Abstract

Web services have been recognized to have the potential to revolutionize e-commerce. The potential for businesses to be able to interact with each other on the fly is very appealing. To date, however, the activity of creating Web processes using Web services has been handled mostly at the syntactic level. Current composition standards focus on building the process based on the interface description of the participating services. The limitation of a rigid approach is that it does not allow businesses to dynamically change partners and services. We enhance the current process composition techniques by using Semantic Process Templates to capture the semantic requirements of the process. The semantic process templates can act as configurable modules for common industry practices maintaining the semantics of the participating activities, control flow, data flow, intermediate calculations and exposing it in a industry accepted interface. The templates are instantiated to form executable processes according to the semantics of the corresponding templates. The use of ontologies in template definition allows much richer description of activity requirements and more effective way of locating services to carry out the activities in the executable Web process. During discovery of services we consider not only functionality, but also the QoS of the services. Our unique approach combines the expressive power of the present Web service composition standards and the advantages of the semantic Web techniques for template definition and service discovery. The prototype implementation of the framework for building the templates and generating the processes is discussed.

Keywords: Web Services Composition, Semantic Web Process, Semantic Web Service Discovery, Semantic Process Templates

1. Introduction

There has been significant excitement over the promise of Web services. The existence of standardized software components over the Internet that can be accessed, described and registered using XML based protocols could lead to powerful applications spanning the Internet. From the perspective of e-commerce, the idea of creating dynamic business processes on the fly (described as “dynamic trading processes” in [1]), would allow corporations to enable full-scale business process integration [2] further leveraging the power of the World Wide Web. There has been a flurry of activity in the area of Web services and in ways to assemble these Web services to create Web processes. Web Processes represent next generation technology for carrying out core business activities, such as e-commerce and e-services, and are created from the composition of Web Services [3] or other software components. Web processes encompass the ideas of both inter and intra organizational workflow. While there has been significant progress in this
area, there are a number of factors that prevent the wide scale deployment of Web services and creation of Web processes. The most inherent problems concern describing and discovering Web services. The current solutions and standards take a structural approach to describing Web services using XML based descriptions [4]. The main problem with this approach is that, it is not possible to explicitly define the meaning intended by the Web service provider. "Formally self-described" [5, 6, 7, 8] semantic Web services are a solution to semantically describe and discover Web services. Most of the composition standards build on top of Web service description standards. Hence semantically describing a service could result in composing a process whose individual components are semantically described. When all the tasks involved in a Web process are semantically described, we may call such process as Semantic Web Processes [9].

As part of the METEOR-S project at the Large Scale Distributed Information Systems (LSDIS) Lab at the University of Georgia, we are using techniques from the semantic Web [10], semantic Web services [11] and earlier research in workflow management as part of the METEOR project [12, 13] to deal with the problems of semantic Web service description, discovery and composition. In particular, the METEOR-S project associates semantics to Web Services, covering input/output, functional/operational descriptions, execution and quality, and exploits them in the entire Web process lifecycle encompassing semantic description/annotation, discovery, composition and enactment (choreography and orchestration) of Web services. The current emphasis of the METEOR-S project has been on semantic annotation of Web services [14], semantic discovery infrastructure for Web services (MWSDI: METEOR-S Web Service Discovery Infrastructure) [15] and semantic composition of Web services (MWSCF: METEOR-S Web Service Composition Framework). This paper focuses on MWSCF.

MWSCF aims at using the power of Web services to allow corporations to create processes that mirror today’s dynamic and ever-changing business needs. Corporations can expose their application software as Web services so that other corporations can dynamically find and invoke them. In order to precisely define a business or workflow process, several process specification languages have been created. These standards are based on WSDL descriptions. Most of these languages have focused on representing the semantics of user requirements, so that partners and Web services can be dynamically discovered before the executable process is created. We have defined Semantic Process Templates (SPTs), which allow us to semantically define each activity involved in a process. With an SPT, a process can be generated that will have a concrete Web service implementation for each activity preserving their semantics. The main focus of this paper is one aspect – that of creating semantically enriched process templates, which can be refined into concrete executable processes based on the users requirements. The most challenging problem in creating a process template is to capture the semantic capabilities of activities in the process, so that relevant Web services can be bound to activities of the process. We extend our previous efforts on semantic Web service description and discovery to describe the process template and to discover relevant Web services and to generate the executable process based on the discovered services. Using the approach stated and framework discussed in this paper, semantic Web processes can be more
effectively designed. This is a straightforward approach that is compatible with present industry standards for Web services.

The key features and contributions of this paper are the following:

- A comprehensive framework for composition of SWPs
- Using process templates to store semantics of each activity in an SWP, and
- Process generation and dynamic discovery of services based on semantics of each activity in the SWP during process generation.

The rest of the paper is organized as follows. Section 2 discusses three of the most popular composition standards. We also present a sample of a typical Web process that will be used for illustrative purposes. Section 3 presents a detailed discussion of the components making up our composition framework. It also briefly summarizes the steps involved in composing a semantic Web process. Key features of MWSCF are highlighted in section 4. Section 5 discusses related work. Conclusions and future work are presented in section 6.

2. Background study

In this section, we first present an overview of the present composition standards. Later, we provide a sample Web process and explain the implementation with respect to each of the presented composition standards.

2.1. Overview of Present Standards for Process Specification

To enable enterprise application integration (EAI) and business process integration within and across organizations, the importance of the languages for Web service composition has increased. So have the number of proposals for the Web service composition standards proposed by different vendors, organizations and consortia. These standards propose XML based open standards for assembling a number of Web services to form a business process. The standards that are currently being considered for building processes using Web service composition include (among others) BPEL4WS [16], BPML [17] and DAML-S [5]. In spite of the fact that all of these standards aim to solve the problems related to process description, they differ in many aspects. Comparing these standards requires an in-depth study of application scenarios of the composed processes and the support of these standards for each of the scenarios. [18] compares these standards using a framework with a set of patterns that are representative of the recurring situations found in different workflow management systems and in the context of enterprise application integration. Comparison based on other criteria is available in [19, 20, 21, 22, 23, 24, 25]. The following sub-sections provide an overview of three major proposed standards; interested readers are referred to the above comparative studies for further information.
2.1.1 BPEL4WS

The Business Process Execution Language for Web Services (BPEL4WS) [16] is a language to specify business processes and business interaction protocols. It superseded XLANG [26] and WSFL [27] as a standard for Web services flow specification. The model and XML-based grammar provided by BPEL define the interactions between a process and its partners using Web services interfaces. BPEL also defines the states and logic of coordination between these interactions and systematic ways of dealing with exceptional conditions. The business interaction protocols are called abstract processes. They are used to specify public and visible message exchange between different parties involved in a business protocol, but they do not reveal the internal behavior or the implementation of the involved parties. The executable processes on the other hand are like workflow descriptions represented using basic and structured activities specifying a pattern of execution of Web services. The process model defined by BPEL is based on the WSDL service description model. The services (described as partners in BPEL) that are invoked by the process and the services that invoke the process are represented using their WSDL description. An executable process can be a Web service by itself and the interface of that process can be represented using WSDL.

2.1.2 BPML

The Business Process Modeling Language (BPML) [17] is based on an abstract model and grammar for expressing abstract and executable business processes. Using BPML, enterprise processes, complex Web services and multi-party collaborations can be defined. A process in BPML is a composition of activities that perform specific functions. The process directs the execution of these activities. It can also be a part of another composition by defining it as a part of a parent process or by invoking it from another process. Each activity (both simple and complex) in the process has a context, which defines common behavior for all activities executing in that context. Hence a process can be defined as a type of complex activity that defines its own context for execution. The BPML specification defines 17 activity types, and three process types. The different process types are nested processes which are defined to execute within a specific context and whose definitions are a part of context definition, exception processes to handle exceptional conditions in executing a parent process and compensation processes to provide compensation logic for their parent processes. Each process definition may specify any of the three ways of instantiating a process: in response to an input message, in response to a raised signal, or invoked from an activity. BPML specifications support importing and referencing service definitions given in WSDL. It also suggests standardizing BPML documents by using RDF for semantic meta-data, XHTML and the Dublin Core metadata to improve human readability and application processability.

