4-22-2008

WS-CDL Based Specification for Web Services Collaboration Testing

Ahmed A. Ugaas

WS-CDL BASED SPECIFICATION FOR WEB SERVICES COLLABORATION TESTING

by

Ahmed Ugaas

Under the Direction of Dr. Xiolin Hu

ABSTRACT

Service Oriented Computing(SOC) is becoming a major paradigm for developing next generation of software systems, and one of the major challenges of Service Oriented Computing is testing interactions and collaborations among the distributed and dynamically integrated web services. To support automated test of web service's collaborations, a formal specification is needed. This thesis proposes a specification of web services collaborations based on Web Services Choreography Description Language (WS-CDL). We identify the basic constructs that can be found in any web services collaboration, and we mapped them to the new WS-CDL based language (WS-CDL+). Finally, A scenario of web services collaboration is developed and specification in WS-CDL+ is provided. This work builds a foundation for automated web services testing in a service oriented computing environment.

INDEX WORDS: WS-CDL, WS-CDL+, Web Service, Service Oriented Computing, Collaboration.

WS-CDL BASED SPECIFICATION FOR WEB SERVICES COLLABORATION TESTING

by

Ahmed Ugaas

A Thesis Submitted in Partial Fulfillment of the Requirements for the degree of

Master of Science

In the College of arts and sciences

Georgia State University

2008

WS-CDL BASED SPECIFICATION FOR WEB SERVICES

COLLABORATION TESTING

by

AHMED UGAAS

Committee Chair: Dr. Xiolin Hu

Committee: Dr. Alex Zelikovsky
Dr. Yanqing Zhang

Electronic version Approved:

Office of Graduate studies
College of Arts and sciences
Georgia State University
May 2008

TABLE OF CONTENTS

1.0 Introduction

Software development and maintenance is very costly; both academia and industry have been working on ways to reduce that cost. Although many approaches have been suggested and some were embraced by industry such as Model Driven Development by the Object Management Group[OMG], most of these efforts did not focus on improving web services testing, but were geared towards rapid applications development[RAD]. Web services and Service Oriented Architecture[SOA] is becoming the standard of software development, and instead of writing new code, SOA based applications are assembled from distributed web service components. Although SOA makes software development more efficient and productive, there are no well defined, automated web services testing procedures in place, therefore web services testing and maintenance is a major impediment. Most systems today require generating new test cases and running a new test cycle for each upgrade. A survey conducted by Gartner, an information and technology research and advisory firm, showed that 35% of an enterprise's software maintenance budget is spent on maintaining the multitude of point-to point application links already in place, and about 42% of almost 500 IT executives polled said their systems were more complex then necessary and maintaining and managing the excess complexity costs them an average of 29% of their IT budgets.

SOA based systems are assembled from distributed components; these components may be developed and hosted by different organizations that expose them in a Web Services Description Language[WSDL]. Test cases of these SOA based systems can not be easily deduced from the systems requirement; it requires a thorough understanding of service contracts and accepted interactions among them. Before we delve into formal definition of these interactions, let us look into the major SOA based applications testing; 1) Controlled Test which

each component is tested in isolation and its behavior compared with the service contract, this type of test is relatively easy to achieve and its test cases are easily identified from service contract (WSDL). 2) Collaborative Test which test cases could be quite large and exhaustive depending on how many components participating in the collaboration and how freely they can interact. There is a lot of research in distributed system collaborative test; although there is a lot of progress in this area over the years, a formal language which can capture the collaboration of services has never been formally defined. SOA based applications, dynamically and free invoking web services, present a challenge when it comes to testing. Different services are invoked dynamically and each interaction my in turn produce another interaction. Therefore, it becomes very difficult to observe the behavior of the application unless there is a formal specification of the services collaborations (rules of the game). A lot has been done to automate web services testing and in this thesis we look into some of the research effort in the SOA based applications testing.

Web Services Testing

A plenty of web services testing tools are available but most of them focus on a controlled test where a service component is tested in isolation. Web Services description Language (WSDL) File is published by service producers to a registry to be referenced by service consumers. These published WSDLs are available through the web in an XML format. The WSDL file provides the mechanism to access the services by either referencing it statically or dynamically, and each provided interface is tested in isolation. The separate testing of each provided operation is fine as long as the operations the services offer are independent from each other, i.e. all operations can be called at any time, without influencing each others behavior. For

instance, think of a web service translating Italian and Spanish words into English, thus offering two independent query operations. Testing such services boils down to testing each operation separately. In many non-trivial services, the invocations of operations are not independent; and services maintain state between service invocations. The result of each service invocation is dependent of state, time and input, in other words it is state machine with a time factor. The interactions of these services is controlled by specification driven configuration data (meta-data) which describes the ordering of service invocation and time delays of the service invocations. The Test Engine will consider as an input the WSDL files of all the orchestrated services and the meta-data file which is driver of the services collaborations. A collection of operation-signatures given in the WSDL only describes a static aspect of invocations; it is not sufficient to specify the allowed sequences of invocations. Thus, it is crucial that the web services offer an additional source of information, which can be accessed to discover the web services protocol a user has to obey. To make clients aware of the protocol, its description should be published to web services registries or passed to partners.

In order to proceed towards our goal of automatic, model-based testing of web services, WS protocols must be formally modeled. One commonly used model for specifying WS protocols is a state machine. We will propose using a special variant of a state machine, namely a Symbolic Transition System (STS) as introduced in [2]. An STS has states and labeled transitions between states modeling the actions, i.e. the inputs and outputs, of the system. Both states and actions can be parameterized with data variables, and predicates on these variables may guard the transitions. The use of STSs for specifications allows exploiting the well established STS based testing theory and algorithms of [2]. These include a precise definition of conformance of a WS implementation with respect to its specification using the implementation

relation ioco [3], an algorithm for the generation of test cases from an STS specification, and notions of soundness and exhaustiveness of the generated test cases. Moreover, several testing tools nowadays implement this test generation algorithm it, e.g. TorX [4] and the TGV-based tools [5]. The tool TorX generates and executes test cases on-the-fly, which means that instead of firstly computing a set of test cases from the STS, and then applying them to the System Under Test (SUT), it generates a single test event in each step, and immediately executes it to the SUT. As a consequence the state space explosion when generating test cases, is avoided, see also [4].

An automated testing framework based on Discrete Event System Specification (DEVS) modeling and simulation formalism is discussed [6]. Extensible Markup Language (XML), and System Entity Structure (SES), is being introduced at DoD's Joint Interoperability Test Command (JITC) for interoperability testing. This framework supports the separation of experimentation, models, and simulators. The experimental frames are developed to support reusable models and simulators based on the DEVS formalism and dynamic system theory. The hierarchical structures of the models are represented by SES and written in XML format to promote extensibility and interoperability. In order to support the separation of models and simulators in the software development, the Model/Simulator/View/Controller design pattern provides the framework to support model execution and multiple network simulation protocols.

In an Automated Test Case Generator (ATC-Gen) research project funded by JITC to support the mission of standards compliance and certification. With the advent of simulation based acquisition, the test requirements in the simulated environment becomes how to automate and more precisely the scope, the completeness, and the methodology for updating conformance testing. The incorporation of the Systems Theory, the modeling and simulation concept, DEVS, XML, SES, and the MSVC design pattern all contribute to the development of ATC-Gen to

automate the TADIL-J conformance testing, increasing the productivity and effectiveness of the software test tools at JITC.

Many testing methodologies were developed to assist the test engineers to perform conformance testing over the past decades [7]. ATC-Gen is interested in a particular methodology in which the test exercises are carefully planned and the participants are given test scripts and roles to play in a simulated environment. ATC-Gen becomes a participant in a testing environment that can monitor a specific function of the MIL-STD 6016C and exchange tactical data messages with the other players. By interpreting the transmissions and the receipt of the tactical data messages, ATC-Gen is able to determine the degree to which a system conforms to the TADIL-J standard.

The automated testing framework is developed based on three concepts: SES, DEVS, and XML. SES can represent a family of hierarchical DEVS models, and serves as a means of organizing the configuration of a model to be designed, which is extracted from a pruning process. Pruning reduces the number of probable models to meet the system requirement. In the automated testing framework, the minimal testable I/O pair and the test model are represented by Pruned Entity Structures (PES). The test models obtained via PES are in executable form. XML uses elements to break up the test model into hierarchical form, and it can be used to represent the SES hierarchical structure. PES is directly mapped into XML, and the three SES modes become XML elements. XMLPES offers simplicity, extensibility, and interoperability. The test models are represented in XML-PES, which can be transformed into DEVS C++ source code.

Significance and Contributions of This Thesis

These above mentioned efforts of the related work do not fully capture the requirement of collaborating services; in this thesis, we propose a new specification language for service collaboration (WS-CDL+). This language is based on The Web Services Choreography Description Language (WS-CDL), an XML-based language that describes peer-to-peer collaborations of parties [8]. The WS-CDL based collaborations specification as defined in this thesis turns out to be rich enough to formally model the interactions among a single distributed services; more useful web services are commonly built by coordinating many simpler web services. This implies that we must take into account service interactions comprising several web services which interact with each other. The goal of this thesis is to discuss specification driven testing techniques that can be applied to testing web services using Web Services Choreography Description Language.

Our proposed language, WS-CDL+, a language that describes peer-to-peer collaborations from a global viewpoint, their common and complementary observable behavior; where ordered message exchanges result in accomplishing a common business goal. The Web Services specifications offer a communication bridge between the heterogeneous computational environments used to develop and host applications. The future of E-Business applications requires the ability to perform long-lived, peer-to-peer collaborations between the participating services, within or across the trusted domains of an organization.

WS-CDL+ extends the W3C Choreography language by introducing dependency constraint, interaction constraint and exchange constraint. These constraints are necessary for guarding and interaction and making sure state-full transactions are captured in the language. A scenario of collaboration of services is presented and captured in the language to illustrate that WS-CDL captures all stateless collaborations, and our extension covers state-full collaborations.

The specification should capture state-fullness of the services to keep them consistent and in valid state before an interaction between them is allowed.

2.0 Web Services and SOA

Service-oriented architecture (SOA) is becoming one of the enabling technologies for the emerging large-scale net-centric systems, such as distributed supply chain management systems, networked crisis response systems, and the military Global Information Grid. This new paradigm brings new design and testing challenges in software development [9, 10]. Among them how to test the collaborations among multiple dynamically integrated services is becoming an increasingly important issue. Requirements for testing service collaborations include those derived from their decision-making, net-centric, large-scale, and dynamic behaviors. Two of the factors underlying the challenges posed by collaboration testing are: 1) the increasing complexity of service-oriented computing results in sophisticated interaction patterns among services. In particular, stateful web services, i.e., services that access and manipulate state and/or stateful resources in a timely fashion are widely deployed. A stateful service may implement a series of operations such that the result of one operation depends on a prior operation and/or prepares for a subsequent operation [11]. This results in temporal and logical dependencies between its input/output messages, which make the test difficult. 2) Service-oriented applications are by nature evolutionary and dynamical. This nature implies that supporting the sustainable long term evolution of service applications should play a central role in developing quality assurance and testing techniques and tools [12]. Service-oriented computing emphasizes service integration and composition where new services are dynamically discovered and integrated. This evolutionary

nature calls for new test mechanisms such as “on demand” test development to support this paradigm.

