to the external database.

The Subscription Servlet extends the HTTP Servlet to handle the seeked services request. The utilities and error handlers are used by all the other modules (not all links are shown in Figure 6.1) for general assistance and to handle errors. Except for the Registry Servlet and Engine module, Inquiry Servlet module, and the Utilities and Error Handler module, all the other modules are extended with new classes and methods to support the extensions. The extended jUDDI uses Apache Axis for communication with the requester and the composer. Apache Axis [136] in an open source implementation of SOAP, and is essentially a SOAP engine.

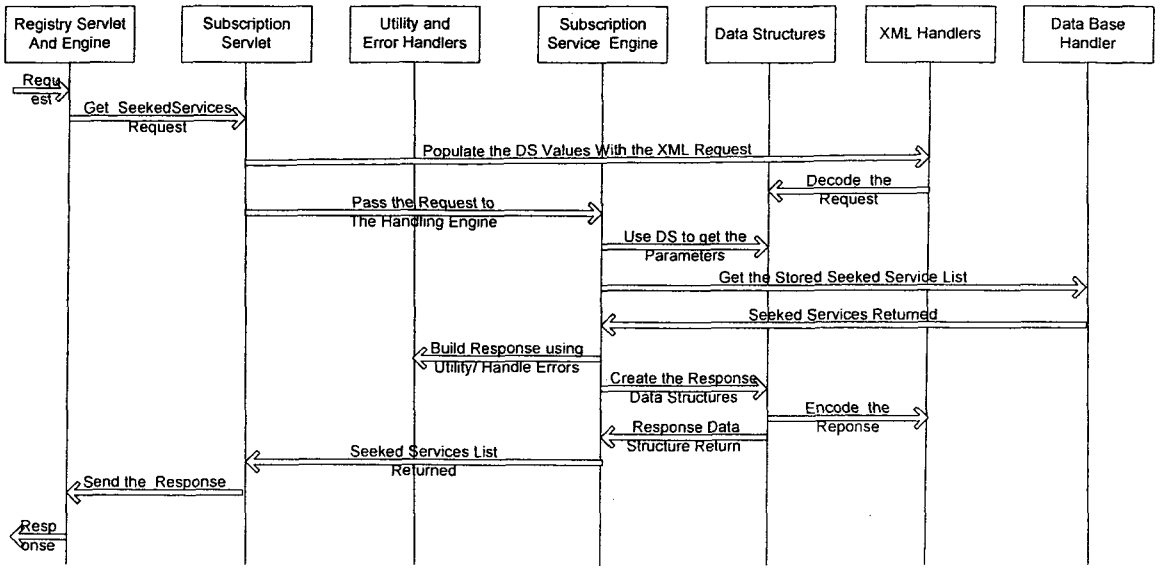


Figure 6.2. Processing of Get_seekedServices Request using the Different Modules

Figure 6.2 shows the communication between different modules/components when the *get_seekedServices* method is invoked by the composer. After passing through the Axis (SOAP) layer, the request is first received by the registry servlet and engine module, which passes the request to the subscription servlet module. The request is ultimately handled by the subscription service engine (GetSeekedServices Handling Engine). The XML handlers un-marshals the requests and populates the data structures that is used by the subscription service engine to process the request. The business logic allows the engine to query the seeked services data base handler to get the services seeked by requesters, which is used to build the response. The engine uses the utility (and error handling) module and the data structures to create the response. It should be noted that the XML handlers marshals the response, which is invoked by the data structures. Finally, the response is sent back to the composer through the subscription servlet, registry servlet and engine, and the SOAP handlers.

6.1.1.2. Performance Analysis

Experiments were performed to determine the performance of this prototype. The experiments that we carried out were to find services that do not exist in UDDI (with and without the extended operation), which is invoked by the requester. We also did experiments where the composer synchronously retrieves information about services sought by the requesters and that do not exist in UDDI. For these experiments, we used jUDDI's JSP console as the client (in case of both requester and composer), because when these experiments were performed the prototypes of requester and the composer were not developed. We extended the jUDDI's JSP console to support the new *get_seekedServices* method.

The performance measurements were taken with the extended jUDDI running on an Apache Tomcat 5.5 Web server. The jUDDI used a MySQL server 5.0 as a database. The whole system ran on a Pentium 4 2.99 GHz machine with 1 GB of RAM and a Windows XP platform. We used another machine with exactly the same configuration (2.99 GHz P4 with 1 GB RAM running XP) to run the clients (JSP console) remotely from the same LAN. Table 6.1 shows the response time and network load of the *find_service* method with and without extended operation, and also for the *get_seekedService* method. These values are average measurements over 15 trials. These measurements were not taken immediately after the Web server was (re) started, as this incurs more response time because of the Java virtual machine initializations.

Table 6.1. Network Load and Response time of the Extended Registry (jUDDI)

Functionality	Response Time (ms)	Network Load (kb)
find_service without extended operation	15.7	5.6

find_service with extended operation	60.5	5.7
get_seekedServices operation	18.5	5.7

From the measurements it is clear that the extended function of *find_ service* penalizes the response time (increased by 44.8 ms), but not the network load (increased only by 0.1 KB). The *get_seekedServices* method incurs an acceptable response time and network load, which is comparable (with an increase of just 2.8 ms response time and 0.1 KB of network load) to the values of the *find_service* method without extended registry operation. It is evident from this analysis that the extensions to the UDDI are indeed very useful from the requester's and composer's perspective, with acceptable penalization to the registry's performance when the synchronous communication mode is used for getting the seeked services.

6.1.2. Prototype of the Extended Web Service Requester

The Web service requester prototype implements the functionality required to invoke the extended *find_Service* method of Inquiry API, and implements the Get-Give API. It also creates the detailed composite request. This prototype is implemented from scratch in Java. However, it extends and uses UDDI4J for invoking the extended *find_service* method at the registry (the extended jUDDI). UDDI4J [137] is an open source Java library that allows interacting with UDDI registry. It uses Apache Axis for communication.

We make simple extensions to UDDI4J to add a new parameter, consumer address (SOAP endpoint for composer to communicate), in the *find_service* request, and to support the new parameter, note (related to information about composer), in the

response. A new data type 'consumer address' is also added to UDDI4J for manipulating the consumer address parameter and to marshal and un-marshal this data. The prototype uses these extensions to communicate with the extended jUDDI using the extended *find_service* method. It should be noted that the address given in the consumer address parameter is an actual SOAP endpoint, which implements the Get-Give API.

The Get-Give API is implemented as a Web service by the requester. Apache Axis 2 is used for communication between the requester and the composer. Axis 2 [138], like Axis, is an open source implementation of SOAP, and it is a Web service engine. However, it is redesigned and redeveloped completely based on the lessons learned from Axis, and it is more flexible and efficient compared to Axis. We use Axis2/Java, the Java implementation of Axis2. The *get_CompositionRequest* and *give_CompositionRequest* methods are implemented synchronously, where the *get_CompositionRequest* message is sent as a reply to the *get_CompositionRequest* method. AXIOM (AXIs Object Model), [139] the light-weight object model of Axis2, is used for processing the get and give messages.

The implemented classes are compiled and wrapped as an Axis2 application and it is deployed in the Axis2 container. Axis2 runs as a Web application in a Web server. This deployed application (the requester) basically receives the get message and responds to it with the give message having the detailed composition request. It should be noted that (a part of the) requester also runs as a standalone application to invoke the *find_service* method.

The interactions from the requester's perspective start when the requester tries to find a service using the extended *find_Service* method. In order to communicate with the

composer, the requester (application) is deployed as a Web service before invoking the *find_service* method. If the service seeked does not exist in the registry, the extended registry responds with no-service found message, which has the note that composer(s) might contact the requester for composing the needed service. The requester then creates the detailed composition request, which in the current prototype is selecting a default request. When the composer sends the *get_CompositionRequest* message, the detailed composition request is sent as response in the *give_CompositionRequest* message.

The software architecture of the extended requester is shown in Figure 6.3. As mentioned before, it uses UDDI4J and Axis2 for its working, and Axis is used by UDDI4J for communication. The three classes extended/introduced in UDDI4J are shown in the architecture. The consumer address class is the new class, and the other two classes (FindService and ServiceList) are extended to support the new functionality. The Extended Find_Service Invoker implements the logic to invoke the extended find method and for handling the response. The composition request creator creates the composition request. The Get-Give API Implementer has the necessary logic to receive the get request and processes it. It also has the logic to send the give message by using AXIOM.