2.1.3 DAML-S

The DAML-based Web Service Ontology (DAML-S) [5] is an initiative to provide an ontology markup language expressive enough to semantically represent
capabilities and properties of Web services. DAML-S is based on DAML+OIL and the aim is to discover, invoke, compose, and monitor Web services. It defines an upper ontology appropriate for declaring and describing services by using a set of basic classes and properties. In DAML-S, each service can be viewed as a Process and its Process Model is used to control the interactions with the service. Using the Process Ontology’s sub-ontologies, Process Ontology and Process Control Ontology, it aims to capture the details of the Web service operation. The Process Ontology describes the inputs, outputs, preconditions, effects, and component sub-processes of the service. Process Control Ontology is used to monitor the execution of a service. However, the current version of DAML-S does not define the Process Control Ontology. DAML-S also categorizes three types of processes. The first type is an atomic process, which do not have any sub-processes and can be executed in a single step. The second type is a simple process, which is not invocable as it is used as an abstraction for representing an atomic or composite process. A composite process is the third type, which is decomposable into sub-processes. A composite process uses several control constructs to specify how inputs are accepted and how outputs are returned.

There is a need to compare and analyze the features of these languages in detail to frame a single and powerful and interoperable standard for composing processes. Web processes should be dynamic and flexible enough to adapt to the changes in demands from customers or market forces. To meet this requirement, BPEL and BPML, abstract the service references in the process from actual service implementations. This helps in selecting a correct service implementation for each activity during process deployment (deployment-time binding) or execution (execution-time binding). However, the present composition standards like BPEL and BPML are lacking in an important aspect, semantically representing the activity components of a process. BPML only suggests adding semantics to the process description.

We handle this problem by capturing semantics of the involved activities in the process template. These semantic descriptions are not bound to Web service implementations. Before deployment of the process, the services that satisfy the semantic requirements are discovered and bound to the activities in the process template. Using the interfaces and message types supported by the services, an executable process is generated which is used for deployment. Any process management system will demand a powerful discovery algorithm integrated into the process management system. This demand becomes critical when the size and the number of services available in the Web are taken into consideration. Our approach supports precise semantics based discovery of Web services. The following section details a sample business process. Composition of this process using our framework is discussed in the subsequent sections.

2.2 Sample Web Process

Let us consider an example for a typical Web process in the e-business domain, an electronic toy manufacturer processing a distributor's order. The manufacturer hosts an application (getOrderPriceAndDeliveryDate) where the distributor can query for the price and date of delivery by specifying an order. The manufacturer, upon receiving the
order details, processes the order and returns the details that the distributor is querying for. Based on the details returned by the application, the distributor can place the actual order \((\text{placeOrder})\) keeping the returned details as the agreement between the manufacturer and the distributor. The applications, \(\text{getOrderPriceAndDeliveryDate}\) and \(\text{placeOrder}\), have several intermediate steps, which are to be carried out by services within and outside the Manufacturer’s organization. For brevity, we have considered only the \(\text{getOrderPriceAndDeliveryDate}\) application in our example. In the \(\text{getOrderPriceAndDeliveryDate}\) application, when the distributor places an order, the manufacturer checks the inventory to verify if it has enough goods to satisfy the order. Based on the price returned by the delivery partner and the fix price methods, the price for the order is finalized. Then, the delivery date and the finalized price are returned to the distributor. In the other case, when there is not enough stock in the inventory, the manufacturer contacts its supplier partner for the required components. Then the manufacturer contacts its delivery partner to arrange for delivery of components to its manufacturing site and later to deliver the products to the distributor. The price and delivery date are returned to the distributor. The following figure depicts this process in detail.

![Figure 1: Sample Web Process](getOrderPriceAndDeliveryDate)

This above-mentioned procedure for order processing can be implemented as an SWP. This process may span across several organizations. In this example, the process design will happen in the Manufacturer’s organization. When the process is statically composed as a workflow, then the supplier partner and delivery partner services are decided before hand and integrated in the process along with manufacturer’s intra organizational services such as fix price, assembly line, inventory, etc. Due to the dynamic nature of the business, hard coding the business logic and participating services may not be efficient. This process should be able to be integrated with any potential

---

\(1\) I, O respectively represent inputs and outputs of each activity. Rec and Rep are the inputs to be received by the process and output produced by the process
delivery partner or supplier partner. The present standards support this kind of design, but they impose a restriction on the interface provided by these potential partners. In BPEL, it is assumed that if a service needs to be a part of a process instance, then it should provide the interface specified by the WSDL’s port type\(^2\) construct in the process definition. We attempt to solve this problem by specifying the process using templates during process design. Such templates are independent of the service description and process definition standards. Hence any service that satisfy the semantic requirements of the activity can be used to carry out that activity in the process. Before execution, the process templates are used to instantiate an executable process in any of the process definition standards and executed accordingly.

The idea of customizable processes and using process templates has been discussed earlier in [1]. It proposes three architectures/modalities for managing inter-organizational business processes. One of the architectures envisions Process Portal hosted by an enterprise or an organization for its customers. It manages a variety of customizable processes in which a subscribing company or a trading partner might do an individual activity. The second architecture is the Process Vortex for specialized markets where interactions are controlled by some third party. The business processes in the process vortex are designed to incorporate different trading models and they are available as templates that can be used to customize processes. The Dynamic Trading Process architecture defined as the third architecture is a virtual market place for different products spanning across multiple industries. In this architecture, processes can be constructed based on customer's needs. It supports flexible and dynamic trading processes that are composed upon requests from customers and are based on the QoS requirements specified by the customer. The framework discussed in our present work can be used to design processes in any of these modalities. In our system, the processes can be defined using semantic templates and the users of the process or provider of the process can customize and generate executable process.

The next section explains the METEOR-S composition framework and demonstrates how it can be used to design the getOrderPriceAndDeliveryDate SWP.

3. METEOR-S Web Service Composition Framework (MWSCF)

This section describes the MWSCF. There are four major components in MWSCF: the Process Builder, the Discovery Infrastructure (MWSDI), XML repositories and the Process Execution Engine. The process designer constitutes a builder and a process generator. It provides a graphical user interface to design/open process templates and provides a process generator, which uses MWSDI and data in XML repositories to convert the template into an executable process. The METEOR-S Web Service Discovery Infrastructure (MWSDI) is used to access a community of Web service registries and semantically search for Web services. The generated executable process is then executed using a process execution engine. The XML repositories in the architecture are used to store ontologies, activity templates and process templates. The details of each of these

\(^2\) PortType construct in WSDL is used to group operations
components are discussed in the following sections. Figure 2 shows the overall architecture of MWSCF.

![Figure 2: Web Service Composition Framework](image)

### 3.1 METEOR-S Web Service Discovery Infrastructure (MWSDI)

Web services are advertised in registries. The initial focus of Universal Description, Discovery and Integration [28] specifications was geared towards working with a Universal Business Registry (UBR), which is a master directory for all publicly available Web services. However, the new version of the UDDI specification [29] recognizes the need for existence of multiple registries and the need for interactions among them. A large number of registry/repository implementations for electronic commerce, each focusing on registering services of interest to respective sponsoring groups, are also anticipated. Hence, the challenge of dealing with hundreds of registries (if not thousands) during service publication and discovery becomes critical. Searching for a particular Web service would be very difficult in an environment consisting of hundreds of registries. This search would involve locating the correct registry in the first place and then locating the appropriate service within that registry.

