This is to certify that the
dissertation entitled

A SERVICE-ORIENTED APPROACH FOR
COLLABORATIVE PROCESS MANAGEMENT

presented by

Woongsup Kim

has been accepted towards fulfillment
of the requirements for the

Ph.D. degree in Computer Science

Wayne N Pyle
Major Professor's Signature

12/14/06
Date

MSU is an Affirmative Action/Equal Opportunity Institution

A SERVICE-ORIENTED APPROACH FOR COLLABORATIVE PROCESS MANAGEMENT

By

Woongsup Kim

A DISSERTATION

Submitted to
Michigan State University
In partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science and Engineering

2006

ABSTRACT

A SERVICE-ORIENTED APPROACH FOR COLLABORATIVE PROCESS MANAGEMENT

By

Woongsup Kim

Service-oriented computing provides organizations with methodologies and technologies that enable them to link multiple software systems using platform independent interfaces and contracts thereby creating a so-called Virtual Organization (VO). Unfortunately, as of today, there are challenges that can limit the implementation of a true service-oriented environment. One of the most noticeable challenges is the issue of "trust". During service-oriented process integration, participants may face uncertainties due to the nature of Web Service technologies. These uncertainties arise from the limitations of current Web Service support that is required for collaboration between participants in the VO. As a result, resolution of such limitations of Web Services is both very necessary and highly desirable.

In this thesis, we propose a new framework that we call the Web Service Collaborative Process Coordinator (WSCPC), which provides support for a reliable service-oriented collaborative environment. Using our proposed approach, partners can share their workspaces through Web Service semantics in heterogeneous processes. Behaviors in the VO can be predicted, monitored, and analyzed so as to determine if a goal can be successfully achieved. Such behaviors can also be enacted on-the-fly by partners based on the needs.

To this end, we first propose a trustworthiness model that can be used to predict performance and reliability of service behavior. The trustworthiness model employs a probabilistic method called probabilistic Latent Sematic Analysis (pLSA) that eliminates noise in the pool of service ratings. Using this methodology, service consumers can expect a particular degree of satisfaction and estimate percentage measure that predicts to what extent which they will be disappointed. Our trustworthiness model can also be integrated into system analysis methodologies such as stochastic Petri nets, and hence support run-time quantitative predictions.

Moreover, we propose several service models including a service definition model, a service registry model, and a service interaction model. These service models are described using the Ontology Web Language (OWL), an XML knowledge representation based on Description Logic (DL). As a result, any peer's capability and behavior can be logically inferred and enacted in a heterogeneous, collaborative software environment. Under these service models, collaborators in a VO can share communicational behaviors and enact partners' behavior in a flexible and platform neutral way.

To my wife Hyo-jeung,

daughter Hyunwoo,

father, and mother.

ACKNOWLEDGEMENT

I wish to thank the countless people—family, friends, colleagues, advisors, and instructors—who have made my years in East Lansing the most precious of my life.

First and foremost I would like to thank my current and former advisors, Dr. Wayne Dyksen and Dr. Moon Jung Chung, whose guidance and tireless support have proven invaluable during my PhD studies throughout graduate school. I am very fortunate to have had the opportunity to work with them and to be the beneficiary of their vast knowledge and friendliness.

I am of course grateful to my thesis committee, Drs. Abdol-Hossein Esfahanian, Charles Owen, Brian Pentland, and Wayne Dyksen.

Others with whom I have had the pleasure of collaborating include Yuan Zhang, Hong-Suk Jung, Ravi Gopalan, Yonggang Qin, Un-Sang Park, Ming Wu, and the members of the Korean Buddhist Association at Michigan State University. Yuan, Ravi, Yonggang, and Hong-Suk worked with me in developing earlier versions of the WSCPC framework. Discussions with Ming and Un-Sang helped me to build the WSCPC trustworthiness model. Members of the Buddhist Association reinvigorated me whenever I felt tired. I am also indebted to visiting professors Sang-Cheol Kim, Youn-Mo Chung, and Young Keun Choi who gave me advice and motivation to further my research.

Finally, I would like to thank my family including my wife Hyo-Jeung and my daughter Hyunwoo as well as my mother and father back in Seoul, Korea. They have provided an

enormous amount of support and encouraged me to stick with my PhD studies as long as I could. I dedicate this thesis to them.

My work had been supported in part by the National Science Foundation (NSF) Grant Number DMI-0313174.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

Information technology is transforming every aspect of our world. In particular, businesses are challenged to compete in rapidly changing environments. A significant aspect of that challenge is the management of the change itself. One approach is that of the agile enterprise.

A Virtual Organization (VO) is a temporary alliance of enterprises that come together to share resources in order to respond to rapidly changing technologies and increasingly complex market competition [1-4]. Organizations within a VO focus on their core strengths, while, turning to partners for supplemental areas of expertise thereby enabling them to read market changes and to react to them quickly.

VO's must share processes in order to develop and deliver new products on demand. To this end, VO's are generally managed by software systems running on widely distributed, heterogeneous computing environments. As a result, VO's require a new paradigm that allows applications to access and share resources across distributed networks. Besides making them more competitive, companies are realizing that VO's can save development and maintenance costs by outsourcing these components to various service providers. This trend has increased the importance of developing distributed applications and the integration of heterogeneous systems.

Service-oriented computing was introduced to deliver methods and technologies to help organizations link their software in order to support business partnerships [5-10]. A Service-Oriented Architecture (SOA) is a component model that interrelates an

application's different functional units—the services—through well-defined interfaces and contracts between these services. The interface is defined in a neutral manner; that is, ideally independent of the hardware platform, operating system, and programming language in which the service is implemented. This allows services that are built on a variety of such systems to interact with each other in a uniform and universal manner.

In a service-oriented architecture, the concept of a service that is not strongly tied to a particular implementation is known as loose coupling. Loosely-coupled systems can provide business applications with agility based upon the needs of the business to adapt to its changing environment such as changing policies, business strengths, business focus, partnerships, industry standing, and other business-related factors that influence the very nature of the business. Such systems are agile because internal implementations represented as services are used to build up the whole application. Tight-coupling, on the other hand, implies that the interfaces between the different components of an application are tightly interrelated in function and form, thus making them brittle when any change is required to parts of or to the whole application.

As expected, there are challenges that can inhibit the implementation of collaboration software based on a service-oriented paradigm. One of the most noticeable challenges is the issue of "trustworthiness". Trustworthiness is an integrative concept, which encompasses the following attributes: availability (readiness of a service), reliability (correctness of a service), safety (absence of catastrophic consequence from using a service), integrity (absence of improper system state alterations), and maintainability (ability to undergo repairs and modification).

During service-oriented collaborative processes, participants may face uncertainties in terms of trustworthiness such as service availability, service reliability, service safety, or service integrity. These uncertainties come from two major limitations of current web service technology. In web services, description logics are used to describe functions and capabilities of services. Description logics are well-suited for such tasks and proved successful in a wide range of applications [11]. However, they are not well-suited for representing quantitative knowledge in terms of performance and reliability. Moreover, malicious attacks residing in a VO cannot be detected solely through description logics. In addition, communications within web services rely on RPC-type messaging via the Simple Object Access Protocol (SOAP); however, SOAP is not sufficient to implement the complex communication required for collaboration between participants in a VO.

Uncertainties in a service oriented environment can be classified in two categories: pre-purchase uncertainties and fulfillment uncertainties. Pre-purchase uncertainties occur when service consumers cannot decide whether or not to purchase a certain service. Even semantically well-written descriptions of a service may not be good enough for users to be convinced of the service usability. Fulfillment uncertainties involve situations in which a user invokes a service only to find out that the service cannot provide the functionality claimed in the descriptions. Service consumers may have concerns about defective or low quality services.

In response to pre-purchase uncertainty, recommender systems and third party certification methods have been proposed [12-14]. Recommender systems are methodologies that utilize evaluations records from past transactions in the community. When a transaction finishes, users in a business community evaluate the results and the

evaluation is later made available to other users seeking to build a VO. Users within recommender systems are evaluated and forced to offer reliable services in order to be selected as partners in later transactions. Third-party certification is a traditional business method. For example, some traditional business transactions require a Letter of Credit (L/C) before the transaction begins. The L/C guarantees secure payment and proves its availability and usability in the business transaction. Even though a L/C is issued by only certified banks, third-party certification of other services rendered in e-business can be provided by any organization within the community.

Although useful, these approaches only provide a list of recommended services. They are not sufficient to provide parametric information in terms of performance or reliability. It is thus hard to build the reliability models or the performance models, which are required to predict and improve trustworthiness of the whole system.

To manage fulfillment uncertainty, agreement-based systems and assertion-based monitoring have been designed [15, 16]. Agreement-based systems utilize a contract created by both parties. The contract describes the required service quality metrics for both parties. One specific example is the Service-Level Agreement (SLA) from IBM and HP [17]. To check contract fulfillment, companies manage additional modules that can send or receive reports regarding attributes described in the contract. Another approach is, assertion-based monitoring, which makes use of assertions embedded in process logic. In this approach, assertions are determined based on client's requirements and then inserted into specific locations before transactions begin [18]. When execution reaches each embedded assertion point, a service fulfillment indication is reported to clients, and analyzed through formal modeling languages like SPIN [19].

Even though these approaches are beneficial, intrinsic anonymity (i.e., the customer's lack of familiarity or relationship with services and the people involved) still makes users hesitant to join a service-oriented collaborative environment. Complex interaction is a way of testing new boundaries and seeing how far one can go when no one knows about such boundaries. Instead of judging services through categories provided by some portal sites, interactive communication allows a service consumer to choose which services to use. Along with communications, service consumers must be able to predict performance and reliability of services. Therefore, systems enabling intensive interactions and behavior predictions between service consumers and providers are highly necessary in a service-oriented environment.

In this thesis, we present a framework Web Service Collaborative Process Coordinator (WSCPC) for supporting reliable, service-oriented collaborative environments. First, we provide a trust model for predicting behaviors of service providers. Our trust model provides probabilities that measure the extent to which a service is reliable. To estimate such probability, we employ ratings by previous service users in order to expect how much the user will be satisfied after the usage of service. The probabilities are used to build the expectation ratings and the degree of service satisfaction. In our model, failure or reliability is defined in terms of the degree of service satisfaction.

In order to support a complex level of interactions between peers, we propose a frame-based approach, rather than the more conventional Finite-State Machine (FSM) approach, which relies on pre-defined sequences and hence has limited flexibility for reconfiguration or adaptation. In a frame-based approach, messages have slots to be filled, and responses fill the appropriate slots. Slots are generated, analyzed, and filled due to

the semantic information provided from both service peers. Peers can create a message with slots filled locally or by remote systems, in case they need to continue interaction. Generating and filling slots requires semantic information from peers. The Process Grammar [20, 21] proposed by Chung together with the Web Service Collaborative Process Coordinator (WSCPC) developed herein work well for this approach as they have hierarchical structure and alternative selection rules.

The remainder of this thesis is organized as follows. In Chapter 2, we present the current paradigm of service oriented computing and the corresponding research issues and solutions. In Chapter 3 we develop trustworthiness issues in terms of a service oriented environment. In particular, we propose a methodology to be used for behaviors prediction and run-time analysis in services. In Chapter 4 we propose service models to be used for service behavior enactment and sharing. The implementation is discussed in Chapter 5. Finally, we design and implement up an experiment to evaluate the proposed models. We do this in the setting of a die casting manufacturing domain and assign artificial data for performance parameters. The results demonstrate that our service model is effective in terms of trustworthy collaboration.

CHAPTER 2 BACKGROUND

What follows in this chapter is a survey and summary of previous work that serves as a springboard for our research regarding trustworthy service oriented collaboration.

2.1 Virtual Organization

A Virtual Organization (VO) is defined as a coordinated network of autonomous production units (factories, firms, etc.) with the goal of producing a product while exchanging all needed information via a computer network [22, 23]. In a VO, organizations can virtualize--convert a tangible, physical aspect of a business to function in a virtual environment—various functions and parts such as processes, operations, groups, or individuals. This enables organizations to focus on their core strengths while turning to partnerships for supplemental areas of expertise. Organizations participating in VO's are generally managed by software systems running in heterogeneous computing environments. As such, they must share processes to deliver new products on demand.

Several studies [24, 25 , 26 , 27 , 28] identify a set of dimensions regarding VO-related implementation issues. These dimensions are described as follows.

1) Tightness and Duration: Two partners are tightly coupled if they are dependent on each other. Loosely coupled partners exchange business information on demand.
2) Similarity: Applications use different data structures and standards. There may also be structural heterogeneity at the processing layer, such as use of API and document exchange protocols. Organizations may use different strategies to conduct process execution.

3) Autonomy: Partner systems may be autonomous in their design, communication, and executions. More autonomous collaboration allows partners to have more local control over implementation and operation of services resulting in flexibility to change their processes without affecting each other.
4) External Manageability: In order to be effectively monitored by external partners, an application must be defined in a way that facilitates the supervision and control of its execution, measurement of its performance, and prediction of its status and availability.
5) Adaptability: Applications operate in a highly dynamic environment where new services can be initiated in the middle of process execution, existing services may be removed, and the contents of services may be updated. A VO must be able to respond to such dynamic changes.
6) Security: Security is a major concern for inter-organization applications. Security measures must be in place to boost confidence between partners such that their transactions are safely handled.

Research in VO's focuses mainly on the aspects of cooperation and integration within heterogeneous software environments. WISE [29, 30] is a paradigm that aims to provide support for cross organizational business processes. In WISE, distribution is modeled in each virtual process definition and defined as a building block that can be posted as an entry of a catalog. WISE also creates an awareness model to be used for load balancing, routing, quality of service, and analysis purposes. CrossFlow [31, 32] is intended to provide high-level support for dynamic workflows in VO environments. The main contribution of CrossFlow is in the use of the concept of a contract as a basic tool for

cooperation. A contract is made by filling out a template. Based on the contract, a service enactment infrastructure is established. Business partners specify their interactions through the resulting contract. The dynamically created modules can be removed after contract completion. Mentor-Lite [33, 34] addresses the problem of distributing the execution of workflow by partitioning the overall workflow specification into several sub-workflows, each encompassing all the activities that are to be executed by a given entity. The basic building block of Mentor-Lite is an interpreter for workflow based on state charts. SELF-SERV [35, 36] proposes a process based language for composing web services based on state charts. SELF-SERV also defines a peer-to-peer service execution in which the responsibility of coordinating the execution of a composite service is distributed across several peer components. Collaboration Management Infrastructure (CMI) [37, 38] extends the traditional workflow concepts with advanced concepts such as placeholder, which enables the dynamic establishment of trading relationships. A placeholder activity is replaced at runtime with a concrete activity having the same input and output. A selection policy is also specified to indicate the activity that should be executed. eFlow [39] is a platform that supports the specification, enactment, and management of composite services. A composite service is described as a process schema and modeled by a graph that defines the order of execution among the nodes in the process. There are service, decision, and event nodes: service nodes execute selection of a reference to a specific service; decision nodes specify the alternatives and rules controlling the execution flow; and event nodes enable service processes to send and receive several types of events. To support heterogeneity of services, eFlow provides adapters for services that support various interactions. WebBase of Internet-accessible

Services (WebBIS) [40] proposes a declarative language for composing Web Services. WebBIS addresses the issue of adaptability by defining a mechanism to propagate changes. All changes performed to a service are propagated to other services that rely on it to ensure global consistency. To this end, WebBIS defines a meta-service called change notifiers, which attach to each service and maintain information about the availability of the related service.

2.2 Trustworthiness Management

The lifecycle of a VO—formation, communication, operation—is strongly connected with the Internet. During communication in a VO, a huge amount of extremely valuable technical data and information (development, product, and process data in addition to business information) moves through the network, making security a vital concern. Enhanced security can contribute to the growth of the number of realized VO solutions. The basic network security aspects [41] are a security issue in a VO environment. The communication between software and users should be protected. Software agents, users, and even nonparticipating entities can potentially eavesdrop or tamper with the communication or impersonate participating persons and organizations. Thus, the typical cryptographic security services—entity authentication, data authentication, and data confidentiality—should be provided. Standard mechanisms are available for this purpose such as SSL/TLS [42] at the transport layer or IPsec [43] at the network layer. In addition, malicious software agents should be prevented from stealing a host's private keys.

2.2.1 Considerations

Since VO's are run primarily on open environments, it is to be expected that there will be participants with malicious intentions. Hence, participants should be restricted to execute in a sandbox environment in which they have limited privileges and in which they are protected from one another [44]. Authentication allows a participant to identify partners. In addition to authentication, partners can carry proof that enables a service provider to determine whether it is safe to work with a particular partner.

Although the problem of identifying partners seems more and less easy to solve, protecting participants from malicious attacks is a very difficult task. Malicious attacks can include masquerading as a partner, denial of service, eavesdropping on interactions, or returning wrong results of service calls.

2.2.2 Potential Application of Concepts from Mobile Agent Systems

Mobile software agents are agents that can travel from one computer to another computer. They are sent by end-users and visit a series of hosts. The mobile agents are executed locally on these hosts to perform their tasks, and will return to the end users to report their results. The autonomous and coordinative nature of mobile agents resembles the dynamics of a VO in that its membership is created and coordinated on demand.

Therefore, it would seem that an approach similar to that used by mobile agents might offer potential solutions to the issues of trust within a VO. The purpose of this section is to present open issues and ideas of mobile agents and secure electronic transactions on untrusted hosts in the context of a VO environment since they are relevant to the specific

problem of trusted computing in a VO environment. These issues and ideas are described as follows.

1) Insecure Networks : The security issues associated with mobile agents are similar to the security issues of the underlying network itself. The communication between agents and users as well as the communication between the agents themselves should be protected. Thus, the typical cryptographic security services such as data authentication, entity authentication, and data confidentiality should be provided.

2) Avoiding the Problem : The problem of malicious hosts can be avoided by simply not working with untrusted hosts. *Reputation* can be used in a VO as a security mechanism. Rasmusson and Johnson [45] describe this as a soft security approach since each participant itself is responsible for the security, as opposed to hard security in which security is forced by external third party. In reputation systems, participants only work with a limited set of partners that they trust.

3) Minimizing the Risk : The basic idea of this scheme is to limit capabilities to that of performing transactions. Romao et al. [46] propose proxy certificates, which restrict access to private digital signature keys. Yi et al. [47] propose a one-time key pair, where a private key is associated with each message and used exactly once. Bellare and Miner [48] and Krawczyk [49] propose a scheme in which the public key is fixed but the private key is updated regularly. All of these schemes are based on the fact that services are executed with a limited amount of time.

4) Execution Integrity : Execution integrity is about ensuring the correctness of execution. State appraisal is proposed to detect tampering to some extent. With state

appraisal, invariants can be expressed that each state must satisfy. Cryptographic traces proposed by Vigna [50] are detailed digitally signed logs of the operations performed by an agent during its lifetime. To reduce overhead, only the hashed log is sent and verified.

2.3 Service-Oriented Computing

Service-oriented computing delivers methodologies and technologies to organizations so as to link multiple software systems using platform independent interfaces and contracts in order to create a Virtual Organization (VO). Service-oriented computing is commonly implemented through the use of Web Services and software components that provide self-contained functionality via internet-based interoperable interfaces. These services are based on a common description of their characteristics, including how they are dynamically discovered, selected, and accessed. The terms "orchestration" and "choreography" have been widely used to describe business process integration comprised of Web Service-based software systems [51, 52].

2.3.1 What Are Web Services?

The World Wide Web Consortium (W3C) defines a Web Service as "a software system designed to support interoperable machine-to-machine interaction over a network" [53]. Web Services are Web-based applications that utilize open XML-based standards and transport protocols to exchange data. Web Services were first introduced in order to enable consistent resource access across multiple heterogeneous platforms [9, 54], and hence to support simple inter-organizational collaboration in process management [55-57].

The purpose of a Web Service is to provide some functionality, either pre-existing or newly defined, on behalf of its owner (e.g. a business or an individual). A Web Service takes such functionality and makes it available to any system.

In order to promote interoperability and extensibility among these applications, as well as to allow them to be combined in order to perform more complex operations, a standard reference architecture is needed; Web Services technology provides such an architecture. For example, a user might develop a purchasing application that can perform all of the following functions: provide price information for various vendors, allow the user to select a vendor and submit the order, and track the shipment until goods are received. The vendor application, in addition to exposing its services on the Web, may in turn use Web Services to check the customer's credit, charge the customer's account, and set up the shipment with a shipping company.

Web Services provide an important instantiation of service-oriented computing and include infrastructure for specifying service properties (WSDL) [58], interaction between services (via SOAP) [59], service invocation through a variety of protocols and messaging systems (Web Service invocation frameworks), support for a services registry (via UDDI) [60], and scheduling through a Web Services Choreography Language [61-63]. Web Services also play an important role in the Semantic Web [64]. By providing semantic metadata to enable machine processing of information, Web Services can provide a useful mechanism to enable automatic interaction between software, thereby also providing a useful environment in which agent systems can interact.

Web Service technologies represent the evolution of distributed software component technologies like RPC, DCOM, CORBA, Java RMI, and even Web applications like Google.com [65]. As application developers struggle with interoperability between various technologies, they turn to the evolving Web as a potential solution for these challenges.

Web Services have two distinct advantages for heterogeneous distributed system integration. First, Web Services enable dynamic discovery and composition of heterogeneous distributed services by providing mechanisms for registering and discovering platform-neutral interface definitions and endpoint implementation descriptions. Second, the widespread adoption of Web Services mechanisms implies that a framework based on Web Services can exploit numerous tools and services, such as workflow systems that sit on top of the Web Service Description Language (WSDL) [58] and hosting environments for Web Services (e.g. Microsoft .NET and Apache Axis).

2.3.2 Web Service Environments

The basic Web Services architecture defines an interaction between software agents as an exchange of messages between service consumers and service providers [53]. Consumers are software agents that request the execution of a service. Providers are software agents that provide a service. Agents can be both service requesters and providers. Providers are responsible for publishing a description of the service(s) they provide. Requesters must be able to find the description(s) of the services.

While a Web Service is an interface described by a service description, its implementation is the service. A service is a software module deployed on network

accessible platforms provided by the service provider. It exists to be invoked by or to interact with a service requestor. The service description contains the details of the interface and implementation of the service. This includes its data types, operations, binding information, and network location. It could also include categorization and other metadata to facilitate discovery and utilization by requestors. The complete description may be realized as a set of XML description documents. The service description may be published to a requestor directly or to a discovery agency.

Software agents in the basic Web Services architecture can take on one or all of the following roles:

1) Service Requester, which requests the execution of a Web Service;

2) Service Consumer, which processes a Web Service request; and

3) Discovery Agent, which publishes and makes discoverable Web Service descriptions..

In order for an application to take advantage of Web Services, three actions must occur: publication of service descriptions, finding and retrieval of service descriptions, and binding or invoking of services based on the service description.

1) Publish: In order to be accessible, a service needs to publish its description such that the requestor can subsequently find it. Where it is published can vary depending upon the requirements of the application.

2) Find: In order to locate other services, a service requestor retrieves a service description directly or queries the registry for the type of service required. The find operation may be involved in two different phases for the service requestor: at design

time in order to retrieve the service's interface description for program development, and at runtime in order to retrieve the service's binding and location description for invocation.

3) Interact: Eventually, a service needs to be invoked. During the interact operation, the service requestor invokes or initiates an interaction with the service at runtime using the binding details in the service description to locate, contact, and invoke the service.

The essential part of Web Services is the interact relationship between a service provider and service requestor. The interactions in service-oriented computing involve publish, find, and bind operations. These roles and operations act upon the Web Service artifacts, the Web Service software module and its description. In a typical scenario, a service provider hosts a network accessible software module as an implementation of a Web Service. The service provider defines a service description for the Web Service and publishes it to a requestor or service discovery agency. The service consumer uses a find operation to retrieve the service description locally or from a registry or repository, and then uses that service description to bind with the service provider and invoke or interact with the Web Service implementation. Together these components become an interface to a vast world of data and query services that provide data about Web Services as well as many other things.

2.3.3 Structural Layers in Web Services

Web Services interact by passing XML data, with types specified using XML schema. Simple Object Access Protocol (SOAP) is commonly used as the communication protocol and WSDL is often used to specify the I/O structures . The underlying structure

for the Web Services paradigm will most likely be guided by already established standards and practices. Some of the current standards are illustrated by the layered structure shown below. All of these can be defined before binding Web Services to each other. Behavioral descriptions of Web Services can be defined using higher level standards such as Business Process Execution Language for Web Services (BPEL) [63], Web Service Choreography Interface (WSCI) [66], Business Process Management Initiative (BPML) [67], or DARPA Agent Markup Language of Services (DAML-S) [68]. Figure 1 illustrates Web Services structural layers.

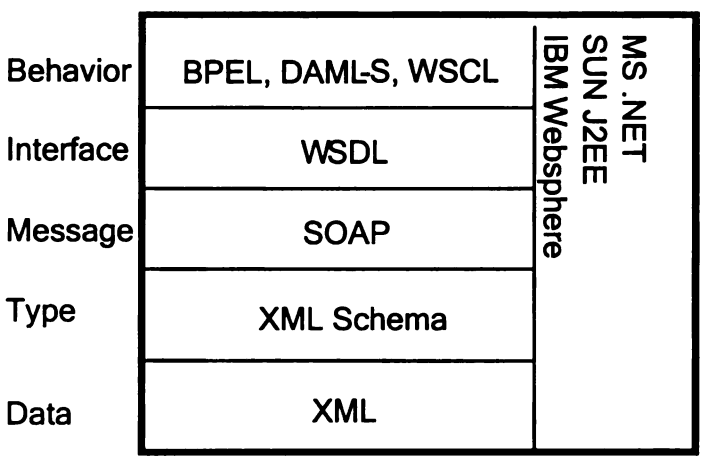


Figure 1 Structural Layer of Web Services

XML and XML Schema

Extensible Markup Language (XML) is an extensible, portable, and structured text format [69]. XML plays an important role in the exchange of a wide variety of data on the Web and elsewhere. XML schemas provides a means of creating a set of rules that can be used to govern the validity of XML documents. Schemas provide a means of

defining the structure, content, and semantics of XML documents that can be shared between different types of computers and documents [70, 71].

One important concern of Web Services is how to transmit data in an interoperable manner. XML and XML schema provide a means of describing and using Web Services. Messages and interfaces in Web Services are defined with XML schema and written with XML such that platform neutral interoperability is possible.

SOAP

Simple Object Access Protocol (SOAP) is a communications protocol for Web Services [59]. There are elements of the SOAP specification that describe how to represent program data as XML and how to use SOAP to do Remote Procedure Calls (RPCs). A SOAP message contains a callable function and the parameters to pass to that function. SOAP also supports document style applications where the SOAP message is just a wrapper around an XML document. Document-style SOAP applications are very flexible and many new XML Web Services take advantage of this flexibility to build services that would be difficult to implement using RPCs. What follows in Figure 2 below is an example of a SOAP message [72]. The example illustrates a request of a method *GetStockPrice* with a parameter *StockName*.

```
POST /InStock HTTP/1.1
Host: www.stock.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn
```

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-
encoding">
::
::
  <soap:Body xmlns:m="http://www.stock.org/stock">
    <m:GetStockPrice>
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
::
::
</soap:Envelope>
```

Figure 2 A SOAP Message

The most distinct feature of SOAP is that it can be implemented on many different hardware and software platforms. This means that SOAP can be used to link disparate systems both inside and outside of an organization. Many attempts have been made in the past to achieve a common communication protocol that could be used for system integration but none of them has had as widespread adoption as SOAP. Since SOAP can use existing XML parsers and HTTP libraries to do most of the hard work, a SOAP implementation can be completed relatively easily. This is why SOAP is becoming so widespread. The popularity of HTTP and the simplicity of SOAP make them an ideal basis for implementing Web Services that can be called from almost any environment.

WSDL

WSDL stands for Web Services Description Language [58]. A WSDL file is an XML document that defines the format of a set of SOAP messages and how those messages should be exchanged. In other words, WSDL is to SOAP what Interface Definition Language (IDL) is to Common Object Request Broker Architecture (CORBA). Since WSDL is based on XML, it is readable and editable.

Figure 3 is a subset of a WSDL example [73]. The *portType* describes a single operation *GetStockPrice*, which uses the message types for input and output.binding indicating that the communication is through SOAP. Moreoever, *portType* associates the binding with the URI http://sample/stockprice where the running service can be accessed.