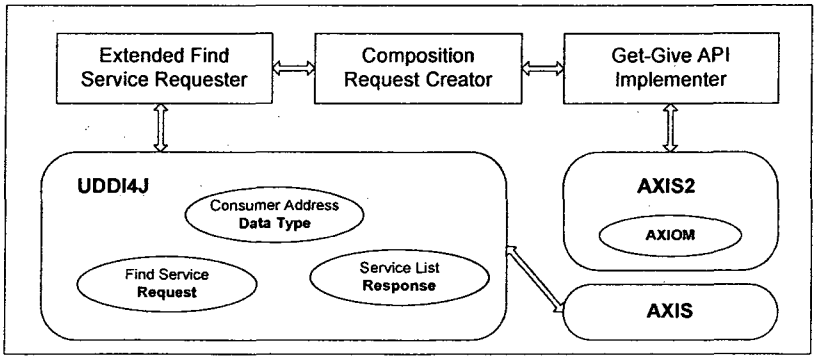


Figure 6.3. Architecture of the Extended Requester Prototype

6.2. Implementation of the Proposed Composition Framework

6.2.1. Prototype

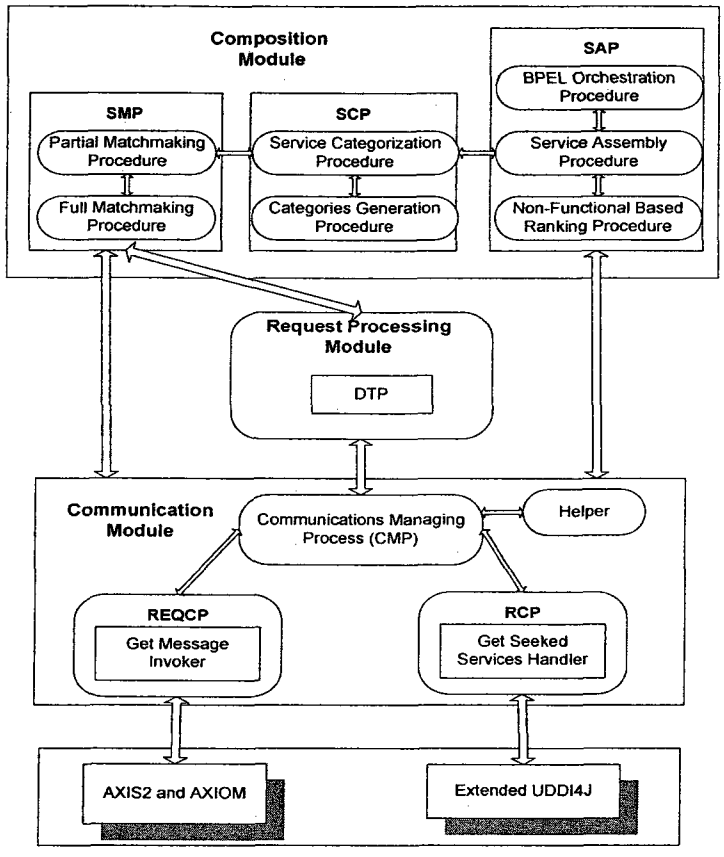


Figure 6.4. Architecture of the Composition Framework Prototype

A part of the composition framework and its techniques has been implemented, as a proof-of-concept prototype in Java. The architecture of this prototype is shown in Figure 6.4. It implements the communication module, the composition module, and a part of the request processing module. The matchmaking technique, the categorization technique, and the assembly technique, which are basically the building blocks of the composition module are implemented. It should be noted that the parts that are not implemented are not essentially required for composing Web services in the framework. The execution time adaptation module is not implemented, as it is not used for composition. The

optional service composition checking process and feature interaction checking sub-process of the composition modules are not implemented, as they are only used for verifying the composed service. Moreover, the c-registry communication process of the communication module is not implemented, as locate and put interactions have not been concretely realized.

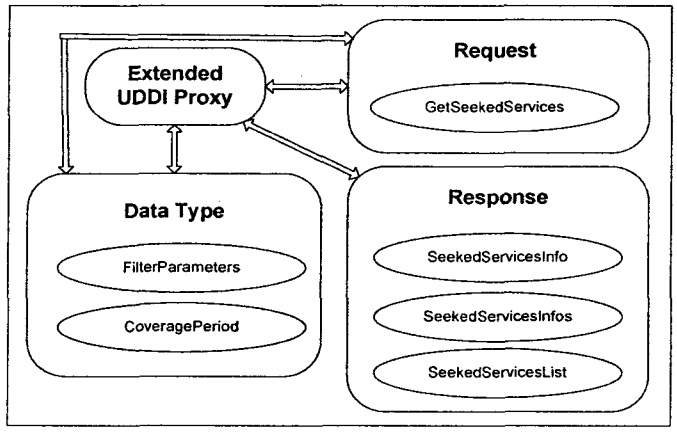


Figure 6.5. Architecture of the Extended UDDI4J for Getting Seeked Services

The RCP of the communication module implements the synchronous *get_seekedServices* method from the composition framework's perspective. It essentially invokes this method at the extended jUDDI and processes the response. It extends and uses UDDI4J for this implementation. The architecture of the extended UDDI4J for getting and processing the seeked services from the registry is shown in Figure 6.5. A new request class for handling and processing the *get_seekedService* method, called as GetSeekedServices is added. Three new classes for handling the response, which has the information about the services seeked by the requesters and their endpoints for communication, are also added.

Two new data types are also added. The basic UDDI client (class), which is a proxy to UDDI called as UDDI Proxy, is extended to handle the *get_seekedServices*

method. The extended UDDI proxy uses the new request, response, and data type classes to process the method.

The REQCP of the communication module invokes the Get-Give API. Similar to the extended requester, the REQCP also uses Java implementation of Axis2 for sending and receiving SOAP messages, and also uses AXIOM for data manipulation. The CMP of the communication module initiates the composition framework by using the RCP for invoking the *get_seekedServices* message. CMP uses a helper component for aid in communications. It then uses the SOAP endpoint of the requester that is got in the *get_seekedServices* response to communicate with the requester by using the REQCP. The response from the requester has the detailed composition request, which is provided to the DTP of the request processing module. Based on the composition request, the DTP basically selects one of the pre created HOL description file and passes it to the SMP of the composition module.

The full and partial service matchmaking procedures of the SMP are implemented. In both these matchmaking procedures, we first check for the behavioral match, then for the functional match and after that we check for the semantic match. In the case of full service matchmaking procedure we also check for the non-functional match. We do not use Isabelle theorem prover for matching. This is because of the fact that there is no interface available for integrating Isabelle with Java. Isabelle is generally used as a standalone theorem proving system. Consequently, we use the basic string matching in the case of behavioral match. In the case of functional match, we check for the number of parameter match, and also the basic 'type' matches. This is because Isabelle theorem prover cannot be used here. Semantic match is performed using Pellet. Pellet [133], an

open source OWL DL reasoner implemented in Java, is used by our prototype for semantic reasoning. Non functional checking is simple and it is performed using the basic logical and numerical operators of Java.

The third party services in HOL format are used from a particular local directory, and they are not discovered from the composition registry. The requested service and the third party services are manipulated and matched as tokens by using Java string tokenizer utility. The fully matched service orchestration procedure that creates the BPEL process of fully matched service is also implemented.

The service categorization procedure and the category generation procedure of the SCP are implemented. A special data structure called MultiValueMap is used for categories representation and manipulation. MultiValueMap enables having more than one value for a key. It is a part of the Collections package of Apache Commons [140], which develops reusable Java components. MultiValueMap was chosen as it allows categories (identified by the key) to hold multiple partially matched services (the values).

In the SAP, all the procedures of the assembly technique are implemented. The main service assembly procedure and its sub procedures are implemented. The BPEL orchestration procedure is implemented, which creates a BPEL process automatically using the partially matched services. The non-functional based ranking procedure is also implemented for choosing the best-assembled service.

6.2.2. Performance Analysis

Two set of experiments were performed with the composition framework prototype. In the experimentations, the requester and the registry prototypes presented in the last section were used for getting the detailed composition request. In the first set of

experiments, a fully matched service that was available from a list of third party services was orchestrated as a BPEL process and it was provided to the CMP. Partially matched services were found, categorized, and assembled using the respective procedures, and finally the best-assembled service (a BPEL process) was selected and provided to the CMP in the second experimentation. In both sets of experiments, the response time is the only metric that was measured, as it is the key for analyzing the performance of the framework, and it was the easiest to measure.

A presence service is used as the requested service in all the experiments. Presence service generally allows entities to find the willingness and availability information of other entities for communication. This particular presence service used for experiments allows getting willingness, availability, and location information of two users, and it is similar to the service used in the case study in the next section. Ten 'similar' third party services were used for the experiments, where in the case of first set of experiments, a fully matching 'exact' presence services was used, and it was replaced with another service in the second set of experiments.

In all the experiments, the composition framework communicated with the registry using the RCP, as expected. Using the SOAP endpoint received from the registry prototype the framework communicated with the REQCP, and got the composition request in the SAWSDL-MSC-NFSL format. In the first set of experiments, the composition framework prototype identified the fully matching presence service correctly. When this service was replaced with the other service it (correctly) failed to find the fully matching service and invoked the partial matching procedure.

In the case of second set of experiments, we chose the third party services in a way so that two compositions were possible. The first composition used three individual services: a willingness-providing service, an availability-providing service, and a location-providing service. The second composition was possible by using, a presence service that provides willingness and availability information together, and a location-providing service.

The partial matchmaking procedure identified these four services. After that, they were rightly categorized in their respective categories by the categorization procedure. The categorized services were then manipulated and the two possible compositions were identified. Based on the non-functional matching, one of the two service sets were orchestrated as BPEL process. Here, again, we changed the non-functional values of the partial matching services, so that in the two sub-sets of experiments, one of the two possible compositions was selected as the best service and then orchestrated.

The performance measurements were taken with the extended registry (jUDDI), the extended requester, and the composition framework running on a same machine, which had a Pentium 4 2.99 GHz processor with 1 GB of RAM and a Windows XP platform. It should be noted that this setup does not change the communications response time measurement, compared to running these entities in different machines in the same LAN. The jUDDI was running on Apache Tomcat 5.5 Web server, and used MySQL server 5.0 as the database. SOAP over HTTP was used for all communications. Table 6.2 and 6.3 shows the response time measurements. These values are average measurements over 10 trials. The measurements in Table 2 are averaged by performing 5 trials using two experiment sub-sets. It should be noted that these measurements were not taken

immediately after the Web server was (re) started, as this incurs more response time because of the JVM initializations. In the case of both Tables, the total response time (value in last row) is calculated based on when final orchestrated service is given to CMP and not to requester.