Finding the right services would be easier if the registries were categorized based on domains with each registry maintaining only the Web services pertaining to that domain. If the registries are specialized like this, search for services in that domain can be carried out in a relevant registry. In addition, adding semantics to the domain-registry association will help in efficiently locating the right registries based on query requirements. In MWSDI, we use a specialized ontology called the Registries Ontology, which maintains relationships between all domains in MWSDI, and associates registries to them (see [15] for details).
Improving service discovery also involves adding semantics to the Web service descriptions and registering these descriptions in the registries. Adding semantics to Web service descriptions can be achieved by using ontologies that support shared vocabularies and domain models for use in the service description. Using domain specific ontologies, the semantics implied by structures in service descriptions, which are known only to the writer of the description (provider of web service), can be made explicit. Hence, while searching for a Web service, we can refer to relevant domain specific ontologies to enable semantic matching of services. MWSDI provides support for this kind of matching by relating both Web service descriptions and user requirements to ontologies. It also provides an infrastructure for accessing multiple registries. The registries are provided by different registry operators and they may support their own domain specific ontologies for their registries. The registries may also want to offer their own version of semantic publication and matching algorithms. Along with that, each operator may also provide value added services for the registry users.

We have implemented MWSDI to demonstrate this scalable infrastructure of Web service registries for semantic publication and discovery of services. It is implemented as a P2P network of UDDI registries. The MWSDI prototype system allows different registries to register in a P2P network and categorize registries based on domains. These registries in turn support domain specific ontologies and provide value added services for performing registry operations. MWSDI supports semantic publication of services. The inputs and outputs of the services are semantically annotated and these annotations are captured in UDDI. To perform semantic discovery according to the original implementation, the users can annotate the inputs and outputs of the service requirements and the discovery process in a UDDI will result in the services that match these semantic requirements. This discovery algorithm has been extended later in [14] that also supports annotating each operation in a WSDL file with a concept in functional ontology along with the annotation of preconditions and effects of that operation. Hence during discovery the service requirements are semantically annotated by associating it with concepts in ontologies that represent operation, inputs, outputs, preconditions and effects of the service. The discovery mechanism supported in MWSCF is based on all these kinds of annotations in addition to input and output semantics. Using the MWSDI with the semantic publication and discovery algorithms can significantly improve upon the current standards in Web service registration and discovery. MWSDI provides the flexibility to search for Web services based on ontologies. The user when designing a process may specify the discovery details of an activity by using the registry ontology (discussed in [15]). MWSDI architecture has been implemented on a cluster of SUN workstations as peer-to-peer network using the JXTA [30] framework. Xindice [31], a native XML database that comes with JWSDP [32] is used for implementing UDDI registries in MWSDI. UDDI4J [33] is used for accessing UDDI registries during publication and discovery.

3.2 Process Builder

The process Builder implemented in Java has a designer that assists in composing semantic Web processes. WSDL4J [34] has been used for processing WSDL files. The
Jena tool kit has been used to building and processing ontologies. The builder supports three different approaches to specify an activity. Each activity in the process can be specified using a

- Web service implementation,
- Web service interface, or
- Semantic activity template.

### 3.2.1 Specifying an Activity using a Web Service

Static composition of a process is done by specifying activities using concrete Web service implementations. This type of composition is discussed in one of our previous projects called SCET [35]. MWSCF allows linking an activity to a WSDL file and a relevant operation in it. For example, in the process discussed in section 2.2, the process designer knows the details of the intra organizational services like `checkInventory`, `fixPrice`, etc. These services can be used to carry out respective activities in a process. The process designer can link the activities to a WSDL file and an operation in it. Even if the service interface or implementation changes, as long as the URL of the WSDL and the name of the operation do not change, MWSCF can associate the activity with that operation. During process generation, the `portType` and `message` details are extracted from the WSDL and used in the generated executable process.

### 3.2.2 Specifying an Activity using a Web Service Interface

An activity can also be specified using a Web service interface. If an activity is linked to a Web service interface, during process generation, a concrete service that implements the interface could be used to carry out the activity. Only the services that implement the interface are discovered. Discovering services that implement interfaces has been suggested in [36]. UDDI is a registry and not a repository and hence it does not allow publishing the interface definitions. Popular or industry specific interfaces can be published in UDDI using tModels that will have references to the interface definitions. All the services that implement that interface will indicate the conformance to that interface by binding the `tModel` using `binding template` constructs in UDDI. During discovery process, the `tModel` that represent an interface, can be used and all the services that implement this interface can be discovered. In MWSCF, during the creation of process templates, process designer can specify activities using Web service interfaces. During process generation a service can be selected from the list of services that implement the interface. The `portType` and `message` details of the implementing services are retrieved and used during process generation. In MWSCF, these interfaces are stored in a XML repository. The interfaces in the interface repository are identified using the same `id` as that of the `tModels` in UDDI that represent each of these interfaces. The user while designing a process can browse (shown in figure 3) through these interfaces and select an interface and an operation in the interface to link to an activity. The identifier (same as `tModel id`) of the interface is used during a UDDI search to retrieve services that implement the interface.
The user in addition to specifying the interface and an operation also specifies the
discovery details and QoS requirements for that activity. The discovery details are the
details based on which UDDI registries can be queried. For example, the UDDI supports
searching for services based on name of the business or services (keywords, wildcard
character and qualifiers), categorization (in taxonomies), characterization (technical
fingerprint), etc. These details in addition to the interface details are used during
discovery of Web services for an activity. In another example, if a $tModel$ representing a
technical fingerprint is specified in the discovery details of an activity, then a $tModelBag$
is constructed using this $tModel$ and the $tModel$ that represents the interface. The
constructed $tModelBag$ is used during service search. The MWSDI specific details can
also be represented in this. The discovery details are given in an XML representation of
the API supported by UDDI specifications.

The QoS requirements may also be given in an XML file. The QoS details are
used in ranking the discovered services. The process designer can use this ranking to
select an appropriate service to carry out an activity. The details are given in section
3.2.7. If there is a standard interface for the Delivery service discussed in sample web
process in section 2.2, it can be cached in the XML repository. Then during the process
template design, the activity for delivery service can be specified using that interface.

### 3.2.3 Specifying an Activity using a Semantic Activity Template

In the third approach, the requirements for an activity are given using its semantic
c characteristics. In the previous approach (section 3.2.2), the activity is specified using an
interface, meaning that an operation in the interface to carry out the activity and the data
type (complex or simple) for input and output of the activity is stated as the activity requirements. When the activity is specified using a semantic activity template, the activity requirements given as the semantics of the IO along with the operational semantics of the activity are specified. The functional semantics of an activity, its IO, its preconditions and effects are represented using ontological concepts. The services that conform to these semantic characteristics are discovered and ranked. The users can select a service from the list of discovered services. This approach of specifying the activities assumes that all services are semantically annotated by which each operation in a WSDL file, its inputs, and outputs are mapped to ontological constructs in addition to having additional tags for preconditions and effects of the operations. With this methodology, capabilities of each operation in a WSDL file can be captured. These semantic details in a WSDL file could be published in a UDDI. Hence given a set of requirements based on these semantic details, the services that match these requirements could be more precisely found. A detailed discussion is presented in [14]. The data types of input and output are optional and could be used as a weighted component by the match algorithm. As in the previous approach, the discovery details and QoS requirements could be specified for the semantic activity template too. These discovery details are combined with the semantic requirements and search is performed within a registry. The QoS requirements may be used to rank the resulting services. The following figure shows the user interface window to specify an activity using semantic activity template.

![Figure 4: Semantic Activity Template Specification Interface](image)

In our example discussed in section 2.2, the activity for Delivery Service can be specified as a semantic template. The input and output of the activity can be represented
using a standard vocabulary or ontology. A domain ontology for the domain Cargo Services could be used for this requirement annotation. Our work like most other work in semantic Web research is based on rich framework for ontology engineering and re-use. The Cargo Services ontology may encompass concepts like Air Cargo Services, Cargo Insurance, Maritime Cargo Services, Rail Cargo Services, Trucking (taken from Yahoo directories). Hence using such ontology, the semantics of an activity are specified. During service discovery, the service implementations that used this ontology and annotated their descriptions will be semantically compared with the requirements and ranked.

3.2.4 Process Composition

When composing a generic process template the user gives the list of activities and control flow constructs to link the activities. After finalizing the template, the user can save the template for future use or can find service implementations for each activity in the template. Though the templates could be used during run-time to find services for each activity, at present MWSCF only support deployment time binding. This is because the initial implementation of MWSCF uses an engine that supports deployment time binding. Though the builder supports different approaches to design, the focus of this work is on the third approach where the process is composed by defining each activity using semantic templates. When all the activities in a process template are specified using semantic templates then it is called a semantic process template.