```
<portType name="StockPricePortType">
    <operation name="GetStockPrice">
      <input message="tns:Name"/>
      <output message="tns:Output"/>
    </operation>
  </portType>
  <binding name="StockPriceSoapBinding"
type="tns:StockPricePortType">
    <soap:binding style="document"
transport="http://sample"/>
    <operation name="GetStockPrice">
      <soap:operation soapAction="http://sample "/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
```

```
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="StockPriceService">
    <documentation>My service</documentation>
    <port name="StockPricePort"
binding="tns:StockPriceSoapBinding">
      <soap:address location="http://sample/stockprice"/>
    </port>
  </service>
</definitions>
```

Figure 3 A WSDL Example

The fact that the format a WSDL file uses to describe messages is based on the XML schema standard implies that it is both programming language neutral and standards-based. This makes WSDL suitable for describing XML Web Service interfaces, which are accessible from a wide variety of platforms and programming languages. In addition to describing message contents, WSDL defines where the service is available and which communication protocol should be used to talk to the service. Hence, the WSDL file defines everything required to write a program to interact with a Web Service.

2.4 Process Management Based on Web Services

Web Services provide functional building blocks that can be discovered, selected, and accessed. On the other hand, application services are composed typically of functional parts coming from multiple service providers and hence include execution logic resembling multilateral interaction procedures among service providers.

To this end, several Web Service based *service composition* frameworks have been proposed. The field of Web Service composition provides means to assemble basic Web Services into composite ones that constitute a considerable step towards application services. Those frameworks are designed to integrate heterogeneous distributed systems easily through Web Services, and hence they enable multi-organizational process execution by allowing service-based collaboration of distributed organizations. Some of the distinctive works are BPEL4WS, BPML, OWL-S, GSFL, and GridAnt, [21, 61, 74-76]. These frameworks can be categorized into three basic approaches: using meta language, using Web Services on grids, and using a semantics approach.

2.4.1 Using Meta Language

This approach provides an open, standards-based methodology for connecting Web Services together to create higher-level business processes. Distinct standards in this category are Business Process Execution Language for Web Service (BPEL4WS), Business Process Modeling Language (BPML), and WS-Choreography Definition Language (WS-CDL). The business process should be designed to be dynamic, flexible, and adaptable in order to meet the changing needs of a business; these standards are designed to support such dynamic business demands.

BPML is a meta-language for modeling business processes. BPML provides an abstract execution model for collaborative and transactional business processes based on the concept of a transactional finite-state machine. BPEL4WS provides an XML-based grammar for describing the control logic required to coordinate Web Services.

BPEL4WS

BPEL4WS is a specification that models the behavior of Web Services in a business process interaction [7]. The specification provides an XML-based grammar for describing the control logic required to coordinate Web Services in a process flow. This grammar can then be interpreted and executed by an orchestration engine, which is controlled by one of the participating parties. The engine coordinates the various activities in the process and compensates the system when errors occur. BPEL4WS is essentially a layer on top of WSDL, with WSDL defining the specific operations allowed and BPEL4WS defining how the operations can be sequenced.

BPEL4WS provides support for both executable and abstract business processes. An executable process models the behavior of participants in a specific business interaction. Abstract processes, modeled as business protocols in BPEL4WS, specify the public message exchanges between the parties. Business protocols are not executable and do not convey the internal details of a process flow. Essentially, executable processes provide the orchestration support described earlier while the business protocols focus more on the choreography of the services.

The specification includes support for both basic and structured activities. Basic activity is defined as a component that interacts with something external to the process. For example, basic activities would handle receiving or replying to message requests as well as invoking external services. By contrast, structured activities manage the overall process flow, specifying which activities should be run and in what order. Structured activities may specify that certain activities should be run sequentially or in parallel.

These activities also provide support for conditional looping and dynamic branching. One can think of structured activities as the underlying programming logic for BPEL4WS.

In BPEL4WS, a set of activities can be grouped into a single transaction through the *scope* tag. The scope tag signifies that the steps enclosed in the scope should either be all completed or all failed. Within the scope, the developer can then specify compensation handlers that should be invoked if there is an error. BPEL4WS also provides an exception handling mechanism through the use of throw and catch clauses, similar to that in Java.

BPML

Business Process Management Language (BPML) is a meta-language for describing business processes [74]. BPML was initially designed to support business processes that can be executed by a BPMS system. BPML was subsequently expanded to incorporate the WSCI protocol. Withing BPML, WSCI can be used to describe the public interactions and choreographies, while the private implementations can be developed with BPML.

The specification can be compared to BPEL4WS as they provide similar process flow. Basic activities such as sending, receiving, and invoking services are available, along with structured activities such as handling conditional choices, sequential and parallel activities, joins, and looping. BPML also supports the scheduling of tasks at specific times. Other features supported by BPML include persistence, roles, instance correlation, and recursive decomposition. BPML also supports recursive composition, the ability to compose sub-processes into a larger business process.

BPML includes both transactional support and exception handling mechanisms. Both short and long running transactions are supported, with compensation techniques used for more complex transactions. BPML uses a scoping technique similar to BPEL4WS in order to manage the compensation rules. It also has the ability to nest processes and transactions, a feature that BPEL currently does not provide. Finally, a robust exception handling mechanism is available within BPML. Timeout constraints can also be specified for specific activities defined within the process.

WS-CDL

The Web Services Choreography Description Language (WS-CDL) describes peer-to-peer collaborations of Web Services participants by defining their common and complementary observable behavior where ordered message exchanges result in accomplishing a common business goal. In WS-CDL, there is no single controlling process managing the interaction among process participants. WS-CDL provides for sequencing and also offers a flexible systemic view of the process. The W3C positions WS-CDL "as a necessary complement to BPEL, Java, and other programming languages which describe one endpoint on a transaction, rather than the full system."[77]

A WS-CDL document describes interoperable, cross-enterprise collaborations that allow participants to perform business transactions. Their collaboration takes place via a commonly agreed set of constraints, whereby documents are exchanged in a synchronized fashion between the participants. Choreography allows constructing compositions of Web Service participants by explicitly asserting their common

observable behaviors, where synchronized information is exchanged through their shared contact-points.

2.4.2 Web Services on Grids

A grid is defined as a layer of network services that allows access to a distributed collection of data and application resources. The grid services provide a collection of services, which users can access from any location. However, many grid architectures suffer from the heterogeneous nature of resource implementation. This is one reason why approaches based on Web Services are gaining more and more attention. The key idea here is to isolate resource deployment from the instantiation of a component in a distributed environment.

OGSA

Open Grid Services Architecture (OGSA) provides technologies that enable people to share computing power, databases, and other on-line tools securely across corporate, institutional, and geographic boundaries without sacrificing local autonomy [78]. OGSA is based on grid services, which are Web Services with additional features of grid computing [79, 80]. OGSA uses WSDL to achieve self-describing and discovering services. It defines a set of standard interfaces that a grid service may export and enables features such as discovery, service lookup, lifetime management, notification, and credential management. They are built from two main technologies: the Globus Toolkit, which has been widely adopted as a grid technology and Web Services.

The Globus Toolkit is a community-based, open-architecture, open-source set of services and software libraries [81]. The toolkit components are the Grid Resource Allocation and Management (GRAM) protocol, its gatekeeper service, the Meta Directory Service (MDS-2), and the Grid Security Infrastructure (GSI). MDS-2 provides information discovery through soft state registration, data modeling, and a local registry (GRAM reporter). The GSI supports single sign on, delegation, and credential mapping. Single sign-on allows a user to authenticate with any remote service on the user's behalf. Delegation allows for the creation and communication to a remote service of delegated proxy credentials that the remote service can use to act on the user's behalf.

XCAT

XCAT is a Common Component Architecture (CCA) compliant application component framework that is built on top of a Web Service [76, 82]. The CCA specification describes the construction of portable software components that may be re-used in any CCA compliant runtime framework. XCAT is designed to support applications built from components that are distributed over a wide-area grid of resources and distributed services. XCAT is based on Globus for its core security and remote task creation and it uses RMI for its communication. XCAT components are software modules that provide part of a distributed application's functionality in a manner similar to that of a class library in a conventional application.

A running instance of an XCAT component is a Web Service that has two types of ports. One type of port, called a *provides-port*, is a normal Web Service port. A provides-port is a service provided by the component. The provides-ports of an XCAT component can be

described by the Web Service Description Language (WSDL) and hence can be accessed by any Web Service client that understands that port type. The second type of port is called a *uses-port*. These are ports that are "outgoing only" and they are used by one component to invoke the services of another, or, as will be described later, to send a message to any waiting listeners. Within the CCA model, a uses-port on one component may be connected to a provides-port of another component if they have the same port interface type.

GridAnt

Makefiles play a role in automating the complex build process in software engineering. It allows users to structure their code and to build sophisticated reusable libraries with ease. For Java, a similar tool called Ant exists under the Apache Project. Ant provides Java with a sophisticated build tool. Dependencies and parallelism can be easily expressed and new tasks can be included with little effort.

The GridAnt [83, 84] is proposed to reuse the Ant framework to develop a client side workflow system for grids. The philosophy adopted by the GridAnt project was to use the workflow engine available with Ant to develop a grid. The GridAnt framework not only allows clients to map complex client-side workflows, but also to test the functionality of different grid services as a client. GridAnt identifies a set of essential grid tasks that include setup, authenticate, copy, delete, execute, query, and checkpoint. GridAnt also contains a control construct for expressing parallel and sequential tasks. Tasks encapsulated in *<sequential>* are executed in sequential order. Tasks encapsulated in *<parallel>* are executed in parallel.

GSFL

Grid Services Flow Language (GSFL) is an XML based language that allows the specification of workflow descriptions for grid services in the OGSA framework [Gannon03]. GSFL is defined using XML schemas and has the following important features.

1) Service Providers are the list of services taking part in the workflow. The service providers are identified throughout the GSFL document by a unique name, which is specified as part of the definition.

2) An Activity Model lists all the operations that belong to the individual service providers that have some role in the workflow. It contains a list of activities, each of which has a name for identification purposes and a source, which is a reference to an operation in a Web Service.

3) A Composition Model describes the interactions among the individual services. It describes the control and data flow among the various operations of the services. Communication occurs in a peer-to-peer fashion.

4) A Lifecycle Model describes the lifecycle for the various activities and services, which are part of the workflow. It addresses the order in which the services and activities should be executed.

2.4.3 Semantic Approaches

Semantic approaches are designed to integrate independently developed heterogeneous resources using semantic information. Semantics helps to identify heterogeneous resources and hence enables a certain level of automated interoperability [85, 86]. The main purpose of a semantic approach is the automated discovery and search, selection, matching, composition and interoperation, and invocation and execution monitoring. OWL and OWL-S are proposed to represent web semantics.

The markup languages such as Ontology Web Language (OWL) [87] and OWL-S [68, 75] enable the creation of arbitrary domain ontologies that support the explicit description of the Web content. These ontologies make Web Service machine understandable and enable tasks such as locating Web Services, invocating Web Services, inter-operating through semantics, composing services to create new functionality, verifying service properties, and monitoring service execution [85, 88, 89].

OWL-S

OWL-S defines a set of classes and properties specific to the description of services within OWL [64, 68, 75]. The class Service is at the top of the OWL-S ontology. Service properties at this level are very general. From the top level, the taxonomy is structured according to functional and domain differences and market needs. For example, one may imagine a broad subclass B2C-transaction, which encompasses services such as purchasing items from retail Web sites, tracking the status of the transaction, establishing and maintaining accounts with the sites, and so on.

The ontology of services provides two essential types of knowledge about a service. The one is *ServiceProfile*, which describes the capabilities and parameters of the service. The other is *ServiceModel*, which describes the workflow and possible execution paths of the service. *ServiceProfile* provides information about a service that an agent can use to determine if the service meets its rough needs, and whether the service satisfies constraints such as security, locality, affordability, quality-requirements, etc. On the other hand, *ServiceModel* enables an agent to do the following: 1) perform a more in-depth analysis of whether the service meets its needs; 2) compose service descriptions from multiple services to perform a specific task; 3) coordinate the activities of different agents; and 4) monitor the execution of the service. *Grounding* of a service specifies the details of how to work with protocol and message formats, serialization, transport, and addressing. *Grounding* can be thought of as a mapping from an abstract to a concrete specification of those service descriptions. In DAML-S, both *ServiceProfile* and *ServiceModel* are conceived as abstract representations, while the service *Grounding* deals with the concrete level of specification.

2.5 Approaches Toward Trustworthy Service Oriented Computing

2.5.1 Service Discovery

Web Service discovery is the process of locating Web Services so that they can be used to request a service that fulfills some user needs [90, 91]. In order to achieve the automated discovery of reliable Web Services, a requester must explicitly state two things: 1) the information the requester wants to receive as the result of the service provision (called "post-conditions") such as airline schedules; and 2) the state of the

world resulting from it (called "effects") such as the booking of a flight. Goals need to be formalized in order to automate the Web Service discovery process.

Generally, Web Services are described using WSDL and cataloged using the Universal Description Discovery and Integration (UDDI) protocol. UDDI registries contain so-called white pages for each registered provider. References to concrete service implementations are available for binding as template models (called "tModels"). Several methodologies are proposed to use tModels to incorporate post-condition and effects into UDDI [92-94].

Description Logics (DLs) are a good candidate for this purpose. Description logics are knowledge representation languages tailored for expressing knowledge about concepts and concept hierarchies. The basic building blocks are concepts, roles, and individuals. Concepts describe the common properties of a collection of individuals and can be considered as unary predicates, which are interpreted as sets of objects. Roles are interpreted as binary relations between objects. DLs also defines a number of language constructs such as intersection, union, and role quantification that can be used to define new concepts and new roles. The main reasoning tasks are classification and satisfiability, subsumption, and instance checking.

Once a set of DL descriptions has been classified, querying the relation of a new DL description to the set of classified descriptions is efficient [95]. Such descriptions characterize post-conditions and effects although they do not deal with how they relate to the input given to the Web Service. Determining whether a Web Service is relevant to the goal at hand is reduced to computing the subsumption relation between the goal

description in DL and the abstract services accessible through the correspondent Web Services. While DLs are a good formalism for classification tasks, they have some limitations [96, 97]. One of these limitations is their limited ability to express the relation between the input given to a Web Service and the post-conditions and effects resulting from the service determined by that input.

On the other hand, to minimize the intervention of designers and end users in trusted service discovery steps, there are efforts that propose a reputation model in Web Service discovery [12-14]. Reputation model provides for recording past transactions and evaluations for each service so that clients can estimate available services to improve future service discovery decisions.

For extended decision help, Quality of Service (QoS) factors are integrated into UDDI. In this model, service behaviors regarding QoS factors are recorded in the service registry, so that future users can refer to them during service selection. To this end, some approaches extend the current UDDI tModel to accommodate QoS factors [93, 94, 98]. Moreover, there are many proposals to define QoS factors in service-oriented environments. QoS factors can be classified largely into two classes, run-time and quality-driven factors. Runtime QoS factors refer to service execution factors. Such factors include capacity, response time, latency, throughput, and service accuracy. Quality-driven QoS factors refer to the service outcomes such as execution prices, service availability, and service reputation.

2.5.2 Service Behavior Analysis

The WSDL standard focuses on passing messages between Web Services. This leads naturally to behavioral models that use messages as the "actions", and use internal states for individual Web Services as the state that messages transition.

The model used to represent the behavior of individual and composite Web Services has fundamental implications on how they can be discovered, combined, and analyzed. When considering a business system with multiple agents and multiple concurrent processes, one would like to have an automated way of checking some fundamental questions.

- Will the process necessarily terminate?
- Will the service respond within a given time?
- Will the service ever deliver something as promised?

This perspective is closely related to work on process algebra such as π-calculus and work in the verification community, both of which study distributed automata with message passing of one form or another [99-102]. Process algebra is used to define specifications from the system configuration and to analyze specifications in order to find any violations from achieving high-level goals.

The workflow community has focused on activity-based models. These represent a process by combining activities with some form of control flow. The typical examples are flowcharts, Petri nets, and finite state machines [103-105].

The semantic Web Services community favors an activity-based perspective. Much of that work assumes that atomic services perform activities, which have the effect of

changing the state of an "external world". These approaches permit the use of logic-based axiomatization and reasoning about how composite Web Services affect their external world and thereby permit the use of goals-based planning algorithms for automated construction of compositions. A situation calculus is typically used to provide formal basis [9, 96, 106]. The PSL standard has also been advanced for this purpose [107].

2.5.3 Service Security

Threats to Web Services involve threats to the host system, the application, and the entire network infrastructure. Web Services implementations may require point-to-point and/or end-to-end security mechanisms, depending upon the degree of threat or risk. Traditional, connection-oriented, point-to-point security mechanisms may not meet the end-to-end security requirements of Web Services. However, security is a balance of assessed risk and cost of implementation. Depending on the implementer's risk tolerance, point-to-point transport level security can provide security solutions. To secure Web Services, a range of security mechanisms that accommodate the presence of intermediaries are proposed to solve problems related to authentication, role-based access control, distributed security policy enforcement, and message layer security.

Policy-Level Security

Policy-level security is principally concerned with the existence of guards and their role in the architecture. There are fundamental concepts related to security from the perspective of policy-level security including the resources that must be secured and the mechanisms by which these resources are secured. The unauthorized access threat may be countered by a mechanism that validates the identity of potential agents who wish to

access the controlled resource. That mechanism is, in turn, controlled by the policy document that expresses what evidence must be offered by which agents before the access is permitted. A given policy document usually contains a mix of obligation and permission policy statements. These two kinds of policies have different enforcement mechanisms including a permission guard and an audit guard. A permission guard is a mechanism that can be used to verify that a requested action or access is permitted while an audit guard can only verify after the fact that an obligation has not been met.

There have been many proposals regarding policy-level security. However, they focus mainly on server-side policy. The server decides the access level to its own resources from policy documents. In Web Service environments, services are spatially dispersed such that the design of a policy-level security system on distributed objects on the network is necessary. The most common approach is to separate policy logics from business logic [108]. Clients obtain certifications from access control server for a specific service. A service provider then examines certification from clients to decide access level. In the Organization for the Advancement of Structured Information Standards (OASIS), a centralized access policy server gets requests with client identification from a service provider and makes a decision on the access level of the clients [109].

Message-Level Security

A common technique for securing SOAP exchanges is to rely on a traditional lower-level secure tunnel between the endpoints such as the Secure Socket Layer/Transport Layer Security (SSL/TLS), Virtual Private Networks (VPNs), and Internet Protocol Security (IPSec). Although this approache works in many situations, it has a serious disadvantage

in relation to transport-level security in a service oriented environment. Web Services use a message-based approach that enables complex interactions that can include the routing of messages between and across various trust domains. A message might travel between various intermediaries before it reaches its destination. Therefore, message-level security is important as the requester agent may be communicating with the ultimate receiver through the use of one or more intermediaries. The traditional lower-level security mechanisms are not appropriate for secure end-to-end communication as the intermediary domains may use different security tunnels.

WS-Security is designed to construct secure SOAP message exchanges though the use of end-to-end security. WS-Security provides an XML vocabulary for designing cryptographic protocols and is undergoing standardization at OASIS [109]. WS-Security defines how to sign or encrypt SOAP messages without relying on a secure transport. A central abstraction is the use of security tokens, which ensure security claims such as user identification, cryptographic keys, or certificates. WS-Security provides syntax for multiple token formats such as XML username tokens, and XML-encoded binary tokens conveying X.509 public key certificates or symmetric keys.

2.5.4 Service Reliablity

In the context of Web Services, the focus on service reliability is not generally on issues such as syntax errors or badly written applications., Rather, the issues of reliability are addressed at several levels including the reliable and predictable delivery of services, such as message transport and service discovery, reliable and predictable interactions between services, and the reliable and predictable behavior of individual requester and

provider agents. This analysis is separate from concerns of fault tolerance, availability, or security.

Regarding service reliability, one cannot guarantee that service provider agents and/or service requester agents will always perform flawlessly. Particularly, since the different agents may be owned by different organizations and subject to different policies and management, it is not possible to engineer complete service reliability. Undesirable behaviors include timeouts, runtime errors, and violation of functional contracts. Therefore, there are efforts to enhance reliability and reduce the cost of failure. One way to incorporate service level reliability would be to use standardized headers containing information such as transactional bracket markers and context information. Such information is added to messages exchanged between service requester agents and service provider agents in such a way that intermediaries can process messages and monitor transactions with only minimally impact on existing applications. Specialized transactional intermediaries then process messages' transaction-specific headers (such as beginning of transaction, commitment, roll-back and so on) and mark messages that they process with the results so that applications can respond appropriately.

Related to transactional monitoring is the monitoring of service choreographies. This approach mainly utlizes annotations inside process logic [15, 18]. In this way, processes like Business Process Execution Language for Web Services (BPEL) remain standard and executable by any standards-compliant engine [63]. This requires steps of incorporating assertions into the standards business logic. This also implies the need for a module that can translate functional contracts into annotated choreography.

Another approach is to deploy specialized intermediary processes whose specific function is to ensure that the choreographic as well as the static requirements of service usage are being met. This is especially important when the provider agent of a service is not in the same ownership domain as the requester agent. Such intermediary processes can be delegated to third-party partners. The key property of the deployment of third-party services is that neither the requesters of services nor the providers of services need to be concerned, since monitoring and message processing are conducted and regulated by a third-party [31]. This is possible because the architecture does not require messages to be consumed by single agents but allows multiple agents to collaborate in the processing of a given message. Each service role establishes a specific functionality, often encoded in specific headers on the messages.

CHAPTER 3 SEMANTICS FOR SERVICE-ORIENTED COLLABORATION

Here we propose and develop our new framework, the Web Service Collaborative Process Coordinator (WSCPC), which is designed to support collaborative processes to integrate design engineering, process engineering, and business planning through Web Services technologies. To this end, we design service models, which a VO can utilize to discover and collaborate with diverse services of design, manufacturing, and logistics. Our service models are a general framework from which to realize specific functionalities, such as defining and enacting services, and monitoring the traces of service execution.

3.1 Requirements for Service-Oriented Collaborative Process Management

In service-oriented collaboration, companies decide which modules will be processed internally versus which ones will be outsourced. Once the company decides to outsource a module, the company searches possible module suppliers through a service registry. Companies that aim to provide components of a product must register their components into the service registry. To locate module suppliers, the company utilizes the functional definitions and interfaces of entries in the service registry. In our service-oriented collaborative process, companies' capabilities are advertised and searched with semantic information published in the service registry. Such semantic information includes service name, type of service, input/output specifications, and pre-/post- conditions as described in the Process Definition Model (PDM) model in Section 3.3. Since there can be multiple service providers based on service requirements and capability matching, service providers should be ranked numerically. One possible solution to the problem of ranking service providers is to use recommender systems. Recommender systems adopt a static

view of a group of recommendations for a given item and use it to predict the value of the item [110]. Through recommender systems, the service providers with the required service capability can be evaluated, ranked, and selected based on the service requester's decision logic.

As partners are formed based on the public semantic information, the next steps in collaborative processes involve monitoring, communication, and enacting between partners. In a service oriented approach, such steps must be executed through platform neutral, Web Service based implementations. Behaviors and messages between partners are described based on universal semantic language so that partners can share them in a heterogeneous software environment.

We describe the requirements that must be considered in order to realize service-oriented collaboration in a heterogeneous VO environment [111, 112].

- Implementing and Deploying Web Services: The service provider's service must be accessible through an interoperable Web interface. Therefore, the service provider's functional capability needs to be implemented with public Web Service interfaces. In addition, the implementation of functional capabilities must be separated from the public interface accessible from the Web, such that an implementation-neutral environment is possible and changes of the internal application logic do not affect the collaborative global process.

- Registering and Discovering Service: The service requester must be able to match its business requirement with what a service provider has posted. Service registration and the discovery model should be able to match needs based on the semantics

exchanged between service requester and service provider. To this end, the published description of functional capability should be well written and understandable.

- Proactive Process Enactment: Process enactment corresponds to process execution. The service-oriented framework should be able to execute the given process flow and to configure process flow at run-time. Thus, the service model must provide sharing functionality that can show the run-time changing configuration of process flow as well as the task execution status.

- Sharing Service Execution: A service-oriented framework should be able not only to execute the given process flow, but also to configure process flow at run-time. Thus, the service model must provide sharing functionality that can show the run-time changing configuration of process flow as well as the task execution status. In addition, side effects from process execution require run-time coordination between partners in order to accomplish an optimal solution. Sharing service execution is therefore necessary for proper collaboration to achieve optimal results. Such service execution can be realized via predefined notification signals or proactive monitoring.

- Support for Multi User Environment: For secure collaboration, it is necessary to have appropriate access control support for multiple users in a distributed environment. The resources and workspaces are distributed and shared at some level. However, the level of sharing of resources must be restricted by certain policies. Based on user policy, access levels such as writing, reading, or removing are assigned to each user. Naturally, this access pattern should be regulated. Such access control must be

implemented in a service-oriented paradigm since the system must maintain platform neutral characteristics.

3.2 A Grammar-Based Approach

Web Service Collaborative Process Coordinator (WSCPC) is a service oriented, web-service based framework to exploit the benefits of modular product design derived with collaborative suppliers in an integrated process. In VO environments, products may have a complex structure assembled from a large number of modules and sub-systems, each with complex and multiple relationships to other modules and systems. Functional sub-systems also run across a number of modules as well as interface with each other.

The process in WSCPC begins with a high-level process design in which a company defines requirements for its desired product. The design process then moves into defining details. The process definition defines a product design process based on a process flow graph and reflects all the external requirements and constraints of a product. Functional sub-systems in a process flow are defined as production rules in Process Grammar [20, 21, 113] along with their relationship to each other. In addition, input and output specifications describe how a module interfaces with other modules and its subsystems and how it enables product variants. Functional subsystems in a process flow can also be decomposed into a set of sub-functional subsystems. Groups of subsystems are also represented with a flow graph; several flow graphs can correspond to one subsystem as an alternative.

Our process flow graph depicts tasks, data, and the relationships among them, describing the sequence of tasks for a large design and manufacturing (D&M) activity [112]. Four

basic symbols are used to represent the process flow: ovals represent logical tasks, two concentric ovals represent atomic tasks, rectangles represent specifications, and diamonds represent selectors. nodes).

A logical task can be decomposed into subtasks, while an atomic task cannot. An output specification produced by a task can be consumed by subsequent tasks as an input specification. These elements can be combined into a process flow graph using directed arcs to indicate the specifications used and produced by each task. Specifications with no incoming arcs are the first inputs to the process flow.

Figure 4 below illustrates the production rule used for a product casting process in a D&M domain. The product casting process has a logical task "Casting Product", three input specifications "SelectedMaterial", "Dies", and "Trim Die" with "Finished Product" as an output specification. We now consider three alternative processes as shown as (a), (b), and (c). Figure 4 (a) depicts a task delegation, while (b) and (c) show two alternatives of possible task decomposition. Figure 4 (b) represents a vacuum die casting process, while (c) illustrates a high pressure die casting process.

Once a task is delegated, the "Casting Product" task is marked as "Outsourcing" and engineers search the appropriate service provider [114]. When engineers decide to decompose the "Casting Product" task, they consider input specifications, design constraints, and properties of each alternative in order to determine the actual process design.

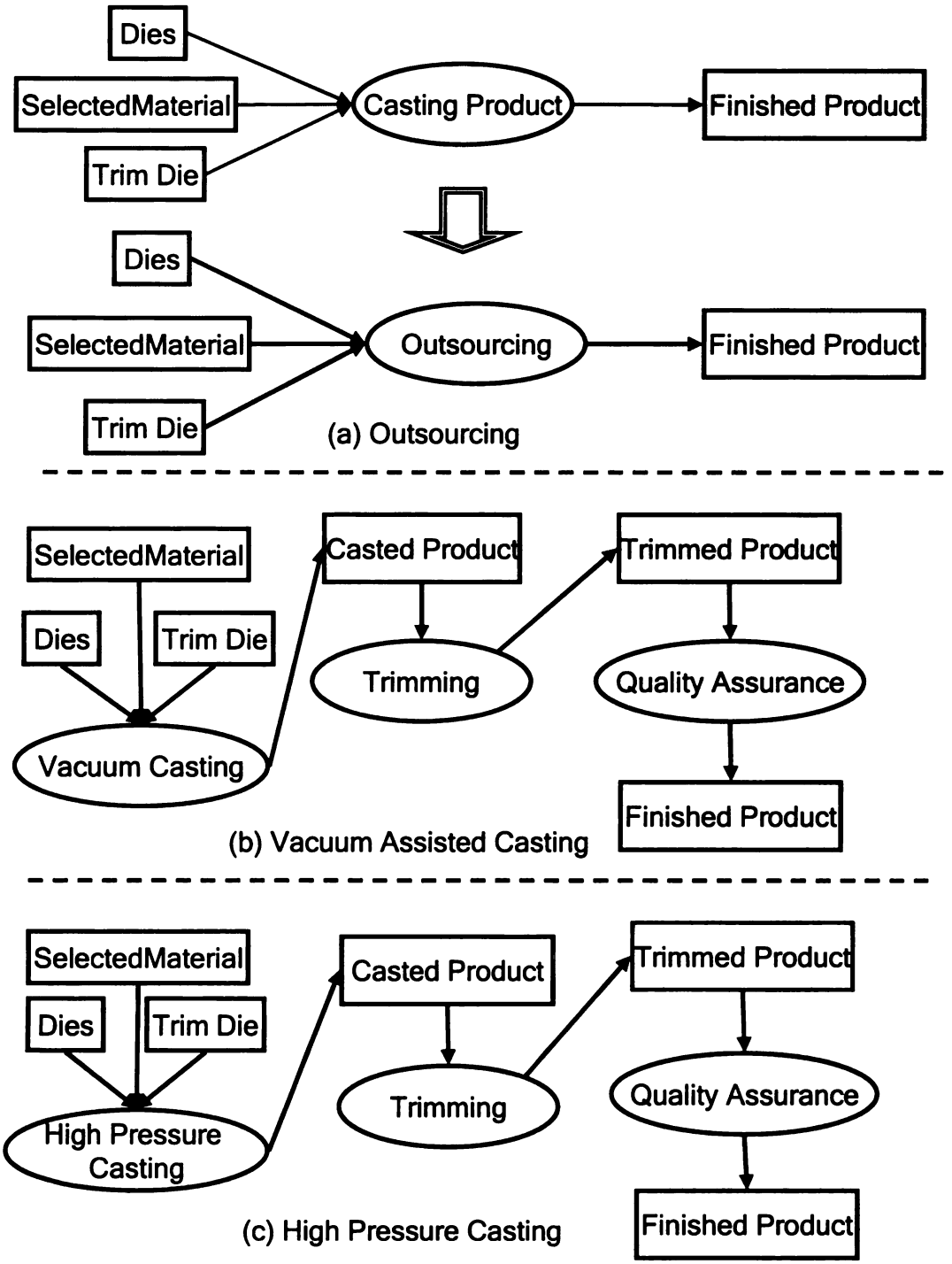


Figure 4 An Example of Production Rule

The grammar-based approach provides the mechanism for transforming a high-level process flow graph into progressively more detailed process flow graphs. The grammar consists of a set of production rules. A production is a substitution that permits the replacement of a logical task node with a flow graph that represents a possible way of performing the task. If there are several production rules with the same left side flow graph, this implies that there are alternative production rules for the logical task. Several production rules for a logical task imply that there are alternative production rules for that task. For example, *Casting* and *Machining* may be, two alternative productions for the task *Manufacturing*. This capability is critical to maintaining the usability and effectiveness of the overall framework. Therefore, Process Grammar naturally captures the hierarchical design methodology and allows systematic exploration of the D&M space.

3.3 Web Service Semantics for Collaborative Process

To support collaborative processes in terms of service-oriented computing, we observed the required characteristics that should be incorporated into service oriented collaborative behavior modeling. To realize such a goal, we identify five types of aspects needed for modeling collaborative process behavior to be shared among partners [114].

- The *functional aspect* describes how a process is decomposed into functional subsystems.
- The *behavioral aspect* depicts process control flow such as when an activity is to be executed.
- The *informational* aspect concerns data exchanged during process execution between peers. Data can be either input/output specifications or activity execution status.

- The *organizational aspect* describes how a process is distributed to a group of participants.
- The *transactional aspect* depicts activity execution and sharing.

Considering these five aspects, we define and propose three types of semantic service description models, which are built on top of OWL-S. [64, 68]. The Process Definition Model (PDM) describes functional and behavioral aspects. The Process Monitor Model (PMM) represents informational and organizational aspects. The Process Enactment Model (PEM) characterizes transactional aspects [111, 115].

3.3.1 The Process Definition Model

The Process Definition Model (PDM) is designed to represent functional and behavioral aspects of services. Figure 5 below describes the ontology of service models that we have developed. In PDM, a process is composed of a set of tasks or sub functional systems. Tasks or sub functional systems can be defined as activities needed to complete a process. Such activities describe functional semantics of a process. In a collaborative environment, each task is associated with a certain role provider; hence, composition of tasks creates a collaborative environment. An activity corresponds to a particular service provided by collaborative partners. Such services can be selected or modified at run-time as needed.

Tasks can represent an internal activity implemented either as a service performed by a single service provider or as composite activities by multiple collaborative partners. In either case, services can be represented as an aggregated service, which is composed of several services that can show all the sub functional units in a service. Since each service

has its own service description, one can infer the aggregated service by seeing all of its sub-functional service descriptions.

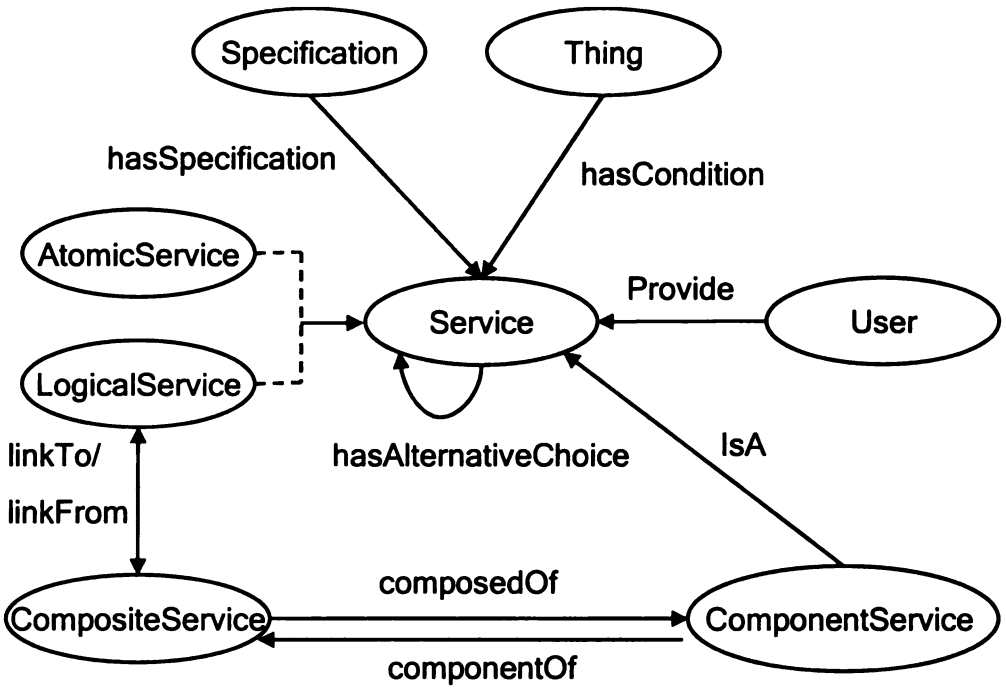


Figure 5 Ontology for the Process Definition Model

Moreover, services in a composite service are related to each other in temporal order. The temporal relation between services can represent interactions between sub functional systems in a service, and hence describe behavioral semantics of a service.

These characteristics lead us to define a meta model $CM = (T, D, U, S, DC, \tau, \rho, \pi, \varphi)$ capturing collaboration aspects in service oriented environment where the components of the model are defined as follows [111, 113].

- T is a finite set of abstract tasks needed for completing a collaborative process.

- D is a finite set of abstract shared data objects needed by multiple tasks.

- U is a finite set of users that can provide and consume services.

- S is a set of available services within a collaborative community. Services can be a logical service that is composed of a set of functionalities or an atomic service that represents one single non-decomposable functionality.

- $DC \subseteq (T \times D) \cup (D \times T)$ is a set of arcs as a temporal relation connecting data and abstract tasks. (T, D, DC) forms a directed graph.

- $\tau : T \rightarrow 2^S$, where task (t) = s represents that a finite set of services s are available to execute task $t \in T$.

- $\rho : U \rightarrow 2^S$, where service (s) = u represents that a finite set of service $s \subset S$ are provided by a community user $u \in U$.

- $\pi : T \rightarrow S$ is an assignment function, which associates each task $t \in T$ with the best service $s \in S$ to complete the entire collaborative process.

- $\varphi : S \rightarrow 2^T$ is a composition relation, where a service S is represented through a composite set of tasks T in order to describe its behavior.

Figure 5 gives a graphical representation for the ontology of the above meta model. Each class in the diagram corresponds to a set and each relation corresponds to a relation in the formal meta-model. The WSCPC process model follows a grammar based approach as

described in Section 3.2. Services can be decomposed by multiple sub components called tasks and each task can represent a service. Therefore, a service can be structured hierarchically based on services and tasks. Relations φ and π represent such hierarchical relation among services and tasks. φ shows a service's behavior by a composition of multiple tasks. π represents which services are assigned to fulfill a specific task. τ stands for alternative relations between services and tasks. To execute a task, a set of services based on τ are considered before execution and an evaluation function may be applied to choose the most appropriate one. In a service oriented environment, each user provides one or more services based on their capability. ρ exhibits such a users-service relationship in a service oriented environment. ρ enables users to identify service providers and get provider information. ρ can be used to enable direct contact when necessary or to predict service quality based on a service provider's reputation.

In order to support web service semantics, the actual behavior model is written in the ordinary semantic language OWL. Figure 6 and Figure 7 illustrate examples of a service description written with OWL based on the service behavior in Figure 4. A composite service represented with PDM can be translated into a directed acyclic graph, which can then determine temporal dependencies of services. If a service A is determined to be dependent on another service B, then the execution of A will be affected by B.

The general ontology of our PDM is given in class *Service*, which refers to the definition of a task that may consist of several other subtasks. *Service* is a super class of two subclasses *LogicalService* and *AtomicService*. *LogicalService* represents the logical task

of Process Grammar. It is expanded into a detailed sub-process flow when it is applied by a *ServiceComposite*. The *hasAlternativeChoices* lists the names of such available *ServiceComposites*. Formally *hasAlternativeChoices* is defined by relation τ. Figure 6 shows how *hasAlternativeChoices* lists *ServiceComposite*.

AtomicService represents an atomic task in Process Grammar. Its semantic description is similar to that of *LogicalService* except that *AtomicService* does not have the *hasAlternativeChoices* property.

Service possesses four properties *hasPreConditon*, *hasPostConditon*, *hasInput*, and *hasOutput*, which are used by a service consumer to identify and utilize the service. These properties are inherited by their subclasses *LogicalService* and *AtomicService*.

In the service model of WSCPC, the representation of a relationship between complex collaborating services is given in terms of *ServiceComposite* and its OWL descriptions. The representation of activities occurring when services collaborate is given in terms of the enacting service and monitoring.

The *ServiceComposite* collates one or more *ServiceComponents* (each being a service) and is analogous to a production rule in Process Grammar. The properties of *ServiceComposite* represent a φ relation. Every service component should be linked to another service component. The *ServiceComposite* uses the *linkTo* and *linkFrom* properties to describe the flow between service components when collected. In Figure 6, a logical service *DesignDies* has three alternatives, *VacuumAssistedCasting*, *HighPressureCasting* and *SemiSolidCasting*. Each alternative is a *ServiceComposite*, a collection of several *ServiceComponents*. Figure 7 shows one *ServiceComposite*

VacuumAssistedCasting is declared as *collection* type and composed of several *ComponentService*.

In OWL-S, the *CompositeProcess* is similar to the production rule but can have only one *composedOf* property that appends a single ControlConstruct (Sequence, Split, Choice) to a single CompositeProcess. This approach seriously restricts representing alternatives with complex workflow clearly and efficiently as a single basic unit. Our model does not have such restrictions

The tags *providedBy* and *boundBy* represent the relation ρ and π. These relations are initially empty and later filled in at run-time since such a relationship is defined during the process enactment stage. The *providedBy* tag represents a service provider, not a service itself. The tag *boundBy* tells which actual service is assigned to the logical or atomic service.

. Figure 5 illustrates the general ontology of the Service Oriented Definition Model.