Software model is considered as a promising technology for supporting service-oriented computing in the open distributed Grid environment [13]. In the related work of applying modeling technology to testing web services and web applications, one focus has been on testing at the function level by exploiting the internal structure and/or behavior of a service component[14, 15]. Test plans are derived from data flow analyses of web applications and test agents are constructed to carry out these test plans [15]. Another focus is at the software component level. For example, [16] utilizes “validation agents” to automate the testing of software components. These validation agents depend on the “component aspects” that specify the functional and non-functional cross-cutting concerns of software components to construct and run automated tests. Test environments for large scale service collaborations are also developed. The Agent cities test-bed [17] is an agent-based test environment developed to support large-scale dynamic service synthesis testing. In relation to the objectives of this paper, two research issues emphasized by current work are: 1) automating the construction of test models, and 2) deploying multiple models in test federations to collaborate in testing complex distributed interaction patterns. To achieve these requires a mature methodology based on a formal mathematical framework.

Web services technology is changing the Internet, augmenting the eyeball web with capabilities to produce the transactional web [19]. The eyeball web is dominated by program-to-user business-to-consumer (B2C) interactions. The transactional web will be dominated by program-to-program business-to-business (B2B) interactions. This transformation is being fueled by the program-to-program communication model of Web services built on existing and

emerging standards such as Hypertext Transfer Protocol (HTTP), Extensible Markup Language (XML), Simple Object Access Protocol (SOAP),Web Services Description Language (WSDL), and the Universal Description, Discovery, and Integration (UDDI) project.

Web services technologies provide a language-neutral, environment-neutral programming model that accelerates application integration inside and outside the enterprise. Application integration through Web services yields flexible loosely coupled business systems. Because Web services are easily applied as a wrappering technology around existing applications and information technology assets, new solutions can be deployed quickly and recomposed to address new opportunities. As adoption of Web services accelerates, the pool of services will grow, fostering development of more dynamic models of just-in-time application and business integration over the Internet.

WS Actors

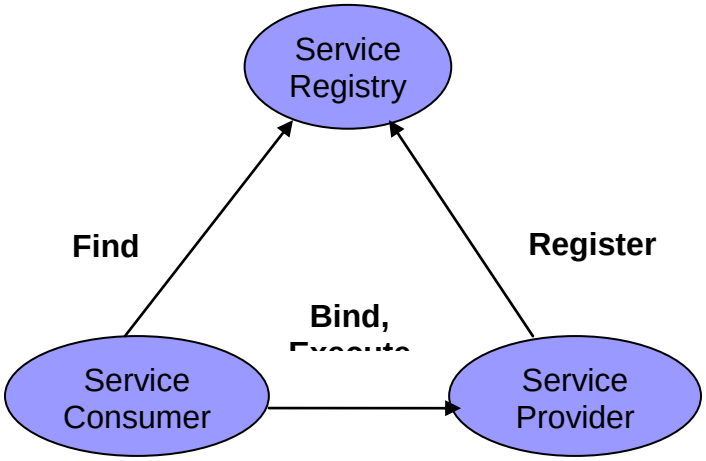

Figure 2.1 WS Actors.

A Web service is an interface that describes a collection of operations that are network-accessible through standardized XML messaging. A Web service performs a specific task or a set of tasks .A Web service is described using a standard, formal XML notation, called service

description, that provides all of the details necessary to interact with the service, including message formats (that detail the operations), transport protocols, and location. Web service descriptions are expressed in WSDL. We describe Web services in terms of a service-oriented architecture. As depicted in Figure 2.1, this architecture sets forth three roles and three operations. The three roles are the service provider, the service requester, and the service registry. The objects acted upon are the service and the service description, and the operations performed by the actors on these objects are publish, find, and bind. A service provider creates a Web service and its service definition and then publishes the service with a service registry based on a standard called the Universal Description, Discovery, and Integration (UDDI) specification. Once a Web service is published, a service requester may find the service via the UDDI interface. The UDDI registry provides the service requester with a WSDL service description and a URL (uniform resource locator) pointing to the service itself. The service requester may then use this information to directly bind to the service and invoke it. For a more complete description of these Web services components, see the architecture document.

Web Services Technology Stack

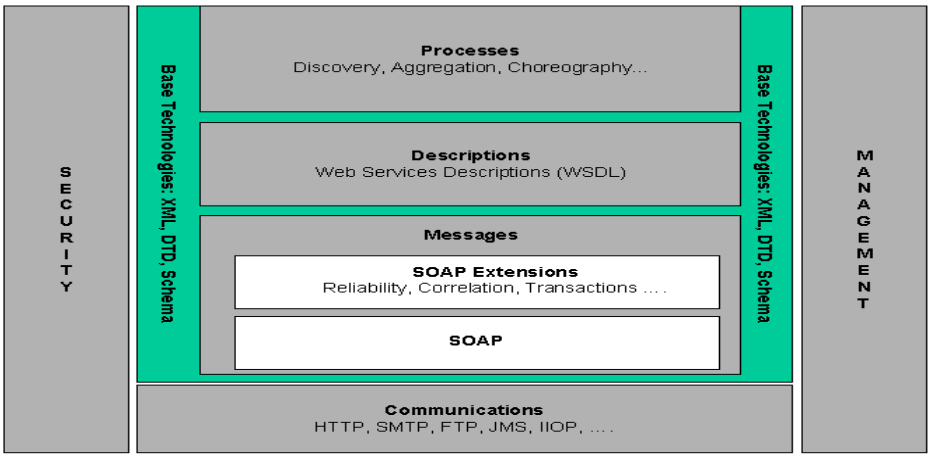

*Figure 2.2 WS Technology Stack [**W3C Web Services Architecture Working Draft, August 2003]***

2.1 Business Process Execution Language (BPEL) and SOA

Business processes can be described in two ways. Executable business processes model actual behavior of a participant in a business interaction. Abstract business processes are partially specified processes that are not intended to be executed. An Abstract Process may hide some of the required concrete operational details. Abstract Processes serve a descriptive role, with more than one possible use case, including observable behavior and process template. WS-BPEL is meant to be used to model the behavior of both Executable and Abstract Processes. [18]WS-BPEL provides a language for the specification of Executable and Abstract business processes. By doing so, it extends the Web Services interaction model and enables it to support business transactions. WS-BPEL defines an interoperable integration model that should facilitate the expansion of automated process integration in both the intra-corporate and the business-to-business spaces.

WS-CDL+ differs from WS-BPEL in the following ways; WS-BPEL is orchestration which implies a centralized control mechanism whereas WS-CDL is choreography which has no centralized control. Instead control is shared between domains. WS_BPEL is used for a single domain while WS-CDL is used cross domains. WS-BPEL lacks formal semantics, WS-CDL is based on formal languages. WS-BPEL is execution language (process oriented), WS-CDL is description language (Design Oriented). BPEL Process is a container where you can declare relationships to external partners, declarations for process data, handlers for various purposes and, most importantly, the activities to be executed. On top, the process container has a couple of attributes, i.e. a (mandatory) name and a (also mandatory) declaration of a namespace – as shown in the example below. You should note that not all possible attributes of the process element are shown in this example.

```
<process name="PrimerProcess"
        targetNamespace="http://oasis-open.org/WSBPEL/Primer/"
        xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable" />
```

BPEL Business Processes offer the possibility to aggregate web services and define the business logic between each of these service interactions. It is also said that BPEL orchestrates such web service interactions. Each service interaction can be regarded as a communication with a business partner. The interaction is described with the help of partner links. Partner links are instances of typed connectors which specify the WSDL port types the process offers to and requires from a partner at the other end of the partner link.

Note that for one partner, there can be a set of partner links. You can regard one partner link as one particular communication channel. Such an interaction is potentially two sided: the process invokes the partner and the partner invokes the process. Therefore, each partnerLink is characterized by a partner link type and a role name. This information identifies the functionality that must be provided by the business process and by the partner service.

```
<partnerLinks>
        <partnerLink name="ClientStartUpLink"
                partnerLinkType="wsdl:ClientStartUpPLT" myRole="Client" />
</partnerLinks>
```

Partner link declarations can take place directly under the process element, which means that they are accessible by all BPEL constructs within the BPEL process, or under a scope element, which means they are only accessible from the children of that scope.

BPEL Composition of Web services

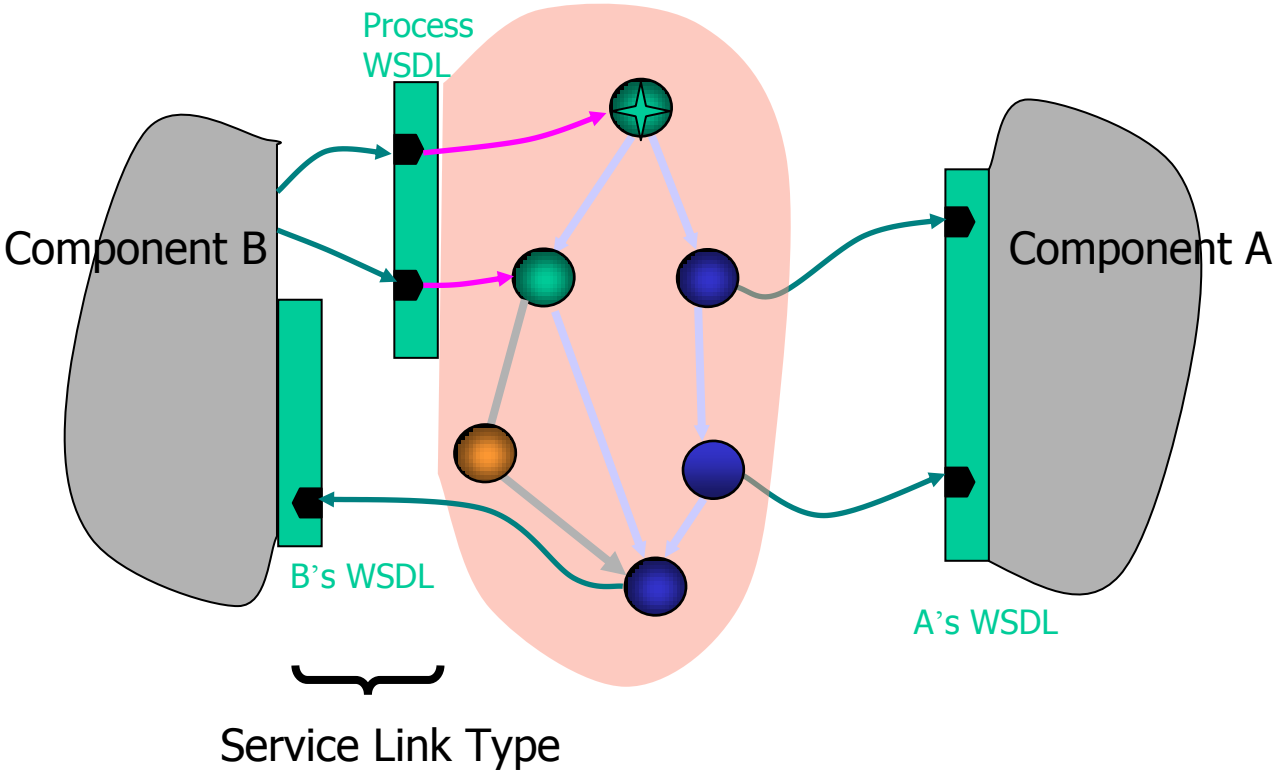

Figure 2.3 Composition of Web Services [Nirmal Mukhi Component Systems DepartmentIBM Research]

3.0 WS-CDL Overview

WS-CDL is a language for specifying peer-to-peer protocols where each party wishes to remain autonomous and in which no party is master over any other – i.e. no centralization point. The description of a peer-to-peer protocol is grounded in what we term an ordered set of interactions, where an interaction is defined loosely as an exchange of messages between parties. It is essential in understanding Web Services Choreography Description Language (WS-CDL) to realize that there is no single point of control. There are no global variables, conditions or work-units. To have them would require centralized storage and orchestration. WS-CDL does permit a shorthand notation to enable variables and conditions to exist in multiple places, but this is syntactic sugar to avoid repetitive definitions. There is also an ability for variables residing in one service to be aligned (synchronized) with the variables residing in another service, giving the illusion of global or shared state.