Table 6.2. Response Time Measurements When Finding a Fully Matched Service

Operation (Component) of the Composition FW	Response Time (ms)
Communication with the Registry (RCP)	1266
Communication with the Requester (REQCP)	397
Communications Management (CMP)	839
All External Communications	2502
Full Matching (SMP)	2594
BPEL Orchestration of the Fully Matched Service	33
Time to Find and Orchestrate a Fully Matched Service	2627

Table 6.3. Response Time Measurements When Composing a Composite Service

Operation (Component) of the Composition FW	Response Time (ms)
Communication with the Registry (RCP)	1264
Communication with the Requester (REQCP)	384
Communications Management (CMP)	836
All External Communications	2484
Partial Matching and Service Categorization (SMP & SCP)	3610
Service Assembly (SAP)	274
Time to Create a Composite Service	3884

It can be observed from the measurements that, the communications measurements in the case of both set of experiments are rather similar. The communication response time with the requester (0.39 sec) is lesser compared with the registry communication (1.265 sec), because of the complex levels of processing involved and the database interactions. As the communications managing process manages all the communications, marshals and un-marshals the data received from both the registry and the requester, its response time is quite high (0.837 sec).

The total response time for finding a fully matched service and then to orchestrate it as a BPEL process is 2.627 sec, which is only 0.125 sec more than the time to get the detailed composition request (2.502 sec). Semantic matching is the main match type that consumes a significant amount of time, as it involves reasoning about the semantic concepts that are from the ontologies. The average response time for checking if two semantic concepts have either equivalent or plug-in or subsume relation is 50-100 milliseconds. The presence service used in the experiments had six semantic concepts, so the semantic matching takes a sizable amount of time. It is important to note that during these experiments, the fully matched service is always placed as the 8^{th} service among the 10 third party services. However, the first 7 services do not have a behavioral match, so for none of these services the semantic matching is performed.

The response time for partial matching in these experiments is more than the full matching (increase of 1.016 sec), because four partially matched services are found (meaning more semantic matching is done) and also the amount of processing involved is more. In addition, this response time also includes the categorization time. The total time to create a composite service is relatively higher than finding and orchestrating a fully

matched service with an increase of 1.257 seconds, but when it is compared with the time to get the detailed composite request where there is a 1.4 second increase, the time is reasonable. Moreover, as we match and compose services by considering all their characteristics, then this response time is fairly good.

6.3. Case Study 1: Presence Service

In this case study, we describe a presence service that allows finding the willingness and availability information of two users. This information can be used for setting up communication between the two users. The service takes the address of two users, and if both are available or willing to communicate it gives a positive response, and negative otherwise. A potential user for this presence service is third-party call control services.

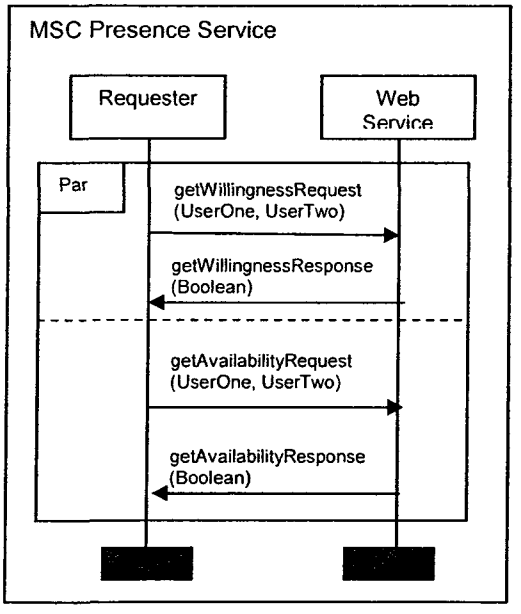


Figure 6.6. MSC Description of the Presence Service

The consumers and other entities describe the Web services using SAWSDL, MSC, and NFSL descriptions. It should be noted that the requesters are generally not end-users; they could be applications or other (software) entities. The three descriptions are then

mapped into HOL theories. The MSC description of the presence service is depicted in Figure 6.6. It shows the set of interactions and their sequencing, by which the requester can interact with the presence Web service. The behavior describes two response messages (getWillingnessResponse, and getAvailabilityResponse) sequentially following their respective request messages (getWillingnessRequest, and getAvailabilityRequest). The message pairs can also be invoked in parallel. It should be noted that the same MSC description can also be specified using its textual syntax.

The messages are described in a SAWSDL document, which is not provided here. In the SAWSDL description of the presence service, the parameters of both the request messages (UserOne, and UserTwo) are described using semantic concepts from some ontology. Figure 6.7 shows the non-functional description of the service using NFSL. <NFSL>, and </NFSL> marks the beginning and end of the definitions, where the Web service is identified by its name, '*PresenceService*'. It should be noted that the Web service name used here is the same name as used in the SAWSDL document to identify the service. The service has a cost of two USD and high reliability.

```
<? Xml version="1.0" encoding="utf-8"?>
< NFSL>
<WSName> PresenceService  </WSName>
<Cost> 2 USD </Cost>
<Reliability> High </Reliability>
< /NFSL>
```

Figure 6.7. NFSL Description of the Presence Service

The MSC, NFSL, and SAWSDL descriptions of the presence service presented above are mapped to HOL theories. First, the semantic and functional descriptions are mapped to the formalized SAWSDL concepts, then the behavioral description is mapped to the formalized process algebra semantics of MSC, and finally the non-functional description

is mapped to the formalized NFSL concepts. The semantic and functional description of the service is mapped by instantiating the records of the SAWSDL formalization.

We show the mapping of only parts of the SAWSDL description, as the full mapping is quite large. The concrete record definition of *'element1'* is of type *'wsdlTypeElement'*, which maps the first parameter in the 'getAvailabilityRequest' message. It maps *'UserOne'* of type *'string'*. It is also mapped with the *'modelReference'* attribute using an ontology concept of 'MachineIP' (defines IP address of a machine) from 'TelecomServices.owl' ontology available at some URL. "http://users.encs.concordia.ca/~r_karuna/Ont/TelecomServices.owl#MachineIP" shows this ontology concept in the definition. The lifting and lowering schema mapping attributes are not mapped as they are not specified in the SAWSDL description of the presence service, and they are shown as empty strings ('').

```
definition
  element1 :: wsdlTypeElement
   where
  element1_def: "element1 = (| elementName = ''UserOne'',
elementType = string, modelReference =
''http://users.encs.concordia.ca/~r_karuna/Ont/TelecomServices.ow
l#MachineIP '', liftingSchemaMapping = '''',
loweringSchemaMapping ='''')"
```

Similar to the above definition, *'messagerequest1'* of type *'message'* definition maps the 'getAvailabilityRequest' message, which uses the *'part1'* definition that has *'element1'* definition as one of its attributes.

```
definition
  part1 :: part
  where
  part1_def: "part1 = (| partName = ''getAvailabilityRequest '',
partType = '''', partElement = element1, modelReference = '''',
liftingSchemaMapping = '''', loweringSchemaMapping = ''''|)"
definition
  messagerequest1 :: message
   where
  messagerequest1_def: "messagerequest1 = (| messageName =
''getAvailabilityRequest '', messagePart= part1|)"
```

The operations *'getAvailability'* and *'getWillingness'* makes the port type definition (see the *'porttypeOperation'* attribute) of the presence service, called as *'presenceService'*. It can be noted from the *'getAvailability'* definition that *'messagerequest1'* is the input message and this operation is of type request-response.

```
definition
  getAvailability :: operation
  where
    getAvailability_def: "getAvailability  = (| operationName =
''getAvailability '', operationType = requestResponse,
operationInputName ='''', operationInputType = messagerequest1,
operationOutputName = '''', operationOutputType =
messageresponse1, operationFaultName = '''', operationFaultType =
emptyMessage, operationExtension = '''' |)"
```

```
definition
  presenceServicePortType :: portType
   where
    presenceServicePortType_def: "presenceServicePortType = (|
porttypeName = ''presenceService'', porttypeOperation =
''getAvailability, getWillingness'', modelReference = '''' |)"
```

The SAWSDL specification of the other parts of the presence service is also mapped similar to the aforementioned definitions, as every definition is a record concretization.

The behavioral description of the presence service is mapped to Isabelle/HOL using the formalized concepts of CSP-Prover as specified below, where the operators ($, ;;, ||) are described in subsection 4.4.2. The messages (messagerequest1, messagerequest2, messageresponse1, and messageresponse2) that are used in the behavior description are of course the messages specified by concrete record definitions of SAWSDL. It should be noted that the $ symbol, which represents the processes, can be used to map the messages as processes, as specified in the definition below.

```
WS_def: "WS == (($messagerequest1 ;; $messageresponse1) ||
($messagerequest2 ;; $messageresponse2))"
```

The non functional description of the service is mapped easily by instantiating the record that formalizes the NFSL concepts. The non-functional description of the presence service in HOL is given below, where the cost is specified as 2 USD with high reliability.

```
definition
  NFofPresenceService :: NFSofWS
  where
```

```
NFofPresenceService_def: "NFofPresenceService = (|
serviceName = ''presenceService'', cost = 2, responseTime = 0,
availability = 0, security = none, reliability = high, reputation
= none |)"
```

All the aforementioned HOL specifications are integrated in a single Isabelle theory file, which imports all the other Isabelle theories that formalizes SAWSDL, MSC, and NFSL. This description of the presence service as an Isabelle theory basically integrates the different characteristics of this service in Isabelle/HOL.