```
<!-definition of "DesignDies"-->
<service:LogicalService rdf:ID="DesignDies">
    <service:providedBy/>
    <service:boundBy/>
      <service:hasPreCondition>
      <service:condition rdf:resource="#Materials"/>
      </service:hasPreCondition>
      <service:hasPreCondition>
     <service:condition rdf:resource="#Finishing"/>
      </service:hasPreCondition>
      <service:hasPostCondition>
```

```
      <service:condition rdf:resource="#Certification"/>
</service:hasPostCondition>
    :   :
    :   :
<service:hasInput>
    <service:Spec rdf:resource="#DesignRequirements"/>
  </service:hasInput>
    :   :
    :   :
<service:hasOutput>
    <service:Spec rdf:resource="#TrimDieDesign"/>
  </service:hasOutput>
  <service:hasOutput>
    <service:Spec rdf:resource="#DesignedDies"/>
  </service:hasOutput>
    :   :
    :   :
<service:hasAlternativeChoices>
    <service:AlternativeChoice rdf:parseType="collection">
      <service:ServiceComposite
rdf:resource="ServiceComposite.daml#VacuumAssistedCasting">
      <service:ServiceComposite
rdf:resource="ServiceComposite.daml#HighPressureCasting">
<service:ServiceComposite
rdf:resource="ServiceComposite.daml#SemiSolidCasting">
    </service:AlternativeChoice>
</service:hasAlternativeChoices>
</service:LogicalService>
```

Figure 6 An Example of Semantic Service Definition

```
<!-- definition of "VacuumAssistedCasting" -->
```

```
<service:ServiceComposite rdf:ID=""
rdf:parseType="collection">
  <service:hasSCPreCondition>
  <service:condiion rdf:resource="#Materials "/>
  </service:hasSCPreCondition>
  <service:hasSCPostCondition>
    <service:condiion rdf:resource="# Certification "/>
  </service:hasSCPostCondition>
<service:hasSCPostCondition>
    <service:condiion rdf:resource="# EndMarkets "/>
</service:hasSCPostCondition>
<service:ComponentService
rdf:resource="Services.owl#DesignRunnersAndGates">
    <service:linkFrom/>
    <service:linkTo>
<service:LogicalService
rdf:resource="Services.owl#OverflowDesign"/>
</service:linkTo>
  </service:ComponentService>
    :   :
    :   :
<service:ComponentService
rdf:resource="Services.owl#Finalizing">
<service:linkFrom>
```

```
<service:LogicalService
rdf:resource="Services.owl#SelectCooling"/>
</service:linkFrom>
<service:linkFrom>
<service:LogicalService
rdf:resource="Services.owl#OverflowDesign"/>
</service:linkFrom>
<service:linkTo/>
</service:ComponentService>
</service:ServiceComposite>
```

Figure 7 An Example of Semantic Definition of ServiceComposite

3.3.2 The Process Enactment Model (PEM)

In order to provide support for transactional aspects in a collaborative environment, we propose the Process Enactment Model [114]. The PEM provides a standardized representation to describe which actions are called on what services and what state changes occur as result of the service call. As we have observed, transactional aspects are about process enactment and sharing. By capturing the semantics of other users' transactions, users in heterogeneous environments can understand the collaborating partners' intentions and then incorporate them into their workspace.

Services in collaborative processes should be defined to represent all the characteristics as follows.

- The *state* of the process execution.
- The *functionality* represented as a set of operations building activities.
- An *activity* that is modeled as a transition of the states and functionalities involved.
- The *provider* that is associated with functionalities.

To this end, our Process Enactment Model defines classes of service states and properties related to state changes due to service enactment. We define five basic classes related to service states. Actions that users invoke tend to change states of services. We implement transactional aspects by defining what service states are and what changes those states. Users of WSCPC can control and capture task execution through this model.

The process execution status of each service is captured by the following five states: *Initial*, *Ready*, *Running*, *Finished*, and *Exception*. These states are described as follows.

- The *Initial* state indicates that nothing has been initialized in a service.
- The *Ready* state shows that input data has been bound to a service, but that service execution is not yet invoked.
- The *Running* state indicates that execution of a task has been invoked and keeps on going.
- The *Finished* state indicates that execution of a service has been finished. Two sub states are *success* and *fail*.

- The *Exception* state indicates that an unexpected event has occurred during the proceeding state.

The functionality affecting state changes in the Process Enactment Model are described as follows.

- *ProvideInput* delivers input data to a service.
- *RetrieveOutput* transports output data from a service to a viewer.
- *InvokeEnactment* sets a cue to start process enactment.
- *Execution* carries out the application of a production event or a tool execution event.
- *Rollback* instantiates a rollback event caused by exception or systems to a service.
- *EnforcedRollback* delivers an abort event by the user to a service.

The activity is modeled as $\phi \underset{\delta=<e,O>}{\Rightarrow} \phi'$, where O is a target object representing either a task or data specification, ϕ encapsulates properties of a target object O before transition, ϕ' encapsulates properities of the object O after transition takes effect, and δ shows a labeling function (enactment property e $\times$ target object O) that enables transitions.

Assume a process is structured linearly in the order data d_0 as input to task *t1* ,which produces output *d1*. A few examples listed below show how activity is modeled in the WSCPC environment.

- Providing necessary inputs to *d0* before starting a task *t1*

initial $\Rightarrow$ *ready.*
$<$*provideInputs,d0*$>$

- Forcing task *t1* to execute

initial $\Rightarrow$ *ready.*
$<$*invokeEnactment,t1*$>$

- Confirm task *t1* in execution

ready $\Rightarrow$ *running*
$<$*execution,t1*$>$

- Bind outputs from task *t1* into a data object *d1*

finished $\Rightarrow$ *finished*
$<$*retrieveOutput,t1*$>$

initial $\Rightarrow$ *ready*
$<$*provideInput,d1*$>$

- Cancel current transition and come back to former status

running $\Rightarrow$ *ready*
$<$*rollback,t1*$>$

3.3.3 The Process Monitor Model (PMM)

During the enactment of the process flow, the user may desire to know how their request is being processed at the service provider. Users of WSCPC can monitor a process by capturing traces of task executions. The Process Monitor Model (PMM) provides a standardized way of capturing and delivering informational and organizational aspects to users. Informational aspects concern data that involve execution trace and input/output

specification. Provisions of PMM enable service consumers to capture changes and effects in a collaborative process environment. Figure 8 illustrates the Process Monitor Model.

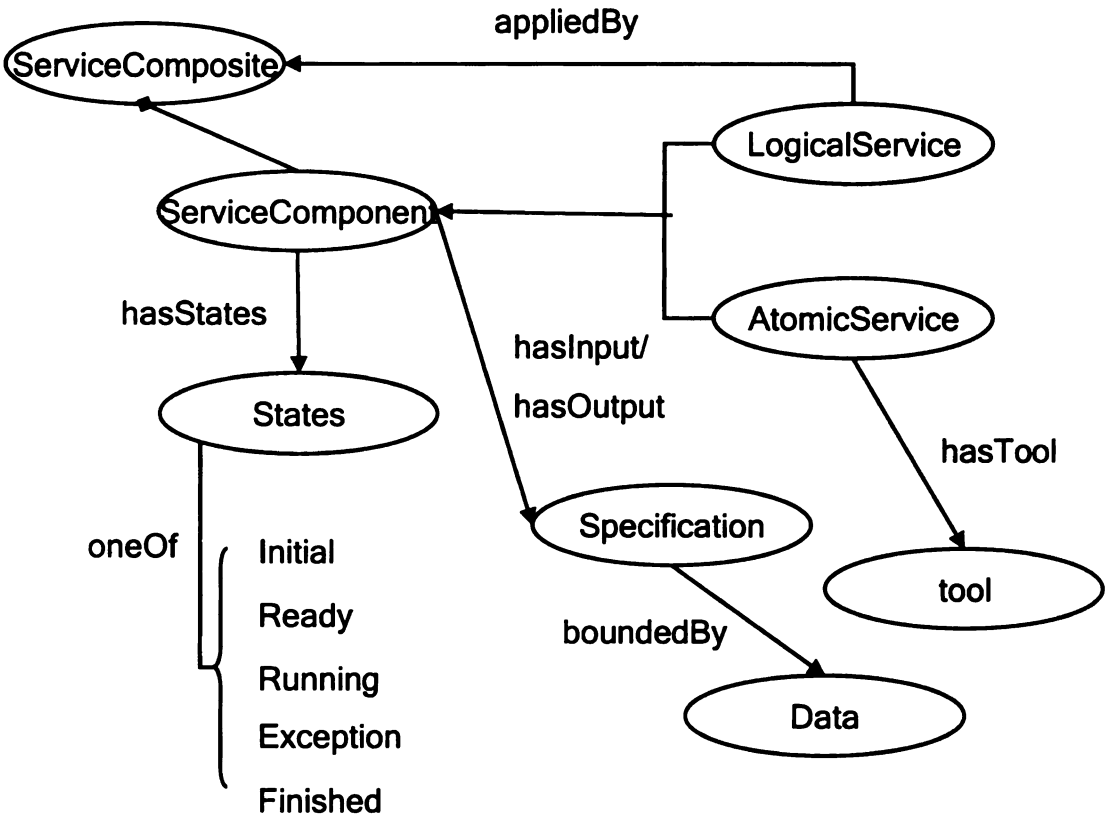


Figure 8 Ontology for the Process Monitor Model

Since there are two kinds of services, the way to capture execution information should vary depending on the service types. Based on the mathematical model of Process Grammar, the execution of a logical task applies the production to the logical task. Therefore, a logical task will be accompanied by a service composite as a mark of execution. The execution marks regarding logical services are represented by the *applied* property. The *applied* property represents production rules in Process Grammar that show

a relationship between a service and its components. The *applied* property shows the selected service's behavior in which services may use a variety of business logic to complete its goal.

The execution of an atomic task executes a given tool for the atomic task. The execution traces regarding atomic tasks are represented by the *hasTool* property. The atomic service's behavior is defined by external tool invocation. Based on the tool used, the behavior of atomic services is defined.

Regarding input/output specification, the trace shows data given by and/or sent to others. The data and specification relationship is explained with the *boundBy* property. Input specifications are bound to input data before task execution and output specifications are bound to output data after task execution. Both specifications can be written with OWL so that the data can be understood in a universal way.

Therefore, PMM is modeled as a triplet $M = (O, \delta, G)$, where $O \in T \cup D$, T and D are task and data objects, $\delta \in \{applied, boundBy, hasTool\}$, and G is the grounding of abstract objects T and D *[116, 117]*. For example, PMM information regarding vacuum assisted casting method is applied for a task *CastingProcess* can be represented as (*castingProcess, applied, vacuumAssistedCasting*). Certain requirements can be bound to the data object *designRequirements*. Figure 9 illustrates the definition of *outputSpecification* written in OWL. The *boundBy* and *specifications* properties are defined in *service* as an object property and a class, respectively. Attributes that are shared among partners are defined along with *specification* class. Those attributes include things such as size, performance factors, and tolerance. Figure 10 shows an example of

an output specification bounded in *outputSpecification*. Partners can locate the meaning of tags through process definition files written in OWL and extract information bounded *outputSpecification* data objects.