It is also important to understand that WS-CDL does not distinguish between observable messages from applications, that might be considered as application or business messages, from the infrastructure upon which an application is based, that might be considered as some form of signal. In WS-CDL all messages are described as information types and have no special significance over each other. All that WS-CDL describes is the ordering rules for the messages which dictate the order in which they should be observed. When these ordering rules are broken WS-CDL considers them to be out-of-sequence messages and this can be viewed as an error in conformance of the services that gave rise to them against the WS-CDL description.

WS-CDL is an XML-based language that can be used to describe the common and collaborative observable behavior of multiple services that need to interact in order to achieve some goal. WS-CDL describes this behavior from a global or neutral perspective rather than

from the perspective of any one party and we call a complete WS-CDL description a global model. Services are any form of computational process with which one may interact, examples are a buying process and a selling process that are implemented as computational services in a Service Oriented Architecture (SOA) or indeed as a Web Services implementation of an SOA. Because WS-CDL is not explicitly bound to WSDL it can play the same global model role for both SOA services and Web Services, that is it is possible to use WS-CDL to describe a global model for services with no WSDL descriptions (perhaps they just have Java interfaces) as easily as it is to describe services that do have or will have WSDL descriptions. The way in which WS-CDL can be used without WSDL descriptions is however implementation dependent.

Common collaborative observable behavior is the phrase we use to describe the behavior of a system of services, for example buyer and seller services, from a global perspective. Each service has an observable behavior that can be described today using WSDL or some other interface description language (e.g. Java). Such observable behavior is described as a set of functions, possibly with parameters, that a service offers coupled with error messages or codes that indicate failure along with the return types for the functions offered. If we used abstract BPEL along with WSDL we can also describe the valid sequences of functions from a single services perspective (i.e. the service we are describing), which is not possible with WSDL or Java alone. We refer to this set as the “observable behavior” for a service. This level, the service level, of “observable behavior” does not describe behavior of a system of services because it only deals with a single service. The composition of a set of "observable behaviors" at a service level is what we call the common collaborative observable behavior. The composition is not simply the set of observable behaviors at the service level operating together because such a composition requires further description of the dependencies that the set of services exhibit in

order to interoperate correctly. If we captured the ordering rules for a set of service then we would have the common collaborative observable behavior fully specified and this is what WS-CDL+ is for.

Individual service behaviors can be used in the composition of wider collaboration in which a set of services with their own behaviors could be effectively used. In order to do so a global model that described the peer to peer observable interactions of such a set of services is required to ensure that the services will in-fact cooperate to a commonly understood script. That script is the global model and that script is what WS-CDL+ is used to describe.

A global model ensures that the common collaborative observable behavior is not biased towards the view of any one of the services. Instead it describes as peers the entire collaborative observable behavior of all of the services such that no one service can be said to exert any control over any other service. In effect it described the services as a complete distributed application in which each service plays a distinct role and has distinct relationships with its peer services.

One may think of WS-CDL as a language for describing the observable activities of a set of services some of which are synchronized through some common understanding realized by a specific business interaction between the services or by a declaration of interest in the progress of one service by another (e.g. has the buyer accepted the price offered by the seller). The least interesting scenario is one in which WS-CDL can be used to describe a set of services that never synchronize at all; that is there is no observable relationships and no statement of an unobservable relationship that exists between the services. In this case the services perform choreography, but effectively on different stages and thus need no form of coordination (e.g. a buyer and seller choreography for Wall-Mart versus Bloomberg Reuter’s choreography for the

exchange of news items). In all other cases the synchronization is what makes life interesting (e.g. buyer and seller choreography coupled with seller credit check choreography or indeed a seller shipper choreography).

In WS-CDL the mechanisms for describing the common observable behavior range from specific information alignment (e.g. when a buyer and seller record the fact that an order has been accepted in variables that reside at the buyer and at the seller), interaction (e.g. when a buyer requests a price from a seller and receives a price as a response from the seller) and a declaration of interest in the progress of a choreography (e.g. has the bartering choreography between buyer and seller "started" or has it "finished"). In the first two cases synchronization is explicit and visible as a business related activity (e.g. the observable recording of information and it's alignment and the description of an information exchange between a buyer and seller) and in the last case (e.g. choreography has "started" or "finished") it is implicit based on the progress of a choreography and not any business relationships.

WS-CDL Overview Diagram

<table>
<tr><td colspan="3">Business Collaboration Language:
Choreography Description Language</td><td rowspan="2">Quality of Service</td></tr>
<tr><td colspan="3">Business Process Languages
BPEL, CI, XPDL, BPML</td></tr>
<tr><td rowspan="3">Reliable Messaging</td><td rowspan="3">Security</td><td>Transactions</td><td rowspan="3">Quality of Service</td></tr>
<tr><td>Security</td></tr>
<tr><td>Context</td></tr>
<tr><td colspan="3">UDDI</td><td>Discovery</td></tr>
<tr><td colspan="3">WSDL</td><td>Description</td></tr>
<tr><td colspan="3">SOAP</td><td rowspan="2">Transport</td></tr>
<tr><td colspan="3">XML, Encodings</td></tr>
<tr><td colspan="3">HTTP, BEEP, IIOP, JMS, SMTP</td><td>Transport</td></tr>
</table>

Figure 3.1 WS-CDL Overview

3.1 WS-CDL Elements

The WS-CDL language is, as are most other languages for describing Web services, based on XML. Given that no graphical syntax exists, descriptions of web service interactions become large for even small examples. The WSCDL language can be understood to have two parts: a part describing static relationships which are invariant, e.g., for characterizing types of XML data, for defining types of communication channels, types of communication partners, and so on. The second part describes dynamic behavior: interactions between communication partners (web services) and their temporal interdependencies.

3.1.1 Static WS-CDL

The static part defines the collaborating entities in a WS-CDL specification. **Role Types:** All interactions take place between different role types, for example between a Customer role type and a Retailer role type. A typical definition of role types can be found on lines 30–39 in figure 1. It would probably have been preferable to call these entities roles instead of role types, as there is no possibility to create separate instances from a role type definition, but in this paper we will continue using the vocabulary of the WS-CDL definition. The same confusion with regards to types versus instances holds also for Participant and Channel Types (covered next).

Participant Types: A participant type groups together role types that are conceptually going to be realized by the same physical entity. For instance, if both the Retailer and the Warehouse role types of a business interaction are implemented by the same organization, the organization could be modeled using one Participant Type. In the example the participant types are defined on lines 9–15.

Relationship Types: A relationship type declares that two Role Types (invariably two) have a need to interact to realize some set of web services, as seen on lines 41–44 in the static part of the example.

Channel Types: Channel types describe the medium used to communicate between role types. A communication always, surprisingly given the supposedly high-level nature of the specification language, involves two parties. In the WS-CDL standard multi-party communication is not possible. WS-CDL interactions have two-phases, a request followed by an optional response. Two channel types for communicating with a retailer, and a channel for directly communicating with a customer, are provided on lines 46–57 of the example.

A channel type identifies the responding role type but not the requesting role type. Furthermore a channel type may include a section that describes how channel type instances may be transmitted over another channel instance in interactions. Interestingly, the channel type may also restrict how many times and in what circumstances channel instances are used; e.g. stipulating for instance that a given instance can only be used once, in a single communication or shared that it can be used in multiple communications. However, the usages of these restriction modes for channel instances is poorly explained in the standards document, nor are such features present in the core π-calculus formalism either. Finally the static part also contains definitions of types, and of functions, defined in the XPath formalism, for retrieving subfields of XML based document data.

3.1.2 Dynamic WS-CDL

The core of WS-CDL can be found in the dynamic part. Here the notion of choreography is defined. In WS-CDL choreography represents, essentially, a use case. Choreography concerns

a set of relationships (interactions between role types) and includes a definition of the set of variables used to realize the data dependent behavior of the choreography. A particular variable is located at a particular participant type, and all role types at a participant type share access to such a variable. A role type that is located at a different participant type from where a variable is located cannot access the variable. Thus, in the normal case, there is no implicit communication between web services located at different participants using shared variables.

The choreography activity construct describes the behavior of the choreography (as seen next), while an exception block defines the handling of exceptional events during the execution of the choreography activity, and a finalize block describes the actions to be taken upon termination of the choreography. In addition choreography can define sub-choreographies, and can make available to these sub choreographies its own variables.

3.1.2.1 Choreography Activities

An activity is the basic programmatic building block of WS-CDL. Essentially there are three classes of activities: structural ones, work-unit activities, and basic activities.

Basic Activities: The principal basic activity is the interaction; an example interaction can be seen on lines 25-53 in figure 2. An interaction describes the binary communication between two role types, as a request–response set of events exchanging information about the value of variables located at the respective role type (or rather: participant types). Other basic activities include the starting of a sub-choreography (using the perform statement) and the assigning of variables (assign).

Work-units: A work-unit is essentially a conditional and a looping construct combined in one, enclosing an activity. A work-unit has a guard, which if true permits the execution of the

enclosed activity. If false, and if a blocking execution strategy has been specified, and the guard has a chance of becoming true in the future, the execution of the guards is halted until it becomes true. Upon successful execution of the work-unit activity the repeat condition of the work-unit is analyzed, and if the condition is true, the work-unit is enabled for future execution. An example work-unit that checks whether a variable has been initialized is shown on lines 15–23 in figure 2.