6.4. Case Study 2: Dating Service

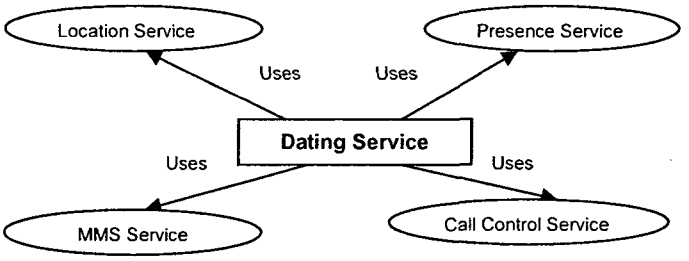


Figure 6.8. The Dating Service

In this case study, we illustrate the architecture, the composition framework, and its techniques using a dating service, which was presented in the Introduction. This service is composed of a call control Web service, a presence Web service, a location Web service, and a MMS Web service, as shown in Figure 6.8. We illustrate the creation of this composite Web service using the framework, starting from the moment the requester searches for the service in the business model. We make two assumptions. First, to illustrate the composition, we assume that a similar dating service does not exist in both the registries of the model. Second, we assume that the third party service provider(s)

have all the primitive services that would make up the requested composite Web service, in HOL format. Moreover, we also assume that these services are published in the c-registry.

Initially, the composition framework registers with the registry using RCP to get notified about the services sought by the requesters and which does not exist in the registry. When the Web service requester tries to find the dating service from the Web service registry, it gets a 'no-service found' response. However, the requester provides an endpoint (its SOAP address) in the find query, where the composition framework can contact it. The registry then informs the RCP about the possibility of dynamically composing a dating service, and gives the requester's endpoint for further communication. The CMP gets this information, and then invokes REQCP to communicate with the requester to get the detailed request. The requester responds using the give interaction with the functional (semantic), behavioral, and non-functional description of the dating service in SAWSDL, MSC, and NFSL descriptions respectively.

The MSC description of the dating service is given in Figure 6.9. It consists of a scenario of interactions the requester is expecting from the composite dating Web service. The description of the messages used in the MSC is provided in a SAWSDL document (not shown here), along with the (domain) semantics of the message parameters using ontologies. The framework now has the detailed composition request to start its internal activities. Subsequently, REQCP forwards the request to the DTP of request processing module.

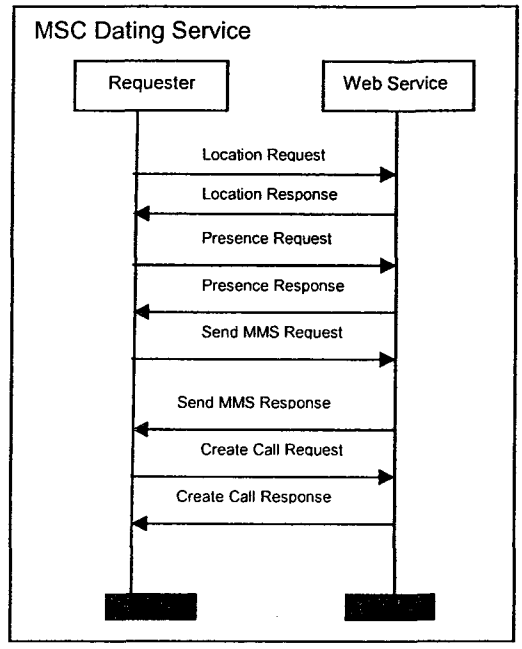


Figure 6.9. The MSC Description of the Requested Dating Service.

The description transformation technique of the DTP maps the SAWSDL, MSC, and NFSL descriptions of the dating service to HOL terms and formulas. The MSC description of the dating service mapped as HOL formulae is shown below.

```
($Location Request ;; $Location Response) ;; ($Presence Request ;;
$Presence Response) ;; ($Send MMS Request ;; $Send MMS Response) ;;
($Create Call Request ;; $Create Call Response)
Where ;; is sequential composition operator
```

The transformation technique first maps the MSC concepts to process algebra concepts, and then maps the process algebra concepts to Isabelle/HOL concepts. The semantics of the sequential composition operator shown in the above HOL formulae is encoded in Isabelle/HOL by CSP Prover. Similarly, the transformation technique also maps the SAWSDL and NFSL description of the dating service to Isabelle/HOL

concepts, based on our formalization of SAWSDL and NFSL concepts in Isabelle/HOL. These mappings allow integrating the different descriptions of this dating service in a single Isabelle theory, like the presence service shown in last section.

The composition module uses the Isabelle/HOL theories of the third party services to create the requested composite service. The composition request is forwarded to the SMP, which uses its service matchmaking technique. We extend our assumption further that by using the existing keyword-based discovery mechanism of UDDI (with the keyword 'telecom services'), we get a list of 20 services from the composition registry that could potentially be checked for matching. The 20 services are further assumed to be a mix of: two location services, two SMS services, four call control services, three conferencing services, two MMS services, two account management services, one terminal status service, two third party call control services, and two presence services.

The 20 services are first checked for full match using the full service matching procedure. If we assume the first service in the list to be a location service, then it is selected and checked for the functional and behavioral matching using Isabelle theorem prover. It is basically checked if there is an equivalence or preorder relation between the location service and the dating service. The location service does not have the expected relation with the dating service, as it offers less than what is expected, so this location service is ignored. Similarly, the next service is selected and checked until all the 20 services are exhausted.

Subsequently, the partial service matchmaking procedure is invoked to check the 20 third party services for partial matches. In this procedure, the dating service request is

first analyzed and the request is decomposed into four component services (subparts) based on the sequential composition operator. It should be noted that though sequential composition operator is used for both the request-response message binding, and also to bind the different parts of the overall behavior, they can be individually identified, as these mappings are performed by the DTP. The decomposed components are location request-response, presence request-response, send MMS request-response, and create call request-response.

The first service (a location service) from the list is then selected and matched for the behavior and functional aspects, with the first (location) component of the requested service. This matching is performed with the Isabelle theorem prover to find if there is an equivalence or preorder relation. We present below in detail how the equivalence checking is performed.

```
lemma Web_service_equivalence_proof: "($LocationRequest ;;
$LocationResponse) =F ($getLocationRequest ;; $getLocationResponse)"
```

In the above lemma named *'Web_service_equivalence_proof'*, the requested service behavior is checked for failure equivalence (=F) with the selected service behavior, which is the part in the lemma after the failure equivalence operator. The equivalence can be proved by data definition unfolding and by simplifications. The first two steps of the theorem proving are shown below.

```
apply (unfold LocationRequest_def LocationResponse_def)
apply (unfold getLocationRequest_def getLocationResponse_def)
```

The first command tells Isabelle to unfold the definitions of *'LocationRequest'* and *'LocationResponse'* using their definitions. It should be noted that the definition of

'LocationRequest' is defined using *'LocationRequest_def'* in its Isabelle theory (not presented here). Basically, the definition of this message is replaced with the message name in the lemma. Similarly, the other definitions are also applied. The second command is exactly similar to the first, except the fact that it is applied using the definitions of the selected service. In the same way all the definition of the other messages are also unfolded.

Simplification is one of the theorem proving methods available with Isabelle [14]. In simplification, term rewriting is done, where some equations are repeatedly applied from left to right. The simplification method is invoked using the keyword 'simp'. Rules can be added or deleted for simplification with the 'add' or 'del' keywords following the 'simp' keyword. We use simplification with addition to prove the rest of the equivalence, one such step is shown below.

```
apply (simp add: LocationRequestPart_def LocationResponsePart_def)
```

The above command basically applies the simplification rule by expanding the part definitions of *'LocationRequest'* and *'LocationResponse'*. Similarly all the other messages are simplified using the part definitions. In the next steps, the element definitions are applied to the lemma using more simplification rules, one such step is shown below.

```
apply (simp add: element1_def element2_def element3_def element4_def)
```

The above command simplifies the lemma by applying the element definitions. When two services are equivalent it can be proved with these simplifications and data definition unfolding. When the equivalence is proved, Isabelle displays a 'true' message, which

tells that the lemma is proved as true with Isabelle. The proof and the lemma can be stored, and the lemma can be used wherever needed by using its name. In our case, as both the services are 'equivalent' we will get a 'true' message from Isabelle. It is important to note that other Isabelle tactics and laws from CSP-Prover [86] can also be used for proving the equivalences or refinements, when needed.

The selected location service is then checked for semantic concept matching with the location component of the dating service using Pellet. Here, we assume that location request message has two parameters (UserOne and UserTwo), which are described using the ontology concepts from an ontology. Moreover, we assume that the same (standard) ontology is also used by the requester to describe the semantic concepts in the dating service. The semantic checking of the two parameters of the location service is done with the two parameters (MachineOne, MachineTwo) of the location component of the dating service. This service is selected as a semantic match, because the two semantic concepts in the location service have a plug-in match with the location component.