3.2.5 Semantic Process Template

The semantic process template is a collection of activities, which can be linked using control flow constructs. A sample process template in XML format is shown in figures 5 and 6.
<xml version="1.0" encoding="UTF-8"?>
  <process-template name="simple process">
    LSDIS-TopManufacturing = "uuid:fa6574b-49f4-a557-0f08-45fa635d8484"
    LSDIS-OrderPlacement = "uuid:fa6574b-49f4-a557-0f08-45fa635d8484"
    LSDIS-FunctionalCost = "uuid:fa6574b-49f4-a557-0f08-45fa635d8484"
    LSDIS-CargoServices = "uuid:fa6574b-49f4-a557-0f08-45fa635d8484"
    url = "http://www.w3.org/2001/XMLSchema#"
  </process-template>

  <sequence name="sequence-1">
    <receive name="receive-1">process-call-operation = "getDetails" /></receive>
    <invoke-activity name="InventoryCheckActivity" type="ServiceImpl">
      url-URL = "http://192.168.1.100/services/inventory/inventory_services.xml"
      operation-name = "checkInventory" />
    </invoke-activity>
  </sequence>

  <switch name="switch-1">
    <case condition = "inventory availability = 'in stock'">
      <sequence name="sequence-2">
        <invoke-activity name="QuerySupplierPart" type="SemanticTemplate">
          semantic-type = "semantic-1"
          qos-spec = "qos-1"
          ranking-weights = "ranking-1"
          discovery-URL = "http://192.168.1.100/services/inventory/inventory_services.xml"
        </invoke-activity>
      </sequence>
    </case>
  </switch>
</xml>
The process templates in MWSCF have BPEL-like syntax. For representing control flow the template, uses the BPEL constructs. First step in template creation is to create a WSDL that represents the description for the desired process is generated using a
WSDL editor. This WSDL is then linked to the process template. The template can be explained as follows:

- **Process-template** is the root element that will enclose the entire template definition. It has different attributes that represent different ontologies or other name spaces.
- The control constructs (like `<sequence>`, `<flow>`, `<switch>` etc.) are used to represent control flow in the template. They do not need translation and they are used as it is during the process generation phase.
- In the example discussed in section 2.2, the inputs and outputs of the respective activities `Receive Order Details` and `Return Details` are represented as messages in the WSDL file (shown in section 3.4) of the process. The `receive` and `reply` constructs in the template are linked to the WSDL description (`process-wsdl-operation`) and an `operation` in the description that the process designer creates. The messages that are to be received and returned by the process are also captured in that WSDL file. They will be translated to containers\(^3\) in the executable process.
- The `invoke-activity` elements in the process template are translated into corresponding `invoke` elements in the generated processes. The `invoke-activity` elements in the template are of three *types*:
  1. `ServiceImpl`, if the activity is specified using a concrete implementation,
  2. `WSInterface`, if the activity is specified using a Web service interface, and
  3. `SemanticTemplate`, if the activity is specified using an semantic activity template.

Based on the discovery\(^4\) (`semantic-spec`, `discovery-URL`, `tModel id`, `discovery-spec`) and QoS (`qos-spec`) criteria given for each of the activities, the relevant services to carry out each activity are discovered and selected (discussed in section 3.2.6). The `invoke` elements in executable process will have other details like `portType`, `operation`, `input` and `output containers`, etc. that are extracted from the WSDL description of the service that is selected to carry out the activity.
- Input and output `container` details of the `invoke` elements in the executable processes are generated from the data flow details provided by the user (shown in figure 11).
- The business protocol data (like `inventory-availability`) that are used in process control (like conditional statements) need not be explicitly assigned in the process template. Instead, during process generation, the user can map output from any of the participating service to that variable. These details are translated into `<assign>` and `<copy>` tags in the final generated executable process.

---
\(^3\) called Variable is BPEL version 1.1
\(^4\) `ServiceImpl` type does not need discovery as `wsdl-URL` and the `operation-name` to invoke are given in the template itself
Other details given by the user for discovery (semantic-spec, discovery-spec and qos-spec) and ranking (ranking-details) for activities during process design are also present in the process template under the criteria element. The discovery-spec element in the template (figure 6) refers to a XML representation of the API used to query for services that are categorized using geo3166-2 taxonomy. Since it is given in conjunction with an activity of type WSInterface, when finding Web services for the activity DeliveryPartnerService, all the services that implement the interface and categorized using the taxonomy are discovered and ranked. semantic-spec element is used to give the semantics of the activity. qos-spec is used to specify the QoS criteria of the activities. ranking-weights is used to assign weights to rank the discovered services. Detailed discussion on discovery and ranking is given in the following section.

### 3.2.6 Service Ranking and Selection

Service selection is a crucial aspect of composition. Hence the discovery algorithms in our system are supplemented with a good ranking scheme. The service selection is based on the discovery details for the activity provided by the user. The user could specify the discovery URL for each activity. This discovery URL could point to a market place registry, a private enterprise registry, a domain registry, or a Universal Business Registry. The service discovery will be carried out in that registry. Registry selection can also be based on the registries ontology as proposed in MWSDI. Since discovery could result in a large number of candidate services, we have implemented a ranking mechanism that will help the process composer to select an appropriate service. The ranking of services for each activity can be based on the semantic matching of activity requirements with the service specifications and on the satisfiability of service in terms of QoS requirements of the activity. Our approach requires that each service registered with UDDI is linked to a semantically annotated WSDL description and that the WSDL descriptions are linked to WSEL [27] (Web Service Endpoint Language) files that have the QoS details of all the operations in the service. WSEL is an XML format for the description of non-operational and behavioral characteristics of service endpoints, like quality-of-service or security properties. This specification is under development and at present no specification exists for this language. WSFL [27] specification envisions the need for this language and suggests using it in conjunction with an activity to describe endpoint properties and enable better matchmaking. We have taken this idea and linked each WSDL file to a WSEL file, which has the QoS specifications of the operations in the WSDL file.

If the activity is specified as a semantic activity template then the overall ranking of a service is the weighted arithmetic mean of the two dimensions. The first dimension is based on the Semantic Matching. The second dimension is the QoS criteria matching. Semantic matching can be considered in ranking only if the activity is specified semantically. However, the QoS based ranking can be done for both interface and semantics based specification of an activity. Matching semantics is done against the operations, inputs, outputs, preconditions and effects of the activity requirements with the
service details. The semantic matching ranking is based on the weights assigned by the user to the individual parts of the requirements. The user can assign the weights for each of the parts in the activity requirement namely operations, inputs, outputs, preconditions and effects. The assigned weights are normalized before calculation (at least one weight must be non-zero). During discovery the semantic criteria of the activity are matched against the semantic details of the services registered in UDDI. The weights corresponding to the matched semantics are used to rank the services. The formula to calculate the ranking value for semantic matching is given as follows:

$$M_S = \frac{\sum_{i=1}^{5} M_i W_i}{\sum_{i=1}^{5} W_i}$$

$M_S$: Semantic Matching Value
$i$: Index of the assigned weights ($0 - 5$)
$M_i$: Semantic Matching Value of $i^{th}$ criterion
$W$: Weight of the $i^{th}$ criterion

**Figure 7: Formula for Calculating Semantic Matching Value**

Let us consider a sample calculation using the semantic specifications of $QuerySupplierPartner$ given in the template in figure 6. The semantic requirements of that activity are given in semantic-spec element named semantic-1. It specifies that the candidate Web services for that activity should have an operation that conforms to the concept $getOrderDetailsForOrderToyParts$ in the ontology represented by name space $LSDIS-FunctionalOnt$ and takes two inputs and produces two outputs. It also specifies the semantics of the inputs and outputs using ontological constructs. From these specifications it can be understood that one of the inputs of the $QuerySupplierPartner$ should conform to the $OrderCount$ concept (class) in the $LSDIS-OrderPlacement$ ontology. The template refers to four different ontologies, namely, $LSDIS-ToyManufacturing$, $LSDIS-OrderPlacement$, $LSDIS-FunctionalOnt$, $LSDIS-CargoServices$ that are published in UDDI and identified by a $tModel id$. The same $tModel id$ is used to index the ontologies in the ontology repository.