Structural Activities: The most basic structural activity is the sequence, specifying that its arguments should be evaluated in sequence. The parallel activity, in contrast, permits parallel execution of its activity arguments. Finally the choice activity selects, based on the excitability of its arguments, one of its arguments for continued execution. If, for instance, one branch continuously blocks (a work-unit waiting for its guard condition to become true) then that choice branch will never be chosen. These constructs bear a strong resemblance to the corresponding process operators in the π-calculus in contrast to the more flow-oriented concurrency constructs available in WS-BPEL.

```
1   <informationType name="purchaseOrderType"
2                    type="tns:PurchaseOrderMsg"/>
3   <informationType name="purchaseOrderAckType"
4                    type="tns:PurchaseOrderAckMsg"/>
5   <informationType name="badPOAckType"
                     type="xsd:QName" />
6   <informationType name="uriType"
                     type="xsd:string" />
7   <informationType name="intType"
                     type="xsd:integer" />
8
9   <participantType name="Consumer">
10          <roleType typeRef="tns:Consumer" />
11  </participantType>
12
13  <participantType name="Retailer">
14          <roleType typeRef="tns:Retailer" />
15  </participantType>
16
17  <token name="purchaseOrderID" informationType="tns:intType"/>
18  <token name="retailerRef" informationType="tns:uriType"/>
19  <token name="consumerRef" informationType="tns:uriType"/>
20
21  <tokenLocator tokenName="tns:purchaseOrderID"
```

```
                    informationType="tns:purchaseOrderType"
                    query="/PO/orderId"/>
<tokenLocator       tokenName="tns:purchaseOrderID"
                    informationType="tns:purchaseOrderAckType"
                    query="/PO/orderId"/>
<tokenLocator       tokenName="tns:retailerRef"
                    query="/PO/retailerId"
                    informationType="tns:purchaseOrderType"/>

<roleType name="Consumer">
        <behavior   name="consumerForRetailer"
                    interface="rns:ConsumerRetailerPT"/>
        <behavior   name="consumerForWarehouse"
                    interface="rns:ConsumerWarehousePT"/>
</roleType>
<roleType name="Retailer">
        <behavior   name="retailerForConsumer"
                    interface="rns:RetailerConsumerPT"/>
</roleType>

<relationshipType   name="ConsumerRetailerRelationship">
        <roleType   typeRef="tns:Consumer"
                    behavior="consumerForRetailer"/>
        <roleType   typeRef="tns:Retailer"
                    behavior="retailerForConsumer"/>
</relationshipType>

<channelType name="ConsumerChannel">
        <roleTypetypeRef="tns:Consumer"/>
        <reference><token name="tns:consumerRef"/></reference>
        <identity><token name="tns:purchaseOrderID"/></identity>
</channelType>

<channelType name="RetailerChannel">
        <passing    channel="ConsumerChannel" action="request" />
        <roleType   typeRef="tns:Retailer"
                    behavior="retailerForConsumer"/>
        <reference><token name="tns:retailerRef"/></reference>
        <identity><token name="tns:purchaseOrderID"/></identity>
</channelType>
```

Figure 1: WS-CDL Example: Static Part

```
1   <choreography   name="ConsumerRetailerChoreography"    root="true">
2       <relationship       type="tns:ConsumerRetailerRelationship"/>
3       <variableDefinitions>
4           <variable       name="purchaseOrder"
                            informationType="tns:purchaseOrderType"
5                                                           silent="false" />
6           <variable       name="purchaseOrderAck"
7                           informationType="tns:purchaseOrderAckType" />
8           <variable       name="retailer-channel"
                            channelType="tns:RetailerChannel"/>
9           <variable       name="consumer-channel"
                            channelType="tns:ConsumerChannel"/>
10          <variable       name="badPurchaseOrderAck"
11                          informationType="tns:badPOAckType" />
12      </variableDefinitions>
13
14      <sequence>
15          <workunit       name="unit" block="false"
```

```
16                              gua rd =" n o t ( c d l : i sVa r i a b l eAv a i l a b l e (
'                                       t n s : pur cha s eOrde r ' , ' t n s : Consumer ' ) )">
17                              <assign  roleType =" tns:Consumer">
18                                      <copy     name=" copy1">
19                                              <source   expression =" cdl:doc (
'                                                       <PO> <orderId name="10" />
                                                <CustomerRef       name="1000" />
                                        <target  variable =" cdl:getVariable (
                                                'tns:purchaseOrder ' , ' ' , ' ' ) " / >
21                                      </copy>
22                                      </ assign>
23                      </workunit>
24
25                      <interaction  name=" createPO "
26                              channelVar iable =" tns:RetailerChannel "
27                              operation =" handlePurchaseOrder">
28                              <p a r t i c i p a t e  r e l a t i o n s h i p T y p e="
                                        t n s : Co n s ume rRe t a i l e rRe l a t i o n s h i p "
29                                      f romRoleTypeRef =" t n s : Consumer "
                                        toRol eTypeRe f=" t n s : R e t a i l e r "/>
30                              <exchange          name=" r e q u e s t "
31                                      i n f o rma t i o nTy p e =" t n s : pur cha s eOrde rType "
                                        a c t i o n =" r e q u e s t">
32                                      <s end v a r i a b l e =" c d l : g e tVa r i a b l e ( '
                                        t n s : pur cha s eOrde r ' , ' ' , ' ' ) " />
33                                      <r e c e i v e v a r i a b l e =" c d l : g e tVa r i a b l e ( '
                                        t n s : pur cha s eOrde r ' , ' ' , ' ' ) "
34                                      r e c o r dRe f e r e n c e=
                                        " r e c o r d−the−channe l−i n f o " />
35                              </exchange>
36                              <exchange          name=" r e s p o n s e "
37                                      informationType =" purchaseOrderAckType"
                                        action =" respond">
38                                      <send     variable =" cdl:getVariable ( '
                                        tns : purchaseOrderAck ' , ' ' , ' ' ) " />
39                                      <receive  variable =" cdl:getVariable('
                                        t n s : pur cha s eOrde rAck ' , ' ' , ' ' ) " />
40                              </exchange>
41                              <exchange          name=" badPurchaseOrde rAckException "
                                        f aul tName =" badPur cha s eOrde rAckEx c ep t i on "
42                                      informationType ="badPOAckType " action =" respon d">
43                              <send      variable =" cdl:getVariable ( '
                                        tns : badPurchaseOrderAck ' , ' ' , ' ' ) "
44                                      causeException =" tns : badPOAck" />
45                              <receive  variable =" cdl:getVariable ( '
                                        tns : badPurchaseOrderAck ' , ' ' , ' ' ) "
46                                       causeException =" tns:badPOAck" />
47                              </exchange>
48                              <record  name=" record−the−channel−info " when=" after ">
49                                      <source variable =" cdl:getVariable ( '
                                                tns:purchaseOrder ' , ' ' ,
50 '                                             /PO/CustomerRef ' )"/>
51                                      <target variable =" cdl:getVariable ( '
                                                tns:consumer−channel ' , ' ' , ' ' ) " / >
52                              </ recor d>
53                      </ interaction>
54              </ sequence>
55 </ choreography>
56 </package>
```

Figure 2: WS-CDL Example: Dynamic Part

3.2 Using WS-CDL

WS-CDL is a description and not an executable language, hence the term “Description” in its name. It is a language that can be used to unambiguously describe observable service collaboration; we might also refer to this as a business protocol. When WS-CDL is focused on describing collaboration within a domain of control (e.g. a single company or enterprise) WS-CDL is used to describe the internal workflows that involve multiple services (also called end-points) that constitute observable collaborative behavior. The value in so doing is to encourage conformance of services to a negotiated choreography description and to improve interoperability of services through an agreed choreography description. This is no more than describing a business protocol that defines an observable collaboration between services. You can think of it as ways of ensuring services are well behaved with respect to the goals you wish to achieve within your domain.

When the focus of WS-CDL is across domains of control, WS-CDL is used to describe the ordering of observable message exchanges across domains such as those that govern vertical protocols such as FPML, FIX, TWIST and SWIFT. These protocols have some form of XML data format definition and then proceed to describe the ordering of message exchanges using a combination of prose and UML sequence diagrams.

3.3 Why use WS-CDL?

WS-CDL can be used to ensure interoperability within and across domains of control to lower interoperability issues, and create solutions within and across domains of control. WS-CDL can be used to ensure that the total cost of software systems in a distributed environment, within a domain of control and across the world-wide-web is lowered by guaranteeing that the

services that participate in choreography are well behaved on a continuous basis. Both of these benefits translate into greater up-time and so increase top line profits. At the same time they translate into less testing time and so reduce cost of delivery which decreases bottom line costs.

3.4 Interaction Based Information Alignment

In some choreographies there may be a requirement that when the interaction is performed, the roleTypes in the choreography have agreement on the outcome. More specifically within an interaction, a roleType may need to have a common understanding of the observable information creations or changes of one or more state capturing variables that are complementary to one or more state capturing variables of its partner roleType. Additionally, within an interaction a roleType may need to have a common understanding of the values of the information exchange capturing variables at the partner roleType. For example, after an interaction happens, both the "Buyer" and the "Seller" want to have a common understanding that: State capturing variables, such as "Order State", that contain observable information at the "Buyer" and "Seller", have values that are complementary to each other, e.g. 'Sent' at the "Buyer" and 'Received' at the "Seller", and Information exchange capturing variables have the same types with the same content, e.g. The "Order" variables at the "Buyer" and "Seller" have the same informationTypes and hold the same order information.

In WS-CDL, an alignment interaction MUST be explicitly used in the cases where two interacting participants require the alignment of their observable information changes and the values of their exchanged information. After the alignment interaction completes, both participants progress at the same time in a lock-step fashion, and the variable information in both participants is aligned. Their variable alignment comes from the fact that the requesting

participant has to be assured that the accepting participant has received the message, and the accepting participant has to be assured that the requesting participant has sent the message before both of them progress. There is no intermediate state, where one participant sends a message and then it proceeds independently, or the other participant receives a message and then it proceeds independently. In other words, after each alignment interaction both participants act on the basis of their shared understanding for the messages exchanged and the information recorded.

3.5 Interaction Life-line

An interaction must complete normally when its message exchanges complete successfully. An interaction must complete abnormally when: An application signals an error condition, during the management of a request or within a participant when processing the request its time-to-complete timeout occurs after the interaction was initiated but before it completed. Some other type of error occurs, such as protocol based exchange failures, security failures, document validation errors, etc.

3.6 Interaction Syntax

The syntax of the interaction construct is:

```
<interaction  name="NCName"
  channelVariable="QName"
          operation="NCName"
          align="true"|"false"? >
            <participate  relationshipType="QName"
                      fromRoleTypeRef="QName" toRoleTypeRef="QName" />

                      <exchange  name="NCName"
                                faultName="QName"?
                                informationType="QName"?|channelType="QName"?
                                action="request"|"respond" >
                                <send      variable="XPath-expression"?
                                                  recordReference="list of NCName"?
                                                  causeException="QName"? />
                                <receive   variable="XPath-expression"?
```

```
                                        recordReference="list of NCName"?
                                        causeException="QName"? />

            </exchange>*

            <timeout  time-to-complete="XPath-expression"
                      fromRoleTypeRecordRef="list of NCName"?
                      toRoleTypeRecordRef="list of NCName"? />?

            <record   name="NCName"
                      when="before"|"after"|"timeout"
                      causeException="QName"? >
                 <source        variable="XPath-expression"? |
                                expression="XPath-expression"? />
                  <target  variable="XPath-expression" />
            </record>*
</interaction>
```

Figure 5.4 Interaction Syntax

The attribute name is used to specify a name for each interaction element declared within choreography. The channelVariable attribute specifies the channel capturing variable used for communicating during this interaction. The channel variable contains information about the participant that is the target of the interaction. The information is used for determining where and how to send and receive information to and from the participant. The channel variable used in an interaction must be available at the two roleTypes before the interaction may occur. At runtime, information about a channel variable is expanded further. This requires that the messages exchanged in the choreography also contain reference and correlation information, for example by: Including a protocol header, such as a SOAP header or Using the actual value of data within a message, for example the "Order Number" of the "Order" that is common to all the messages sent over the channel.