The location service is then checked to find if it matches further with any other components of the requested service, however, this checking fails. The service is marked as partial match and details of the matching are given to the categorization process. The sequence and the level of the behavioral match (matching of one component, which also happens to be the first component), and the semantic concept matching of concepts (plug-in match for both concepts) are the details that are provided to the categorization process. The service categorization process uses the service categorization procedure for categorizing the partially matched services. The categorization procedure initially generates the categories and their addresses with the dating service request using the

categories generation procedure. The categories generation procedure analyzes and decomposes the composite request and finds the component number as 4. It also determines the number of categories as 9. The addresses of the categories are then generated and represented as 3[1], 3[2], 2[1], 2[2], 2[3], 1[1], 1[2], 1[3], and 1[4].

The categorization procedure uses the details provided by the matchmaking procedure to fix the address of the matched service. The first level address for the category for the first partially matched service is generated as 1 based on the level of the behavioral (and functional) match. The second level address for the category is generated as 1 based on the sequence of the behavioral (and functional) match. The address for this matched service is fixed as 1[1]. Next, the semantic rank for this matched service is determined as 2. The value is obtained by averaging semantic match values of the 2 individual concepts (plug-in match), which is 4 divided by the number of concepts (2). This matched service is placed in the category with address 1[1] and it is ranked first in this category as it is the only service in this category at present. Basically, the first matched service is categorized in its category.

Similarly, the other 19 services are checked for partial matching and the matched services and categorized in their respective categories. For simplicity purposes, we assume that only one presence, MMS, and call control services from the list matches with the other three decomposed components of the request. Subsequently, their categories are determined and categorized by the categorization procedure. The categories for the presence, MMS, and call control services are 1[2], 1[3], and 1[4] respectively. Obviously, their semantic ranks are determined and the services are ranked within their categories. As all the services are the only services in their categories they are placed first in their

respective categories. If, for example, we assume the service with behavioral match level of 2 and sequence of 2 is a matched service then its category would be determined as 2[2].

After all the matched services are categorized for this dating service the assembly process is invoked, which uses the assembly procedure for orchestrating the possible compositions. The assembly procedure virtually determines the number of categories as 9 and finds its addresses for manipulation purposes. Then, the assembly value is determined as 2. The replacement list is then generated using its procedure. After that, all the possible compositions, if present, are generated from categories whose level is above 2. However, as none of the categories has any matched services, no assemblies are generated from the first and second assembly sub procedures. In this case study, categories that are below assembly value are only last level categories, so no assemblies are generated using third assembly sub procedure. After that, all the services in the categories of last level, which in our case is 1[1], 1[2], 1[3], and 1[4] are used for assembly. A service orchestration is possible, which is generated using BPEL orchestration procedure.

In the BPEL orchestration procedure, the four services from the four categories are obtained; they become the partner services for the dating service BPEL process. The requested service behavior is analyzed and the requirement for the sequential composition operator is determined, as all the services are invoked sequentially. The first service to be used in the orchestration (location service) is specified as a partner service and the right method is invoked using the 'invoke' activity. Then, to simulate the requested behavior, the 'sequence' activity is used for the sequential composition behavior. The next service, a presence service, is then orchestrated using another 'invoke'

activity within the 'sequence' activity. Similarly, the MMS and the call control services are 'invoked' after that. This basically completes the creation of the BPEL process for the dating service.

Finally, the only assembled service is checked for non-functional matching using the best-assembled service selection procedure. The non-functionalities based best-assembled service selection procedure first finds the non-functional values of the dating service, which we assume to be specified with a cost of 5 USD, and medium security. The procedure then finds if the non-functional values for all the third party services of this assembly is available, which we assume to be true. We further assume that the four third party services have a cost of 1 USD each, and 2 of the services have medium security and reliability, and 2 of them having high security and reliability. Now, the total non-functional value of the assembled service is calculated, which is cost of 4 USD, and medium security (lowest scale of the 4 services). As the calculated values of the assembled service are better than what is requested for, this assembled service is marked as acceptable from the non-functional perspective. This assembled service is placed in the ranking list as first as this is the only assembled service, and no other service assemblies are generated. This assembled service is then selected and given to the communication module for providing it to the requester.

In this case study, we assumed for simplicity purposes that services that match only one component of the (decomposed) request are available for composition. However, if there are services available that matches two or more components together (in any sequence) then they will obviously be matched, categorized, selected, and assembled with the other services for composition. For example, if we assume there is a service available

that satisfies for both the presence and MMS component and is combined using a sequential operator, then this will be selected as a partially matched service by the matchmaking procedure. The categorization procedures will place the service in its appropriate category, which is 2[2] and its semantic rank will be calculated. This will enable the assembly procedure to select this service and combine it with the other two services, location service and call control service, to create an orchestration using BPEL. Assuming the non-functional requirement of this assembled is satisfied; it is then compared with the already assembled service using four individually matched services for non-functional rank, and ultimately the best of the two services is given to requester. It is important to note that for all the experiments performed and reported in Section 6.2, all the steps explained here have been performed, except for the Isabelle-based matching.

Chapter 7

Conclusion

7.1. Contributions of this thesis

Web services are software components that are described, published, found, used, and composed into new services. Web service composition is considered as the cornerstone for Web service development. The descriptions expose a mixture of functional, non-functional, behavioral, and semantic characteristics of Web services. Appropriate descriptions enable Web services to be discovered and reused for creation of new services. In order to compose a requested service the component services have to be discovered using matchmaking techniques. The component services have to be selected and orchestrated/choreographed suitably to achieve the desired functionalities and behaviors as expected from the composite service. In this thesis, we tackled the Web service composition problem by considering their functional, non-functional, behavioral, and semantic characteristics together. The contributions of this thesis are summarized in the rest of this section.

- ✓ **Identified issues for composing Web services considering all their aspects** – We have identified three basic issues in composing Web services by considering their different characteristics. First, the Web service composition should be supported from the architectural perspective. In order to use composition as one of the fundamental mechanisms for services provisioning the architecture should explicitly support composition, and must have the capabilities in terms of entities,

their interactions, and mechanisms. Second, there should be a technique to describe Web services with all their characteristics. To compose Web services with all their characteristics, a suitable description technique is a prerequisite. Third, there should be precise mechanisms for matchmaking, selection, and orchestration/choreography, which can manipulate services considering all their characteristics together, as they are the basic building blocks of the composition method.

- ✓ **Classified, derived requirements, and evaluated the architectures and techniques for Web services description and composition** – We classified existing techniques for the description and composition of Web services, which helps in understanding and evaluating these techniques. We have derived requirements for the architectures to support Web services composition. We also derived requirements for the description and the composition techniques to support composition considering all their characteristics. We evaluated the existing architectures/business models with respect to the derived requirements and found that none of the existing ones support the derived requirements. We have also evaluated the existing description and composition techniques based on the set of derived requirements and by using our proposed classification. The evaluation of these techniques showed that none of them support describing and composing Web services with all the characteristics together.

- ✓ **Proposed architecture for Web service composition and proposed realization schema for the component interactions** – We proposed an architecture for Web services composition by extending the standard Web services architecture. This is one of the core contributions of this thesis. The extended architecture introduces

three new entities: Web service composer, Web service composition registry, and third party Web service provider. It has the Web service consumer and the Web service registry entities from the original architecture. Register, inform, get, give, put, and locate interactions are proposed. Publish, find, and bind interactions are reused from the original architecture. Realization schema for register, inform, get, and give interactions is proposed as APIs. The subscription API of UDDI V3 is extended to realize the register and inform interactions, while the other two interactions are realized as a new API. This architecture satisfies all the derived requirements in supporting Web service composition.

- ✓ **Proposed a description framework for Web services taking into account all their characteristics** – We proposed a framework for describing the four aspects of Web services together, which uses three different languages: SAWSDL, MSC, and NFSL. The languages are integrated in the common semantic domain of HOL for reasoning about Web services with all the characteristics. We have formalized SAWSDL and NFSL in HOL. Work on CSP-Prover [13] is reused for formalizing MSC in HOL. We proposed a mapping of MSC to HOL using its process algebra semantics.

- ✓ **Proposed a composition framework for composing Web services considering their different characteristics** – We have proposed a framework for composing Web services, where they are considered and manipulated with all their characteristics. This framework is a realization of the Web service composer component in our proposed architecture. It has components and techniques for performing Web services composition.

- ✓ **Proposed techniques for matchmaking, categorization, and assembly** – We proposed techniques for matchmaking, categorization, and assembly, which are the core composition-related techniques of the composition framework. The formal matchmaking technique discovers the third party services that fully or partially match with the requested service. This technique uses an existing HOL theorem prover, Isabelle [14], for finding matching services formally. However, because of the interactive aspect of Isabelle the matchmaking technique cannot be fully automated, human intervention may be needed at regular intervals. The categorization technique organizes the partially matched services into appropriate categories based on their different levels of match. The categorized services are manipulated to select and orchestrate the appropriate services, and finally the best-orchestrated service is selected as the 'composite service' by assembly technique.

- ✓ **Developed prototypes for the architectural components, the composition framework and its techniques and evaluated them** – We have developed prototypes of the extended requester, extended registry, the composition framework and its techniques as proof-of-concept. The get and give interaction as new API, and the extended find interaction is implemented by the extended requester. The extended registry implements register and inform interactions, as an extension to Apache jUDDI registry. The communication module and the composition module of the framework are implemented. The communication module mainly implements the proposed realization schema of the get, give, register, and inform interactions. The matchmaking technique, categorization technique, and the assembly techniques

are implemented by the composition module. Performance evaluation of these prototypes was performed with specific scenarios.