For $QuerySupplierPartner$, the inputs are annotated as $LSDIS-ToyManufacturing:ToyIdentifier$ and $LSDIS-OrderPlacement:OrderCount$. Similarly the outputs and operation are also annotated. So the user can assign weights to inputs, outputs and operations. The semantics of preconditions and effects are not specified in the semantic requirement specification of $QuerySupplierPartner$ and hence weights are also not assigned for the precondition and effects. In this case the weights are assigned only to three parts ($i=3$) operations, inputs and outputs. The weights assigned in the template are 30, 20 and 50 respectively. If there is an operation in a WSDL file that takes two inputs and produces two outputs with the following semantics:
1. The functionality of the operation is mapped to \textit{LSDIS-FunctionalOnt:getDetailsForOrderToyParts}. The functional semantics of the service exactly matches with the functional semantics of the activity. Hence the semantic matching value $M_1$ is 1.

2. One of the inputs is annotated using \textit{LSDIS-ToyManufacturing:ToyIdentifier} and the other using \textit{LSDIS-OrderPlacement:OrderCount}. In this case the input semantics of the service exactly match to the inputs semantics of the activity. The semantic matching value $M_2$ is 1.

3. One of the outputs is annotated with \textit{LSDIS-CargoServices:PickupDate} and the other output is annotated with \textit{LSDIS-CargoServices:PickupLocationDetails} which is a concept in the ontology that is 2 levels up in the hierarchy created by subClassOf relationships. One of the outputs match exactly and the other output does not match exactly. For non-exact matches the semantic matching value is calculated using a linear function that decides the semantic matching value based on the subClassOf hierarchy. The final semantic matching value $M_3$ is the average of semantic matching values of the two outputs. In the example the value of $M_3$ is $(1 + 0.8)/2 = 0.9$. The subClassOf hierarchy and linear function used is this implementation can be extended with a better function that can be used to characterize two concepts separated in an ontology by a number of named relationship properties.

4. The value of $M_5$ is $(1 \times 50 + 1 \times 30 + 0.9 \times 20) / (50 + 30 + 20) = 0.98$.

The next dimension in ranking is based on the QoS requirements of each activity. Each activity can be linked to requirements specification that defines the QoS parameters of that activity regardless of the different design approaches discussed in section 3.2. For simplicity of discussion, our present work considers five different QoS parameters, which are the subset of QoS details that we have identified in one of our previous work [37]. The different QoS specification parts used in ranking are:

- Task Delay Time
- Task Process Time
- Task Realization Cost
- Task Reliability Measure

In addition to QoS the WSEL file can also support representing boundary values and possible values for the input and output parameters of each operation. Our present discovery method does not take boundary values into consideration. The following figure shows a sample WSEL file.

\footnote{For example if a service 'TakeOrder' has an operation that takes order for some product X and if the maximum order that it can take is for 1000, it can specify that the value for the input parameter "numberOfProducts" should be less than 1000. Also if the input takes any enumerated value set it could be specified in the WSEL file. These details help users in properly selecting appropriate services.}
When specifying activity in a process the designer can link a QoS requirement specification to it. The QoS requirements are specified with qualifiers. A sample QoS requirements specification is shown in the template in figure 6 under qos-spec element. Each of the QoS parameter in the requirements description is given a weight. For every service that is discovered, the QoS compatibility is checked and using the weight for each QoS parameter the ranking is calculated. The formula for calculating QoS matching of an activity is shown below:

$$M_Q = \frac{\sum_{i=1}^{4} M_i W_i}{\sum_{i=1}^{4} W_i}$$

- $M_Q$: QoS Matching Value
- $i$: Index of the assigned weights (0 – 4)
- $M_i$: QoS Matching Value of $i^{th}$ criterion
- $W$: Weight of the $i^{th}$ criterion

Considering the sample template and the activity named QuerySupplierPartner in it, the qos-spec indicates that there is only one QoS requirement and that delay-time for the activity should be less than 90 milliseconds. The QoS matching value of the candidate services are calculated using a matching function. If for a service the matching value is 0.9 then the value of $M_Q$ is calculated as $(0.9 * 25) / 25 = 0.9$

After calculating the ranking values ($M_S$ and $M_Q$) for the Semantic Matching and QoS dimensions, weights assigned by the user for each of these dimensions are used to calculate the overall ranking. The overall ranking value is the weighted arithmetic mean of the ranking values in each dimension. In the example activity QuerySupplierPartner, the weights for Semantic matching and QoS matching are respectively 75 and 25. The overall ranking value is hence calculated as $(0.98 * 75 + 0.9 * 25) / (75 + 25) = 0.96$. The services are ranked based on the calculated value and the process designer can select one from the list of services (see figure 10):
3.2.7 Process Generation

After designing the template, the process designer can generate an executable process. This involves finding services pertinent to each activity in the process, retrieving their WSDL file, and extracting relevant information to generate the process. The discovery of services is done for each activity independent of other activities. The builder helps in finding Web services for each activity in the process. The user can then link services to incorporate data flow in the process. For this purpose, after obtaining data link requirements between two services, the process generator assists the user in establishing explicit data flow link between output parameters of one service to the input parameters of the other.

The interface in MWSCF that assists in establishing data flow is shown in the figure 11. If the process creator wants to specify the data flow link between two activities, the respective Web service descriptions can be fed to the interface that can be used to establish the data flow in the process. The process creator has to explicitly state the mapping between output parameters of one service to the input parameters of the other.

The port type data extracted from the WSDL of the process (created by process creator), the data flow requirements obtained from the user and the control flow constructs in the templates are used to generate an executable process. The WSDL descriptions of the participating Web services are also retrieved. These WSDL files will be used during deployment time for binding. During the process generation phase the
process WSDL file\(^6\) is updated with the service link details. A sample executable BPEL process that is generated from the template shown in section 3.2.5 can be found in the appendix.

### 3.3 XML Repositories

In MWSCF, we have a pool of XML repositories that are used for managing (storing, searching and reusing/sharing) the following

- Ontologies which are used during annotation of services and annotation of semantic activity requirements,
- Semantic process templates which can be opened and edited using the process designer during process design, and
- Activity/service interfaces.

All data (ontologies, interface definitions and process templates) are based on XML and since they are meant for sharing and re-use, using XML repositories will be of much help in this regard. The ontologies in the repositories are identified using the tModelId of the published tModels that represent these ontologies. In another XML repository, the process templates are categorized and stored. The categorization (taxonomy or ontology) is stored in the same XML repository as that of the process templates. Each category in the categorization has an identifier. The XML repository for storing process templates has different collections based on these identifiers and the process template is categorized by storing the template in a particular collection that represents a category. There is also a interface repository that stores the WSDL interface definitions which can be browsed and selected to link to an activity during the process design. Using these repositories during process design, service discovery or process generation gives the user a powerful environment to compose a process. Xindice is used for implementing these XML repositories.

### 3.4 Execution Engine

The generated executable process can be executed in any respective execution engine. As the work to develop our own execution engine is underway, at present MWSCF has been implemented and tested using BPWS4J orchestration server. Deploying a process in BPWS4J [38] engine requires a BPEL file containing the process definition, a WSDL description\(^7\) for the process and the other WSDL descriptions of the web services that a part of the process. The WSDL descriptions of the participating services are needed for deployment time binding. Using the WSDL files of the participating services and other details, an in-memory model [39] of the process is created. Since the present release of BPWS4J does not allow deploying a process without a BPEL file, we have used the BPWS4J printer class to write the in-memory model of the process into a BPEL file. This file is then validated using the tool validator that comes

---

\(^6\) WSDL file describing the process. It is designed by the process designer. It has the details of the inputs, outputs and operations of the process

\(^7\) created by the process composer (shown in Figure 14)
with BPWS4J. Once it is validated, it is deployed in the BPWS4J engine and it can be invoked like any other Web service.