The operation attribute specifies the name of the operation that is associated with this interaction. The specified operation belongs to the interface, as identified by the roleType and behavior elements of the channelType specified in the channel variable used in this interaction. The optional align attribute, when set to "true", means that this alignment interaction results in

the common understanding of both the information exchanged and the resulting observable information creations or changes at the ends of the interaction, as specified in the fromRoleTypeRef and the toRoleTypeRef elements. In other words, after each alignment interaction both participants act on the basis of their shared understanding for the messages exchanged and the information recorded. The default value for this attribute is "false". Within the participate element, the relationshipType attribute specifies the relationshipType this interaction participates in, and the fromRoleTypeRef and toRoleTypeRef attributes specify the requesting and the accepting roleTypes respectively. The type of the roleType identified by the toRoleTypeRef attribute must be the same as the roleType identified by the roleType element of the channelType specified in the channel variable used in the interaction.

The optional exchange element allows information to be exchanged during an interaction. The attribute name is used to specify a name for this exchange element. When the action attribute is set to "request", then the information exchange happens from the 'from' roleType to the 'to' roleType. When the action attribute is set to "respond", then the information exchange happens from the 'to' roleType to the 'from' roleType. Within the exchange element, the send element shows that information is sent from a roleType and the receive element shows that information is received at a roleType respectively in the interaction.

The send and the receive elements must only use the WS-CDL function getVariable within the variable attribute. The optional variables specified within the send and receive elements must be of type as described in the informationType or channelType attributes. When the action element is set to "request", then the variable specified within the send element using the variable attribute must be defined at the 'from' roleType and the variable specified within the receive element using the variable attribute must be defined at the 'to' roleType. When the action

element is set to "respond", then the variable specified within the send element using the variable attribute must be defined at the 'to' roleType and the variable specified within the receive element using the variable attribute must be defined at 'from' roleType. Within the send or the receive elements of an exchange element, the recordReference attribute contains an XML-Schema list of references to record elements in the same interaction. The same record element may be referenced from different send or the receive elements within the same interaction thus enabling re-use. Within the send or the receive elements of an exchange element, if the optional causeException attribute is set, it specifies that an exception must be caused at the respective roleTypes. In this case, the "QName" value of this attribute will identify the exception that must be caused.

4.0 Mapping Collaborating Services to WS-CDL

WS-CDL is an XML based description language for describing interactions between web services. Unlike WS-BPEL[20], WS-CDL there is no unique web services being defined through other web services (WS Choreography). In WS-CDL the services are defined in a peer to peer manner and there is no notion of a coordinating web services. In popular terms the developers of WS-CDL like to distinguish between traditional orchestration languages, like WS-BPEL that uses existing services to orchestrate a new service, and between choreography languages, like WS-CDL, that describe the interaction points between existing Web Services without defining new such services[21]. WS-CDL is based on formal method π-calculus process algebra, and therefore the language is well suited for describing concurrent processes and dynamic interaction scenarios.

4.1 A Formal Model for WS-CDL

WS-CDL models choreography with a set of participant roles and the collaboration among them. We give the syntax and an operational semantics here. We use meta-variable R for role declarations; A and B for activity declarations; r, f and t for role names; x, y, u and v for variables; e, e_1 and e_2 for XPath expressions; g, g_1, g_2 and p for XPath Boolean expressions; op for operations offered by the roles. We use R as a shorthand for $R_1, \cdots, R_n$, for some n, (similarly, for x, op, e, etc.), r.x to refer to the variable x in role r, and r.x := e for $r.x_1 := e_1, \cdots, r.x_n := e_n$. A choreography declaration includes a name C, some participant roles R, and an activity A with the form: C[R, A]. Each role R has some local variables x and behaviors represented as a set of operations op. A role with name r is defined as:

R ::= r[x , op]

The basic activities in CDL are as follows:

BA ::= skip (skip)

| r.x := e (assign)

| comm (f.x → t.y, rec, op) (request)

| comm (f.x ← t.y, rec, op) (response)

| comm (f.x → t.y, f.u ← t.v, rec, op) (req-resp)

Here op specifies what the recipient does when receiving the message, rec represents the assignments f.x := e_1, t.y := e_2, where x and y are lists of state variables on the roles f and t respectively. The syntax of the activities is:

A,B ::= BA (basic)

| p?A (condition)

| p * A (repeat)

| g :A:p (workunit)

| A;B (sequence)

| A ∩ B (non-deterministic)

| g1 =>A[] g_2 =>B (general-choice)

| A || B (parallel)

The WS-CDL specification includes many well-formedness rules. In the semantic definition, we will always assume that the WS-CDL program is well-formed [8]. From this formal CDL Model let us assume we have n collaborating services, S_1, S_2, …, S_n, m service states E_{11}, E_{12}, …, E_{nm} such that service x {S_x, x=1,..,n} is in state y {E_{xy}, x=1,…,n nad y= 1,…,m}, Let us also assume the interaction between S_x and S_y such that x ≠ y is $I_{xy.}$ The

interaction between S_x and S_y (I_{xy}) is determined by operation I_{xy} which is provided by service y (S_y). To model and interaction between services with state, time and sequence can be defined by this general scenario [22]:

Service Sx calls operation Oxy on Sy with paramets Py. If Service Sy is in State Ey and it is called within time T it sends Response Ryx to Sx otherwise it sends Exception.

```
<roleType name=”SampleRequestor”>
        <behavior name=”Oxy” interface=”Sx”          //the role played by service x
</roleType>
<roleType name=”SampleResponder”>
        <behavior name=”Oyx” interface=”Sy”         //the role played by service y
</roleType>

<relationshipType name=”SampleRelationship” >       //relationship between two services req/resp
        <role type=”Sx”  behavior=”Oxy”/>
        <role type=”Sy”  behavior=”Oyx”/>
</relationshipType>

<channelType name=”smapleReqChannel>
        <role type=”SampleRequestor”/>
</channelType>

<channelType name=”smapleRespChannel>
        <role type=”SampleResponder”/>
</channelType>


<choreography name=”sampleChoreography  root=”true”>
        <relationship type=”Samplerelationship”
        <variabledefinitions
                <variable  name=”Pxy”
                        informationType=’tns:string”/>          //parameters passed to the operation

                <variable name =”Ryx”
                        informationType=”tns:string”/>          //response sent back to Service Sx

                <variable name=”Ex1”
                        informationType=tns;string/>             /serice status
        </vaiabledefinitions>

        <sequence>
                <workunit name=”unit”
                        guard="cdl:getVariable("Ex1", "", "",      //controls the service to proceed
                                "tns:Sx") =”normal”)'”>          // only if codition is met

                </workunit>
                <interaction name = “ServiceXToServiceY”
                    channellVariable=”SampleReqChannel” operation=”Oxy”>     // protocol to send data

                        <participate relationshipType="SampleReqSampleResp"
                                fromRole="tns:SampleRequestor" toRole="SampleResponder"/>

                        <exchange name=”request”
                                    informationType=”tns:string” action=”request”>
                                <send variable=”cdl;getVariable(“tns:Pxy”. “”, “”)”/>
```

```
                    <receive variable="cdl:getVariable("tns:Pxy","","")"/>
                <exchange>

                <exchange name="response"
                    informationType="tns:string" action="response">
                    <send variable="cdl;getVariable("tns:Pyx". "", "")"/>
                    <receive variable="cdl:getVariable("tns:Pyx","","")"/>
                <exchange>
                <timeout  time-to-complete="T"/>
            </interaction>
        </sequence>
</choreography>
```

4.2 WS-CDL Exception Handling

A choreography can sometimes fail as a result of an exceptional circumstance or an "error" that occurred during its performance. Different types of *exceptions* are possible including this non-exhaustive list:

- *Interaction failures* - for example, the sending of a message did not succeed
- *Protocol based exchange failures* - for example, no acknowledgement was received as part of a reliable messaging protocol
- *Security failures* - for example, a message was rejected by a recipient because the digital signature was not valid
- *Timeout errors* - for example, an interaction did not complete within the required time
- *Validation errors* - for example, an XML "Order" document was not well formed or did not conform to its XML-Schema definition
- *Application failures* - for example, the "Goods Ordered" were 'Out of stock'

4.3 Collaborations Constructs and WS-CDL Element Mapping

Collaboration in general is the process of working together to achieve common objective, it describes the dynamic relationship that exists among objects. Web Service collaboration in particular is the process of DISTRUBUTED synchronous/asynchronous processes working together to achieve a common objective. Large scale software development is a complex task; hence software architects decompose the overall system into smaller components. The task of building these components may be assigned to different teams and deployed to different nodes, web services is a common technology used to develop these distributed components.

Web services are services that are made available from a business's Web Service Engines for Web users or other Web-connected programs. Providers of Web services are generally known as application service providers. Web services are ubiquitous in applications development especially Service Oriented Architecture (SOA) shops and the accelerating creation and availability of these services is a major Web trend. Users can access some Web services through a peer-to-peer arrangement rather than by going to a central server. Besides the standardization and wide availability to users and businesses of the Internet itself, Web services are also increasingly enabled by the use of the Extensible Markup Language (XML) as a means of standardizing data formats and exchanging data. XML is the foundation for the Web Services Description Language. Distributed services encompass the functionality required to complete a task or a use case for a business requirement. A one use case (business requirement) is realized by the collaboration among two or more services. We can classify the participant of collaboration into two categories: Initiator and Receptors (one or more collaboration participants).

Initiators initiate the collaboration among participants of a use case (business requirement), it exchanges message with other participants and there is no constraint on the initiator on a given collaboration. Receptors on the other hand receive messages from the initiator or other receptors participating in collaboration, it exchanges messages with Initiators and other receptors and there are often constraints on when and how to participate on a given collaboration.

Messages are exchanged among participants of collaboration and there are two types of exchanges possible among collaboration participants. One way messages which the Initiator sends messages but does not receive message back; two way messages which the Initiator sends messages and receives one back. Messages returned by services to the Initiator are classified into two: Normal messages which hold normal response data and Exception messages which hold exception data.

There are two major constraints on web services collaboration, interactions constraints which one participant checks if a condition is satisfied before it can interact with other participant; exchange constraint which one exchange type will send a particular message depending on certain condition. A specification for collaboration should capture all the collaborations constraints and constructs; participants, messages, exchanges, interaction, exchange type, exchange constraints, interaction constraints, condition variables, request message, response message and exceptions. There is a specification language called WS-CDL which captures all of these constructs. We can also specify different choices of interactions we can execute.