- ✓ **Case studies for describing and Composing Web services considering all their Characteristics** – We have developed case studies from telecom domain for illustrating the whole work: the architecture, the description framework, composition framework, and the techniques.

7.2. Future Work

Different directions for future work are possible, as discussed below.

7.2.1. Architecture Related Issues

The realization scheme for put and locate interactions could be developed. As these interactions were inspired from publish and find interactions, it would be interesting to see if existing UDDI APIs for publish and find can be extended in this case. The realization of the composition registry and third party Web service provider could also be developed. For example, composition registry could be realized by reusing and extending the UDDI concepts and data structures. The architecture of the third party Web service provider can also be developed. Implementation of these components and their interactions could be done, which would allow evaluating the performance of the whole architecture.

7.2.2. Description Framework Related Issues

The languages of the description framework are integrated based on their formalization in HOL. Formalization of the MSC concepts directly in Isabelle/HOL could be explored, as

an alternative to the current two-step mapping. As MSC is a language for expressing behaviors, this formalization has to be performed more carefully to enable proper reasoning about the formalized concepts. Works like [141], where UML state charts are formalized directly in HOL can be used as a starting point for formalizing MSC. OWL concepts can also be directly formalized in HOL, which would allow us to develop and reason with ontologies directly in HOL. Works like [142] can be reused for this.

7.2.3. Composition Framework Related Issues

Work on the description transformation module and execution time adaptation module could be done. Techniques and procedures for automatically converting the specifications in SAWSDL, MSC, and NFSL format to HOL specifications can be developed. Mappings from SAWSDL and NFSL to HOL could be specified, similar to what was specified for MSC. The execution time adaptation module's processes (techniques) could be developed. Existing work on execution time adaptation of BPEL processes like [115] and [116] can be used as reference for this work. Implementation of these techniques and components in the composition framework would be interesting. Also, integrating HOL theorem prover for matchmaking with the existing prototype could be carried out, which would also help in complete evaluation of the prototype.

Bibliography

[1] Reference Model for Service Oriented Architecture 1.0, OASIS Standard, 2006.

[2] Web Services Architecture Specification, W3C Note, 2004, http://www.w3.org/TR/ws-arch/.

[3] Web Services Description Language (WSDL) Version 1.1, W3C Note, 2001, http://www.w3.org/TR/wsdl.

[4] UDDI Version 3.0.2, UDDI Technical Committee Draft, http://uddi.org/pubs/uddi_v3.htm.

[5] SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), W3C Recommendation, 2007, http://www.w3.org/TR/soap12-part1/.

[6] M. N. Huhns, and M. P. Singh, "Service-Oriented Computing: Key Concepts and Principles," *IEEE Internet Computing*, vol. 9, no.1, pp.75-81, 2005.

[7] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-Oriented Computing: A Research Roadmap," Service Oriented Computing (SOC), number 05462 in Dagstuhl Seminar Proceedings, 2006.

[8] R. Karunamurthy, F. Khendek, and R. H. Glitho, "A Novel Business Model for Web Service Composition," In the Proceedings of the IEEE International Conference on Services Computing (SCC 2006), IEEE CS Print, pp. 431-437, 2006.

[9] R. Karunamurthy, F. Khendek, and R. H. Glitho, "A Business Model for Dynamic Composition of Telecommunication Web Services," *IEEE Communications Magazine*, Special issue on 'Web Services for Telecommunications', vol. 45, no. 7, pp. 36-43, July 2007.

[10] R. Karunamurthy, F. Khendek, and R. H. Glitho, "A Formal Description Framework and A Matchmaking Technique for Web Service Composition," Submitted for *Service Oriented Computing and Applications,* Springer.

[11] R. Karunamurthy, F. Khendek, and R. H. Glitho, "A Framework for Web Service Composition," Submitted for *IEEE Transactions on Parallel and Distributed Systems*.

[12] R. Karunamurthy, F. Khendek, and R. H. Glitho, "Categorizing and Assembling Web Services in a Composition Framework," In the Proceedings of the 3rd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2008).

[13] Y. Isobe and M. Roggenbach, "A generic theorem prover of CSP refinement," In the Proceedings of 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2005), LNCS 3440, pp.108-123, 2005.

[14] T. Nipkow, L. C. Paulson, and M. Wenzel, *Isabelle/HOL - A Proof Assistant for Higher Order Logic*, LNCS 2283, 2002.

[15] M. P. Papazoglou and D. Georgakopoulos, "Introduction to Special Issue on Service-Oriented Computing," *Communications of ACM*, vol. 46, no. 10, pp.24–28, 2003.

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

[17] E. Newcomer, *Understanding Web Services: XML, WSDL, SOAP, and UDDI,* Addison Wesley Professional, 2002.

[18] Semantic Annotations for WSDL and XML Schema, W3C Recommendation, 2007, http://www.w3.org/TR/sawsdl/.

[19] OWL-S: Semantic Markup for Web Services, W3C Member Submission, 2004, http://www.w3.org/Submission/OWL-S/.

[20] Web Service Semantics – WSDL-S Version 1.0, W3C Member Submission, 2005, http://www.w3.org/Submission/WSDL-S/.

[21] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation,* Second Edition, Addison Wesley, 2000.

[22] OMG Unified Modeling Language (OMG UML) Superstructure V2.1.2, 2007, http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF.

[23] Z.120, Message Sequence Chart (MSC), ITU-T Recommendation, 2004.

[24] Business Process Execution Language for Web Services (BPEL4WS) Version 1.1, 2003, http://www.ibm.com/developerworks/library/specification/ws-bpel/.

[25] WS-CDL Version 1.0, W3C Candidate Recommendation, 2005, http://www.w3.org/TR/ws-cdl-10/.

[26] S. R. Ponnekanti and A. Fox, "Sword: A developer toolkit for Web service composition," In the Proceedings of the 11th International World Wide Web Conference (WWW 2002), Alternate Paper Track, http://www2002.org/CDROM/alternate/786/.

[27] N. Milanovic and M. Malek, "Current Solutions for Web Service Composition," *IEEE Internet Computing*, vol. 8, no. 6, pp. 51-59, Nov/Dec 2004.

[28] J. Rao and X. Su, "A Survey of Automated Web Service Composition Methods," In the Proceedings of the 1st International Semantic Web Services and Web Process Composition Workshop (SWSWPC 2004), LNCS 3387, pp. 43-54, 2004.

[29] S. Dustdar, and W. Schreiner, "A survey on Web services composition," *International Journal of Web and Grid Services*, Inderscience Publishers, vol. 1, No. 1, pp.1–30, 2005.

[30] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara, "Semantic Matching of Web Services Capabilities," In the Proceedings of the International Semantic Web Conference (ISWC 2002), LNCS 2342, pp.333-347, 2002.

[31] M. Paolucci, T. Kawamura, T. R. Payne, and K. P. Sycara, "Importing the Semantic Web in UDDI" In the Proceedings of the International Workshop on Web Services E-Business and the Semantic Web (WES2002), LNCS 2512, pp. 225-236, 2002.

[32] TINA Business Model and Reference Points, Version 4.0, May 1997, http://www.tinac.com/specifications/documents/bm_rp.pdf.

[33] ETSI ES 203 915-3 Version 1.1.1, Parlay 5.0, "Part 3: Framework," April 2005.

[34] H. Berndt, T. Hamada, and P. Graubmann, "TINA: Its Achievements and Its Future Directions," *IEEE Communication Surveys & Tutorials*, vol. 3, no. 1, pp. 2-16, 2000.

[35] A. J. Moerdijk and L. Klostermann, "Opening the Networks with Parlay/OSA: Standards and Aspects Behind the APIs," *IEEE Network*, vol. 17, no. 3, pp. 58-64, May/June 2003.

[36] 3GPP TS 23.198, "Open Service Access (OSA)," June 2006.

[37] K. Gottschalk, S. Graham, H. Kreger, and J. Snell, "Introduction to Web Services Architecture," *IBM Systems Journal*, vol. 41, no.2, pp.170-177, 2002.

[38] M. P. Papazoglou, and W-J. V. D. Heuvel, "Service oriented architectures: approaches, technologies and research issues," *The VLDB Journal*, vol.16, no.3, pp. 389-415, 2007.

[39] T. Thompson, R. Wiel, and M. D. Wood, "CPXe: Web Services for Internet Imaging", *IEEE Computer*, vol. 36, no. 10, pp. 54-62, October 2003.

[40] M. P. Papazoglou, "Extending the Service-Oriented Architecture," *Business Integration Journal*, pp. 18-21, February 2005.

[41] Semantic Web Services Interest Group Webpage, http://www.w3.org/2002/ws/swsig/.

[42] WSMO W3C Member Submission, 2005. http://www.w3.org/Submission/WSMO/.

[43] WSML W3C Member Submission, 2005. http://www.w3.org/Submission/WSML/.

[44] SWSO W3C Member Submission, 2005. http://www.w3.org/Submission/SWSF-SWSO/.

[45] SWSL W3C Member Submission, 2005. http://www.w3.org/Submission/SWSF-SWSL/.

[46] D. Berardi, F. De Rosa, L. De Santis, and M. Macella, "Finite State Machine as Conceptual Model for E-Services," *Transactions of the SDPS: Journal of Integrated Design and Process Science*, vol.8, no. 2, pp.105–121, 2004.