```
<owl:Ontology rdf:about="">
  <owl:imports rdf:resource="&service;" />
    :
    :
  <owl:imports rdf:resource="&specification;" />
</owl:Ontology>
    :
    :
  <owl:ObjectProperty rdf:about="&OutputSpecification;#
DesignSpec">
    <rdfs:range rdf:resource="#Specification"/>
  </owl:ObjectProperty>
<owl:Class rdf:ID="Specification">
  <rdfs:label>DefaultSpec</rdfs:label>
  <rdfs:subClassOf rdf:resource="&owl;#Thing"/>
  <rdfs:comment>
    Specification represents a specification needed to
define a process requirement.
  </rdfs:comment>
</owl:Class>
    :   :
    :   :
<owl:DatatypeProperty rdf:ID="Size">
    <rdfs:domain rdf:resource="#Specification"/>
    <rdfs:range rdf:resource="&xsd;#string"/>
  </owl:DatatypeProperty>
```

```
  <owl:DatatypeProperty rdf:ID="tolerance">
    <rdfs:domain rdf:resource="#Specification"/>
    <rdfs:range rdf:resource="&xsd;#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="Density">
    <rdfs:domain rdf:resource="#Specification"/>
    <rdfs:range rdf:resource="&xsd;#string"/>
  </owl:DatatypeProperty>
    :   :
    :   :
  <owl:DatatypeProperty rdf:ID="email">
    <rdfs:domain rdf:resource="#Specification"/>
    <rdfs:range rdf:resource="&xsd;#string"/>
  </owl:DatatypeProperty>
```

Figure 9 Definition of Design Requirements

```
<Service:BoundedBy>
  <OutputSpecification:DesignSpec
rdf:ID="DesignRequirementsSpec">
  <Specification:Size Diameter="14" Weight="0.124" … />
  <Specification:Tolerance dimentional="0.005"
range="plusminus" />
  <Specification:performanceFactor>high thermal
conductivity</DesignOutputSpec:performanceFactor >
  <Specification:performanceFactor >uniform metal fill</
DesignOutputSpec:performanceFactor >
  <Specification:Density range="LT">10</
DesignOutputSpec:Density>
  <Specification:TensilYieldStrenth range="GT">20</
DesignOutputSpec Tensi- …>
    :   :
```

```
    :   :
  </Specification:Specification>
</Service:boundBy>
```

Figure 10 An Example of Output Specification

3.3.4 Service Registry Model

WSCPC includes a model for registering and discovering the semantics of services, which should be written with OWL. The WSCPC service registry model provides functionalities that allow service providers to register service capabilities and users to locate the available services [111]. As illustrated in Figure 11, the service users can reference the semantic information from the service registry and compare the pre- and post-conditions as well as the input and output specifications with the semantics presented.

Service information in the service registry includes the following: name, type of service, input and output specifications, and the pre- and post-conditions. Pre- and post-conditions are used for maintaining consistency in distributed design and manufacturing processes. The principal idea of pre-/post- condition can be described as follows. The service requester must request that the service provider guarantee certain qualities before calling a service specialized on the component (pre-conditions. The service provider also guarantees certain properties after the service returns outputs to the service requester (post-conditions).

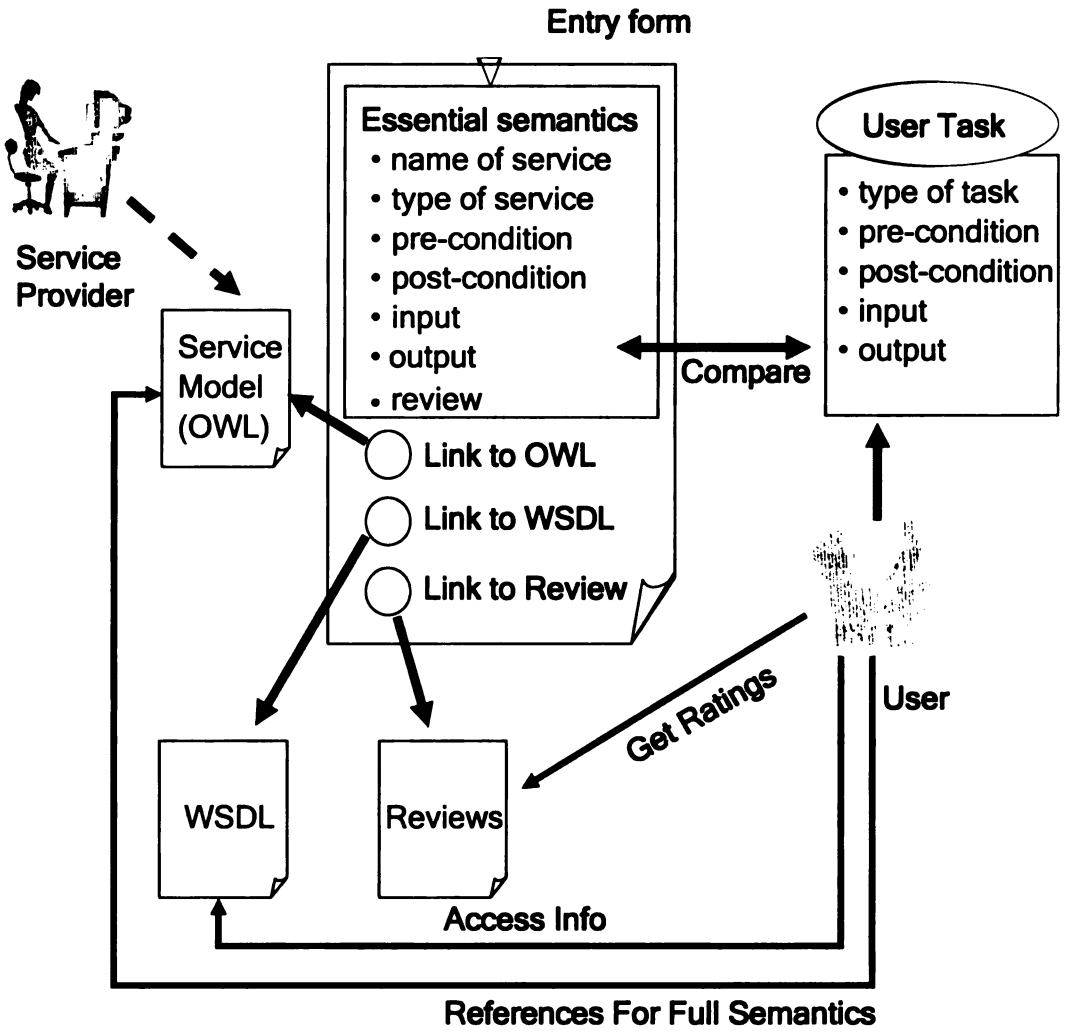


Figure 11 Service Registry Model

The service provider posts pre-conditions to inform the service requester of the requirements that the service provider must satisfy to invoke the service. For example, the service provider can restrict the qualifications of the service requester and specify the quality of inputs to deliver service invocation results. On the other hand, the service provider describes the quality matrix of service execution results in the post- condition

entry. The service provider can publish its quality of service level such as product specifications and processing time.

Since the service registry does not provide full-scale semantics, the OWL bounding provides the user with detailed semantics of a service. The full description of a service is given by a separate document, and the link to this document is provided to service searchers as well. The technical information of the service, such as the URL, the operation list, and type of required parameters, is provided by a separate WSDL document and URL links. Service users can see the technical information and full service description by following the links provided in the individual service information

Figure 12 illustrates an example of a service entry that is registered into the service registry. The example shows a die casting service that is posted by a company located in Troy, Michigan. The service capability that the service provider can deliver is described as a post-condition—in this example, machining, material support, etc.

```
<entry>
    <entryID> 00001 </entryID>
    <serviceType> CastingProducts </serviceType>
    <profile>
        <serviceName>Die Casting </serviceName>
      <Address>Troy, MI, USA</Address>
      <Telephone>517-xxx-xxxx</Telephone>
    :   :
    :   :
    </profile>
    <precondition>
        <Materials>Aluminum</Materials>
```

```
        <Machining>CNC</Machining>
        <Machining>Tapping</Machining>
        <Finishing>Powder Coating</Finishing>
    :   :
    :   :
    </precondition>
    <postcondition>
        <Certifications>ISO9002</Certifications>
        <EndMarkets>Automotive</EndMarkets>
        <EndMarkets>Furniture</EndMarkets>
    :   :
    :   :
    </postconditon>
    <inputSpec>TrimDies</inputSpec>
    <inputSpec>Dies</inputSpec>
    <inputSpec>SelectedMaterial</inputSpec>
    <outputSpec> FinishedProduct </outputSpec>
    <reviews>
        <average/>
        <standardDeviation/>
        <detail_info> http://wscpc/review/detail.xml
        </detail_info>
    </reviews>
    <wsdl_binding> http://wscpc/wsdl/DieDesign.wsdl
    </wsdl_binding>
    <owl_binding> http://wscpc/wsdl/services.daml
    </owl_binding>
</entry>
```

Figure 12 Entries in Service Registry

To manage the design and manufacturing (D&M) process using Web Services, service providers should register their functional capabilities into a public registry so that the service consumer is able to discover it if it needs a collaborator. For this purpose, we propose a service registry model for registration and discovery of the semantics of services written on top of OWL-S [64]. The service consumer references the semantic information in the entries of the service registry and compares the service type, the pre- and post-conditions, and the input and output specifications of users' activities with the semantics presented. Pre- and post-conditions are used for maintaining consistency in a distributed D&M process. A service requester is guaranteed certain qualities from the service provider before calling a service (pre-condition), and the service provider guarantees certain properties after the service return outputs to the service requester (post-condition) [118]. The service provider posts pre-conditions to inform the service requester of the requirements that the service provider must satisfy to invoke the service. On the other hand, the service provider can describe the quality matrix of service execution in the post- condition entry. The service registry model also includes reference of the Semantic Service Description Model of services such that the service consumer can understand the expected service behaviors using OWL-S inference engine [115].

However, semantics including pre- and post-conditions have limitations when used in the collaborative process since semantics can at most provide the specifications of required inputs/outputs and their descriptions of services. Due to the extreme competitiveness of service providers, it is possible that multiple providers may claim they can deliver the required services satisfying all the specifications. In such case, semantics may not support service consumers' selection processes sufficiently and consumers should do additional

work beyond the service oriented environment to find the right partners. Even if there are not many choices from a consumer's side, the service consumer may want to know how the service performs as well as what the service delivers before one decides to use a particular service. Therefore, the registry must offer an additional way to help with selection when multiple providers present similar specs.

In WSCPC, reputation is used to assist with a consumer's selection for that purpose. After service usage, each peer (service consumer and provider) can score partners' behaviors and leave evaluations inside the registry at will. Reviews are composed of three parts: average, standard deviations, and a list of individual review scores. The average and standard deviations are calculated and stored in order to enable a brief overview. Whole reviews are also available to allow individual review analysis. The whole reviews are stored in separate files assigned to each user, and a link is provided so they are accessible. For privacy, a reviewer's identity is hidden from outside view.

3.4 Communication Behavior

3.4.1 Support for Interoperability

Modern applications like electronic commerce and information retrieval require software components to be interoperable. Most of these applications assume that components will be added dynamically and that they will be autonomous and heterogeneous. Service-oriented concepts are proposed to meet such requirements, by providing platform neutral interfaces and self-encapsulation of functionality. Modulated components with platform neutral interfaces can deliver software components that work autonomously in

heterogeneous environments and that can be composed dynamically in order to provide aggregative services or value-added services.

These concepts set Web Services applications apart from conventional components-based applications, which always fulfill any methods invoked on them. In a service oriented environment, applications should be able to perceive their environments and act with other software components. Web service applications should be able to refuse an action when necessary. Applications should be able to communicate with each other to decide what information to retrieve or what physical action to take, such as shutting down an assembly line or avoiding a collision with another invocation.

Currently, many standards such as ARCOL [119], CNP [120], XLBC [121], KQML [122], and FBCL [123] have been proposed for use in software collaboration. These languages are designed for agents to talk with other agents and work together. Typically, they concern inter-agent dependencies, namely the configuration of a system in terms of the basic interaction means, agent generation and destruction, and organizations of the environments. They are based on considering the collaboration process essentially as a problem of communication [124] since communication play a fundamental role in collaboration. These standards rely on agents communicating through asynchronous messages that are structured in terms of a performative verb and contents [125, 126]. The contents along with the performative verb associate a meaning with a message. Structuring messages in terms of a performative verb and a content item is rooted in the speed-act theory [127, 128]. Formal semantics for KQML have been proposed to associate a meaning to a message [129]. The semantics are captured by feasibility conditions and a rational effect. The feasibility condition states what must be true before

a message is sent, and rational effect shows what will be achieved with such a message. Semantics enable a purely goal oriented coordination, i.e. interaction is performed only to achieve goals and interaction itself is not an activity on its own.

However, even though these approaches can be employed in Web Service-based applications, none are fully implemented. Furthermore, they have severe limitations when used in certain environments such as collaborative design and manufacturing processes. The reason comes from the fact that they are not designed to represent quantitative aspects (e.g. performance) with the functional specification of such system. If semantics allow arbitrary quantification, the reasoning process is unlikely to be practicable.

3.4.2 The Service Interaction Model

The Service Interaction Model is designed to assist service consumers during the service invocation by controlling and guiding the invocation of the operations at each inter-action step [116, 117]. The service interaction model represents communicative behaviors between partners in distributed and heterogeneous collaborative process management. As an elemental interaction model, we employ speech act theory [128], which is widely used in agent communication languages.

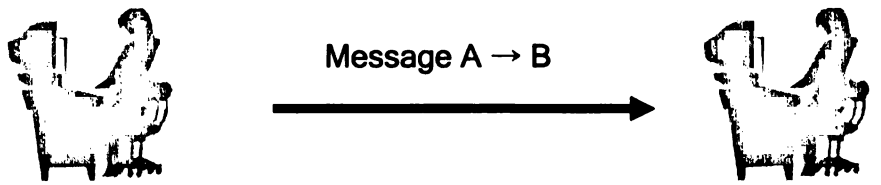
The WSCPC interaction model is structured as a triple *<Intention, Action, Target>*. *Intention* tells what semantic intention you have on the message. *Intention* is composed of *request*, *propose*, *accept*, *reject*, *query*, *answer*, *assertion*, and *declare*. The receiver can identify a message type by checking the *Intention* field. For example, if a message has *query* as an intention, then a service provider may have to prepare answers. If a message has *request* as an intention, then a service provider may consider it as a remote service

enactment or service invocation. If a service requester has the authority to control a service provider, then it can use a stronger message such as *declare*. The service provider can answer the query with a message with *Intention* of *answer*, and report its execution as *Intention* of *assertion*. *Intention* of *declare* can be used for notification purposes. Messages can be broadcasted through the *declare* message. The target can be any object whose state/value can be changed by time.

Target can be anything instantiated such as output specifications, operations to perform, message exchanged, etc. The *Target* field defines a namespace and is used to locate ontology in the message. *Target* is designed to remove ambiguity in communication as much as possible. *Target* may use OWL, RDF, or other standards specifications as long as there is a definition that can be found and understood among partners. *Target* with the WSCPC Service Interaction Model is modeled based on the Process Definition Model described in Section 3.3.1. It can be easily translated into a Petri Net based model together with the Process Monitor Model. Since Petri nets are widely used in quantitative analysis of system behavior, the WSCPC Service Interaction Model can be used for system behavior analysis and predictions, contrary to other communication languages. *Target*'s can be described with attributes associated. Attributes and their bounded data/tools use the Process Monitor Model. In case the target has no values associated with its own attributes, then the message can be generated with empty attributes.

Action's are classes that can change an object's state or property. That is, actions are classes that have some of the functional, behavioral, and transactional aspects described in Section 3.3, which are neither informational nor organizational aspects. *Action*'s are written with the Process Enactment Model presented in Section 3.3.2.

Service consumers can combine *Target* and *Action* to describe complex communicative behaviors in collaborative process management. Figure 13 illustrates two users, A and B, communicating using the WSCPC Service Interaction Model. Based on the scenario in Figure 4, user A wants to enact a task *CastingProcess* by applying a composition service *VacuumAssisted* casting. Assume both A and B know terminology *CastingProcess*, *VacuumAssisted*, and *finishedProcess*. User A first creates a message indicating that A wants task *castingProcess* to be a *VacuumAssisted* casting. Since A does not have full authority to control B, A's intention is marked as *propose*, which B later can accept or refuse. A's actual meaning is represented in the action field. (See Section 3.3.2). The *Target* field describes what *VacuumAssisted* or *finishedProcess* entails, which A includes in the message.



Intention	Propose
Target	LogicalTask *CastingProcess* Logical Task VacuumAssisted
Action	*initial* ⇒ <*assign*,*casting* Pr*ocess*> *appliedBy(VacuumAssisted)*

Figure 13 Messages Based on Service Interaction Model

Figure 14 shows an example of message to be exchanged. Service consumers locate where the service ontology is described and build messages with the intention, desirable action, and target object information. The terminology used from user A is defined in *Target* so that user B can locate information about the terminology that A is using. The actual action A implies for B to do is described in *Action*. From the example in Figure 13, only a logical task *CastingProcess* is affected by the desired action, while data specification is not. So the example in Figure 14 shows the *applied* property is bound only to the *CastingProcess* task while none of properties are bound to the *finishedProcess* specification. In this way, complex dialogue between service provider and consumers is possible and communicational behavior is enabled within the WSCPC environment.

```
<service:conversation rdf:ID="message1">
    <service:hasIntention>
        <service:intention rdf:ID="intentionID">
            <service:parameterType rdf:about="#propose"/>
        </service:Intention>
    </service:hasIntention>
    <service:Target rdf:ID="objectClass">
        <service:LogicalService
rdf:about="providerNS:#CastingProcess"/>
        <service:LogicalService
rdf:about="providerNS:#VacuummAssisted"/>
        <service:OutputSpecification
rdf:about="providerNS:#finishedProcess"/>
    </service:Target>
    <service:Action rdf:ID="actionClass">
        <service:InvokeEnactment
rdf:resource="consumerNS:#castingProcess">
```

```
            <service:applied
rdf:resource="providerNS:#VacuummAssisted"/>
        <service:InvokeEnactment/>
        <service:InvokeEnactment
rdf:resource="consumerNS:#finishedProcess"/>
    </service:Action>
</service:conversation>
```

Figure 14 A Message Example Exchanged During Service-Oriented Collaboration

3.4.3 WSDL Support

Collaboration requires support for complex interactions among companies. For example, while the D&M process is being enacted autonomously across the heterogeneous systems, companies may want to force certain decision modification on partners or highlight problematic parts from the design specification before they start the outsourced task. However, Web Services are subject to several limitations that reduce their applicability to enact such complex interactions. WSDL only supports the invocation of operations characterized by very specific signatures that fail to enable the peers to dynamically specify the realistic business interactions. Such characteristics in the WSDL interface do not enable users to specify the management of interactions between service consumer and provider.

To support interactions in service invocation, the ability of supporting interactive invocation must be defined and declared at WSDL [116, 117]. Figure 15 shows a portion of a WSDL declaration to support interactive service invocation based on the example described in Figure 13. Given the public operational specifications in WSDL, the service provider informs the references of the service semantic description, pointing out where

the ontology of a service description is located. Moreover, service providers should explicitly declare that their services support interactive invocation. Service consumers should prepare conversational input messages from the declaration of interaction type.

```
<definitions ..
xmlns:pns="url_where_object_ontology_is_defined"
xmlns:tns="url_where_intention_is_defined" ... />
:
:
<types>
   <schema ...>
:
:
      <complexType name="interaction">
         <element                    name="service:Intention"
type="tns:#intention"/>
         <element name="service:Action" type = "anyType"/>
         <element     name="service:Object"     type     =
"pns:#object"/>
   </complexType>
   </schema>
</types>
:
:
<portType name="invokeEnactment">
   <operation name="enactment_operation" >
      <input message="invoking" type="interaction" />
   </operation>
</portType>
<service name="DieCastingService">
```