Interaction State Chart Diagram:

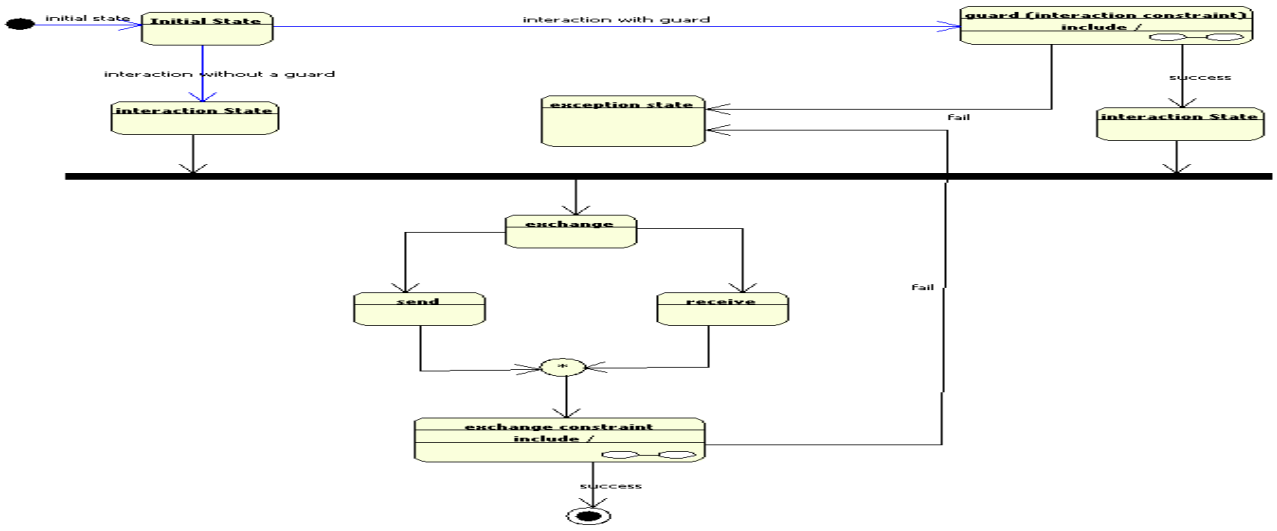

Figure 4.1 Service Interaction State Chart

Web Service Choreography Diagram:

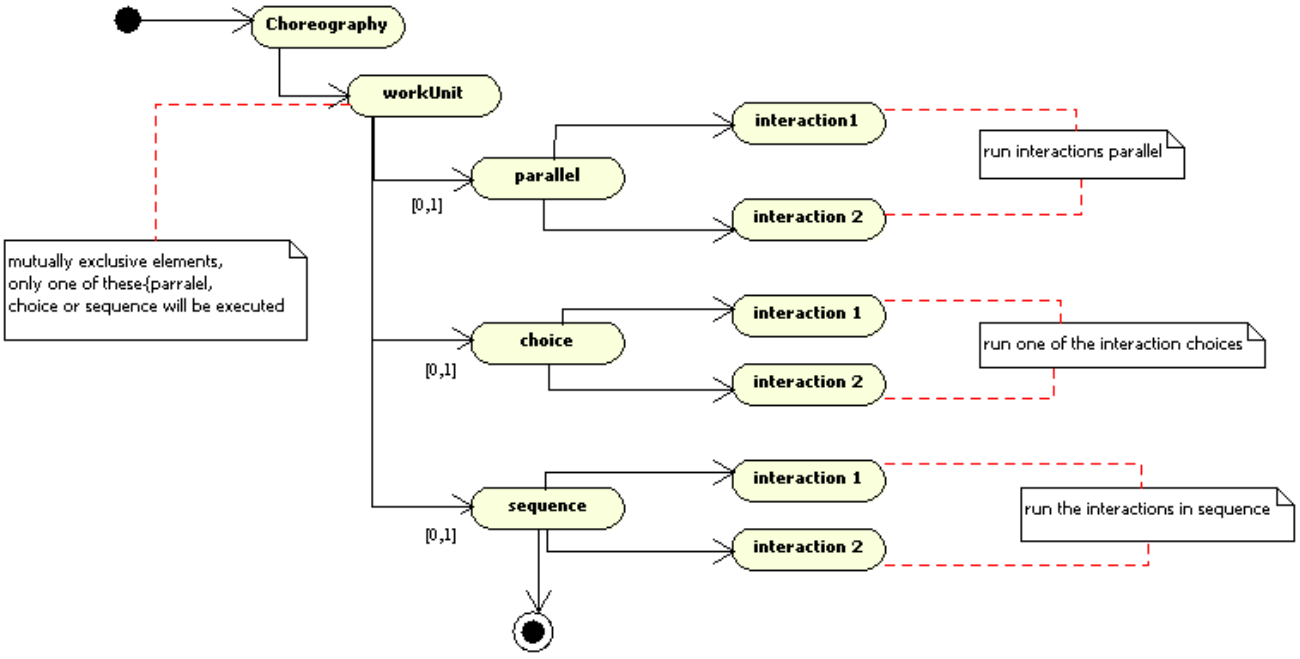

Figure 4.2 Choreography diagram

WS-CDL captures interaction by defining it like this:

```
1     <interaction  name=""
2           channelVariable="qname"
3                 operation="ncname"
4                 align="true"|"false"
5                 initiate="true"|"false" >
6           <participate relationshipType="qname"
7                         fromRole="qname"
8                         toRole="qname" />
9           <exchange   name="ncname"
10                      informationType="qname" |
11                      channelType="qname"
12                      action="request" >
13                <send variable="XPath-expression"
14                      recordReference="list of ncname"
15                      causeException="true"|"false" />
16                <receive    variable="XPath-expression"
17                            recordReference="list of ncname"
18                            causeException="true"|"false" />
19          </exchange>
20          <exchange   name="ncname"
21                      informationType="qname" |
22                      channelType="qname"
23                      action="response" >
24                <send variable="XPath-expression"
25                      recordReference="list of ncname"
26                      causeException="true"|"false" />
27                <receive    variable="XPath-expression"
28                            recordReference="list of ncname"
29                            causeException="true"|"false" />
30          </exchange>
31          <timeout    time-to-complete="XPath-expression"
32                      fromRoleRecordReference="list of ncname"
33                      toRoleRecordReference="list of ncname" />
34          <record     name="ncname"
35                      when="before"|"after"|"timeout"
36                      causeException="true"|"false" >
37                <source     variable="XPath-expression" |
38                            expression="Xpath-expression" />
39                <target     variable="XPath-expression" />
40          </record>
41    </interaction>
```

Figure 4.3 interaction WS-CDL

Lines 9 – 19 request exchange

Lines 20 – 30 response exchange

WS-CDL captures interaction constraint (guard) by defining it like this:

```
1     <workunit   name=""
2           guard= "cdl:getVariable
                  (tns:connectionBResp,
                  tns:ServiceBRole)==ready"/>  //test condition
3           repeat="xsd:boolean XPath-expression"
4           block="true|false" >
5           <interaction name="" channelVariable="qname"
6                 operation="ncname" >
7           </iteraction>
8     </workunit>
```

Figure 4.4 interaction constraint (guard) WS-CDL construct

WS-CDL captures exchange constraint by defining it like this:

```
1           <exchange   name="ncname"
2                       informationType="qname" |
3                       channelType="qname"
4                       action="request"|"respond" >
5                 <send variable= "(cdl:getVariable(
                              tns:connectionBResp,
                              tns:ServiceBRole)==available
                              ?passed:failed)"/>
5                       recordReference="list of ncname"
6                       causeException="true"|"false" />
7                 <receive    variable="XPath-expression"
8                             recordReference="list of ncname"
9                             causeException="true"|"false" />
10          </exchange>
```

Figure 4.5 exchange constraint ws-cdl construct

```
1     <workunit   name=""
2           guard= "cdl:getVariable
                  (tns:connectionBResp,
                  tns:ServiceBRole)==ready"/>  //test condition
3           repeat="xsd:boolean XPath-expression"
4           block="true|false" >
            <choice>
6                 <interaction name="inter1" channelVariable="qname"
7                       operation="ncname" >
8                 </iteraction>
9                 <interaction name="inter2" channelVariable="qname"
10                      operation="ncname" >
11                </iteraction>
12          </choice>
16    </workunit>
```

Figure 4.6 processing choice WS-CDL construct

channelVariable(2):communications channel declared for each participant

operation(3): name of the operation in the participant to be invoked

align(4): indicates if participant variables be aligned

initiate(5): indicates if this interaction is from Initiator or this is intermediary interaction.

Participate(6): indicates the two services participating the interaction and what role each one plays for the choreography

Exchange(9): message exchange mechanism defines the send and receive mechanism and the message types exchanged.

causeException(18): if this interchange throws exception or not.

Timeout(20): time to wait for the interaction to complete

5.0 Service Collaboration Scenario

In this section we are going to see three components which are collaborating and we will do both the Controlled Test and Collaborative Test. In this section we are going to demonstrate:

1. How to specify and test individual services
2. How to specify and test a collaboration.

This Test Scenerio will use the Three WS Components.

a) Service_A -- as a Service_B services consumer

b) Service_B -- service_B service produces

c) Security_Manager -- a proxy service between A and B

Services Sequence Diagram:

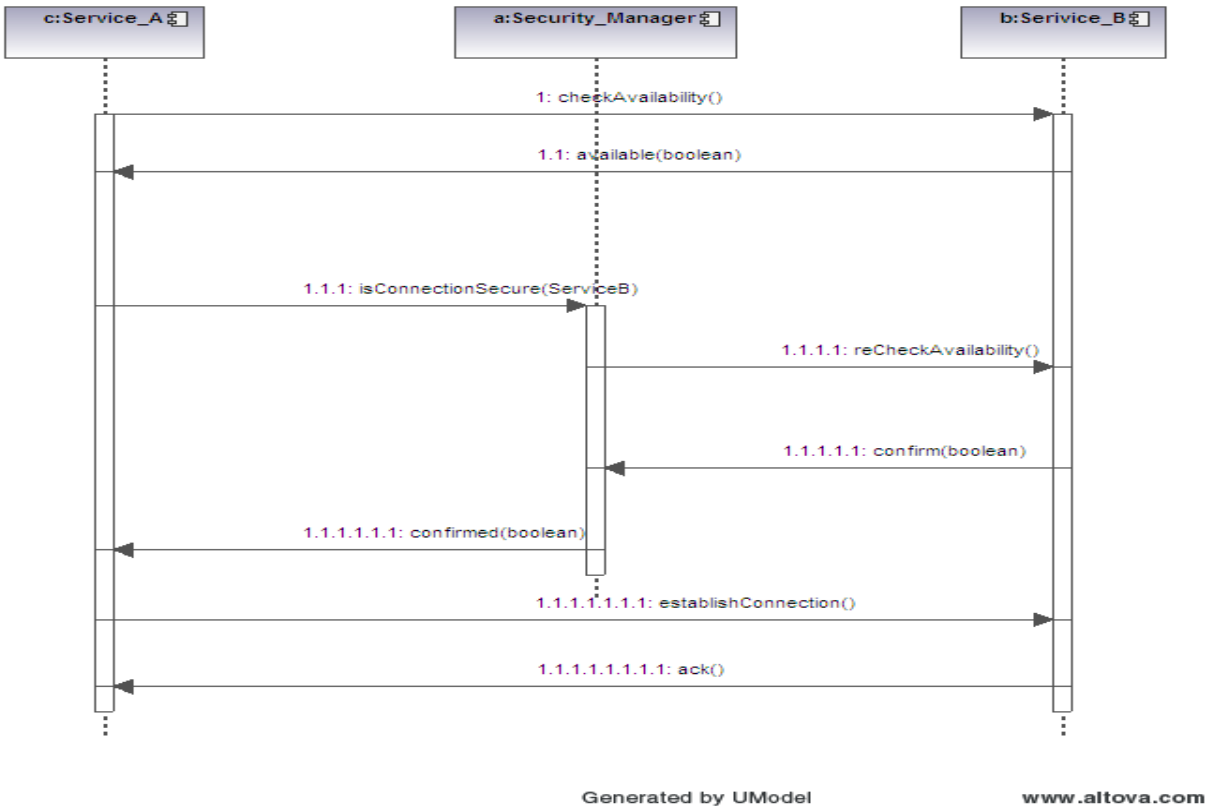

Figure 5.1, Sequence Diagram

Let us assume the following are three of the policies that govern the collaboration.