[47] T. Bultan, X. Fu, R. Hull, and J. Su, "Conversation Specification: A New Approach to Design and Analysis of E-Service Composition," In the Proceedings of the 12th International World Wide Web Conference (WWW 2003), pp. 403-410, 2003.

[48] D. Berardi, D. Calvanesa, G. De Giacomo, R. Hull, and M. Mecella, "Automatic Composition of Transition-based Semantic Web Services with Messaging," In the Proceedings of the 31st International Conference on Very Large Data Bases (VLDB 2005), pp. 613-624, 2005.

[49] J. Pathak, S. Basu, and V. Honavar, "Modeling Web Service Composition using Symbolic Transition Systems," In the Proceedings of the AAAI Workshop on AI-Driven Technologies for Service-Oriented Computing, AAAI Press Technical Report WS-06-01, pp. 44–51, 2006.

[50] B. Benatallah, Q.Z. Sheng, and M. Dumas, "The Self-Serv Environment for Web Services Composition," *IEEE Internet Computing*, Vol. 7, No. 1, pp. 40 – 48, 2003.

[51] B. Medjahed, A. Bouguettaya, and A. K. Elmagarmid, "Composing Web services on the Semantic Web", *Very Large Data Base Journal (VLDB)*, vol. 12, no. 4, pp. 333–351, 2003.

[52] R. Gronmo and M. C. Jaeger, "Model-Driven Semantic Web Service Composition," In the Proceedings of the 12th Asia-pacific Software Engineering Conference (APSEC 2005), pp. 79-86.

[53] H. Foster, S. Uchitel, J. Magee, and J. Kramer," Model-based Verification of Web Service Compositions," In the Proceedings of the 18th IEEE International Conference on Automated Software Engineering (ASE 2003), pp. 152-163.

[54] G. Salaün, L. Bordeaux, and M. Schaerf, "Describing and Reasoning on Web Services using Process Algebra," In the Proceedings of the International Conference on Web Services (ICWS 2004), pp. 43-51.

[55] A. Ferrara, "Web services: a process algebra approach," In the Proceedings of the International Conference on Service Oriented Computing (ICSOC 2004), pp. 242-251.

[56] G. Salaün, A. Ferrara, and A. Chirichiello, "Negotiation Among Web Services Using LOTOS/CADP," In the Proceedings of the European Conference on Web Services (ECOWS 2004), pp.198-212.

[57] R. Hamadi and B. Benatallah "A Petri Net-based Model for Web Service Composition," In the Proceedings of the 13th Australian Database Conference (ADC 2003), pp. 191-200.

[58] C. A. R. Hoare, *Communicating Sequential Processes*, Prentice Hall, 1985.

[59] LOTOS: A Formal Description Technique based on the Temporal Ordering of Observational Behavior, ISO/IEC 8807, International Organization for Standardization (ISO) Standard, 1989.

[60] R. Lucchi, and M. Mazzara, "A pi-calculus based semantics for WS-BPEL" *Journal of Logic and Algebraic Programming*, Elsevier, vol. 70, no.1, pp.96-118, 2007.

[61] Web Services Business Process Execution Language Version 2.0, OASIS Standard, http://docs.oasis-open.org/sbpel/2.0/wsbpel-v2.0.pdf.

[62] N. Lohmann, P. Massuthe, C. Stahl, and D. Weinberg, "Analyzing interacting WS-BPEL processes using flexible model generation," *Data and Knowledge Engineering*, Elsevier, vol.64, no.1, pp. 38-54, 2008.

[63] S. Narayanan and S. McIlraith, "Analysis and Simulation of Web Services," *Computer Networks*, vol. 42, no.5, pp. 675–693, 2003.

[64] K. Fujii and T. Suda, "Component Service Model with Semantics (CoSMoS): A New Component Model for Dynamic Service Composition," In the Proceedings of the IEEE International Symposium on Applications and the Internet (SAINT2004) Workshop on Service Oriented Computing, pp. 348-354, 2004.

[65] J. Rao, P. Küngas, and M. Matskin, "Composition of Semantic Web services using Linear Logic theorem proving," *Information Systems*, vol. 31, no.4-5, pp. 340-360, July 2006.

[66] M. Klein, B. Knig-Ries, and M. Mssig, "What is needed for semantic service descriptions - a proposal for suitable language constructs," *International Journal of Web and Grid Services*, vol. 1, no. 3-4, pp.328-364, 2005.

[67] A. Bansal, S. Kona, L. Simon, and T. D. Hite, "A Universal Service-Semantics Description Language," In the Proceedings of the European Conference on Web Services (ECOWS 2005), pp.214-225.

[68] Web Services Policy 1.5 Framework 1.5, W3C Recommendation, 2007, http://www.w3.org/TR/ws-policy/.

[69] Web Service Level Agreement Language Specification Version 1.0, 2003, http://www.research.ibm.com/wsla/.

[70] V. Tosic, B. Pagurek, and K. Patel, "WSOL - A Language for the Formal Specification of Classes of Service for Web Services," In the Proceedings of the International Conference on Web Services (ICWS 2003), pp. 375-381, 2003.

[71] M. Tian, A. Gramm, T. Naumowicz, H. Ritter, and J. Schiller, "A Concept for QoS Integration in Web Services," In the Proceedings of the 4th International Conference on Web Information Systems Engineering Workshops (WISEW'03), pp. 149-155, 2003.

[72] WSCI Version 1.0, W3C Note, 2002, http://www.w3.org/TR/wsci/.

[73] WSCL Version 1.0, W3C Note, 2002, http://www.w3.org/TR/wscl10/.

[74] R. Duke, G. Rose, and G. Smith, "Object-Z: A specification language advocated for the description of standards," *Computer Standards and Interfaces*, Elsevier, vol.17, no. 5-6, pp.511-533, 1995.

[75] G. Smith, "A Semantic Integration of Object-Z and CSP for the Specification of Concurrent Systems," In the Proceedings of the 4th International Symposium of Formal Methods Europe (FME' 97), LNCS 1313, pp. 62-81, 1997.

[76] J. Hoenicke, and E. Olderog, "Combining Specification Techniques for Processes, Data and Time," In the Proceedings of the 3rd International Conference on Integrated Formal Methods (IFM 2002), LNCS 2335, pp.245-266, 2002.

[77] C. Zhou, C. A. R. Hoare, and A. P. Ravn, "A calculus for durations," *Information Processing Letters*, Elsevier, vol. 40, no.5, pp. 269-276, 1991.

[78] C. Attiogbe, "Formal Methods Integration for Software Development: Some Locks and Outlines," Research Report, RR-IRIN-008, 2000.

[79] J. M. Spivey, *The Z Notation: a reference manual*, Prentice Hall, 2001.

[80] J. R. Abrial, *The B Book: Assigning Programs to Meanings*, Cambridge University Press, 1996.

[81] Web Services Security: SOAP Message Security 1.0 (WS-Security), OASIS Standard, 2004.

[82] Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) Version 2.0, OASIS Standard, 2005.

[83] S. Shapiro, "Classical Logic II: Higher-Order Logic," *The Blackwell Guide to Philosophical Logic*, Blackwell, Lou Gable ed., 2001, pp. 33-54.

[84] Z.120 Annex B, Formal Semantics of Message Sequence Charts, ITU-T Recommendation, 1998.

[85] Isabelle Theorem Prover Webpage, http://www.cl.cam.ac.uk/research/hvg/Isabelle/.

[86] Y. Isobe and M. Roggenbach, "User Guide CSP-Prover Version 4.0", http://staff.aist.go.jp/y-isobe/CSP-Prover/CSP-Prover.html.

[87] Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, W3C Recommendation, 2007, http://www.w3.org/TR/wsdl20/.

[88] M. Vukovic and P. Robinson, "SHOP2 and TLPlan for Proactive Service Composition," In the Proceedings of the UK-Russia Workshop on Proactive Computing, 2005.

[89] E. Sirin, J. A. Hendler, and B. Parsia, "Filtering and Selecting Semantic Web Services with Interactive Composition Techniques," *IEEE Intelligent Systems*, Vol. 19, No.4, pp. 42-49, 2004.

[90] D. Sell, F. Hakimpour, J. Domingue, E. Motta and R. C. S. Pacheco, " Interactive Composition of WSMO-based Semantic Web Services in IRS-III," In the Proceedings of the 1st AKT Workshop on Semantic Web Services (AKT-SWS 2004).

[91] Q. Liang, L. N. Chakrapani, S. Y. W. Su, R. N. Chikkamagular, and H. Lam, "A Semi-Automatic Approach to Composite Web Services Discovery, Description and Invocation," *International Journal of Web Services Research*, Vol.1, No.4, pp. 64-89, 2004.

[92] A. Charfi and M. Mezini, "Aspect-Oriented Web Service Composition with AO4BPEL," In the Proceedings of the European Conference on Web Services (ECOWS 2004), LNCS 3250, pp. 168-182, 2004.

[93] D. Wu, B. Parsia, E. Sirin, J. A. Hendler, and D. S. Nau, "Automating DAML-S Web Services Composition Using SHOP2," In the Proceedings of the 2nd International Semantic Web Conference (ISWC 2003), LNCS 2870, pp. 195-210, 2003.