```
    <port                                    name="invokeEnactment"
binding="tns:conversationBinding">
        <soap:address location="URL_of_Company_D"/>
    </port>
</service>
```

Figure 15 A WSDL Declaration for Interactive Communication

CHAPTER 4 TRUSTWORTHY ENVIRONMENT FOR SERVICE-ORIENTED COLLABORATION

4.1 Motivations for Trustworthiness in Service Oriented Environment

Service-oriented computing is utilitzed by organizations to support business partnerships by linking their software systemns. [5-10]. A Service-Oriented Architecture (SOA) is a component model that inter-relates an application's different functional units viewed as serviced services through well-defined interfaces and contracts. This allows services that, are built on a variety of such systems to interact with each other in a uniform and universal manner. This concept of a service that is not strongly tied to a particular implementation is known as loose coupling. Loosely-coupled systems can provide business applications with agility based upon the needs of the business to adapt to its changing environment since an entire, global application can be built out of a variety of distributed, local implementations represented as services.

As expected, there are challenges that can limit the implementation of a true service-oriented collaboration. One of the most noticeable challenges is that of "trustworthiness". Trustworthiness is the reliability of a person to others because of their integrity, truthfulness and trustfulness, traits that can encourage someone to depend on them [130]. In a service oriented paradigm, trustworthiness can be defined as an ability to deliver service that can be justifiably trusted.

In order to develop systems that are trustworthy, four techniques are used: fault prevention, fault tolerance, fault removal, and fault forecasting [131]. Fault prevention techniques are intended to prevent faults from ever occurring by using tools and fault-

preventing methodologies such as structured programming and modularization in software development, design rules for hardware development, training for information shielding, and firewalls for malicious attacks from hackers. Fault tolerance involves techniques designed to preserve the delivery of correct service even in the presence of active faults. Fault tolerance techniques generally involve two steps to deliver correct services, error detection and subsequent error recovery. Through fault tolerance, fault handling may be required for faults to be reactivated. Fault handling follows four stages such as 1) fault diagnosis, 2) fault isolation, 3) system reconfiguration, and 4) system re-initialization. Fault removal is performed during the development stage and operational life of a system. System verification, validation, testing, and system updates fall into this category. Fault forecasting performs evaluations of the system behavior with respect to fault activation. Evaluation can be both qualitative and quantitative. Qualitative evaluation identifies, classifies, and ranks the failure modes. Quantitative evaluation computes the probabilities that some attributes of trustworthiness are satisfied.

In order to support a trustworthy service oriented environment, there have been several efforts to reduce the number of faults and enhance reliability. These approaches can be classified into two broad categories:

- Failures are anticipated and removed as much as possible at specification modeling time. Potential obstacles are identified through model checking. (Section 2.5 discusses this approach in a service oriented paradigm.)
- Failures are detected and resolved at runtime [15-18]. Resolution of trustworthiness violations is resolved on-the-fly by making "acceptable" changes that satisfy the high-

level goals underpinning the requirements being violated. Detecting violation at runtime sometimes may involve warning the users and reconfiguring the system.

Even though those solutions are valuable—perhaps even the best solution in many cases—they are limited in their ability to remove and resolve failure. These limitations are the result of three factors: 1) RPC based messaging, 2) distributed service control, and 3) restrictions of describing services.

Every time a failure is anticipated or happens, regulation of control and corresponding information exchange is required since the overall process execution path might need a modification to attain the final goal state. RPC-based invoke/return type communication on which current Web Services rely is not capable of providing the information required for data exchange and conflict resolution among partners. To resolve failures, current RPC-based communication would require additional interfaces or shared modules for fault removal and fault forecasting. However, having additional interfaces in the presence of the activation of a new failure may harm consistency of the Web Service implementation since interfaces needs to be updated whenever new fault factors are activated. Moreover, sharing fault handling modules between partners may break the philosophy of platform independency.

Furthermore, current Web Services cannot guarantee that service provider or requester will always perform flawlessly as expected. This is especially the case when the different agents are owned by different organizations and subject to varying policies and capabilities. As a result, traditional fault removal approaches using models and specifications cannot be directly applied in service oriented applications, since traditional

fault removal approaches work based on specifications in the development phase of a process and system cannot guarantee functional correctness and non-functional performance.

Moreover, Web Services are published in a service description language which is based on formal description logic. The name description logic refers to concept descriptions used to describe a domain and to the logic-based semantics which can be given by a translation into first-order predicate logic. Therefore, although service description can be used for qualitative evaluation such as effect analysis, it is hard to apply quantitative evaluation or probabilistic performance prediction to service correctness.

To address these limitations, we propose a trust model that aims for reliability prediction and performance evaluation used in service oriented environment. Our trust model utilizes reputations from the user community to extract the expected degree of service satisfaction and hence estimates risks to reach failure. We use Probabilistic Latent Semantic Indexing to compute the expected degree of reliability in services and estimate risks (the probability of a service failing or unsatisfied service delivery) based on our probabilistic model. Risks from our trust model are then combined with net based methods to analyze non-functional quantitative parameters and hence to improve system trustworthiness.

4.2 Predicting Risks from Recommendations

Our trust model is designed to predict service risks before a particular user is actively involved in a specific service. To predict risks, for each of the possible ratings for each of the services available, we first estimate the probability that users will give a particular

rating to a particular service. These predicted risks will be used during the service analysis stage for performance prediction. We compute an expected value for each user-service pair before actual usage experience for that pair is known. Collaborators in the WSCPC environment can use the expected value during the service selection stage. The domains we consider consists of a set of users $U = \{u_1, u_2, ..., u_n\}$, a set of services $S = \{s_1, s_2, ..., s_m\}$, and a set of ratings $R \in \aleph$, the set of natural numbers. We assume that observations are available for user-service pair (u, s), and that users made explicit ratings V as a part of an observation.

4.2.1 Models for Expected Service Ratings

The first step of our risk prediction is to compute the expected numerical ratings of (u, v) before the actual ratings are known. To this end, we use a technique known as a probabilistic Latent Semantic Analysis (pLSA) [132, 133]. This is a special case of a collaborative filtering technique with implicit preference data.

The key idea of our prediction model is the introduction of an hypothesis space Z consisting of hidden variables $z \in Z$, so that users and services are set conditionally dependent. The possible size of the space Z is assumed to be finite and of order k. In our trust model, the hypothesis space Z is considered as a space for service evaluation pattern considering non-functional aspects, and has the size of k features. For a given space, each co-occurrence observation triplet (s_i, u_i, r_i) is associated with the factor $z_k \in Z$. From the viewpoint of the pLSA model, it can be inferred that there are existing different relationships among <user, service> pairs related to different factors, which can be considered to represent users' service evaluation patterns.

Consider first the following probability definitions.

- Let $P(r_l \mid s_j, u_i)$ denote the conditional probability distribution of ratings r_l over the condition that a service $s_j \in S$ is evaluated by a user $u_i \in U$.

- Let $P(r_l \mid s_j, z_k)$ denote the class-conditional probability distribution of ratings r_l over the joint condition of the latent variable $z_k \in Z$, and service $s_j \in S$.

- Let $P(z_k \mid u_i)$ denote a user specific probability distribution on the unobserved class factor $z_k \in Z$.

Then, the dependency structure of the co-occurrence model with additional variable z is formally given by

$$P(r_l \mid s_j, u_i) = \sum_{z \in Z} P(r_l \mid z, s_j) P(z \mid u_i). \tag{4.1}$$

In addition to specifying the dependency structure, we define a parametric form for conditional probability density $p(r \mid s, z)$. To this end, we use a Gaussian model for $p(r, s \mid z)$. Thus, we introduce a location parameter $\mu_{s,z} \in \Re$ for the mean rating and scale parameter $\sigma_{s,z} \in \Re$ for the spread of ratings. This defines a Gaussian mixture model given by

$$P(r \mid u, s) = \sum_{z \in Z} P(z \mid u) p(r; u_{s,z} \sigma_{s,z}), \tag{4.2}$$

where $p(r,\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left[-\frac{(v-\mu)^2}{2\sigma^2}\right]$

Finally, the expected ratings can be calculated as

$$E[r \mid u,s] = \int_R rP(r \mid u,s)dv = \sum_Z P(z \mid u)\mu_{s,z} \,. \tag{4.3}$$

In addition, the percentage r that a service s will be equal or less than c is computed as

$$P(rating(u,s) < c) = \int_R^c P(r \mid u,s)dv \approx \sum_Z P(z \mid u) \sum_R^{r=c} p(r;\mu,\sigma) \tag{4.4}$$

$$= \sum_Z \sum_r^{r=c} P(z \mid u)p(r;\mu,\sigma).$$

4.2.2 Expectation Maximization Algorithm

In our trust model, we adopt a pLSA model to model the relationship among users, services, and service ratings. Using the definition from Section 4.2.1, we define the probability of an observed pair (u_i, s_j, η) by adapting the latent factor.

Following the likelihood principle, the total likelihood L can be determined by

$$L = \prod_{<u,s,r>:} (P(r \mid s,z)P(z \mid u))^{p(z|u,s)}. \tag{4.5}$$

In order to maximize the total likelihood, we make use of the Expectation Maximization (EM) algorithm to perform a maximum likelihood estimation in the latent variable model.

The Expected Maximization algorithm is a standard technique for statistical inference that can be used to maximize log-likelihood [134-136]. Generally, two steps are needed to implement this algorithm. First, in the Expectation step (E-step) one calculates posterior probabilities for the latent factors based on the current estimates of conditional probability. Second, in the Maximization step (M-step) one updates the estimated conditional probabilities and uses them to maximize the likelihood based on the posterior probabilities computed in the E-step. The first task in deriving an EM algorithm is to specify a complete data model. A complete data model treats the hidden variables as if they were actually observed.

In the E-step, we can simply apply Bayes' formula to compute the posterior probabilities based on ratings observations triplet $<u, s, r>$. Since we do not know the states of the latent variable z, we introduce a posterior probability

$$p(z \mid u, s, r), \text{ such that } \sum_{z} p(z \mid u, s, r) = 1,$$

which needs to be computed for triplets $<u, s, r>$ that have actually been observed.

$p(z \mid u, s)$ can be computed using Lagrange multipliers. To this end, we derive a new likelihood function L' given by

$$L' = - \prod_{z \in Z} P(z \mid u, s, r)^{P(z|u)P(r|z,s)}.$$

Using Lagrange multipliers, we obtain

$$OBJ = \log L' = - \sum_{z \in Z} P(z|u)P(r|z,s)\log(z|u,s,r) \; + \; \lambda (\sum_{z \in Z} P(z|u,s,r) - 1).$$

At its maximum, the derivative of the function *OBJ* is zero. Thus, we have

$$\frac{\partial OBJ}{\partial p(z'|u,s,r)} = -\frac{P(z'|u)P(r|z',s)}{p(z'|u,s,r)} + \lambda \; = 0, \text{ where } z' \; \in \; Z$$

from which it follows that

$$p(z|u,s,r) \; = \; \frac{P(r|z',s)P(z'|u)}{\lambda}.$$

Next we need to compute λ in order to compute $p(z|u,s,r)$.

Again, from

$$\frac{P(z'|u)P(r|z',s)}{p(z'|u,s,r)} - \lambda \; = 0$$

it follows that

$$P(r|z,s)P(z|u) \; = \lambda \, p(z'|u,s,r)$$

so that

$$\sum_{z \in Z} P(r|z,s)P(z|u) \; = \; \lambda \sum_{z \in Z} P(z|u,s,r).$$

However, since

$$\sum_{z \in Z} P(z|u,s,r) = 1$$

it follows directly that

$$\sum_{z \in Z} P(r|z,s)P(z|u) = \lambda .$$

Hence, $p(z'|u,s,r) = \dfrac{P(r \mid z',s)P(z' \mid u)}{\sum_{z \in Z} P(r|z,s)P(z|u)}$. (4.6)

We can also obtain the parameter $P(z|u)$ for the M steps using Lagrange multipler addition in the log likelihood function L'' maximizing the sum of log $P(u,s,r)$. L'' is defined as follows.

$$L'' = log\ L = \sum_{<u,s,r>:u \in U, s \in S, r \in R} \log P(u,s,r)$$

$$= \sum_{<u,s,r,>:u \in U, s \in S, r \in R} (\log P(r \mid s,z)P(z \mid u)) .$$

Using (4.1) and the fact that $\sum_{z \in Z} P(z \mid u,s,r) = 1$, it follows that

$$L'' = \sum_{<u,s,r,>:u \in U, s \in S, r \in R} \sum_{z \in Z} P(z \mid u,s,r) \log(P(r \mid s,z) + P(z \mid u))$$

$$= \sum_{<u,s,r,>:u \in U, s \in S, r \in R} \log(P(r \mid s,z) + P(z \mid u)) .$$

where $log\ L$ is maximized w.r.t. parameters $P(z|u)$ subject to the constraints $\sum_{z \in Z} P(z \mid u) = 1$ and $\sum_{z \in Z} P(r \mid s,z) = 1$.

From (4.6), we obtain the for the parameters of the Gaussian distribution given by

$$P(z \mid u) = \frac{\sum_{<u,s,r>} P(z \mid u,s,r)}{\sum_{z \in Z} \sum_{<u,s,r>} P(z \mid u,s,r)} \tag{4.7}$$

and

$$P(r \mid s,z) = \frac{\sum_{<u,s,r>} P(z \mid u,s,r)}{\sum_{z \in Z} \sum_{<u,s,r>} P(z \mid u,s,r)}$$

from which it follows that

$$\mu_{r,z} = \frac{\sum_{<u,s,r>:u=u'} rP(z \mid u',s,r)}{\sum_{z \in Z} \sum_{<u,s,r>:u=u'} P(z \mid u',s,r)} \tag{4.8}$$

and

$$\sigma^2{}_{r,s} = \frac{\sum_{<u,s,r>:u=u'} (r-\mu^2_{r,z}) P(z \mid u',s,r)}{\sum_{z \in Z} \sum_{<u,s,r>:u=u'} P(z \mid u',s,r)}.$$

Iterating over (4.6), (4.7), and (4.8) one can reach the maximal value of $P(z|u,s,r)$, $P(z|u)$, and $u_{r,z}$. Hence, we can compute both the expected value $E(r|u,s)$ and the failure probability $P(r|u,s)$.

4.2.3 Service Selection and Estimated Risks

Service selection steps involve utilizing expectation values and risk probabilities obtained from (4.3) and (4.4). Service consumers may use service rating expectations $E(r|u,s)$ and service failure probabilities to select a service. Service failure probabilities can be obtained by setting a threshold t, which specifies that ratings less than t are considered as failure. In other words, if a rating is scored less than the threshold, the service usage is considered a failure.

In order to evaluate our probabilistic model, we first run our model on MovieLens [137]. MovieLens is a dataset that contains 100,000 ratings by 943 users on 1,682 movies. The ratings were collected by the MovieLens web site from September 1997 to April 1998. All of the users in this data set rated at least 20 movies.

To simplify the experiments, we used five base-test set split data instead of full data set. The split data consists of 80% of base data and 20 % of test data. Each data set has disjoint data set. We learned probabilities using base set, and estimate probabilities for movies in test rating set. Difference is measured through comparison of ratings between our expectation and ratings in test set. This results are from setting $k = 30$. Increasing k improves the accuracy but the requires more memory size, and at $k > 30$ we cannot conduct experiments due to limitation of system memory size. All the experiments are conducted on a PC with Intel Pentium IV 2.8 GHz CPU and 1GB of RAM.

Then we compared our results with averaging methods (AVG). Averaging methods are a method that computes each movie's average ratings from test set and measures difference between the average ratings of a particular movie and movie ratings in test data set. In

addition, we also compare root square of differences between average ratings and user ratings. The resulting gain of pLSA is around 5.1% in terms of average and 11.2% of square root differences, and .

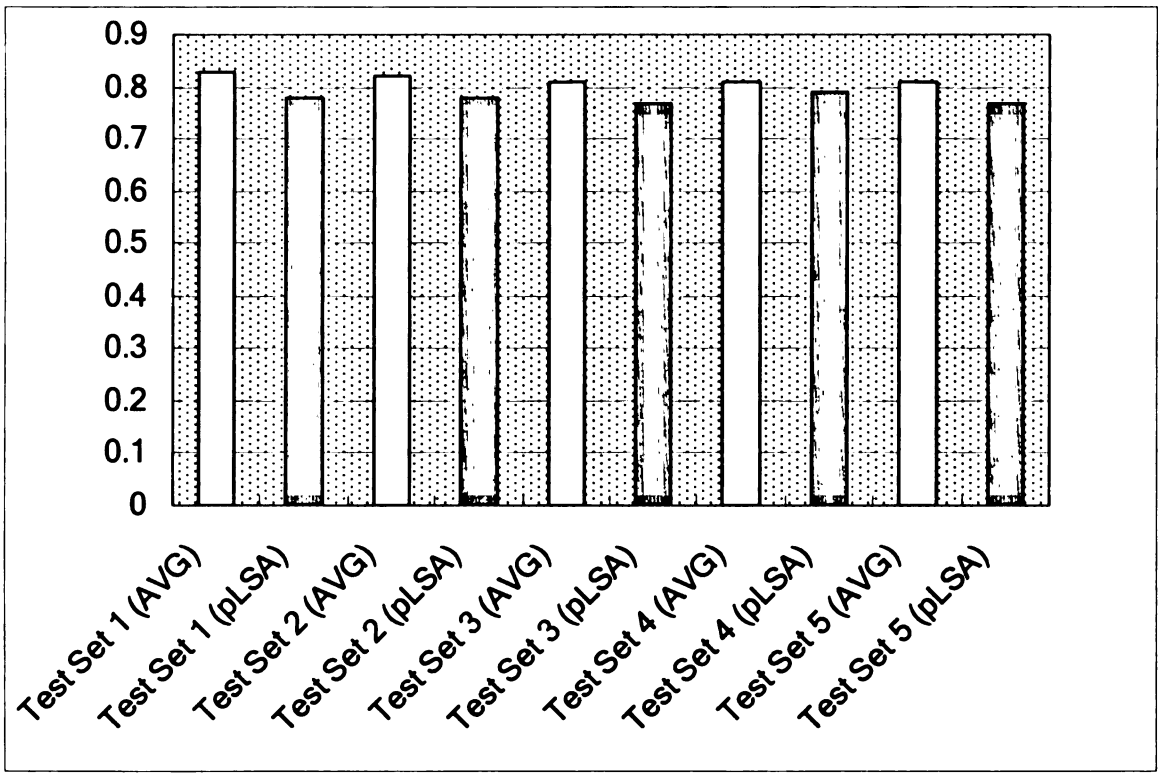


Figure 16 Performance Comparisons (Average Prediction Errors)

The results are shown in Figure 16 and Figure 17. To evaluate our method, we define a prediction to be acceptable if the difference between our prediction and the actual rating is less than or equal to 1.0. That is, we assume that our prediction is acceptable if it is within a certain range, here between 0.0 and 1.0). The comparison is shown in Figure 18. pLSA also improves the acceptability rate compared to the averaging method.

4.3 Analysis of Service Correctness

Service correctness in a service-oriented environment focuses on detecting either malicious or faulty behavior of collaborative services. Correctness can be classified into two categories, functional and non-functional behavior. Functional behavior describes

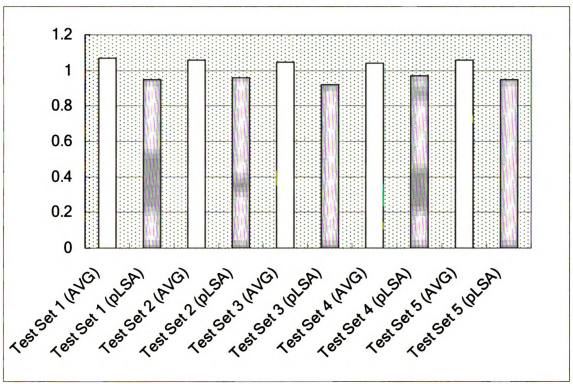


Figure 17 Performance Comparisons (Root Square Prediction Errors)

what task the service accomplishes whereas non-functional behaviors describe how the service accomplishes its task. Functional behavior is related to specifications published by the service provider, while non-functional behavior includes QoS properties such as timeliness, accuracy, and cost. Peers can see the functional correctness by examining equality between the service description and execution status by checking possible

violation of constraints. Non-functional properties can be verified by checking performance factors regularly reported from service providers.

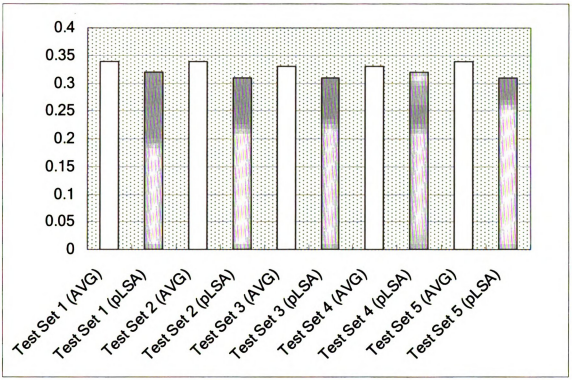


Figure 18 Performance Comparison (Non-Acceptability Rate)

4.3.1 Quantitative Analysis for Non-Functional Behavior

In this thesis, we propose a methodology for analysis of non-functional behavior of collaborative services. To this end, we adopt a net-based quantitative analysis technique in a service oriented environment. Quantitative analysis enables one to trace, analyze and predict quantitative information, by examining its numerical, measurable characteristics like performance factors. Therefore non-functional behavior can be examined through

quantitative analysis. The target of quantitative analysis in service oriented environment would include, for example, the following.

- The expected probability that a service is active.
- The expected value of the throughput of a service execution.
- The expected probability that a service is available.
- The expected probability that a service is invoked.

Current analysis methodologies for functional and non-functional behaviors have limited usage since the analysis can only be carried out by checking violation of pre-determined constraints [15-18, 138-140]. It is, however, hard to predict non-functional behavior before service failure happens using current analysis methodology.

It is therefore imperative to investigate new methodologies. In the proposed methodology, both functional and non-functional behaviors can be analyzed through the query-based dialogue between partners. The functional correctness can be checked through the answers from service providers. Malicious behavior or cheating can also be detected by applying several agent-based methodologies already proposed [141, 142].

In WSCPC, the quantitative analysis is conducted based on the transformation from our Process Grammar model to stochastic Petri nets with timing and stochastic extensions. Petri nets are a widely accepted formalism for modeling and analysis of distributed systems [143]. Although there are also other methods for quantitative analysis like queuing networks [144] and stochastic process algebra [145]), Petri nets are still

considered to be the most useful in terms of efficiency and the number of available existing tools [146]. Moreover, our Process Grammar is easily transformed due to its pre-conditions, post-conditions, and bi-partite graphical structure.

Accordingly, we use stochastic Petri nets for non-functional behavior analysis support in the WSCPC framework. Stochastic Petri nets generalize classical Petri nets by adding rewards and by assigning guards and distributions of firing time to transitions [145]. By definition, stochastic a Petri net is a 10-tuple consisting of: a finite set of places, a finite set of transitions, a finite set of in-arcs, a finite set of out-arcs, an integer weight for every arc, a guarding function for every transition, an initial marking, a distribution of the firing time for every transition, a priority relation such as transitive among the transition, and a finite set of measures. A stochastic Petri net transition is enabled if and only if i) the guard is true, ii) no other transition has higher priority, and iii) at every place there are no fewer tokens in a given marking than the weight of the in-arc from the place to the transition. When a transition fires, the number of tokens increases by the number of in-arc weights and decreases by the number of out-arcs weights.

The validation of quantitative properties in WSCPC is based on quantitative net models. Stochastic Petri nets are used to model and predict the probability distribution of service behaviors. The quantitative properties are provided by communications between partners, using the WSCPC service interaction model. Queries and answers should be designed based on the properties to be validated.

4.3.2 A Net Based Method to Predict Non-Functional Behavior

The quantitative parameters for non-functional behavior are classified as performance parameters and reliability parameters defined as follows..

- *Performance parameter* is the (average) time or cost to execute a given set of services. (See Figure 19 $t_{i,j}^{n}$.)

- *Reliability parameter* is the probability of entering and executing successfully a given service. (See Figure 19 $p_{i,j}^{n}$.)

The execution time or cost can be measured by monitoring services. The monitoring process is explained below in Section 5.4. The probability of unsuccessful services depends on the service consumer's criterion. A service consumer may define a service as a success whenever they get output regardless of the quality or whenever they experience a particular level of flaws from the service. The probability can be assessed from past experience of services such as testing the same services several times or having a history of service usage.

Figure 19 illustrates a net structure evaluated by the WSCPC framework. Oval objects connected to a rectangle object represent *n* service candidates available from an input specification object *i*. The notation $p_{i,j}^{n}$ over the dashed arrow indicates the probability that the service *n* can be executed successfully. Hence, $p_{i,j}^{n}$ represents service

reliability. The notation $t_{i,j}^{n}$ represents performance parameters such as cost, execution time, etc. Thus, $t_{i,j}^{n}$ represents service performance.

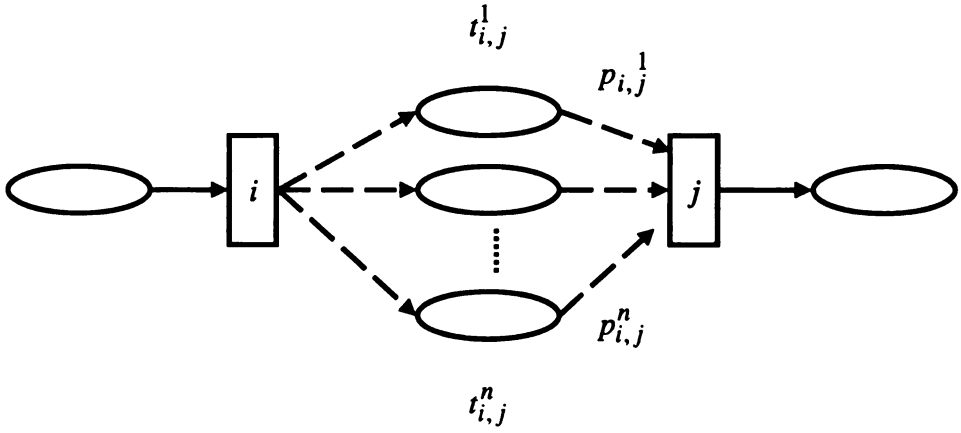


Figure 19 Quantitative Parameters in Net Based Model

Provided there is no information about the probability distribution of any inputs, then the assumption of our model is that the probability of successful service execution is determined by the expected ratings. In our trust model, a service consumer can set a threshold as a required minimum level of service quality. If the threshold is set to $n \in \aleph$, then, to be successful, the service consumer's expected rating should be more than n. If the threshold is not set, then the service is regarded as successful as long as service itself does not fail in its operation. How to obtain such probability is explained in Section 4.2.

Obtaining $t_{i,j}^{n}$ is more complex. Assuming that a service provider posts their particular service performance using a parameters T, we wish to compute the delay of service

performance. The more reliable a service is, the less the service delay would be. We model service performance by

$$t_{i,j}^{n} = T + delay = T + T \cdot \frac{\lambda e^{-\lambda(E(r|s,u))}}{K},$$

where $\lambda = \frac{1}{\sigma_{r,s}}$, C = threshold, K is constant, $E(r \mid s,u)$ is an expected service rating from a user u for a service s, and $\sigma_{r,s}$ is the standard deviation for the probability distribution of ratings r on a service s (see Section 4.2.1.)

From the above equation, *delay* increases exponentially as the service expected rating decreases. (See Figure 20.) For example, if the expectation is less than the threshold, the the service delay will increase considerably so that the service will be deemeded as barely useful. The basic consideration of the above equation is to maximize the expected *delay* on a service whose expectation is mediocre. Moreover, we also note that very minor factors (even if not related to non-functional behavior) can affect evaluation on high-quality service. Here, λ represents rate parameters of the exponential distribution. The rate parameters become large when σ is small. That means that services whose σ is small are assumed to have small degrees of performance variance when their service qualities are expected as high or mediocre.

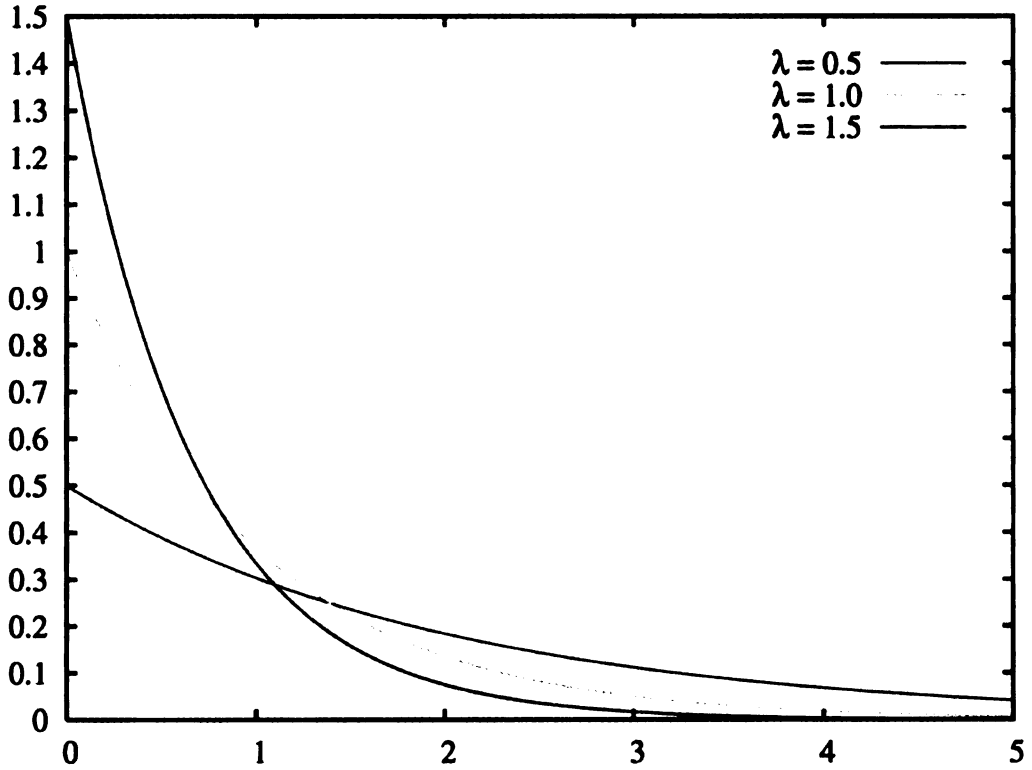
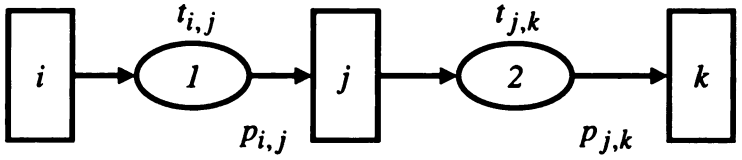


Figure 20 Exponential Distribution

Service oriented computing offers an aggregated service using a composite service. Services in a composite service are structured with parallel and serial connected services. To predict non-functional behavior in a composite service, we use the following equation shown in Figure 21. Similarly we can estimate the performance of a sub-component in a service, assuming that reliability and performance values of each sub-component are evenly distributed.

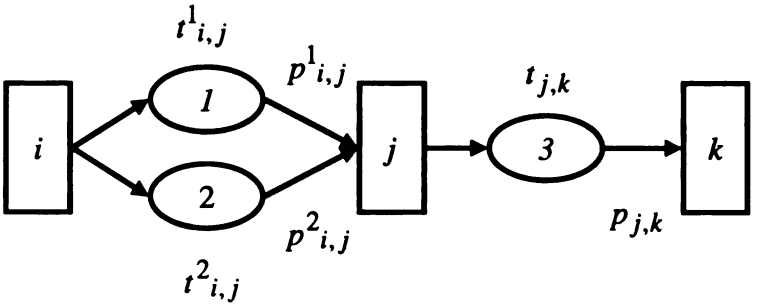
Serial Connection



$$p_{i,j,k} = P(2|1)P(1) = p_{i,j}p_{j,k}$$

$$t_{i,j,k} = t_{i,j} + t_{j,k}$$

Parallel Connection



$$p_{i,k} = P(3|1,2)P(1,2) = p^1_{i,j}\ p^2_{i,j}p_{j,k}$$

$$t_{i,j} = \max(t^1_{i,j},\ t^2_{i,j}) + t_{j,k}$$

Figure 21 Non-functional Behavior in a Composite Service

CHAPTER 5 IMPLEMENTATION

5.1 The WSCPC General Architecture

The Web Service Collaborative Process Coordinator (WSCPC) is a collaborative engineering framework built on Java and Apache Axis. Figure 22 illustrates the WSCPC general architecture. WSCPC utilizes the semantics of Process Grammar and OWL-S for process specification and manipulation. Hence, WSCPC is capable of managing Design and Manufacturing (D&M) processes that are distributed over enterprises. WSCPC consists of the following basic components [112]:

- Process Engine (PE): The Process Engine's primary purpose is to create, manage, and enact a process instance. The Process Engine can load up a process definition from two different sources, a user's private repository and the publicly accessible process library. The Process Engine instantiates the process after it loads up the definition. Once instantiated, a process is managed and executed by the Process Engine. The Process Engine handles the process execution and maintains the current status of execution. It has two modules inside. One is the Communication Server and the other is the Process Controller. The Communication Server handles all of the message creation and reception. The messages through service invocation are passed into the Communication Server and then processed. The Communication Server delivers a user's request to the Process Controller so that it can query services in the Service Registry for the logical task. Matched results are first filtered out by the Process Controller if there are too many available services and then sent to the Cockpit waiting for user's decision. Figure 23 illustrates steps in finding suitable

services for a particular logical task. After the user choses a particular service for outsourcing, the Web Service Module invokes the corresponding Web

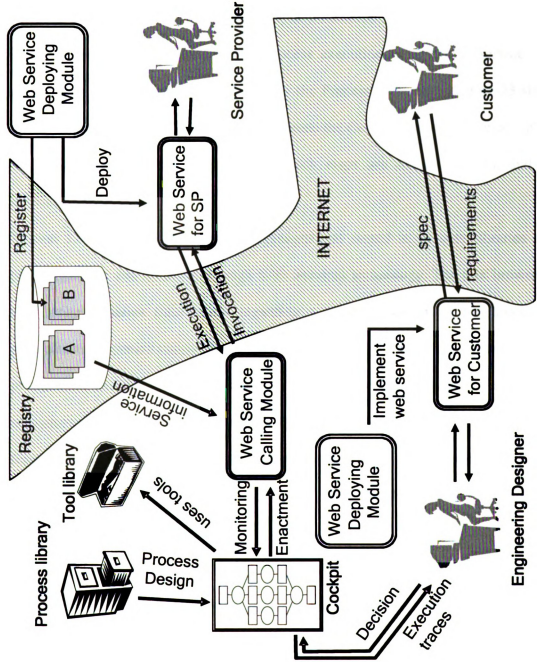


Figure 22 WSCPC General Architecture

Service. The Communication Server maintains a message queue itself. All the messages within a local group are placed in the message queue and then sent to the message receivers. By maintaining the message queue, all of the events of a process are ordered such that systems guarantee synchronization in collaborative workspace. The Process Controller manages the process execution of a logical service. The atomic task execution is also processed by the Process Controller. Figure 23 shows how services are searched and invoked between the Cockpit and other components in the WSCPC framework. The service search steps are marked as (1), and the invocation steps as (2).

- Process Library: Process design is organized and stored in process libraries as a production rule and accessed through Web Services technology. The task browser in the Cockpit queries information regarding a task and organizes the tasks in a generalization-specialization hierarchy. The Process Library can show all the productions available for each logical task. The Process Library is a kind of repository that holds and distributes process flow definitions. The WSCPC framework uses two kinds of process libraries, the JAVA RMI server-based library and the Web Service-based library. The JAVA RMI server-based library is implemented without a common interface. The access to the library must be done only via a JAVA RMI client. On the other hand, the Web Service-based library can be used for collaborative workflow management among heterogeneous systems. As one might expect, the Web Service-based library provides a globally acceptable common interface. Furthermore, the process definition will be distributed in an OWL document in order to achieve interoperability.

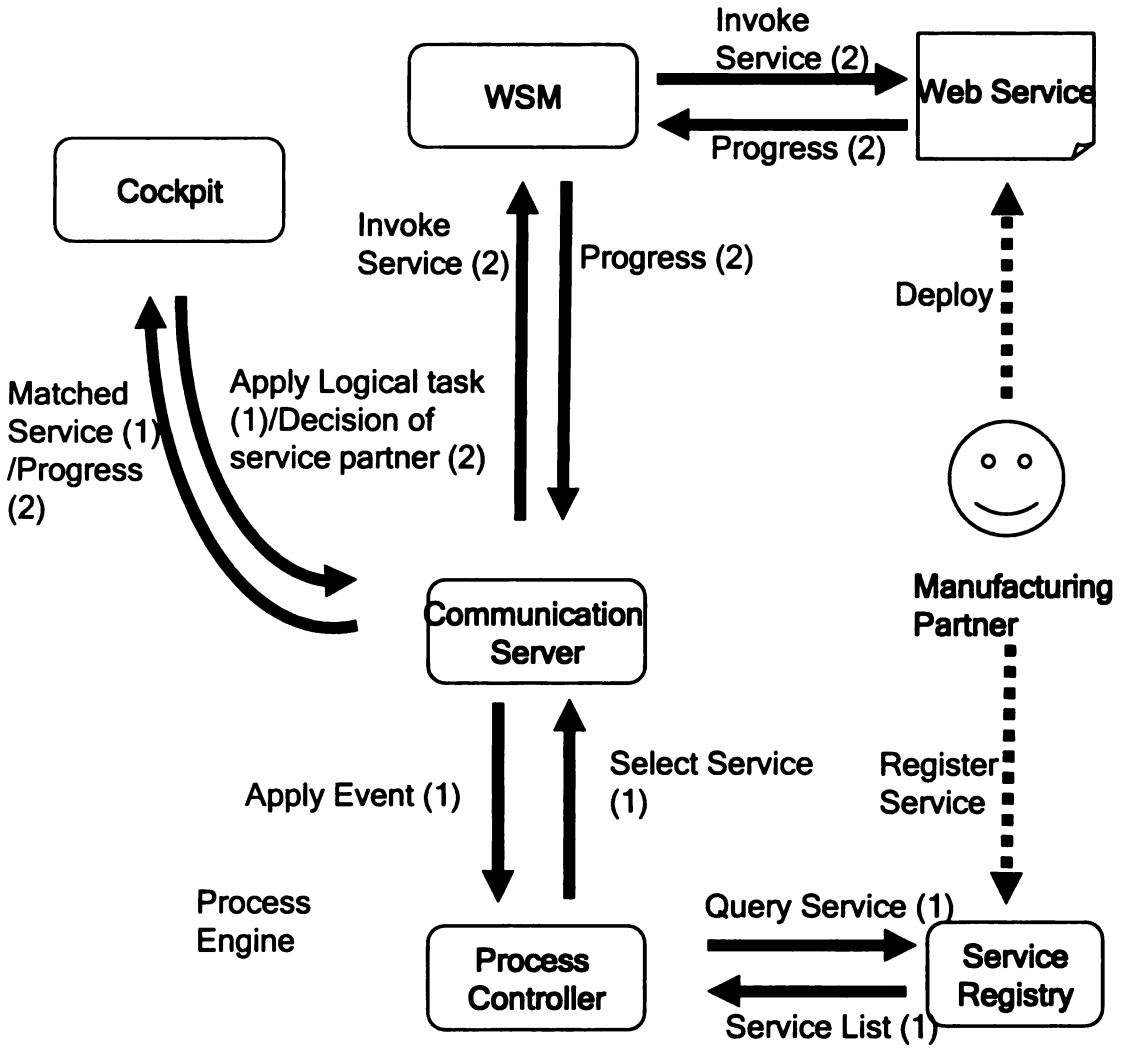


Figure 23 Service Selection and Invocation

- Service Registry: The Service Registry stores the descriptions of Web Services that specify the design and manufacturing capabilities, represented in the Process Definition Module (PDM). The service registry includes the type of a service, input specification, output specification, pre- and post condition, and a link to service evaluation results. Each collaborating organization has its own copy of each of the

components except for the Service Registry, which is shared among all of the organizations.

- Cockpit: The Cockpit is a communication interface connecting a user to the Process Engine. It couples a user and an engine by transmitting the user's decision on process creation and enactment to the Process Engine. The Cockpit also provides graphical information about the process definition and enactment to the user. The Cockpit interacts with the human user when creating process definitions, displaying and maintaining process information archivea, and when playing enactment sequences of a process. When a user defines a process, the Cockpit provides a graphical authoring environment. A user can actually draw a graph of a process flow, which the Cockpit has the ability to document in OWL. The Cockpit has been implemented as a downloadable JAVA applet. Since the Cockpit must connect to the JAVA RMI server, which runs WSCPC process enactment engine, the Cockpit is a JAVA RMI client as well.
- Web Service Module (WSM): The Web Service Module connects the Process Engine with Web Services, so the Process Engine can locate the suitable services for certain task through the Web Service Module and invoke those services. The Web Service Module includes OWL an inference engine, which captures user's actions and generates messages using the interaction model. The Web Service Module also polls the invoked service,and captures runtime service execution through the Process Monitoring Model (PMM). The Web Service Module includes two modules, the Web Service Calling Module and the Web Service Deploying Module. The service provider, after making the semantic description of a service (OWL), deploys the

service on its Web Service server and registers the service into the Service Registry using the Web Service Deploying Module. WSCPC searches and calls the suitable service using the Web Service Calling Module inside the Web Service Module.

5.2 WSCPC Execution Environment

The Web Service Module is composed of four components: the OWL-to-service implementation tool, the Web Service deployment tool, the Web Service registry tool, and the service requester [115]. Figure 24 illustrates the architecture of the Web Service Module in detail. The components used in Web Service Module are as follows.

- The *OWL-to-service implementation tool* encodes the semantic description into the service implementation. It has five operations. The operation *provideInput*() takes the input specifications from the service user and delivers it to the Web Service. The operation *invokeEnacting*() invokes the enactment of Web Services. The operation *getOutput*() delivers output specifications to the service user. The operation *getGraph*() delivers the graphical information of a workflow enacted to the service user. Finally, the operation *getServiceComposite*() delivers the *ServicComposite*() in the service provider's local library to the service user.

- The *Web Service deployment tool* compiles the service implementation and deploys it onto the Web Service server.

- The *Web Service register tool* reads WSDL documents and semantic model descriptions from the service provider's workspace and enrolls the service to the registry.

- The *Service requester* discovers and executes Web Services by using the Web Service Calling Module. It consists of the following components. The *service discovering tool* discovers a Web Service by sending a query to the service registry. The *semantic reader* helps the service user with browsing the service semantics. The *WSDL reader* enables the service caller to understand documents written with WSDL and stored in service registry. Finally, the *service caller* calls one of the operations of a chosen Web Service.

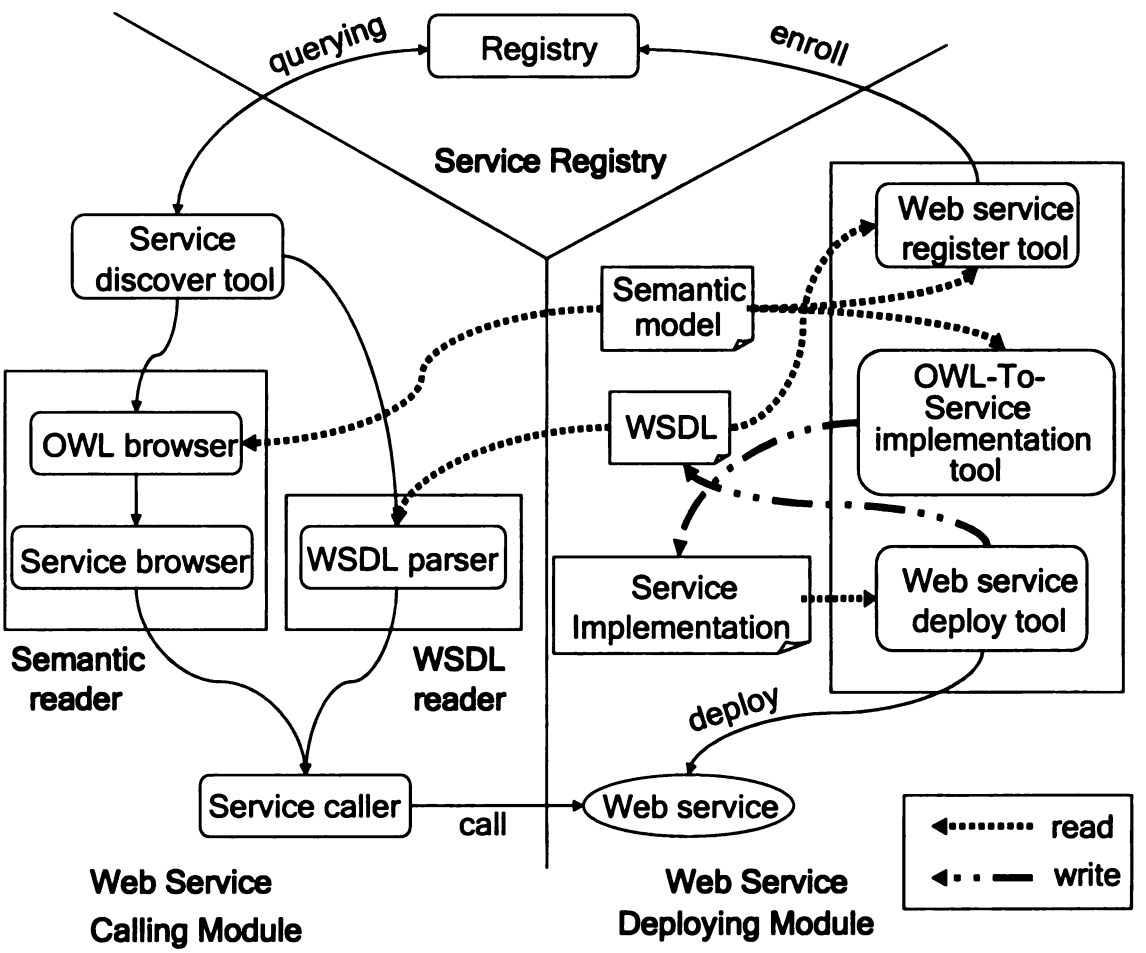


Figure 24 Web Service Module

5.3 Authoring Environment

The primary goal of the WSCPC authoring environment is to provide a process flow authoring facility to a user. In the WSCPC authoring environment, a user can create a process definition by authoring logical tasks and atomic tasks, and then combining those tasks into a process flow. Once a process definition has been created, the process definition is stored into the private repository or public library.

The Cockpit is the major component that is responsible for the authoring process. The Cockpit provides a user with a sophisticated authoring tool that includes a graphical interface, which enables a user to create a process flow graph and convert the process flow graph into a process definition document written in OWL.

The WSCPC authoring environment has the ability to deploy and publish a Web Service to support the Web Service-oriented interoperability [112]. The WSCPC Web Service Registering Module and Web Service Deploying Module together realize a Web Service-oriented interoperability. The Web Service Deploying Module generates an implementation of a Web Service based on the process definition. If the authoring environment needs to advertise its Web Services, the Web Service Registering Module registers Web Services to the public service registry.

Web Service registration is only necessary when inter-organizational collaboration proceeds since the WSCPC framework assumes that every participant within one organization know one another very well so that it is not necessary for them to advertise any Web Services or tasks. However, during the collaboration between different organizations, both organizations need a rendezvous point to meet with one another. So

in this case, the WSCPC authoring environment registers Web Services as a process flow or a tool at the Service Registry.

5.3.1 Defining a Process

WSCPC provides users with a GUI-based process defining tool. Users can first draw service definitions on the Cockpit and then convert it to an OWL-based process definition using the Web Service Module. The final step is deploying the service definition into the registry so that partners can locate the functionality that the service definition provides.

The WSCPC authoring environment supports two patterns of process definition in association with a Web Service. Users can create a process definition either for an atomic service or for a logical service and its service composites. These separate patterns involve two different types of Web Services, either an atomic service in which Web Service based software tools are invoked or a logical service in which a process is assigned or a process library is used.

As discussed above in Section 3.2, an atomic task—an atomic service in terms of the service-oriented model–is responsible for invoking a software tool. Hence, the first step is to create an atomic task in the Cockpit. Through the Cockpit, a user creates an atomic task by adding input data and output data specifications. After that, the atomic task must be bound by an actual tool application. Once the atomic task has been created, the Cockpit writes out the OWL document into temporary storage.

In addition to an atomic service, a logical service performs an important role in Web Service oriented collaborative workflow management. A logical service is to be assigned

for process enactment or is to serve as a public process library function. In either case, a logical service is designed to serve one or more service composites to the requester. So, the process definition should be placed where the Web Service can reach it and get its definitions.

The process definition for an atomic service is rather simple since it specifies only a single tool application. By contrast, process definition for a logical service is not that simple since one must specify service composites and its components. The Cockpit helps users complete this complicated job easily and quickly. Before being combined into a process graph, the user must prepare all of the tasks. Once the process flow graph is ready, the Cockpit writes out a process definition in OWL and stores it in the repository.

5.3.2 Deploying Web Service

In the WSCPC framework, the deployment of a Web Service is accomplished by a combination of the Web Service Deploying Module with the Tomcat-AXIS server system. The Web Service Deploying Module prepares the Web Service implementation while the Tomcat-AXIS server actually compiles the implementation and deploys it as a publicly accessible Web Service on the Internet. Figure 25 illustrates the process for Web Service creation and deployment.

The Web Service Deploying Module performs three steps to prepare a Web Service deployment: 1) it reads a process definition; 2) it generates the implementation of the Web Service from the definition; and 3) it copies the implementation of the Web Service into a specialized directory of AXIS. Throughout Steps 1 and 2, the Web Service Deploying Module uses the OWL parser to extract semantic information about the

process from the OWL document. Based on the semantic information, the Web Service Deploying Module generates a Java Web Service (jws) file automatically. At Step 3, the Web Service Deploying Module copies the Java Web Service = file into a specialized directory of AXIS.

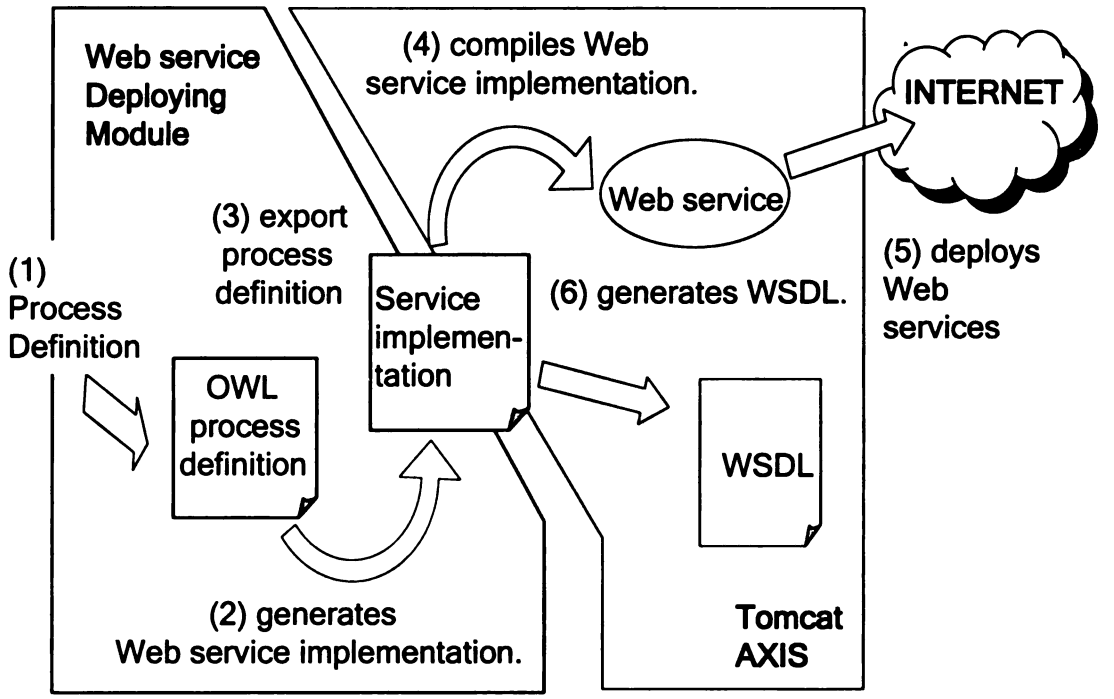


Figure 25 Deploying Web Service

As discussed in Section 3.3.4, the remote procedure that enrolls the service has been implemented as part of the Service Registry. The Web Service Registering Module interacts with that remote procedure of the Service Registry to enroll its service. As a result of the remote procedure call, a new entry is registered in the registry.

5.4 WSCPC Interactions in Collaboration

5.4.1 Frame Based Messaging

As introduced in Section 3.4, WSCPC can support flexible message formats and hence support complex communicative behaviors. In order to support automated interactions between partners, the WSCPC interaction mechanism employs a frame-based approach in order to exchange meaningful messages [147, 148].

A frame is defined as a data structure that represents a typical situation. A frame forms the semantics of a concept. A frame consists of properties and the contents of each property. A frame may have system-defined or user-defined properties. In WSCPC, service models such as the Process Definition Model, the Process Enactment Model, and the Process Monitor Model are represented as system-defined properties. User-defined properties describe a specific feature for some concepts. For example, non-functional parameters regarding QoS factors are represented as user-defined properties.

Formally a frame can be defined via a λ-expression.

$$\lambda\ x(\lambda\ y_0\ \ldots\ y_{n-1}\ ((x,\ z_1)\ y_1\ (x,\ z_2)\ \ldots\ y_{n-1}\ (x,\ z_{n-1}))), \tag{5.1}$$

where x is the frame name, y_i, $0 < i \leq n$ are the property names, z_i, $0 < i \leq$ n are y_i's contents bounded with types, and where n is the number of properties.

In WSCPC, a frame name is interpreted as a service name. Properties are services' functional or non-functional process flow properties such as *composeOf*, *hasStates*, and user-defined *QoS*. The frame name and its related properties are queried from the service

registry which stores the service provider's semantic description. Some properties may be acquired based on user-to-user agreements.

With a frame-based approach, an instance of WSCPC services can be defined with a lambda expression as follows.

$$\exists\ x\ (service\ (x) \rightarrow \exists\ y_0\ y_1\ \ldots\ y_{n-1}\ (property_0\ (x,\ y_0) \wedge property_1\ (x,\ y_1) \wedge \ldots \wedge property_{n-1}\ (x,\ y_{n-1}))), \quad (5.2)$$

where $property_i$, $0 \leq i < n$ is either a property defined in a WSCPC service model or user-defined non-functional parameters.

Consider a process in Figure 4 (b). A task *VacuumAssistedCasting* is created with a λ expression as follows.

$$\lambda\ x\ (\ service(x) \rightarrow \lambda\ y_0\ \ldots\ y_{n-1}(\ y_0\ (\ x,\ z_0\ )\ y_1\ (x,\ z_1\ )\ \ldots\ y_{n-1}\ (x,\ z_{n-1})))$$

(composedOf hasStatus providedBy ... processingTime ...) VacuumAssistedCasting

where z_i represents a content for a corresponding frame property and should be associated with certain type. For example, z_0 should be typed as a list since the *composedOf* property shows how a service can be decomposed. The property *processingTime* must be typed as numeric type. These types should be predefined and stored in public registry so that no erroneous typing happens in communication. Finally the above expression is reduced as follows.

composedOf (*VacuumAssistedCasting* z_0 : ***type of String list***) *hasStatus* (*VacuumAssistedCasting* z_1 : ***type of states***) *providedBy* (*VacuumAssistedCasting* z_2: ***type of String***) ... *processingTime* (*VacuumAssistedCasting* z_i : ***type of Integer***),

where the contents z_i are filled with the following contents.

z_0 = (*VacuumCasting Trimming QualityAssurance SelectedMaterial Dies TrimDies CastedProduct TrimmedProduct FinishedProduct*): *type of String list*

z_1 = *initial : type of states* (*states* are defined in PMM. see section *0*)

z_2 = *"John's. Casting Inc.": type of String*

...

z_i = *null*

...

Some contents are defined and published before a service is deployed. On the other hand, some contents have a *null* value since they will be set during run-time.

In a collaborative environment, it is necessary either to obtain behavioral semantics from a list of service providing candidates before assigning tasks or to know current execution traces after assigning tasks. Execution traces may be either functional behaviors such as service composition, service assignment, and bounded data or they may be non-functional behaviors such as processing time and operational costs. To this end, a service

consumer may need to generate automated querying and answering mechanisms to monitor execution traces.

To create messages that can obtain behavioral semantics and execution traces from partners, the query for the above example can be formally represented for (5.1) as follows.

$\lambda\ x\ (\lambda\ y_0\ \ldots\ y_{k-1}\ ((\ x,\ z_1\ )\ y_1\ (x,\ z_2\ )\ \ldots\ y_{k-1}\ (x,\ z_{k-1})))\ \ p_0\ \ p_1\ \ \ldots\ \ p_{k-1}\ \ s$

where s is the service name and p_i, $0\ \leq\ i < k$, are the properties to query.

For example, suppose that a partner wants to query the behavioral semantics of a service *vacuumAssistedCasting*.such as service composition and the service's execution status. Then the corresponding query is given by

$\lambda\ \ x\ \ (\ \lambda\ \ y_0\ \ y_1\ (\ \ y_0\ \ (\ x,\ \ z_0\ \ )\ \ y_1\ \ (x,\ \ z_1\ \ )))$ ***composedOf hasStatus vacuumAssistedCasting***

$\underset{\beta - reduction}{\overset{\Rightarrow}{}}$ (*composedOf* (*VacuumAssistedCasting,* z_0) *hasStatus* (*VacuumAssistedCasting,* z_1)).

To answer queries, partners can provide contents with values that can be called as needed. The answering message is represented as follows.

(*composedOf* (*VacuumAssistedCasting,* (***VacuumCasting Trimming QualityAssurance SelectedMaterial Dies TrimDies CastedProduct TrimmedProduct FinishedProduct***) **:** ***type of String list***) *hasStatus* (*VacuumAssistedCasting,* ***applied : type of states***))

In order to query non functional behavior for each of the sub functional systems, for example, the corresponding queries take the following forms.

$\lambda\ x(\lambda\ y_0\ (\ y_0\ (x,\ z_0\ )))$ ***processingTime vacuumAssistedCasting***

$$\underset{\beta - reduction}{\overset{\Rightarrow}{}} \quad (\textbf{\textit{processingTime}}\ (\textbf{\textit{VacuumAssistedCasting}},\ z_0))$$

To answer queries, a service provider can provide values for z_0, which can be called as needed by a service consumer. For example, the answer in this case will be

(*processingTime* (*VacuumAssistedCasting,* ***15 : type of integer***)).

Since in the WSCPC framework, a process is recursively defined and a service sub component is recursively defined as an individual service, querying behaviors to sub components involve a similar procedure. For example, to query processing time for a sub component *Trimming* task, the query would be

$\lambda\ x(\lambda\ y_0\ (\ y_0\ (x,\ z_0\ )))$ ***processingTime Trimming.***

The above example assumes both parties know that *Trimming* is a part of the *vacuumAssistedCasting* process and that the namespace for *Trimming* is defined clearly between both parties.

5.4.2 Process Enactment

Figure 27 illustrates a Web Service invocation and the corresponding message exchanges in a WSCPC collaborative process enactment [114]. Once a service consumer decides to

outsource one of its functional subsystems and locates an appropriate outsourcing service, the consumer needs to create a message to invoke the service. The service description is located in the partner's side linked by a registry entry from which the service consumer fetches the necessary information. The invocation message is composed of three components: *intention*, *action* and *target*.

To enact the partner's execution, the *intention* field must be either *request* or *propose*. The value *propose* is used to set up a negotiable rule or condition. For example, when a message initiator offers a price, then the value of the *intention* field must be *propose*. When the delivery time on a contract has not been decided by either side—service consumer or service provider—and neither has full authority to decide, then *propose* can be used. On the other hand, *request* is used to specify the desired actions on authorized subject. The value *request* will be used when a message initiator has a certain level of authority of task execution. For example, if a service consumer wants a die casting method to be *vacuumAssistedCasting* and has the authority to choose a casting method, then *request* will be used.

Formally, the *action* field can be defined based on (5.1) in Section 5.4.1. For example, suppose that a service consumer creates a message proposing a die casting service price for $10,000. Then the contents of *action* will be defined as follows.

$\lambda\ x(\lambda\ y_0(\lambda\ z_0\ (\ y_0\ (x\ z_0\ ))))$ ***10,00 price DieCasting***

$\overset{\Rightarrow}{\beta - reduction}$ ***(price (DieCasting 10,000))***

Therefore the entire message that is sent to service provider will be formally a triplet <*propose*, (*price* (*DieCasting* _)), *price*(*DieCasting 10000*)>. The message is composed using the XML SOAP message format and sent to the service provider. Figure 26 shows the example for the above message, which are sent to receiver. In Figure 26, *price* is a user-defined property and described in the document which *ProviderNS* specifies. The logical service *DieCasting* is also described in *ProvideNS*.