1. Service_A shall check Service_B's availability(within a time period T) before it asks for secure connection from Security_Manager. However, the reverse may not be true, i.e. Service A may not ask for secure connection even if it checked the availability of Service_B.
2. SM has to reCheck B's availability within a time period after receiving A's get secure connection
3. After answering available to Service_A, Service_B shall reply "confirm" to the security_Manager if it receives the recheck message (with in a time period T) from the Security Manager.
4. if Security_Manager receives 'ask for secure connection" from A, it will check B or C's availability but not D's.

In this thesis we focus on defining a language which captures the specification of this system in a formal way. This will make us easy to develop a test engine which is fed to this formal specification to observe that the collaborating parties adhere to the rules of the game (collaborations allowed by the specification).

1. Service_A shall check Service_B's availability(within a time period T) before it asks for secure connection from Security_Manager. However, the reverse may not be true, i.e. Service A may not ask for secure connection even if it checked the availability of Service_B.

NetworkAvailabilityChoreography does not depend on any other choreography and we define it this way:

```
<choreography name="NetworkAvailChoreography" root="true">
      <relationship type="ServiceA_serviceB"/>
      <variableDefinitions>
            <variable    name="netAvail"
```

```
                        informationType="networkAvailabilityType" />
            <variable   name="netAvailResp"
                        informationType="networkAvailabilityRespType" />
            <variable   name="network"
                        channelType="tns:ServiceBChannel"/>
      </variableDefinitions>

      <sequence>
            <interaction  name="checkAvailability"
                              channelVariable="tns:network"
                              operation="checkAvailability" initiate="true">
                  <participate relationshipType="tns:ServiceA_ServiceB"
                               fromRole="tns:ServiceARole"
                               toRole="tns:ServiceBRole"/>
                  <exchange   name="networkAvailabilityExchange"
                              informationType=
                                    "tns:networkAvailabilityType"
                              action="request">
                        <send variable=
                              "cdl:getVariable(tns:netAvail,
                                          tns:ServiceARole)"/>
                        <receive variable=
                              "cdl:getVariable(tns:netAvail,
                                          tns:ServiceBRole)"/>
                  </exchange>
                  <exchange   name="networkAvailabilityRespExchange"
                              informationType=
                                    "tns:networkAvailabilityRespType"
                              action="respond">
                        <send variable=
                              "cdl:getVariable(tns:netAvailResp,
                                    tns:ServiceBRole)"/>
                        <receive variable=
                                    "cdl:getVariable(tns:netAvailResp,
                                          tns:ServiceARole)"/>
                  </exchange>
            </interaction>
      </sequence>
</choreography>


<choreography name="ConnectionSecureChoreography" root="false">
      <relationship type="ServiceA_SecurityManager"/>

      <variableDefinitions>
            <variable   name="connection"
                        informationType="connectionType" free="true"
                        mutable="false"/>
            <variable   name="connectionResp"
                        informationType="connectionRespType"
                        mutable="false"/>
            <variable   name="secureConn"
                        channelType="tns:SecurityManagerChannel"/>
      </variableDefinitions>
      <workunit name="SecureConnection"
            guard="cdl:getVariable("netAvailableCalled")==true &&
                  ( cdl:getVariable("avilableTime")-
                        cdl:getCurrentTime()) < 20"
                  block="true”>
            <interaction name="connectionSecure"
                  channelVariable="tns:secureConn"
                  operation="connectionSecure" initiate="true">
```

```
                <participate
                        relationshipType="tns:ServiceA_SecurityManager"
                        fromRole="tns:ServiceARole"
                        toRole="tns:SecurityManagerRole"/>

                <exchange     name="connectionExchange"
                              informationType="tns:connectionType"
                              action="request">
                        <send variable="cdl:getVariable(
                              tns:connection, tns:ServiceARole)"/>
                        <receive variable=
                              "cdl:getVariable(tns:connection,
                                    tns:SecurityManagerRole)"/>
                </exchange>

                <exchange name="connectionRespExchange"
                              informationType="tns:connectionRespType"
                              action="respond">
                        <send variable=
                              "cdl:getVariable(tns:connectionResp,
                                    tns:SecurityManagerRole)"/>
                        <receive variable=
                              "cdl:getVariable(tns:connectionResp,
                        tns:ServiceARole)"/>
                </exchange>
            </interaction>
        </workunit>
</choreography>
```

Figure 5.2 SecureConnection and networkAvailability choreographies

2. SM has to reCheck B's availability within a time period after receiving A's get secure connection
3. After answering available to Service_A, Service_B shall reply "confirm" to the security_Manager if it receives the recheck message (with in a time period T) from the Security Manager.

Specification,#2 recheckAvailability (from ServiceA to serviceB), depends on the time ServiceA call (getSecureConnection) to SecurityManager, the two calls has to be within T time. SM has to reCheck B's availability within a time period after receiving A's get secure connection.

Specification #3 recheckavailability depends on the networkAvailable varaiable if true it returns accept otherwise reject.

```
<choreography name="RecheckAvailabilityChor" root="false">
      <relationship type="SecurityManager_serviceB"/>

      <variableDefinitions>
            <variable name="connectionB"
                  informationType="connectionBType" free="true" mutable="false"/>
            <variable     name="recheckAvailabilityResp"
                        informationType="reCheckAvailabilityRespType"
                        mutable="false"/>
                  <variable     name="secureConn"
                              channelType="tns:ServiceBChannel"/>
      </variableDefinitions>
      <choice>
         <workunit name="reCheckAvailability" guard="(cdl:getCurrentTime() –
                              cdl:getVariable("secureConnTime")" < 10)”
                                    block="true" >
            <interaction name="connectionSecure"
                        channelVariable="tns:secureConn"
                        operation="connectionSecure" initiate="true">
                  <participate
                        relationshipType="tns:SecurityManager_ServiceB"
                        fromRole="tns:SecurityManagerRole"
                        toRole="tns:ServiceBRole"/>
                  <exchange name="recheckAvailabiltyExchange"
                              informationType="tns:rechekAvailabiltyType"
                              action="request">
                        <send variable=
                              "cdl:getVariable(tns:reCheckAvailability,
                              tns:SecurityManagerRole)/>
                        <receive variable=
                              "cdl:getVariable(tns:reCheckAvailability,
                              tns:ServiceBRole)"/>
                  </exchange>
                  <exchange name="reCheckAvailabilityRespExchange"
                        informationType="tns:reCheckAvailabilityRespType"
                              action="respond">
                        <send variable=
                              "cdl:getVariable(tns:reCheckAvailabiltyResp,
                              tns:ServiceBRole)==true?confirm:reject”/>
                        <receive variable=
                              "cdl:getVariable(tns:reCheckAvailabilityResp,
                                    tns:SecurityManagerRole)"/>
                  </exchange>
            </interaction>
      </workunit>
    </choice>
</choreography>
```

Figure 5.3 recheckavailability choreography

Activities describe the actions performed within a choreography. The Activity-Notation is used to define activities as either: An ordering structure - which combines activities with other ordering structures in a nested way to express the ordering rules of actions performed within a choreography. A WorkUnit-Notation - which is used to guard and/or provide a means of repetition of those activities enclosed within the workunit. A basic activity - which is used to

describe the lowest level actions performed within a choreography. A basic activity is then either: An interaction activity , which results in an exchange of information between participants and possible synchronization of their observable information changes and the actual values of the exchanged information. A perform activity, which means that a complete, separately defined choreography is performed. An assign activity, which assigns, within one roleType, the value of one variable to another variable. A silentAction activity, which provides an explicit designator used for specifying the point where participant specific actions with non-observable operational details are performed. A noAction activity, which provides an explicit designator used for specifying the point where a participant does not perform any action. A finalize activity, which enables a particular finalizerBlock in a particular instance of an immediately enclosed choreography and thus brings that choreography to a defined conclusion.

Ordering structures combine activities with other ordering structures in a nested structure to express the ordering rules of actions performed within a choreography. An ordering structure is one of the following: sequence, parallel, choice: the sequence ordering structure contains one or more Activity-Notations. When the sequence activity is enabled, the sequence element MUST restrict the series of enclosed activities (as defined by one or more Activity-Notations) to be enabled sequentially, in the same order that they are defined.

The syntax of this construct is:

<sequence>

Activity-Notation+

</sequence>

The parallel ordering structure contains one or more Activity-Notations that are enabled concurrently when the parallel activity is enabled. The parallel activity MUST complete successfully when all activities (as defined by one or more Activity-Notations) performing work within it complete successfully.

The syntax of this construct is:

<parallel>

Activity-Notation+

</parallel>

The choice ordering structure enables specifying that only one of two or more activities (as defined by two or more Activity-Notations) MUST be performed. If no activities are selected, this MUST result in a runtime exception. When two or more activities are specified in a choice element, only one activity is selected and the other activities are disabled. If the choice has workunits with guard conditions, the first workunit that matches the guard condition is selected and the other workunits are disabled. Where there is more than one match, lexical ordering is used to select a match. If the choice has other activities, it is assumed that the selection criteria for those activities are non-observable.

The syntax of this construct is:

<choice>

Activity-Notation+

</choice>

In the example below, choice element has two interactions, "processGoodCredit" and "processBadCredit". The interactions have the same directionality, participate within the same

relationshipType and have the same fromRoleTypeRef and toRoleTypeRef names. If one interaction happens, then the other one is disabled.

```
<choice>
  <interaction name=""processGoodCredit"
      channelVariable="goodCredit-channel" operation="doCredit">
     ...
  </interaction>

  <interaction name=""processBadCredit"
      channelVariable="badCredit-channel" operation="doBadCredit">
     ...
  </interaction>
<choice>
```

Interacting

An interaction is the basic building block of a choreography. It results in information exchanged between collaborating participants and possibly the synchronization of their observable information changes and the values of the exchanged information. An interaction forms the base atom of the choreography composition. Multiple interactions are combined to form a choreography, which can then be used in different business contexts. An interaction is initiated when one of the roleTypes participating in the interaction sends a message through a common channel to another roleType that is participating in the interaction. If the initial message is a request, then the accepting roleType can optionally respond with a normal response message or a fault message, which will be received by the initiating roleType. An interaction also contains

"references" to: The channel capturing variable that describes where and how the message is to be sent to and received into the accepting roleType. The operation that specifies what the recipient of the message should do with the message when it is received. The ‘from' roleType and 'to' roleType that are involved the informationType or channelType that is being exchanged.The information exchange capturing variables at the 'from' roleType and 'to' roleType that are the source and destination for the message content. A list of potential state capturing variable recordings that capture observable information changes that can occur as a result of carrying out the interaction.