### 3.4. Summary of Steps in Semantic Process Composition

This section summarizes how Semantic Web Processes are designed, composed and executed.

1. The WSDL description for the desired process is generated using a WSDL editor. In the example discussed in section 2.2, the activities *Receive Order Details* and *Return Details* will be represented in the WSDL file of the process. The process template will be linked to this file. The process template shown in section 3.2.5 is linked to this file and the operation *getOrderPriceAndDeliveryDate*. This WSDL file can be annotated and published as a Web service.

```xml
<definitions
  name="sample"
  targetNamespace="http://lsdis.cis.uga.edu"
  xmlns="http://schemas.xmlsoap.org/wsdl"
  xmlns:soap="http://schemas.xmlsoap.org/soap/"
  xmlns:urn="http://www.w3.org/2001/XMLSchema"
>
  <message name="getDetailsRequest">
    <part name="OrderId" type="xsd:integer"/>
    <part name="OrderId" type="xsd:integer"/>
    <part name="LocationIdToDeliver" type="xsd:string"/>
  </message>

  <message name="getDetailsResponse">
    <part name="OrderPrice" type="xsd:double"/>
    <part name="PossibleOrderDeliveryDate" type="xsd:date"/>
  </message>

  <portType name="ArrayOfGetDetailsPort">
    <operation name="getDetails">
      <documentation>Distributor's can use this file to input some order on toys to get the price and possible price for the order</documentation>
      <input message="tns:getDetailsRequest"/>
      <output message="tns:getDetailsResponse"/>
    </operation>
  </portType>

  <message name="getDetails">
    <part name="getDetailsRequest">
      <part name="getDetailsResponse">
    </message>

  <portType name="getDetails">
    <operation name="getDetails">
      <documentation>This service is a business process in itself. clients just need to call this service to execute the process</documentation>
      <input message="tns:getDetailsRequest"/>
      <output message="tns:getDetailsResponse"/>
    </operation>
  </portType>

  <message name="getDetails">
    <part name="getDetails">
      <part name="getDetails">
    </message>

  <portType name="ArrayOfGetDetails">
    <operation name="getDetails">
      <documentation>
```
2. A process template is created or opened in the process designer.
3. Activities are added to the template and control flow constructs are added to the templates (if needed).
4. Each activity is semantically annotated and two XML files representing discovery details and QoS specifications are linked to them.
5. Services are discovered (using discovery criteria), ranked (using semantic matching and QoS criteria) and selected for each activity.
6. Data flows between services are established.
7. The executable process is generated by the process generator using the WSDL of the process, the process template and the WSDL files of the participating services.
8. The process is validated, deployed and it is ready for invocation.

The above-mentioned steps considered only specifying the activities using Semantic templates. However, any type of activity specification technique discussed in section 3.2 is allowed. Figure 13 shows different phases in the semantic Web process composition. In step 1 (design phase), the figure shows only designing a process using Semantic Activity Templates. Other approaches (specifying an activity using a Web service interface or a concrete Web service implementation) are not shown. Steps 2 and 3 depict that for each activity a service is to be selected and the data flow is established if needed. These steps are done repeatedly until all the activities are bound to a Web service implementation and data flow is established. After this, the actual executable process is generated.
4. Features of MWSCF

The increasing pressure from the market and competitors force the companies to always strive for efficiency and improvement in their processes. This creates the need for an environment that helps building, analyzing and executing processes that are integrated deeply into the business itself. This environment has to provide interactive features for building processes based on business requirements and application semantics. As a step towards this aim, we have completed an initial prototype implementation of the framework as described in section 3 that has the following features:

**Semantically Enriched Service and Process Descriptions**

This framework supports building processes using activities that are semantically described. This helps in building the overall process, preserving the semantics of each activity. Each activity will be achieved using a Web service. Binding a service to an activity may not be easy considering the number of services that will be available in the future and their heterogeneity. Present standards like BPEL and BPML provide limited forms of dynamic binding. These standards include `portType` descriptions in the process definitions. Hence, for an activity, only the services that implement the `portType` can be used. This type of hard coded binding is not always suitable for ever-changing e-business applications. Using semantic descriptions not only helps in efficiently finding relevant Web services to execute each activity, it also helps in reasoning about them. Using semantic templates optimized flow or process execution could be calculated and QoS trade-off analysis could be done.

**Configurable Processes**

MWSCF provides an architecture where the processes are represented using templates, which are not bound to concrete service implementations. This feature is present in almost all process modeling languages. Our designer tool, in addition, allows users to build a process template and configure each activity in the process template to build an executable process according to the requirements. Using the builder, users can open an existing process template and configure activities in that process. This improves the usability of generic processes for different situations and, in addition, it also enables personalization of the processes based on the configuration parameters. This will help in the success of new business models like outsourcing of Web processes. Off-the-shelf ready made template can be bought and used instead of building it from the scratch. Common processes can be made available for rent/lease using value added service features like easy configuration, friendly interface for configuration, etc. For example, let us consider a typical composite Web service example "Conference Booking". This is a Web process by itself as it involves several activities like booking travel tickets, booking hotel, registering for the conference, etc. An organization that specializes in this type of business can create an abstract Web process template. The clients of the organization can use this template to feed in parameters like QoS of each activity, ranking and discovery details, etc. to build the actual process. The organization can then execute this configured process in its own service execution engine and return the results back to user.
Considering another example, in intra-enterprise process that takes an order and delivers some product to the customer. Depending on the price that the customer is willing to pay for the delivery service QoS parameters of the task that is involved in product delivery can be configured so that the execution engine selects a relevant service that handles the product delivery order satisfying the required QoS criteria.

**Comprehensive Framework for Web Service Composition**

The MWSCF provides a comprehensive framework for Web Service composition. The MWSCA extends full life-cycle support for achieving semantic Web processes by providing an infrastructure to assist a user starting from service publication and discovery (MWSDI) to designing a process (Process Designer) and executing it (Execution Engine). The designer interface that we have developed as a part of MWSCF provides to easy-to-use interface. Using the built-in powerful discovery technique in the process builder, users can effectively compose a process. The framework provides interfaces where users can view the ranking of services and select the services that are appropriate for their requirements. A unique aspect of the framework is that it automatically retrieves the WSDL files for deployment and extracts information from them to generate the process. The generated executable process will preserve the semantics specified in the process template. All these user interfaces and features implemented, as a part of MWSCF will help in rapid and efficient development of Web processes.

**Adaptability to any Process Specification Standard**

The present implementation of MWSCF allows constructing process template with BPEL-like syntax. However, the generated executable process can be made independent of BPEL syntax. The process generator is implemented as a separate module independent of other modules in the MWSCF. Hence a pluggable process generator module can replace the existing module that generates BPEL process. If the new process generator module is able to generate executable processes in other specifications based on the template, then it can be used in the framework to generate a process following a different standard. In fact, the best case is when these modules co-exist to help the user generate processes in any standard.

5. Related Work

There have been few research efforts and commercial tools to automate business processes to dynamically compose Web services and to use semantic Web techniques in Web service composition. In this section, we will discuss the related work that deals with Web service composition and other research efforts that are related to our work.

The main focus of our paper is creating semantic process templates. There are a few papers that discuss process templates. For example, [40] discusses reusable process skeletons that implement some conversational logic. It argues that the notion of process templates and service libraries would help to speed up and ease the development of processes that incorporate B2B capability. In our work, we have discussed creating
templates and storing them in a repository. To ease the discovery and usage of these templates we have categorized them. Taxonomies of processes have been discussed in [41]. It discusses classifying process instances depending on their characteristics, such as outcome of the process instance, duration of execution, etc. Our work discusses classifying the process templates based on their semantic capabilities.