```
<service:conversation rdf:ID="message2">
    <service:hasIntention>
        <service:intention rdf:ID="intentionID">
            <service:parameterType rdf:about="#propose"/>
        </service:Intention>
    </service:hasIntention>
    <service:Target rdf:ID="objectClass">
        <service:LogicalService
rdf:about="providerNS:#DieCasting"/>
    </service:Target>
    <service:Action rdf:ID="actionClass">
        <service:LogicalService
rdf:about="providerNS:#DieCasting"/>
            <service:price
rdf:resource="providerNS#contract">10.000</service:price>
            <service:price/>
        </service:LogicalService>
    </service:Action>
</service:conversation>
```

Figure 26 A Message for Process Enactment

The enactment steps using communication model are illustrated in Figure 27. For example, to suggest a service price, a message in Figure 27 is sent (Steps (1) – (4)). Then the service provider figures out the semantics of the received message (Step (5)). The service provider can either follow a service consumer's action or initiate negotiation steps based on a local decision logic. In either case, the service provider also creates a message with its communicational decision and behavior, and sends it (Step (6)).

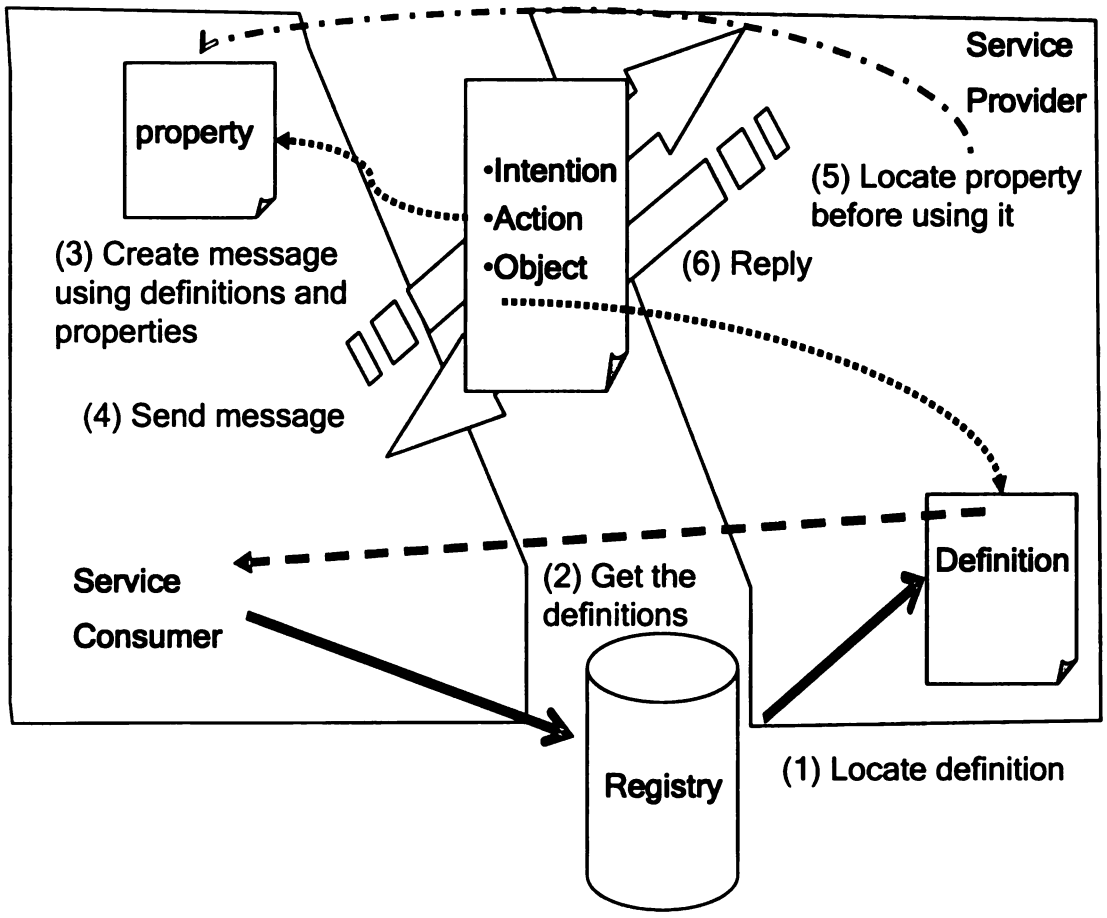


Figure 27 Service Enactment Using Service Interaction Model

In WSCPC, the negotiation proceeds through direct contact between a service consumer and a service provider. Partners exchange messages using the Service Interaction Model

at each negotiation round. For example, a company can send message to initialize a negotiation with <*request*, (*price,A*), *rollback*(*A*)>, where rollback is a property in PEM and (*price, A*) is a class written with OWL-S. Negotiation ends when a partner sends the message <*accept*, (*price, A*), *none*>.

5.4.3 Process Sharing

The platform neutral nature of Web Services enables supporting concurrent and consistent access to heterogeneous software applications. Semantic integration of Web Services provides semantic based process sharing, where people continue to use their specialized applications but collaborative activities are shared with semantic information.

WSCPC utilizes query-based monitoring, where a service consumer builds a query based on the service provider's service representation and the service provider answers with details about its internal behaviors [114]. In the WSCPC framework, organizational and informational aspects are captured by the Process Monitor Model and hence process sharing among multiple participants is possible through the semantics in the Process Monitor Model class.

To monitor a partner's execution, the *intention* field must be set to *query*. For example, suppose that a service consumer creates a message querying *Trimming* service execution time. Then the contents of *action* will be defined as follows.

$$\lambda\ x(\lambda\ y_0\ (\ y_0\ (x\ z_0\ )))\ \textbf{\textit{executionTime Trimming}}$$

$$\overset{\Rightarrow}{\beta - reduction}\ (\textbf{\textit{executionTime}}\ (\textbf{\textit{Trimming}}\ \ z_0))$$

Therefore the entire message that is sent to service provider will be formally a triplet <*query*, (*executionTime* (*Trimming*)), *none*>. In the query message, the z_0 field is empty. Figure 28 shows the message example.