Web services Collaboration Engine Diagram:

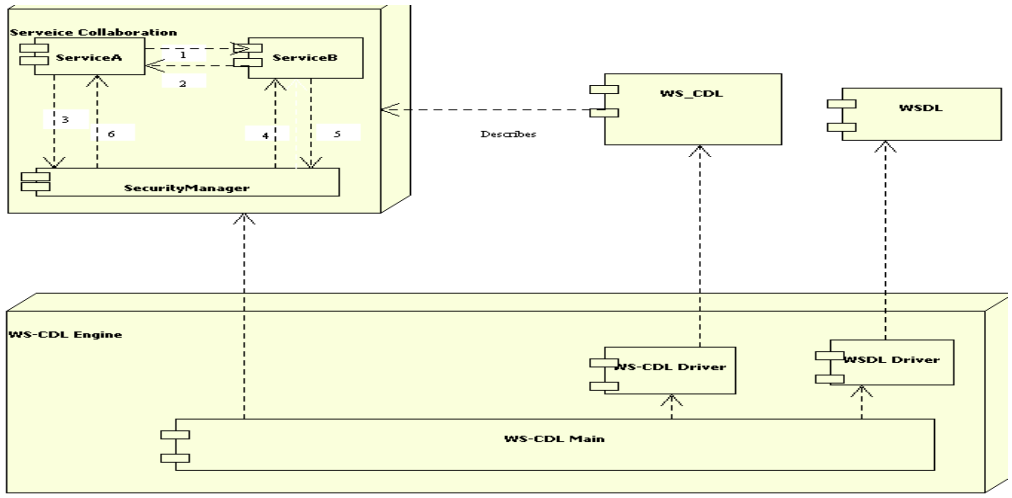

Figure 5.3, Web Services Collaboration Engine

5.1 Services Interfaces

```
package edu.gsu.cs.thesis;
public interface ServiceB {
	public boolean checkAvailability();
	public boolean reCheckAvailability();
	public boolean establishConnection();
	public boolean getAvailable();
	public long getAccessedTime();
	public void resetServiceB(boolean available, long accessedTime);
}
```

```
package edu.gsu.cs.thesis;

public class ServiceBImpl  implements java.io.Serializable {
	boolean available=true;
	long accessedTime=0;

	public boolean checkAvailability() {
		boolean ret=false;
		if (available) {
			accessedTime = System.currentTimeMillis();
			available = false;
			ret = true;
		} else {
			ret = false;
		}
		return ret;
	}
	public boolean reCheckAvailability() {
		boolean ret=false;
		if(available){
			accessedTime = System.currentTimeMillis();
			available = false;
			ret =true;
		} else {
			if((System.currentTimeMillis() - accessedTime) < 20000) {
				accessedTime = System.currentTimeMillis();
				ret=true;
			} else {
				ret=false;
			}
			return ret;
		}
	}

	public boolean establishConnection() {
		boolean ret;
		if(reCheckAvailability()) {
			ret=true;
		} else {
			ret=false;
		}
		return ret;
	}

	public void resetServiceB(boolean available, long accessedTime){
		this.available=available;
		this.accessedTime=accessedTime;
	}

	public boolean getAvailable(){
		return available;
	}

	public long getAccessedTime(){
		return accessedTime;
```

```
	}
}


package edu.gsu.cs.thesis;
public interface SecurityManager {
	public boolean getSecureConnection(boolean available, long accessedTime);
}
package edu.gsu.cs.thesis;
import edu.gsu.cs.thesis.proxy.*;

public class SecurityManagerImpl {
	public SecurityManagerImpl() {
	}

	public boolean getSecureConnection(boolean available, long accessedTime) {
		boolean ret= false;
		ServiceBProxy proxy = new ServiceBProxy();
		try {
			proxy.resetServiceB(available, accessedTime);
			if(proxy.reCheckAvailability()) {
				ret=true;
			} else {
				ret = false;
			}
		} catch(Exception ex) {
		}
		return ret;
	}
}
```

Figure 5.5 Service Interfaces

6.0 Conclusion

Web Services is the preferred implementation choice of Service Oriented Architecture(SOA). The Web Services components that make up SOA application is usually implemented by different organization and deployed into different nodes (distributed). Describing formally the observable behavior of these services as they interact with each other was the focus of this thesis; we have looked into few formal languages (BPEL, CDL, and WSDL) which address related problems. Our goal was to propose a language which can capture any collaboration among dynamic, distributed web services. We defined a WS-CDL based language which captures the elements of collaboration such as interaction, exchange state, time, constraints, dependence, parallel, sequence, choice and others. We have also demonstrated how the elements of the collaborations among the service components can be mapped into the constructs of this language. In the end, we showed how a test engine generates test cases out of the WS-CDL based specification. One of the major challenges of SOA based systems is its lack of formal testing methods, the service components which make up the applications are distributed, possibly developed by different groups or organizations and because of that the interactions among these components are not well defined especially stateful interactions. This thesis presents a solution by proposing a language which captures the interactions among service components, and this language is XML based so it can be easily processed by test engine to generate a test case and run it on the fly.

This work sets a foundation for more research of automating SOA based applications testing, The test cases generated from this specification will reduce drastically the time it takes to generate test cases and test the SOA based applications and will improve the return of investment(ROI). This specification addresses all kind of collaborations, especially dynamically

invoked and freely interacting web services components. In this kind of environment the rules of the game is not known ahead of time and a language which defines formally for these collaborations is needed. This specification has value in state-full e-commerce system, especially in the B2B world, where services are invoked dynamically according to the result of the previous invocation. Applications can not implement the rules of game since it can not anticipate what sequence of web services invocations are required to complete a business requirement. These rules are captured in WS-CDL+ language and it can be used to generate test cases or it can be fed into a controller agent which observes rules of the game for conformance.

6.1 Future Work

The work on this thesis is based on the work of previous research in the area of Service collaboration and automated testing, W3C, The World Wide Web Consortium is one of the leading authorities in this area and they had already proposed WS-CDL and WS-BPEL which both contribute to thesis, two area that I see as a potential future research is extending this WS-CDL+ with annotations so that tools can generate a complete test clients, another potential research is design and implementation of an agent which observes the rules of the game and logs if violated, this agent will require it is own WS-CDL+ and protocol to capture request and response of each participant.

References:

[1] Lars Frantzen, Jan Tretmans, Towards Model-Based Testing of Web Services. Institute for Computing and Information Sciences Radboud University Nijmegen The Netherlands

[2] L. Frantzen, J. Tretmans, and T.A.C. Willemse. Test generation based on symbolic specifications. In J. Grabowski and B. Nielsen, editors, *FATES 2004*, number 3395 in LNCS, pages 1–15. Springer-Verlag, 2005.

[3] J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software—Concepts and Tools*, 17(3):103–120, 1996.

[4] A. Belinfante, J. Feenstra, R.G. de Vries, J. Tretmans, N. Goga, L. Feijs, S. Mauw, and L. Heerink. Formal test automation: A simple experiment. In G. Csopaki, S. Dibuz, and K. Tarnay, editors, 12th *Int. Workshop on Testing of Communicating Systems*,pages 179–196. Kluwer Academic Publishers, 1999.

[5] C. Jard and T. Jéron. TGV: theory, principles and algorithms. In *IDPT ’02*, Pasadena, California, USA, June 2002. Society for Design and Process Science.

[6] Eddie Chee-Hung Mak, Automated Testing Using XML AND DEVS. A Thesis Submitted to the Faculty of the Department of Electrical and Computer Engineering, University of Arizona, 2006.

[7] Zeigler, B., Hammonds, P., Fulton D., Nunn K., and Mak, E., “Simulation-based Testing of Emerging Defense Information System,” In progress.

[8] Nickolas Kavantzas, David Burdett, Gregory Ritzinger, Tony Fletcher, Yves Lafon, W3C Working Draft 19 June 2006.

[9] Manes, A.T. 2005, VantagePoint 2005-2006 SOA Reality Check Version: 1.0, Burton Group Publication, Jun 29

[10] Hull R., and Su, J., Tools for Composite Web Services: A Short Overview, ACM SIGMOD Record, Vol 34, No. 2, June, 2005.

[11] Foster, I., Frey, J., Graham, S., Tuecke, S., Czajkowski, K., Ferguson, D., Leymann, F., Nally, M., Sedukhin, I., Snelling, D., Storey, T. and Weerawaranna, S. Modeling Stateful Resources with Web Services. Globus Alliance, 2004.

[12] Zhu, H., Cooperative Model Approach to Quality Assurance and Testing Web Software, compsac, pp. 110-113, 28th Annual International Computer Software and Applications Conference (COMPSAC'04), 2004.

[13] Foster, I., Jennings, N. R., and Kesselman, C., Brain meets brawn: Why grid and agents need each other. Autonomous Agents and Multi-Agent Systems (AAMAS'04), 2004.

[14] Qi, Y., Kung, D., Wong, E., An Agent-Based Testing Approach for Web Applications, compsac, pp. 45-50, 29th Annual International Computer Software and Applications Conference (COMPSAC'05) Volume 2, 2005

[15] Huo, Q., Zhu, H., Greenwood, S., A Multi-Model Software Environment for Testing Web-based Applications, compsac, p. 210, 27th Annual International Computer Software and Applications Conference, 2003

[16] Grundy, J.C., Ding, G., and Hosking, J.G., Deployed Software Component Testing using Dynamic Validation Agents, Journal of Systems and Software: Special Issue on Automated Component-based Software Engineering, vol.74, no. 1, January 2005, Elsevier, pp. 5-14

[17] Willmott, S., Thompson, S., Bonnefoy, D., Charlton, P., Constantinescu, I., Dale, J., Zhang, T., Model Based Dynamic Service Synthesis in Large-Scale Open Environments: Experiences from the Agentcities Testbed, pp. 1318-1319, AAMAS'04, 2004

[18]. Boston, MA, USA; 12 April 2007 -- OASIS, The International Standards Consortium

[19] K. Gottschalk, S. Graham, H. Kreger, J. Snell, IBM Systems Journal, Vol 41, NO 2, 2002

[20] Web Services Business Process Execution Language Version 2.0 (draft). Technical report, OASIS, 2006.

[21] Lars-Ake Fredlund, Implementing WS-CDL, LSIIS, Facultad de Inform´atica, Universidad Polit´ecnica de Madrid.

[22] I. B. Arpinar, R. Zhang, B. Aleman and A. Maduko. Ontology-Driven Web Services Composition. IEEE E-Commerce Technology,July 6-9, San Diego, CA.