There are several systems that help in Web service composition. [42] discusses a platform where Web services are declaratively composed and executed in a peer-to-peer environment. It does not perform dynamic discovery. The user has to discover services and then define composition using state-chart diagrams. Then the composition can be executed in a P2P fashion with the help of peer software components coordinators for each constituent Web service. [43] discusses a case based reasoning mechanism for discovery of services to form a composite Web service. The participating services are selected based on the relationships between the services and the constraints involved. eFlow [44] is another system that allows composing, customizing and deploying e-services. The process is modeled as a graph that defines the control and data flow. The nodes represent services in the process. A node could be either an ordinary node which is statically bound to a service or it could be a generic node which is specified with a configuration parameter and a list of services out of which one is bound to the node during run time. They have the notion of process templates and categorizing them in a hierarchy. eFlow also supports dynamic discovery technique using a mediator and the associated rules. Unlike our system neither the discovery mechanism nor the process templates in these systems is semantic.

The fundamental assumption in our work of annotating process definition with ontologies is that the candidate services are semantically described and discovered. The need for semantics in service description has been discussed in quite a few of previous publications [5, 14, 45, 46]. DAML-S is a popular initiative in this direction to describe service capabilities using ontologies. A DAML-S based prototype for semi-automatic composition of services has been discussed in [47]. It provides a composer that will aid the user to select services for each activity in the composition and to create flow specifications to link them. Upon selecting a service, the services that can produce output that could be fed as the input of the selected service are listed after filtering based on profile descriptions. The user can manually select the service that he wants to fit in at a particular activity. After selecting all the services, the system generates a composite process in DAML-S. The execution is done by calling each service separately and passing the results between services according the flow specifications. Step-by-step composition of Web services using DAML-S descriptions has also been discussed in one of our previous papers [48]. It discusses using ontologies to solve the problems in discovery of Web services and to resolve the structural and semantic heterogeneity among these services. The discovery methodology and the set of algorithms discussed in this related work supports discovery of services based on functional requirements and operational metrics. Though all of these related work attempt to solve the same problem (semantic discovery and composition) as that our work, the approach is different. Our work, instead of DAML-S, is based on the industry standards for Web services namely WSDL, UDDI and BPEL4WS. Our work does not discuss sequential composition where
the service for an activity in the process is decided before deciding for the next activity. In our work, the entire composition can be defined as a template of semantically annotated activities, and the discovery of services for the activities need not be sequential.

We believe that our composition methodology is better than other present frameworks, because the richness needed in representing services and data in the e-business domain is captured well using ontologies. The use of ontologies to aggregate the products, services, processes and practices within the industry to realize successful net markets has been discussed in [49]. It argues that elements of commerce and relationships between them are used to model market places and identifies ontologies as the means to do that. It states that ontological engineering is the prime requisite for information and services aggregation. It encourages developing and using domain/industry specific ontologies. Representing the products and services using ontologies will help to understand them from the different viewpoints and roles within that domain/industry. Another related work [50] discusses developing a Universal Business Language. UBL aims to define Business Information Entities at a semantic level. It is something similar to an ontology with a few restrictions [51]. Like ontologies, UBL is aimed to model the real world focusing on a domain (business) to enable semantic interoperability. Our paper discusses using standard vocabularies/ontologies to markup process templates for better interoperability and process generation. [52] identifies semantics as one of the important aspect that B2B protocol standards aims to standardize. It also lists business content or vocabulary as one of the facets of semantics in B2B standards. In another related work, [53] states that interoperating services need to agree upon vocabularies, document formats and conversation definitions. They add that, in addition to this, agreement has to be there between various horizontal and vertical industry segments to use the standard vocabularies and conversations. [8] proposes Web Services Modeling Framework (WSMF) to enable flexible and scalable e-commerce using Web Services. It discusses a conceptual model for developing, describing and composing Web services. It advocates using semantic Web techniques to deal with the problems of heterogeneity and scalability in e-commerce. It also discusses different types and approaches for scalable mediation between trading partners in e-commerce. We realize the importance of the mediation mechanism to deal with inherent heterogeneity in an open and flexible environment. This itself is a separate research direction. Hence we have deferred that for future work. However, our work reiterates the need for using semantics both in service description and process description to help make the vision of universal interoperability in e-business a reality.

6. Conclusions and Future Work

Web services have created a major wave in the IT industry. Several standards are being proposed, consortia have been created and academic research has increased rapidly. The obvious reasons are the immense power of Web services with regards to e-business and the commercial value behind them. The Web services are indeed useful for easier, faster integration, good in terms of return of investment (ROI), establishing friction free markets, rapid value added assembly of services, etc. However, some of the inherent
problems of e-business like scalability and semantic interoperability are not solved by the service-oriented architecture provided by Web services. The convergence of ideas, findings, and results from various initiatives like ebXML, Semantic Web and Web services can bring about better solutions.

Our work is on applying Semantic Web techniques to design e-business processes. Applying [54] Semantic Web findings to Web services technologies and use them for e-business could revolutionize the existing business models and the way they are carried out. This idea has received much attention and support both from academia and industry. For example, [55] addresses these problems with a framework that aims to align concepts known from the Semantic Web and the ebXML initiative and [56] discusses using Web services for implementing business processes and the need for Business Process Management to use Web services for dynamic e-businesses.

Using Semantic Process Templates in MWSCF provides following advantages:

- It helps in rapid process composition. The discovery of services can be performed later and can be delegated to the system.
- It is not necessary to build a process from scratch. Templates can be configured and reused.
- Process design is flexible as it is independent of web service portTypes. The change in partner interface without change in the semantics of the interface does not affect the process template or the discovery of services.
- Process re-design is greatly facilitated.
- Ready made templates can act as business/reference models and could be re-used by different organizations that want to implement same process with different services.

Our work does not take into consideration the B2B protocol standards that are crucial for inter-enterprise collaboration and achieving public processes. Our work focuses on the private processes of the enterprises that need to incorporate external services. Typically business transactions occur using some B2B protocol standard [52] that defines the message formats exchanged, sequencing, security, etc. There are several domain specific (Rosetta Net [57]) and domain independent standards (ebXML BPSS [58]) that focus on different aspects of the B2B protocol. In addition, there are Web services standards (WSCI8 [59], WS-CS [60]) to define message exchange in a business collaboration and choreograph the activities in the collaboration. These standards aim to improve the stateless synchronous/uncorrelated asynchronous model of interaction supported by WSDL. A multi-party collaboration is carried out either by using a global controller that coordinates various activities on behalf of the involved parties, or using a mechanism [61] that links a data and messages from visible public processes to the private organizational processes. In any case this is outside the scope of a process

---

8 The convergence of complementary standards happened in the case of WSCI and BPML aims to provide comprehensive view of role of businesses in a collaboration and the flow of activities that characterize each business
specification language like BPML or BPEL\(^9\) and as well as outside the scope of our work too. We foresee the convergence of standards like BPEL, WSCI, ebXML to realize a powerful e-commerce model. This convergence in conjunction with Semantic Web techniques will revolutionize the way businesses are done over the Internet.

Future research directions that we are beginning to explore include the following:

- Specifying collaboration in the process template and using an algorithm [62] to check the compatibility of a service to engage in a conversation.
- Building own engine for better control over the process and performing sub-conversation in a process.
- Extending process templates to capture the behavior of the intended process at high level and providing direct mapping to the formalisms like state charts or Petri nets etc. to enable process verification [63] and simulation [64, 65].
- Using a template in conjunction with other template [66] and a related analysis.
- Incorporating important e-business aspects like negotiation [67], Service Level Agreements [68] and contracts [69] in template design and service discovery.
- Specifying goal definition as a part of process template. Goal definitions will represent a business goal in highlevel representation in some standard format like UML, and
- Investigating possibilities to represent preconditions and effects in a more expressive way.

References


\(^9\) Abstract processes in BPEL are aimed to describe business protocols that specify the sequencing of messages exchanged by one particular partner with its other partners to achieve a business goal. However, an abstract BPEL process defines the business protocol from the perspective of a single entity in the collaboration, while real world business collaboration need a peer-to-peer conversational model.