```
<service:conversation rdf:ID="message3">
    <service:hasIntention>
        <service:intention rdf:ID="intentionID">
            <service:parameterType rdf:about="#query"/>
        </service:Intention>
    </service:hasIntention>
    <service:Target rdf:ID="objectClass">
        <service:LogicalService
rdf:about="providerNS:#Trimming"/>
    </service:Target>
    <service:Action rdf:ID="actionClass">
        <service:LogicalService
rdf:about="providerNS:#Trimming"/>
            <service:executionTime
rdf:resource="providerNS#contract"/>
        </service:LogicalService>
    </service:Action>
</service:conversation>
```

Figure 28 A Query Message for Execution Traces

The corresponding answering message is generated by filling in the contents requested by the query message. Figure 29 shows answering message for the monitoring request. Note that most of the message contents are similar except those marked with boldface.

```
<service:conversation rdf:ID="message4">
    <service:hasIntention>
        <service:intention rdf:ID="intentionID">
            <service:parameterType rdf:about="#answer"/>
        </service:Intention>
    </service:hasIntention>
    <service:Target rdf:ID="objectClass">
        <service:LogicalService
rdf:about="providerNS:#Trimming"/>
    </service:Target>
    <service:Action rdf:ID="actionClass">
        <service:LogicalService
rdf:about="providerNS:#Trimming"/>
            <service:executionTime
rdf:resource="providerNS#contract">200
            </service:executionTime>
        </service:LogicalService>
    </service:Action>
</service:conversation>
```

Figure 29 Execution Trace Reporting

Figure 30 illustrates how WSCPC supports process sharing. Once tasks are allocated to geographically dispersed service providers, the Process Engine in WSCPC regularly polls the service execution status through the Web Service Module. When the query for a service execution status reaches the service provider, the current service execution status is fetched from that software application tools and then sent to the service requester in the form of a Proccess Monitor Model grounding. Since a Proccess Monitor Model grounding is acceptable and understandable to all the participants, The Process Engine can translate it for the domain specific applications and update the task status information.

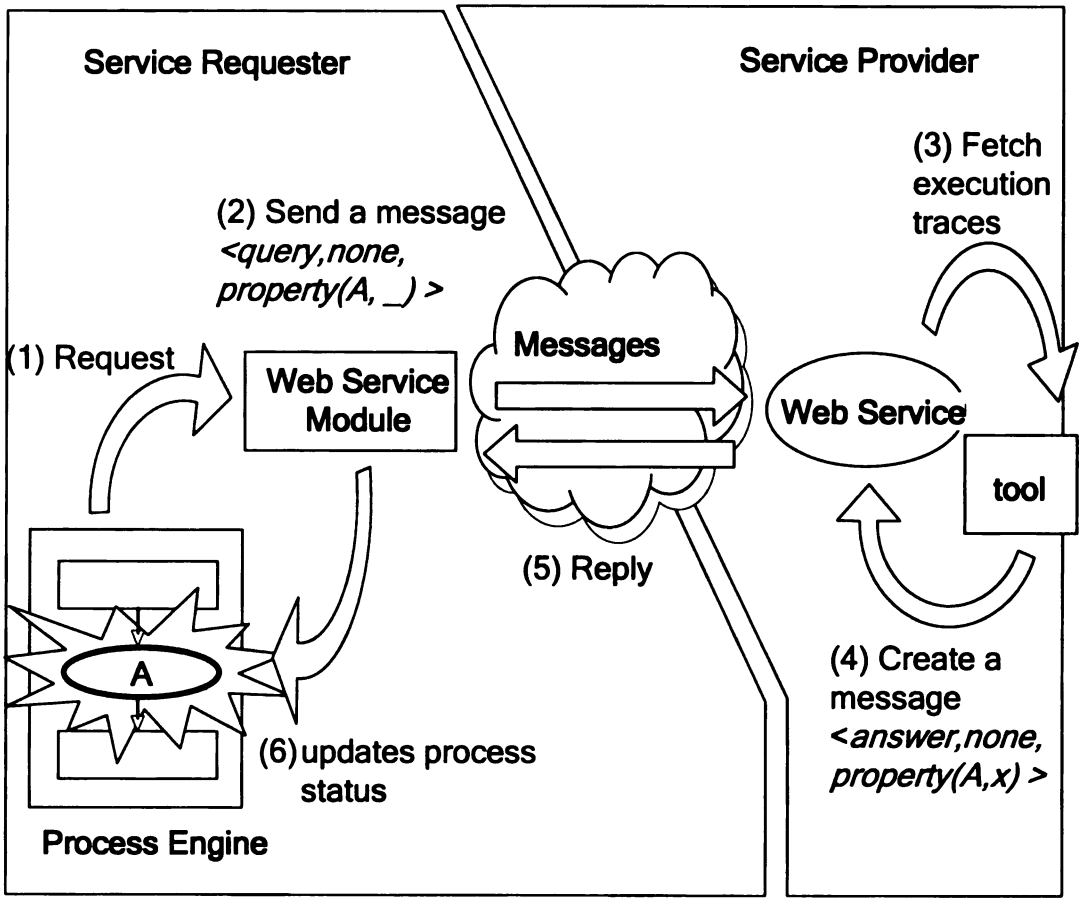


Figure 30 Sharing Display for Concurrent and Consistent Collaboration

CHAPTER 6 EVALUATION

In this section, we will discuss how WSCPC supports Web Service based collaboration. As an illustration of our work, we consider a case study [149] that describes a die casting process for thermoelectric fan housing.

6.1 A Scenario

Figure 31 illustrates this case study to test our implementation of WSCPC [112, 116]. The example in Figure 31 is from [21, 150]. The die casting process for thermoelectric fan housings can be decomposed into several sub tasks with each task being distributed to different companies. In Figure 31, four companies are collaborating at process startup with each company having its own dedicated task such as defining requirements, material selection, die making, and casting products. The left side of Figure 31 shows the initial functional systems, while the right side illustrates task distribution results among the participating companies.

Tasks in the process are interdependent since one task in one company affects other companies' tasks. At the requirements setting stage, company A designs a fan housing product. Based on a study [149], the key factors of product considerations are high thermal conductivity, uniform metal fill, free of porosity, and precise tolerance with +/- 0.005". An ontology for the die casting process is defined with OWL. The design requirements are translated into OWL and posted to be shared with companies B and C together as output specifications. The sample OWL file is shown in Figure 32. Tolerance and thermal conductivity are defined here while other requirements such as uniform metal fill are described as *additional_spec*.

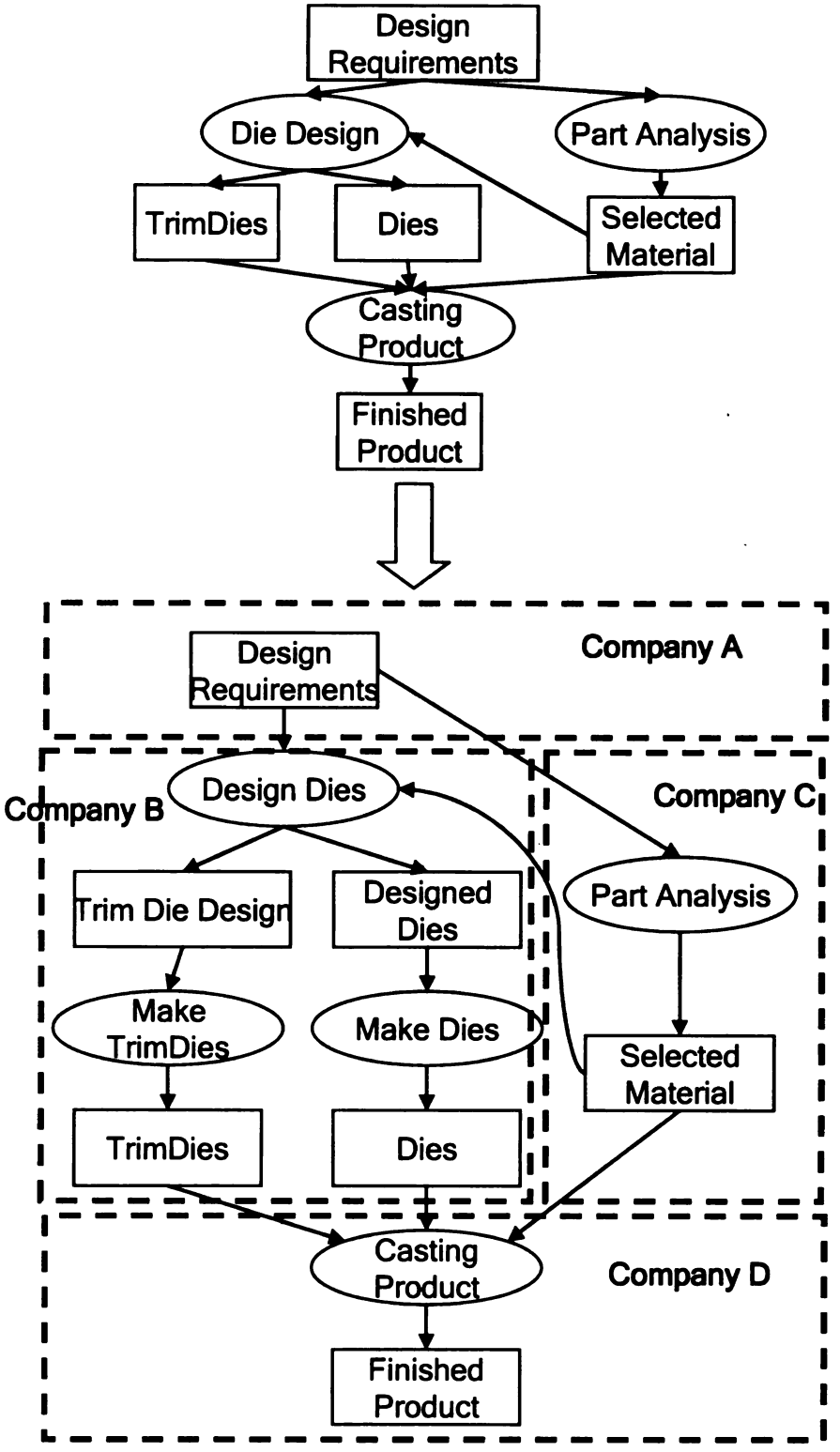


Figure 31 A Design and Manufacturing (D&M) Process Example

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-
ns#"      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xml:base="http://www.wscpc.fake/req_ont#1">
:
:
<rdfs:Class rdf:ID="Requirements ">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#name"/>
      <owl:allValuesFrom
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#tolerance"/>
      <owl:allValuesFrom
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
:
:
</rdfs:subClassOf>
</rdfs:Class>
<owl:DatatypeProperty rdf:ID="tolerance"/>
<owl:DatatypeProperty rdf:ID="name"/>
</rdf:RDF>
```

Figure 32 Design Requirements Written in OWL.

During the requirements setting task, company A designs the fan housing product and sets performance specifications. Based on a study [149], company A designs the fan housing with a diameter of 14", a height of 10" and a weight of 7.5 lbs. The wall thickness of the product ranges from 0.880" at the base to 0.124" on the side walls. The key factors of product considerations are high thermal conductivity, uniform metal fill, free of porosity, and precise tolerance with +/- 0.005". In addition, company A sets the material property requirements: Density < 0.10 (#/inch3), Thermal Conductivity > 120 (W/mK), Ultimate Tensile Strength > 40 ksi, Tensile Yield Strength > 20 ksi. Terminologies for the manufacturing domain are defined with OWL. Design requirements are translated into XML and sent to company B and C together as output specifications. Figure 33 shows an example of output specifications created by company A.

```
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:req="http://www.wscpc.fake/req_ont#">
<rdf:Description
 rdf:about="http://www.wscpc.fake/Outputs ">
    <req:name>Fan Housing</ req:name>
    < req:tolerance>USA</ req:tolerance >
    < req:height>10 inch </req:height >
    < req:diameter>10 inch </req:diemeter >
    :
    :
</rdf:Description>
```

Figure 33 Output Specification

After company A constructs output data and sends it to B and C, company B needs to select the right materials for the fan housing. Company B selects an Aluminum 130 alloy among several alternatives. Subsequently, company C starts the die design process using A's design specifications and B's material selections as described in the output specifications. Company C's task is decomposed into subtasks: *DesignDies*, *MakeTrimDies*, and *MakeDies*. Each subtask owned by C can be decomposed into detailed subtasks with several alternatives. For example, the *DesignDies* task has several alternative processes such as *VacuumAssistedCasting*, *highPressureCasting*, and *Semi-SolidCasting*. Figure 4 shows an example of a production rule for one alternative process *VacuumAssistedCasting* for the *DesignDies* task. Company D's is to implement the casting process as per the output specifications provided by companies B and C. Company D can make a "buy" or "make" decision.

Suppose that company D decides to outsource the *CastingProduct* task. Then D will utilize WSCPC to locate and assign the *CastingProduct* task via Web Services. In our scenario, company D is outsourcing the task "casting product" and locating an appropriate service provider. In order to outsource a *CastingProduct* task, the die casting manufacturing capabilities must be published and advertised in an understandable way such that every OEM will be able to find a provider that can satisfy the design requirements.

When company D decides to outsource the *CastingProduct* task, the company defines the desired attributes for the product and searches the service registry using such attributes. The attributes include *Materials*, *Finishing*, and *Certifications* as described in our Service Oriented Process Model. As illustrated in Figure 31, the Semantic Service

Definition defines pre-conditions and post-conditions that will be used for locating and evaluating service providers. The reviews left in service registry will help to know the trustworthiness of service providers. For our scenario, company D will create a query to sthe ervice registry with "Materials=Aluminium," "Certifications=ISO9002," and "Finishing=power coating." In WSCPC, such queries can be either generated autonomously based on the predefined evaluation functions or typed manually.

After the appropriate search, the service registry notifies company D that the manufacturing capability of a casting company in Troy, Michigan satisfies the query and hence can fulfill the *CastingProduct* task. Subsequently, company D evaluates other attributes including reputation (which is not part of the original query), starts negotiating if necessary, and outsources *CastingProduct* by invoking the services offered by the casting company in Troy. Our Service Enactment Model provides support for various service invocation types and hence facilitates dynamic process enactment, which is required for implementing collaborative processes in distributed and heterogeneous manufacturing environments.

Collaborative interactions among multiple companies often encounter problems since multiple decision centers may exist in the collaborative distributed environment. Each company has its own local mechanism to decide how their tasks are executed. Decisions from a single company may hinder the process of obtaining an efficient solution or even result in a failure due to each company's self-interested tendency. Therefore, decisions should be coordinated between various functional groups. That is, task decision should be allowed to be cancelled with an alternative in a distributed collaborative design and manufacturing environment.

Consider the process in Figure 31. Company B completes the *PartAnalysis* task and generates its output result. After a certain period, company C finishes the *DieDesign* task. Company D then starts its task by outsourcing *CastingProduct* task. Support later that company D faces a design problem on the final product. For example, suppose that the final product fails in a mechanical stress testing. After a certain diagnosis, company D concludes that the product should be designed stronger and that the materials used in the product should be replaced. Company B then changes the material specifications into a stronger one with a minimum loss in thermal conductivity. This change affects both companies C and D's process execution since they are using the A130 alloy originally selected by company B. In such a case, companies C and D must rollback their task status by re-initializing their process executions. Therefore, company B must send several messages about its own rollback process. Messages by B are described as follows.

m_1 = <*declare, hasStatus*(*PartAnalysis*), *finished* ⇒ *ready*>
< *rollback*, *PartAnalysis* >

m_2 = <*declare, hasStatus*(*PartAnalysis*), *ready* ⇒
< *invokeEnactment*, *PartAnalysis* >
running>

m_3 = <*declare, hasStatus*(*PartAnalysis*), *running* ⇒ *finished*>
< *apply*, *PartAnalysis* >

m_4 = <*declare, boundBy(SelectedMaterial),initial*

$\Rightarrow$ *ready*

< *apply, SelectedMaterial* >

6.2 Prediction for Service Trustworthiness

In this section, we apply our trustworthiness estimation scheme to the scenario depicted in Figure 31. The prediction has three phases. First, we model service behaviors. The service behavior can be published in the form of our Process Definition Model or designed using a set of component services. For the die casting process example, Figure 31 provides the service behavior model. The next phase is to build the model of the dynamic service behavior from the model in the first phase. In the second phase, new properties are added such as rollback and selection with transition probabilities p_{ij} and transition costs t_{ij}.

Figure 34 illustrates an example of dynamic service behavior based on the scenario given in the Section 6.1. We do not implement the actual net model given in Figure 34. Rather, we introduce Figure 34 in order to show the dynamic properties that need to be added in the second phase.

From the scenario of the previous section, tasks *DieDesign* and *CastingProduct* will be outsourced by the invoking services. There are *n* and *m* available services for those tasks, respectively. To represent tasks and their available services, we add dummy data objects without breaking the WSCPC service model. The black dashed arrows represent selections that are available and the grey dashed arrows indicate the probability of

satisfactory service execution. Roll back happens when a service fails or service execution is not satisfied. Rollback arrows therefore include the probabilities, which are based on ratings that users determine to be the minimum level of service quality.

The available services are also associated with performance parameters based on our WSCPC trust model. Since *DieDesign* and *CastingProduct* are about to be outsourced, their performance values are set to 0.0 and their tasks are filled with dots as a placeholder for the future relationship. The grey data and task objects indicate the finished work. If the model is evaluated before the execution begins, there will be no grey objects in the graph. In the example of Figure 34, we assume that we are in the middle of execution of the whole process. We are in the step where the task *PartAnalysis* is finished and the two tasks *DieDesign* and *CastingProduct* need to be associated with particular services. The probabilities and performances can be obtained through our trust model presented in the Section 4.2.

The third phase of our prediction is to estimate the trustworthiness of the process. It is also capable of offering analysis aimed at identifying the impact of various factors. To this end, we can run simulations based on available choices or build the corresponding Petri net models that can be run on the available stochastic Petri net tools.

In order to evaluate the scenario given in Section 6.1, we randomly chose 20 items from MovieLens with average ratings between 3.0 and 4.0 [131] After extracting their ratings and probabilistic distributions, ten items are assigned to DD_i while the others are assigned to CP_j. Failure ratings are set to 2.5. We assume the service consumer satisfies the service outcome when the consumer's rating is over 3.5. Table 1 shows the sample

data and its associated expected probabilities used in our evaluation. The failure rate indicates the expected probability that the service delivery brings a rating less than 2.5, and the quality rate tells the expected probability that service consumer may like the service quality (rating $\geq$ 3.5).

We identify three types of evaluation criteria: 1) the least failure rate (LF), 2) the best satisfaction (BS), and 3) the maximal quality (MQ). LF specifies the selection with the minimum failure probability, BS indicates the selection for the best rated service, and MQ gives the selection for maximum probability satisfying a certain rating. In order to simplify the evaluation, we assume that all the service providers posted same values for performance factors such as cost and delivery time. Table 2 shows the prediction results based on each criteria. Table 3 illustrates the accuracy of prediction.

Based on Table 2, selections may vary based on criteria. Selection for "Casting Product" may be either CP_2 or CP_6 based on the criteria. If more non functional factors are considered, the selection may become more complex since a service consumer must build a utility function to evaluate the expected performance matrix.. Table 3 illustrates the ranking differences between predictions and actual results. Ranking provide the ordered list of service recommendation. Therefore ranking correctness is also important as ranking affects selection results. From the samples tested, the overall ranking errors for the expected failure rate and the expected quality rate are 1.4 and 1.2 respectively. These results demonstrate that our WSCPC methods are acceptable to predict trustworthiness.

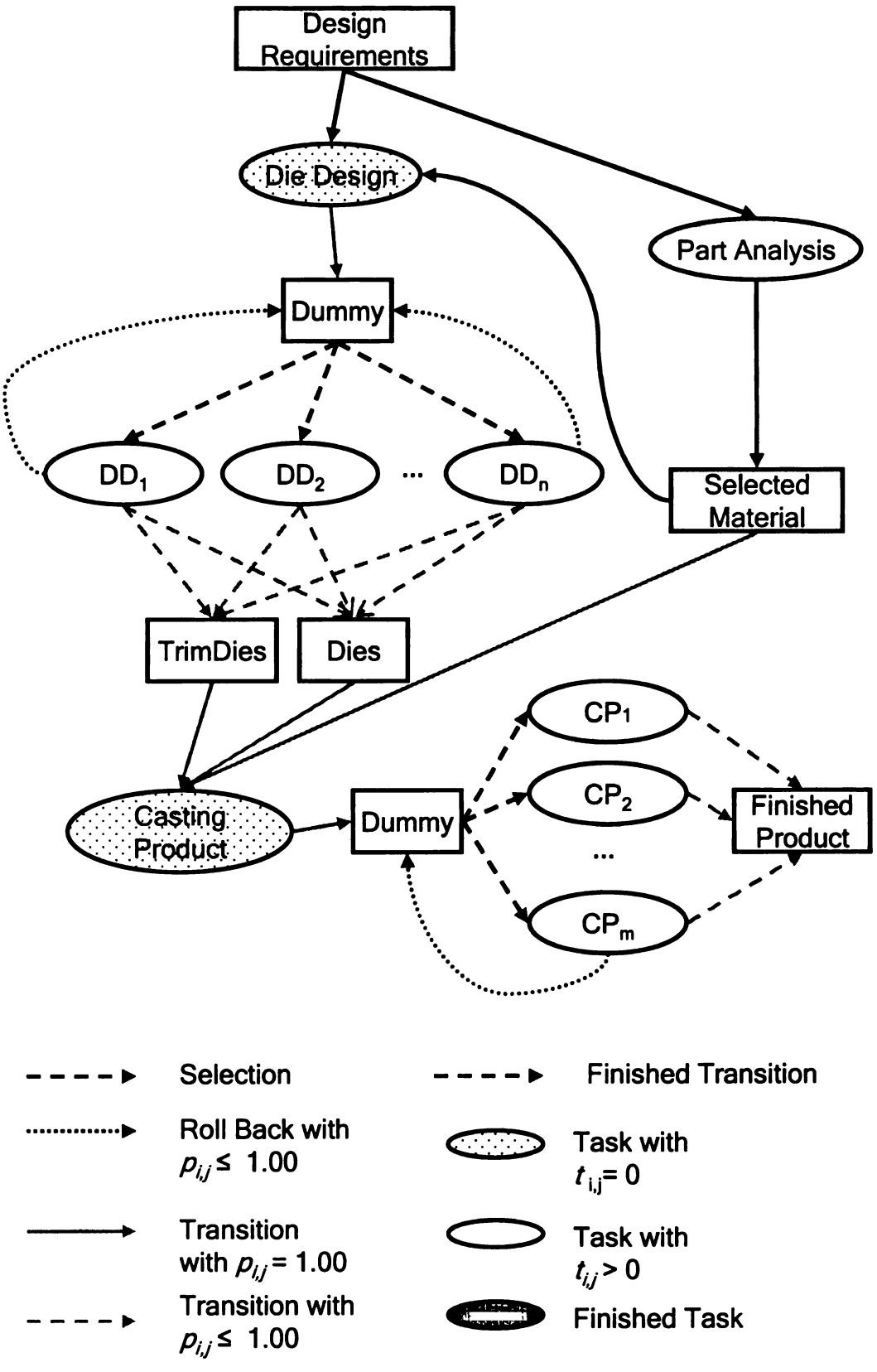


Figure 34 Dynamic Net Model

Task	Expected Rating	Expected Failure Rate (%)	Expected Quality Rate (%)
DD_1	3.16	22.98	39.75
DD_2	3.45	18.41	52.17
DD_3	3.34	17.82	47.52
DD_4	3.16	24.37	40.36
DD_5	3.29	24.06	49.73
DD_6	3.48	15.92	56.05
DD_7	3.29	21.38	44.83
DD_8	3.11	22.96	31.85
DD_9	3.09	24.71	33.53
DD_{10}	3.32	22.52	49.01
CP_1	3.37	19.42	46.76
CP_2	3.42	18.99	51.48
CP_3	3.14	32.08	40.57
CP_4	2.77	39.53	27.91
CP_5	3.39	18.24	49.06
CP_6	3.45	16.54	48.87
CP_7	3.31	19.64	43.93
CP_8	3.24	21.85	45.38
CP_9	3.10	26.19	33.33
CP_{10}	3.09	27.05	35.25

Table 1 Sample Data for Evaluation

Criteria	Selection	Failure Rate (%)	Quality Rate (%)
BS	DD_6, CP_6	0.70	0.27
LF	DD_6, CP_6	0.70	0.27
MR	DD_6, CP_2	0.68	0.29

Table 2 Prediction of Service Execution

Selection	Ranking in Failure Rate	Ranking in Quality Rate	Rank Difference in Failure Rate	Rank Difference in Quality Rate
DD_6	1	1	0	0
CP_2	1	4	1	3
CP_6	1	2	1	0

Table 3 Accuracy of Prediction

CHAPTER 7 CONCLUSION

In this thesis, we present a new framework to provide collaborative process management for service-oriented computing. Our work focuses on issues of service trustworthiness. There has been considerable work done in the area of trustworthiness issues covering service behaviors, service securities and reliabilities. Some of this work has been proposed as new standards. Other work has been proposed solely as extensions to current systems and standards. Our work provides a new, solid foundation on which to develop new methods for reliable and safe service-oriented computing in collaborative virtual organization systems.

In particular, we have developed a framework that we call the Web Service Collaborative Process Coordinator (WSCPC). WSCPC is built on top of JAVA and the Axis framework. We first provide service models for realizing a service oriented collaborative environment. Then, we propose a methodology for predicting service behavior and hence enhancing system trustworthiness. Based on the proposed methodology, service behavior is analyzed stochastically. Adopting our frame-based approach enables partners to be monitored and enacted so as to hold the status that the service is valid and that services are appropriately configured based on the consumer's needs through purely semantic dialogue interpretation. Compositional services are also able to be scheduled such that service behavior can ideally achieve an optimal outcome. Moreover, our proposed approach is designed in a platform neutral way, retaining the spirit of a service-oriented paradigm. We submit that our approach will satisfy the need of users to control ability and customization availability in Web Services.

Our implementation is based on a standalone server-application. So the service based collaboration and its reliability can be realized by using the WSCPC framework. However, our proposed methods can also be integrated into current legacy systems or existing Web Service based standards as we have proposed a platform neutral, message oriented protocol and data formats. We believe that software modules can collaborate with each other using our proposed methodologies. Since our messages follow the SOAP format and since the semantics based on OWL is integrated into messages, software modules or server applications with any inference engine can be implemented to support the proposed method.

In addition, we provide an evaluation based on the example of a design and manufacturing domain. Although the preliminary experiments on the artificial data sets were conducted to evaluate the effectiveness of the proposed framework in the design and manufacturing domain, we believe that our work will contribute to goal of realizing true service oriented collaboration, not only in the design and manufacturing arena, but also over broad ranges of collaborative process domains.