[94] M. Sheshagiri, M. DesJardins, and T. Finin, "A Planner for Composing Service Described in DAML-S," In the Proceedings of the International Conference on Automated Planning and Scheduling, Workshop on Planning for Web Services, 2003.

[95] M. Klusch, A. Gerber, and M. Schmidt, "Semantic Web Service Composition Planning with OWLS-Xplan," In the Proceedings of the 1st International AAAI Fall Symposium on Agents and the Semantic Web, 2005.

[96] V. Agarwal, K. Dasgupta, N. Karnik, A. Kumar, A. Kundu, S. Mittal, and B. Srivastava, "Synthy: A system for end to end composition of Web services," *International Journal of Web Semantics*, Vol. 3, No. 4, pp. 311-339, 2005.

[97] D. McDermott, "Estimated-regression planning for interactions with Web services," In the proc. of the 6th International Conference on AI Planning Systems, pp.204-211, 2002.

[98] E. Martinez and Y. Lespérance, "Web Service Composition as a Planning Task: Experiments using Knowledge-Based Planning," In the Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS-2004), Workshop on Planning and Scheduling for Web and Grid Services, pp. 62-69, 2004.

[99] J. Peer, "A POP-Based Replanning Agent for Automatic Web Service Composition," In the Proceedings of the 2nd European Semantic Web Conference (ESWC 2005), LNCS 3531, pp. 47-61, 2005.

[100] P. Traverso, and M. Pistore, "Automated Composition of Semantic Web Services into Executable Processes," In the Proceedings of the 3rd International Semantic Web Conference (ISWC2004), pp. 380-394, 2004.

[101] S. McIlraith and T. C. Son, "Adapting Golog for composition of semantic Web services," In the Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning (KR2002), pp. 482-496, 2002.

[102] K. Fujii and T. Suda, "Semantics-based Dynamic Service Composition," *IEEE Journal on Selected Areas in Communications (JSAC)*, Special issue on Autonomic Communication Systems, Vol. 23, No. 12, pp. 2361 – 2372, December 2005.

[103] F. Casati, S. Ilnicki, and L. Jin, "Adaptive and dynamic service composition in eFlow," In the Proceedings of the 12th International Conference on Advanced Information Systems Engineering (CAiSE 2000), pp. 13-31.

[104] R. Aggarwal, K. Verma, J. A. Miller, and W. Milnor, "Constraint Driven Web Service Composition in METEOR-S," In the Proceedings of the IEEE International Conference on Services Computing (SCC 2004), pp. 23-30, 2004.

[105] D. Berardi, "Automatic Composition of Finite State E-Services," Technical report, American Association for Artificial Intelligence (AAAI), 2004.

[106] D. Berardi, D. Calvanesa, G. De Giacomo, R. Hull, and M. Mecella, "Automatic Composition of Web Services in Colombo," In the Proceedings of the 13th Italian Symposium on Advanced Database Systems (SEBD 2005), pp. 8-15, 2005.

[107] J. Pathak, S. Basu, R.R. Lutz, and V. Honavar, "MOSCOE: an Approach for Composing Web Services through Iterative Reformulation of Functional Specifications," *International Journal on Artificial Intelligence Tools*, World Scientific, vol.17, no.1, pp.109-138,2008.

[108] S. Laemmermann, "Runtime Service Composition via Logic-Based Program Synthesis," PhD thesis, Royal Institute of Technology, 2002.

[109] Z. Maamar, S. K. Mostefaoui, and H. Yahyaoui, "Toward an Agent-Based and Context-Oriented Approach for Web Services Composition," *IEEE Transactions on Knowledge and Data Engineering*, Vol.17, No.5, pp. 686-697, 2005.

[110] V. Ermolayev, N. Keberle, S. Plaksin, O. Konoenko, and V.Y. Terziyan, "Towards a Framework for Agent-Enabled Semantic Web Service Composition," *International Journal of Web Services Research*, Vol. 1, No.3, pp.63-87, 2004.

[111] U. Küster, B. König-Ries, M. Stern, and M. Klein, "DIANE: an integrated approach to automated service discovery, matchmaking and composition," In the Proceedings of the 16th International Conference on World Wide Web (WWW 2007), pp.1033-1042.

[112] S. Kona, A. Bansal, and G. Gupta, "Automatic Composition of Semantic Web Services," In the Proceedings of the International Conference on Web Services (ICWS 2007), pp.150-158.

[113] H. Sun, X. Wang, B. Zhou, and P. Zou, "Research and Implementation of Dynamic Web Services Composition," In the Proceedings of the 5th International Workshop of Advanced Parallel Programming Technologies (APPT 2003), LNCS 2834, pp. 457-466, 2003.

[114] L. Baresi, E. Di Nitto, C. Ghezzi, and S. Guinea, "A framework for the deployment of adaptable Web service compositions," Service *Oriented Computing and Applications*, Springer, vol. 1, no.1, pp.75-91, 2007.

[115] O. Moser, F. Rosenberg, and S. Dustdar, "Non-intrusive monitoring and service adaptation for WS-BPEL," In the Proceedings of the 17th International Conference on World Wide Web (WWW 2008), pp. 815-824.

[116] W. Kongdenfha, R. Saint-Paul, B. Benatallah, F. Casati, "An Aspect-Oriented Framework for Service Adaptation," In the Proceedings of International Conference on Service Oriented Computing (ICSOC 2006), pp.15-26.

[117] X. Fu, T. Bultan, and J. Su, "Analysis of interacting BPEL Web services," In the Proceedings of the 13th International Conference on World Wide Web (WWW 2004), pp.621–630.

[118] L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea and P. Spoletini, "Validation of Web Service Compositions," *IET Software,* vol.1, no.6, pp.219-232, 2007.

[119] S. Ran, "A model for Web services discovery with QoS," *SIGecom Exchanges*, SIGecom, vol.4, no.1, pp.1-10, 2003.

[120] E. Stroulia, and Y. Wang, "Structural and Semantic Matching for Assessing Web-service Similarity," *International Journal of Cooperative Information Systems*, World Scientific, vol.14, no.4, 2005, pp. 407-438.

[121] J. Wu, and Z. Wu, "Similarity-based web service matchmaking," In the Proceedings of the International Conference on Services Computing, Volume 1, (SCC 2005) pp.287-294.

[122] N. Kokash, W-J V. D. Heuvel, and V. D'Andrea, "Leveraging Web Services Discovery with Customizable Hybrid Matching," In the Proceedings of the International Conference on Service Oriented Computing (ICSOC 2006), pp. 522-528.

[123] X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang, "Similarity search for web services," In the Proceedings of the 30th Very Large Data Base Conference (VLDB 2004), pp. 372-383.

[124] J. Zhang, S. Yu, X. Ge, and G. Wu, "Automatic Web Service Composition Based on Service Interface Description," In the Proceedings of the International Conference on Internet Computing 2006, pp.120-126.

[125] R. Akkiraju, R. Goodwin, P. Doshi, and S. Roeder, "A Method for Semantically Enhancing the Service Discovery Capabilities of UDDI," In the Proceedings of the IJCAI 2003 Workshop on Information Integration on the Web (IIWeb-03), pp. 87-92.

[126] E. M. Maximilien, and M. P. Singh, "Towards Autonomic Web Services Trust and Selection," In the Proceedings of the International Conference on Service Oriented Computing (ICSOC 2004), pp. 212-221.

[127] A. Brogi, and S. Corfini, "Behaviour-aware discovery of Web service Compositions," *International Journal of Web Services Research*, vol. 4, no.3, pp. 1-25, 2007.

[128] M. Baldoni, C. Baroglio, A. Martelli, and V. Patti, "Reasoning about interaction protocols for customizing web service selection and composition," *Journal of Logic and Algebraic Programming*, vol.70, no.1, pp. 53-73, 2007.

[129] L. Bordeaux, G. Salaün, D. Berardi, M. Mecella "When are Two Web Services Compatible?," In the Proceedings of the 5th International Workshop on Technologies for E-Services (TES 2004), pp. 15-28.

[130] M. Weiss and B. Esfandiari, "On Feature Interactions Among Web Services", *International Journal of Web Services Research*, vol. 2, no. 4, pp. 21-45 Oct-Dec. 2005.

[131] A.W. Roscoe, *The Theory and Practice of Concurrency*, Prentice Hall, 1997.

[132] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical OWL-DL reasoner," *Journal of Web Semantics*, vol. 5, no. 2, pp. 51-53, 2007.

[133] Pellet Webpage, http://pellet.owldl.com/.

[134] J. Pasley, "How BPEL and SOA Are Changing Web Services Development," *IEEE Internet Computing*, vol. 9, no. 3, pp. 60-67, 2005.

[135] jUDDI Webpage, http://ws.apache.org/juddi/.

[136] Apache Axis Webpage, http://ws.apache.org/axis/.

[137] UDDI4J Webpage, http://uddi4j.sourceforge.net/.

[138] Apache Axis2 Webpage, http://ws.apache.org/axis2/.

[139] Apache AXIOM Webpage, http://ws.apache.org/commons/axiom/.

[140] Apache Commons Webpage, http://commons.apache.org/.

[141] I. Traore, D. B. Aredo and H. Ye, "An Integrated Framework for Formal Development of Open Distributed Systems", *Journal of Information and Software Technology (IST)*, Elsevier, vol. 46, no. 5, pp. 281-286, 2004.

[142] D. Turner, and J. J. Carroll, "Comparing OWL Semantics," HP Technical Report, HPL-2007-146, 2007.