[28] Universal Description, Discovery and Integration of Web Services, http://www.uddi.org/


[40] Sayal M., Casati F., Dayal U., Shan M-C., Integrating Workflow Management Systems with Business-to-Business Interaction Standards, 18th International Conference on Data Engineering (ICDE’02), p. 0287.


Appendix A: Generated Process follows

The following listing shows an executable process generated from the template shown in figures 5 and 6. During process generation 2 files are created. The first file has the executable BPEL process and the second file is a WSDL file (sample-utils.wsdl in this example) that has definition for all the service link types that are given in partner elements.
A sample executable process that is generated is as follows:

```xml
<process name = "sample"
    targetNamespace = "http://lsdis.cs.uga.edu"
    xmlns:tns = "http://lsdis.cs.uga.edu"
    xmlns:act1="http://lsdis.cs.uga.edu:7001/axis/services/sample/axis/services/sample"
    xmlns:act2="http://lsdis.cs.uga.edu:7001/axis/services/sample/axis/services/sample"
    xmlns:act3="http://lsdis.cs.uga.edu:7001/axis/services/sample/axis/services/sample"
    xmlns:act5="http://decatur.cs.uga.edu:8080/axis/services/sampleSupplierService7/axis/services/sampleSupplierService7"
    xmlns:act7="http://lsdis.cs.uga.edu:7001/axis/services/sample/axis/services/sample"

<partners>
    <partner name = "caller" serviceLinkType = "tns:StringConcatSLT"/>
    <partner name = "provider-1" serviceLinkType = "tns-utils:Provider1SLT"/>
    <partner name = "provider-2" serviceLinkType = "tns-utils:Provider2SLT"/>
    <partner name = "provider-3" serviceLinkType = "tns-utils:Provider3SLT"/>
    <partner name = "provider-4" serviceLinkType = "tns-utils:Provider4SLT"/>
    <partner name = "provider-5" serviceLinkType = "tns-utils:Provider5SLT"/>
    <partner name = "provider-6" serviceLinkType = "tns-utils:Provider6SLT"/>
    <partner name = "provider-7" serviceLinkType = "tns-utils:Provider7SLT"/>
</partners>

<containers>
    <container name = "request" messageType = "tns:getDetailsRequest"/>
    <container name = "response" messageType = "tns:getDetailsResponse"/>
    <container name = "act1-request" messageType = "act1:checkInventoryRequest"/>
    <container name = "act1-response" messageType = "act1:checkInventoryResponse"/>
    <container name = "act2-request" messageType = "act2:arrangeDeliveryRequest"/>
    <container name = "act2-response" messageType = "act2:arrangeDeliveryResponse"/>
    <container name = "act3-request" messageType = "act3:fixPriceRequest"/>
    <container name = "act3-response" messageType = "act3:fixPriceResponse"/>
    <container name = "act4-request" messageType = "act4:finalizePriceRequest"/>
    <container name = "act4-response" messageType = "act4:finalizePriceResponse"/>
    <container name = "act5-request" messageType = "act5:orderToyPartsRequest"/>
    <container name = "act5-response" messageType = "act5:orderToyPartsResponse"/>
    <container name = "act6-request" messageType = "act6:arrangeDeliveryRequest"/>
    <container name = "act6-response" messageType = "act6:arrangeDeliveryResponse"/>
    <container name = "act7-request" messageType = "act7:assemblyServiceRequest"/>
    <container name = "act7-response" messageType = "act7:assemblyServiceResponse"/>
```
<sequence name = "sequence-1">
<receive name = "receive" partner = "caller" portType = "tns:samplePortType"
  operation = "getDetails" container = "request"
  createInstance = "yes"/>
<assign>
<copy>
  <from container = "request" part = "ToyId"/>
  <to container = "act1-request" part = "ToyIdentifier"/>
</copy>
<copy>
  <from container = "request" part = "ToyCount"/>
  <to container = "act1-request" part = "NoOfToys"/>
</copy>
</assign>
<invoke name = "InventoryCheckActivity" partner = "provider-1"
  portType = "act1:IntraOrgServicesPortType"
  operation = "checkInventory" inputContainer = "act1-request"
  outputContainer = "act1-response">
<assign>
<copy>
  <from container = "act1-response" part = "ToysCountToOrder"/>
  <to container = "act5-request" part = "OrderCount"/>
</copy>
<copy>
  <from container = "request" part = "ToyId"/>
  <to container = "act5-request" part = "IdentifierForToy"/>
</copy>
</assign>
<switch name = "switch-1">
<case condition = "bpws:getContainerData('act1-response', 'return') = 'low'">
<sequence name = "sequence-3">
<invoke name = "QuerySupplierPartner" partner = "provider-5"
  portType = "act5:SupplierPartnerPT"
  operation = "orderToyParts" inputContainer = "act5-request"
  outputContainer = "act5-response">
<assign>
<copy>
  <from container = "act5-response" part = "PickUpDate"/>
  <to container = "act6-request" part = "CollectDate"/>
</copy>
<copy>
  <from container = "act5-response" part = "PickUpLocationIdentifier"/>
  <to container = "act6-request" part = "CollectLocationId"/>
</assign>
</sequence>
</case>
</switch>
<copy>
<from expression = "'AA-465'"/>
<to container = "act6-request" part = "DeliveryLocationId"/>
</copy>
<copy>
<from expression = "'AIR-CARGO'"/>
<to container = "act6-request" part = "DeliveryMeans"/>
</copy>
<assign>
<invoke name = "DeliveryPartnerService" partner = "provider-6"
portType = "act6:DeliveryPartnerPortType"
operation = "arrangeDelivery" inputContainer = "act6-request"
outputContainer = "act6-response">
<assign>
<copy>
<from container = "act6-response" part = "DeliveryDate"/>
<to container = "act7-request" part = "InDate"/>
</copy>
<assign>
<invoke name = "AskAssemblyLine" partner = "provider-7"
portType = "act7:IntraOrgServicesPortType"
operation = "assemblyService"
inputContainer = "act7-request"
outputContainer = "act7-response">
</sequence>
</case>
</switch>
<assign>
<copy>
<from container = "request" part = "DeliveryLocation"/>
<to container = "act2-request" part = "DeliveryLocationId"/>
</copy>
<copy>
<from container = "request" part = "DeliveryMeans"/>
<to container = "act2-request" part = "DeliveryMeans"/>
</copy>
<copy>
<from expression = "'AL-465'"/>
<to container = "act2-request" part = "PickUpLocationId"/>
</copy>
<copy>
<from container = "act7-response" part = "OutDate"/>
<to container = "act2-request" part = "CollectDate"/>
</copy>
</copy>
</copy>
<from container = "request" part = "ToyCount"/>
<to container = "act3-request" part = "ToyCount"/>
</copy>
</assign>

<sequence name = "sequence-2">
<flow name = "flow-1">
<invoke name = "DeliveringToysToDistributor" partner = "provider-2"
portType = "act2:IntraOrgServicesPortType"
operation = "deliveryService" inputContainer = "act2-request"
outputContainer = "act2-response">
<invoke name = "FixPrice" partner = "provider-3"
portType = "act3: IntraOrgServicesPortType"
operation = "fixPrice" inputContainer = "act3-request"
outputContainer = "act3-response">
<assign>
<copy>
<from container = "act3-response" part = "Price"/>
<to container = "act4-request" part = "DeliveryCost"/>
</copy>
<copy>
<from container = "act2-response" part = "Cost"/>
<to container = "act4-request" part = "ToysCost"/>
</copy>
</assign>
<invoke name = "FinalizePrice" partner = "provider-4"
portType = "act4:IntraOrgServicesPortType"
operation = "finalizePrice" inputContainer = "act4-request"
outputContainer = "act4-response">
<assign>
<copy>
<from container = "act2-response" part = "DeliveryDate"/>
<to container = "response" part = "PossibleOrderDeliveryDate"/>
</copy>
<copy>
<from container = "act4-response" part = "FinalCost"/>
<to container = "response" part = "FinalCost"/>
</copy>
</assign>
<reply name = "reply-1" partner = "caller" portType = "tns:samplePortType"
operation = "getDetails" container = "response"/>
</sequence>
</process>