BIBLIOGRAPHY

[1] J. N. Lee and Y. G. Kim, "Exploring a Causal Model for the Understanding of Outsourcing Partnership," *HICSS 2003,* vol. 268, 2003.

[2] T. D. Floyd, *Winning the New Product Development Battle*. New York: Institution of Electrical and Electronics Engineers, 1993.

[3] A. Donald and O'Neill, "Offshore Outsourcing Takes Hold," Mortgage Bank Dec 2003.

[4] M. Terk, E. Subrahamanian, C. Kasabach, F. Prinz, D. P. Siewiorek, A. Smailagic, J. Stivoric, and L. Weiss, "Rapid Design and Manufacture of Wearable Computers," *Communications of the ACM,* vol. 38, pp. 63-70, 1996.

[5] R. Akkiraju, D. Flaxer, H. Chang, T. Chao, L. J. Zhang, F. Wu, and J. J. Jeng, "A Framework for Facilitating Dynamic e-Business Via Web Services," in *OOPSLA 2001 Workshop on Object-Oriented Web Services*, Florida USA, 2001.

[6] S. Tsur, "Are Web Services the Next Revolution in E-commerce?," in *the 27th VLDB conference*, Roma, Italy, 2001.

[7] F. Curbera, R. Khalaf, N. Mukhi, S. Tai, and S. Weerawarana, "The Next Step in Web Services," *Communication of The ACM,* vol. 46, pp. 29-34, 2003.

[8] F. Curbera, W. A. Nagy, and S. Weerawarana, "Web Services: Why and How," in *OOPSLA 2001 Workshop on Object-Oriented Web Services*, Florida USA, 2001.

[9] S. Narayanan and S. McIlraith, "Simulation, verification and automated composition of web services," *In Proceedings of the 11th international conference of World Wide Web,* pp. 77-88, 2002.

[10] M. Pierce, C. Youn, G. Fox, S. Mock, K. Mueller, and O. Balsoy, "Interoperable Web Services for Computational Portals," in *the IEEE/ACM SC2002 Conference*, Baltimore, USA, 2002.

[11] F. Baader and W. Nutt, "Basic Description Logics," in *The description logic handbook: theory, implementation, and applications*: Cambridge Uniersity Press, 2003, pp. 43-95.

[12] J. Day and R. Deters, "Selecting the best web service," in *the 2004 conference for Advanced Studies on Collaborative research*, Markham, Ontario Canada, 2004, pp. 293-307.

[13] Y. Liu, A. H. Ngu, and L. Z. Zeng, "QoS computation and policing in dynamic web service selection," in *the 13th international World Wide Web conference on Alternate track papers & posters*, New York, NY, 2004, pp. 66-73.

[14] E. Maximilien and M. Singh, "Agent-based trust model involving multiple qualities," in *the fourth international joint conference on Autonomous agents and multiagent systems (AAMAS'05)*, Netherlands, 2005, pp. 519-526.

[15] A. Lazovik, M. Aiello, and M. Papazoglou, "Associating assertions with business processes and monitoring their execution," in *the 2nd international conference on service oriented computing* New York NY USA, 2004, pp. 94-104.

[16] W. Ma, V. Tosic, B. Esfandiari, and B. Pargurek, "Extending Apache Axis for monitoring Web Service Offering," in *international workshop on Business services networks (BSN'05)*, Hong Kong, 2005, pp. 7-7.

[17] L. J. Zhang and D. Anrdagna, "SLA based profit optimization in automatic computing systems," in *the 2nd international conference on Service oriented computing*, New York NY USA, 2004, pp. 173-182.

[18] L. Baresi, C. Ghezzi, and S. Guinea, "Smart monitors for composed services," in *the 2nd international conference on service oriented computing*, New York, NY USA, 2004, pp. 193-202.

[19] G. J. Holzmann, *SPIN Sources, Version 3.4.1*: available with SPIN, 2000.

[20] R. Baldwin and M. J. Chung, "Design Methodology Management," *IEEE Computer*, pp. 54-63, February 1995.

[21] M. J. Chung, P. Kwon, and B. Pentland, "Making Process Visible: A Grammartical Approach ot Managing Design Processes," *ASME Transaction Journal of Mechanical Design,* vol. 124, pp. 364-374, 2002.

[22] A. Mowshowitz, "Virtual Organization," *Communications of the ACM,* vol. 40, pp. 30-37, 1997.

[23] M. Wolters and M. Hoogeweegen, "Management Support for Globally Operating Virtual Organizations: the Case of KLM Distribution," in *the 32th Hawaii International Conference on System Science*, 1999.

[24] B. Medjahed, B. Benatallah, A. Bouguettaya, A. H. Ngu, and A. K. Elmagarmid, "Business-to-business interactions: issues and enabling technologies," *The VLDB Journal - The International Journal on Very Large Data Bases,* vol. 12, May 2003.

[25] S. Y. Choi, D. O. Stahl, and A. B. Whinston, *The Economics of Electronic Commerce*. Indianapolis, IN: Macmilan Technical Publishing, 1997.

[26] M. J. Cronin, *Unchained Value: The New Logic of Digital Business*. Boston, MA: Harvard Business Press, 2000.

[27] P. Timmers, *Electronic Commerce: Strategies and Models for Business-to-Business Trading*. New York: John Wiley & Sons, 1999.

[28] B. Travica, "Virtual Organization and Electronic Commerce," *The DATABASE for Advances in Information Systems,* vol. 36, 2005.

[29] A. Lazcano, G. Alonso, H. Schuldt, and C. Schulder, "The WISE approach to electronic commerce," *International Journal of System Science Engineering,* vol. 15, pp. 343-355, 2000.

[30] C. Schuler, H. Schuldt, G. Alonso, and H. J. Schek, "Workflows over workflows: practical experiences with the integration of SAP R/3 business workflows in WISE," in *Informatik'99 Workshop: Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications*, 1999.

[31] H. Ludwig and Y. Hoffner, "Contract-based cross-organisational workflow - the Crossflow Project," in *International Joint Conference on Work Actitivies Coordination and Collaboration (WACC'99)*, 1999.

[32] Y. Hoffner, H. Ludwig, P. Grefen, and K. Arberer, "CrossFlow: integrating workflow management and electronic commerce," *ACM SIGecom Exchange,* vol. 2, pp. 1-10, 2001.

[33] J. Weissenfels, M. Gillmann, O. Roth, G. Shegalov, and W. Wonner, "The mentor-lite prototype: a light-weight workflow management system," in *ICDE Conference*, San Diego, CA USA, 2000, pp. 658-686.

[34] G. Shegalov, M. Gillmann, and G. Weikum, "XML-enabled workflow management for e-services across heterogeneous platforms," *The VLDB Journal - The International Journal on Very Large Data Bases,* vol. 10, pp. 91-103, 2001.

[35] H. Skogsrud, B. Benatallah, and F. Casati, "Trust-serv: model driven lifecycle management of trust negotiation policies for web services," in *International World Wide Web Conference*, New York, NY USA, 2004, pp. 53-62.

[36] M. Shen, B. Benatallah, M. Dumas, and E. O. Y. Mak, "SELF-SERV: A platform for rapid composition of web services in a peer-to-peer environment," *VLDB Conference,* pp. 1051-1054, 2002.

[37] D. Georgakopoulos, H. Schuster, A. Cichocki, and D. Baker, "Managing process and service fusion in virtual enterprises," *Information Systems,* vol. 24, pp. 429-456, 1999.

[38] H. Schuster, D. Baker, A. Cichocki, D. Georgakopoulos, and M. Rusinkiewicz, "The collaboration management infrastructure," in *ICDE Conference*, San Diego CA USA, 2000, pp. 677-678.

[39] F. Casati, S. Ilnicki, L. J. Jin, V. Krishnamoorthy, and M. C. Shan, "eFlow: a platform for developing and managing composite e-services," in *Technical Report HPL-2000-36*, H. Laboratoris, Ed. Palo Alto, CA, 2000.

[40] B. Benatallah, B. Medjahed, A. Bouguettaya, A. K. Elmagarmid, and J. Beard, "Composing and maintaining Web-based virtual enterprises," in *1st VLDB workshop*, Cairo, Egypt, 2000, pp. 71-90.

[41] S. M. Bellovin, "Security problems in the TCP/IP protocol suite," *Computer Communication,* vol. 19, pp. 32-48, 1989.

[42] T. Dierks and C. Allen, "The TLS Protocol Version 1.0. IETF Request for Comments " *RFC 2246,* 1999.

[43] N. Doraswamy and D. Harkins, *IPSec: The New Security Standard for the Internet, Interanets, and Virtual Private Networks.* Englewood Cliffs, NJ: Prentice-Hall, 1999.

[44] P. Volpano and G. Smith, "Language issues in mobile program security," *Mobile Agents and Security, Lecture Notes in Computer Science,* vol. 1419, pp. 137-153, 1998.

[45] L. Rasmusson and S. Jansson, "Simulated social control for secure Internet commerce," in *ACM Workshop on New Security Paradigms*, 1996, pp. 18-25.

[46] A. Romao and M. M. Da Silva, "Proxy certification: A mechanism for delegating digital signature power to mobile agents," in *the Workshop on Agents in Electronic Commerce*, 1999, pp. 131-140.

[47] X. Yi, C. K. Siew, and M. R. Syed, "Digital signature with one-time pair of keys," *Electron Letter,* vol. 36, pp. 130-131, 2000.

[48] M. Bellare and S. K. Miner, "A forward-secure digital signature scheme," *Advances in Cryptology-CRYPTO'99, Lecture Notes in Computer Science,* vol. 1666, 1999.

[49] H. Krawczyk, "Simple forward-secure signatures from any signature scheme," in *the Seventh ACM Conference on Computer and Communications Security*, 2000, pp. 108-115.

[50] G. Vigna, "Cryptographic traces for mobile agents," *Mobile Agents and Security, Lecture Notes in Computer Science,* vol. 1419, pp. 137-153, 1998.

[51] M. Bravetti, C. Guidi, R. Lucci, and G. Zavattaro, "Supporting e-commerce systems formalization with choreography languages," in *the 2005 ACM symposium on Applied computing*, Santa Fe, New Mexico, 2005, pp. 831-835.

[52] G. B. Chafle, S. Chandra, V. Mann, and M. G. Nanda, "Business processes and conversations: Decentralized orchestration of composite web services," in *the 13th international WWW on Alternate track paper and posters*, New York USA, 2004, pp. 134-143.

[53] "Web Service Architecture: W3C Working Draft 8," *http://www.w3c.org/TR/2003/WD-ws-arch-20030808*.

[54] P. Sandoz, S. Pericas-Ceertsen, K. Kawaguchi, M. Hadley, and E. Pelegri-Llopart, "Fast Web Services," Sun Microsystems, August 2003.

[55] F. Casati, E. Shan, U. Dayal, and M. Shan, "Business-oriented management of web services," *Communication of The ACM,* vol. 46, October 2003.

[56] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: Enabling scalable virtual organizations," *Suppercomputer Application,* vol. 15, 2001.

[57] M. P. Papazoglou, "Service-Oriented Computing: Concepts, Characteristics and Directions," in *4th International Conference on Web Information Systems Engineering (WISE'03)*, Rome, Italy, 2003.

[58] "Web Service Description Language (WSDL) 1.1," *http://www.w3c.org/TR/2001/NOTE-wsdl-20010305*.

[59] "SOAP: Simple Object Access Protocol 1.1," *W3C Note,* 2000.

[60] "UDDI specification version 2.04," *http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm*.

[61] C. Peltz, "Web Services Orchestration and Choreography," *IEEE Computer (October),* pp. 46-52, 2003.

[62] R. Shapiro, "A Comparison of XPDL, BPML, and BPEL4WS.," *xml.coverpages.org/Shapiro-XPDL.pdf,* 2002.

[63] S. Weerawarana and C. Francisco, "Business Process with BPEL4WS: Understanding BPEL4WS, Part1," IBM Developer Works., 2002.

[64] "OWL-S," *http://www.daml.org/services*.

[65] G. Alonso, "A Note About RPC: Myths Around Web Services," *Data Engineering,* vol. 25(4), December 2002.

[66] "Web Services Composition Interface 1.0," *http://ifr.sap.com/wsci/specification/wsci-spec-10.htm*.

[67] A. Arkin, S. Askary, S. Fordin, W. Jekeli, K. Kawaguchi, D. Orchard, S. Pogliani, K. Riemer, S. Struble, P. Takacsi-Nagy, I. Trickovic, and S. Zimek, "WSCI," *http://www.w3.org/TR/2002/NOTE-wsci-20020808*

[68] M. Paolucci, N. Srinivasan, K. Sycara, M. Solanki, O. Lassila, D. McGuinness, G. Denker, D. Martin, B. Parsia, E. Sirin, T. Payne, S. McIlraith, J. Hobbs, M. Sabou, and D. McDermott, "OWL-S," *http://www.daml.org/services/owl-s/1.0/owl-s.pdf* 2003.

[69] "Namespaces in XML," *W3C Recommendation,* 14 January 1999.

[70] "XML Schema Part 1: Structures," *W3C Recommendation,* 2001.

[71] "XML Schema Part 2: Datatypes," *W3C Recommendation,* 2001.

[72] "SOAP Tutorial," *http://www.w3schools.com/soap/default.asp,* 2006.

[73] "WSDL Tutorial," *http://www.w3schools.com/wsdl/default.asp,* 2006.

[74] "BPML," *http://www.bpmi.org/bpml.esp*.

[75] I. Horrocks, "Daml+Oil: a description logic for the semantic web," *IEEE Data Engineering Bulletin,* vol. 25(1), pp. 4-9, 2002.

[76] D. Gannon, R. Ananthakrishnan, S. Krishnan, M. Govindaraju, L. Ramakrishnan, and A. Slominski, *Grid Web Services and Application Factories, chapter 9*, 2003.

[77] N. Kavantzas, D. Butdett, and G. Ritzinger, "Web Services Choreography Description Language Version 1.0, Working Draft WD-ws-cdl-10-20040427," April 2004.

[78] G. Xue, M. Fairman, G. E. Pound, and S. J. Cox, "Implementation of a Grid Computation Toolkit for Design and Optimization with Matlab and Condor," *In Euto-Par 2003 parallel Processing, Lecture Notes in Computer Science,* pp. 357-365, 2003.

[79] S. Tuecke, K. Czajkowski, I. Foster, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, P. Vanderbilt, and D. Snelling, "Open Grid Services Infrastructure (OGSI) Version 1.0," *Global Grid Forum Draft Recommentation,* 2003.

[80] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "Grid Services for Distributed System Integration," *Computer,* vol. 35(6), 2002.

[81] R. A. v. Engelen and K. Gallivan, "The GSOAP toolkit for Web Services and Peer-to-peer Computing Networks," in *the 2nd IEEE International Symposium on Cluster Computing and the Grid (CCGrid2002)*, Berlin, Germany, May 2004, pp. 128-135.

[82] S. Krishnan and D. Gannon, "Xcat3: A Framework for CCA Components as OGSA Services," *HIPS 2004, 9th International Workshop on High-Level Parallel Programming Models and Supportive Environments,* April 2003.

[83] G. von Laszewski, B. Balunkal, A. Kaizar, S. Hampton, and S. Nijsure, *GridAnt-Client-side Workflow Management with Ant*, 2002.

[84] K. Amin and G. von Laszewski, "GridAnt: A Grid Workflow System," in *Manual*, 2003.

[85] C. B. Trastour and C. Priest, "Semantic Web Support for the Business-to-business E-commerce Lifecycle," in *the 11th International Conference on World Wide Web*, Honolulu Hawaii, May 2002, pp. 89-98.

[86] C. Goble and D. Roure, "The grid: an application of the semantic web," *ACM SIGMOD,* vol. 31(4), pp. 65-70, 2002.

[87] "OWL," *http://www.w3c.org/2001/sw/WebOnt*.

[88] B. Medjahed, A. Bouguettaya, and A. K. Elmagarmid, "Composing Web Services on the Semantic Web," *VLDB Journal,* vol. 12(4), pp. 331-351, November 2003.

[89] S. McIlraith, T. Son, and H. Zeng, "Semantic Web Services," *IEEE Intelligent System,* vol. 16(2), pp. 46-53, 2001.

[90] J. Gonzaliz-Castillo, D. Trastour, and C. Bartolini, "Description logics for matchmaking of services," in *the KI-2001 workshop on applications of description logics*, Vienna, Austria, 2001.

[91] M. Paolucci, T. Kawamura, T. R. Payne, and K. P. Sycara, "Semantic matching of Web services capabilities," in *the international Semantic Web conference*, Sardinia, Italy, 2002, pp. 333-347.

[92] A. Dogac, Y. Tambag, P. Pembecioglu, S. Pektas, G. Lateci, G. Kurt, S. Toprak, and Y. Kabak, "An ebXML infrastructure implementation through UDDI registries and RosettaNet PIPs," in *the 2002 ACM SIGMOD international conference on Management of data*, Madison, Wisconsin, 2002, pp. 512-523.

[93] M. Keidl and A. Kemper, "Towards context-aware adaptable web services," in *the 13th international WWW conference on Alternate track papars and posters*, New York NY USA, 2004.

[94] S. Ran, "A model for web services discovery with QoS," *ACM SIGecom Exchange,* vol. 4, pp. 1-10, 2003.

[95] B. Benatallah, M. Hacid, A. Leger, C. Rey, and F. Tourmani, "On automating Web services discovery," *VLDB Journal,* vol. 14, pp. 84-96, 2005.

[96] M. Solanki, A. Cau, and H. Zedan, "Semantic web services: Augmenting semantic web service description with compositional specification," in *the 13th international conference on World Wide Web* New York NY USA, 2004, pp. 544-552.

[97] B. Grosof, I. Horrocks, R. Volz, and S. Decker, "Description logic program combining logic programs with description logic," in *the 12th international conference on World Wide Web*, Budapest, Hungary, 2003, pp. 28-37.

[98] E. Maximilien and M. Singh, "Reputation and endorsement for web services," *ACM SIGecom Exchange,* vol. 3, 2002.

[99] H. Davulcu, M. Kilfer, C. R. Ramarkrishnan, and I. V. Ramarkrishnan, "Logic based modeling and analysis of workflows," in *ACM symposium on Principles of Database Systems*, 1998, pp. 25-33.

[100] A. Ferrar, "Web services: a process algebra approach," in *the 2nd international conference on service oriented computing*, New York USA, 2004, pp. 242-251.

[101] M. Singh, "Semantical considerations on workflows: An algebra for intertask dependencies," in *workshop on database programming langauge (DBPL)*, 1995.

[102] J. A. Fisteus, L. S. Fernandez, and C. D. Kloos, "Applying model checking to BPEL4WS business collaboration," in *the 2005 ACM symposium on Applied computing*, Santa Fe, New Mexico USA, 2005, pp. 826-830.

[103] W. M. van der Alast, "On the automatic generation of workflow processes on product structures," *Computer in Industry,* vol. 39, pp. 97-111, 1999.

[104] H. Foster, S. Uchitel, J. Magee, and J. Kramer, "Model-based verification of web service compositions," in *the 13th international conference on World Wide Web*, New York, USA, 2003, pp. 621-630.

[105] X. Fu, T. Bultan, and J. Su, "Analysis of interacting BPEL web services," in *the 13th international conference on World Wide Web*, New York USA, 2004, pp. 621-630.

[106] R. Hull and J. Su, "Tools for composite web services: a short overview," *ACM SIGMOD record,* vol. 34, 2005.

[107] "PSL standards group," *http://ats.nist.gov/psl*.

[108] K. Bhar, C. Fournet, and A. D. Gordon, "Verifying policy-based security for web services," in *the 11th ACM conference on Computer and communication security*, Washington DC USA, 2004, pp. 268-277.

[109] "Web Service Security (WS-Security) version 1.1," *http://www-128.ibm.com/developerworks/webservices/library/ws-secure*.

[110] G. Shani, D. Heckerman, and R. Brafman, "An MDP-Based Recommender System," *Journal of Machine Learning Research,* vol. 6, pp. 1265-1296, 2005.

[111] M. J. Chung, W. Kim, H. S. Jung, R. Gopalan, and H. Kim, "Service Model for Collaborating Distributed Design and Manufacturing," in *WWW 2004 Workshop on Application Design, Development and Implementation Issues in the Semantic Web*, New York NY, 2004.

[112] W. Kim, M. J. Chung, S. C. Kim, and H. Kim, "Service Oriented Collaboration in Design and Manufacturing Process," in *FIDJI2004*, Luxembourg, 2004.

[113] M. J. Chung, W. Kim, H. S. Jung, and H. Kim, "Web Service Based Process Management Model for Collaborative Product Commerce," in *the 10th International Conference on Concurrent Enterprising*, Seville, Spain, 2004.

[114] W. Kim and M. J. Chung, "A Web Service Support to Collaborative Process with Semantic Information," in *the 6th international conference on Web Information System Engineering (WISE'05)*, New York, NY USA, 2005, pp. 217-230.

[115] M. J. Chung, H. S. Jung, W. Kim, R. Gopalan, and H. Kim, "A Framework for Collaborative Product Commerce Using Web Services," in *IEEE International Conference on Web Services (ICWS 2004)*, San Diego CA USA, 2004.

[116] W. Kim, M. J. Chung, and J. Lloyd, "Automated Outsourcing Partnership Management Using Semantic Web Service," *Computer Supported Cooperative Work in Design II, Lecture Notes in Computer Science 3865,* pp. 184-193, 2006.

[117] W. Kim and M. J. Chung, "Collaboration in Design and Manufacturing Process Using Web Services Semantics," in *the 9th International Conference on CSCW in Design (CSCWD'05)*, Coventry, United Kingdom, 2005, pp. 247-252.

[118] B. Meyer, "Building bug-free o-o software: An introduction to design by contract: Object Currents," *SIGS Publication,* vol. 1(3), March 1996.

[119] M. D. Sadek, "Dialogue acts are rational plans," in *ESCA/ETRW Workshop on the Structure of Multimedia Dialogue*, Maratea, Italy, 1991, pp. 1-29.

[120] R. G. Smith, "The contract net protocols: High-level communication and control in a distributed problem solver," *IEEE Transactions on Computers,* vol. C-29, pp. 1104-1113, 1981.

[121] W. J. v. d. Heuvel and Z. Maamar, "Intelligent Web services moving toward a framework to compose," *Communications of the ACM,* vol. 46, 2003.

[122] T. Finin, R. Fritzson, D. McKay, and R. McEntire, "KQML as an agent communication language," in *the third international conference on Information and knowledge management* Gaithersburg, Maryland, United States 1994, pp. pp 465-463.

[123] H. Weigand and W. J. v. d. Heuval, "Meta-patterns for electronic commerce transactions based on the formal language for business communication (FLBC)," *International Journal of Electronic Commerce,* vol. 2, pp. 45-66, 1999.

[124] M. J. Wooldridge, "Agent-based software engineering," *IEEE Proceedings Software Engtineering,* vol. 144, pp. 26-37, 1997.

[125] F. Bergenti and A. Rocci, "Coordination models, languages and applications: Three approaches to the coordination of multiagent systems," in *the 2002 ACM symposium on Applied computing*, Madrid, Spain, 2002, pp. 367 - 372.

[126] M. Schumacher, "Objective Coordination in Multi-Agent System Engineering-Design and Implementation," *LNAI,* vol. 2039, 2001.

[127] K. Bach and R. M. Harnish, *Linguistic Communication and Speech Acts*: MIT press, 1979.

[128] J. Shearle, "In direct Speech Act," *Syntax and Semantics: Speech Acts,* vol. 3, pp. 59-82, 1975.

[129] Y. Labrou and T. Finin, "Semantics for an agent communication language," *LNCS,* vol. 1365, pp. 209-214, 1998.

[130] "Dependabily," *http://en.wikipedia.org/wiki/Dependability*.

[131] A. Aviziecaron, "Fault Tolerant Systems," *IEEE Transaction of Computer,* vol. C-25, pp. 1304-1311, 1976.

[132] T. Hofmann, "Latent Semantic Models for Collaborative Filtering," *ACM Transactions of Information Systems (TOIS),* vol. 22, pp. 89-115, 2004.

[133] T. Hofmann, "Probabilistic Latent Semantic Indexing," in *the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, Berkeley, California, 1999, pp. 50-57.

[134] A. Dempster, N. Laid, and D. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society, Series B,* vol. 39, pp. 1-38, 1977.

[135] R. Hogg, J. Mckean, and A. Craig, *Introduction to Mathmatical Statistics*. Upper Saddle River, NJ: Pearson Prentice Hall, 2005.

[136] R. Neal and G. Hinton, *A view of the EM algorithm that justifies incremental, sparse, and other variants*. Cambridge, MA: MIT Press, 1999.

[137] "MovieLens," http://movielens.umn.edu.

[138] H. Davulcu, M. Kifer, and I. V. Ramakrishnan, "CTR-S: a logic for specifying contracts in a semantic web services," in *International World Wide Web Conference*, New York, NY USA, 2004, pp. 144-153.

[139] A. Gupta, L. Zhang, and S. Kalyanaraman, "Simulations for risk management: a two component spot pricing framework for loss-rate guaranteed internet service contracts," in *Winter Simulation Conference*, New Orleans, Lousiana, 2003, pp. 372-380.

[140] L. G. Meredith and S. Bjorg, "Service-oriented Computing: Contracts and types," *Communications of the ACM,* vol. 46, pp. 41-47, 2003.

[141] J. Claessens, B. Preneel, and J. Vandewalle, "(How) can mobile agents do secure electronic transactions on untrust hosts? A survey of the security issues and the current solutions," *ACM Transactions on Internet Technology (TOIT),* vol. 3, pp. 28-48, 2003.

[142] S. Pleisch and A. Schiper, "Approaches of fault-tolerant and transactional mobile agent execution-an algorithmic view," *ACM Computing Survey (CSUR),* vol. 36, pp. 219-262, 2004.

[143] G. Huszerl, I. Majzik, A. Pataricza, K. Kosmidis, and M. D. Cin, "Quantitative Analysis of UML Statechart Models of Dependable Systems," *The Computer Journal,* vol. 45, 2002.

[144] F. Bause, P. Buchholz, and P. Kemper, "Hierarchically combined queueing Petri nets," in *11th International Conference on Analysis and Optimization of SystemsL Discrete Event Systems*, Sophia-Antipolis, France, 1994, pp. 176-182.

[145] M. Bernardo and R. Gorrieri, "Extended Markovian process algebra," in *7th International Conference on Concurrency Theory (CONCUR'96)*, Pisa, Italy, 1996, pp. 315-330.

[146] S. Donatelli, J. Hillston, and M. Ribaudo, "A comparison of performance evaluation process algebra and generalized stochastic Petri nets," in *6th International Workshop on Petri Nets and Performance Models (PNPM'95)*, Duke University, NC, 1995.

[147] M. J. Chung, W. Kim, H. S. Jung, and H. Kim, "A Service-oriented Framework for Collaborative Product Commerce," in *the 8th International Conference on Computer Supported Cooperative Work in Design (CSCWD 2004)*, Xiamen, China, 2004.

[148] W. Kim, M. J. Chung, K. Querishi, and Y.-K. Choi, "WSCPC: An Architecture Using Semantic Web Services for Collaborative Product Commerce," *Computers in Industry,* vol. 57, pp. 787-796, 2006.

[149] "Steel Founder's Society of America," *www.sfsa.org,* 2004.

[150] M. J. Chung, P. Kwon, B. Pentland, and S. Kim, "A process management system for collaborative manufacturing," in *IMECE'02, Symposium on Reconfigurable Manufacturing Systems, 2002 ASME International Mechanical Engineering Congress & Exposition*, New Orleans, Louisiana, 2002.