

Modeling Semantic Web Services with OPM/S – A Human and Machine-Interpretable Language

Dov Dori

Technion - Israel Institute of Technology
Haifa 32000, Israel
972-4-8294409

dori@ie.technion.ac.il

Eran Toch

Technion - Israel Institute of Technology
Haifa 32000, Israel
972-4-8292853

erant@tx.technion.ac.il

Iris Reinhartz-Berger

University of Haifa
Carmel Mt., Haifa 31905, Israel
972-4-8288502

iris@mis.hevra.haifa.ac.il

ABSTRACT

The World-Wide-Web is now a ubiquitous, global tool, used for finding information, communicating ideas, carrying out distributed computation, and conducting business, learning and science. Web services and the Semantic Web are emerging as a powerful infrastructure for distributed computing. However, even though standard methods that define semantics of Web services, such as OWL-S, may aid in the development and deployment of these services, they are hardly designed to be easily understandable and usable by developers. Complexity and lack of accessibility of Web services and the Semantic Web hinders their usage by the information industry. OPM/S, which is based on Object-Process Methodology (OPM), offers a bi-modal visual-lingual representation that is both intuitive for humans and formal for machines. Utilization of ontologies and interoperability are two issues addressed by the OPM/S modeling environment. Ontologies are expressed as meta-libraries, which are specified in OPM or OWL, and can be dynamically linked to semantic Web services in a distributed environment. Interoperability is achieved using a transparent reuse method that enables dynamic development of Web services and their integration into more complex Web services. Using a running example, the paper presents OPM/S and its mapping to OWL-S. The benefits and shortcomings are discussed and compared with other OWL-S modeling methods.

1. INTRODUCTION

The Web is highly dynamic in the quantity and nature of the information that it encompasses, posing a host of challenges in managing distributed information and computation over the Web. Access to the Web may be from a variety of devices and interfaces, by different users at different locations, and at varying times. Thus, there is a need for standard languages that capture information semantics and not just syntax. Web services and the Semantic Web are powerful infrastructures that enable easy integration, automatization, and reuse of different information and process formats in a distributed environment. Since Web services are

distributed, dynamic, and can be utilized as lightweight components within a broader and loosely coupled framework, common semantics is required so differences in terminology and definitions can be automatically resolved. While the World Wide Web provides a repository for Web services, the Semantic Web [2] is designed to be the foundation for semantic markup of Web services. The OWL-S initiative [1] (formally DAML-S) provides an ontology for Web services that enables automatic discovery, invocation, and interoperation of Web services.

As of today, the usage of OWL-S has not crossed the boundary between academia and industry. Based on their experience, Sabou et al. [21] reported that OWL-S is difficult to learn, partly due to the lack of supporting tools, and that it has an imprecise conceptual model, which is composed of multiple ontologies (models). To overcome these shortcomings, we propose to wrap OWL-S with a higher-level modeling language, called OPM/S. Based on Object-Process Methodology (OPM) [9], OPM/S integrates software engineering and semantics engineering practices in order to establish a single framework for modeling Web services. OPM/S contains two main mechanisms that enable modeling the dynamic interoperability between Web services. The first mechanism handles ontologies and descriptions of the concepts that are shared among various semantic Web services. Ontologies are captured as meta-libraries, which are specified in OPM or OWL, and can be utilized in the specification process of Web services. Meta-libraries are developed and maintained separately from the Web service models, and are dynamically linked to the Web services. Alterations in a meta-library are reflected in the particular Web service models that use it.

The second OPM/S mechanism is transparent reuse [20]. Usually, reuse of Web services does not in-

volve the installation of separate copies of the components, but rather utilization of runtime calls to the services. Therefore, modeling frameworks for Web services should take into account scenarios in which services may change after being reused. Transparent reuse enables ongoing development of Web services and their integration into more complex Web services. The reused models can be referred to using either URI or UDDI entries. A supervision mechanism warns the designer if changes in a reused model affect the entire model and automatically suggests possible solutions, based on a shared semantics.

The rest of the article is structured as follows. Section 2 reviews existing OWL-S engineering methods and briefly presents OPM. Section 3 introduces the OPM/S framework and explains how it supports the different OWL-S ontologies. Finally, Section 4 discusses the benefits and shortcomings of OPM/S and refers to future research plans.

2. BACKGROUND

2.1 Semantics of Web Services

Web services are automated resources accessed via the Internet, using an XML interface. The exponential growth of the Web and the progress of Internet-based architectures have set the stage for the proliferation of Web services and corresponding ontologies. However, despite the fact that these systems employ common XML or even RDF interfaces, semantic gaps among these ontologies cause problems in various scenarios. The problem of semantic reconciliation arises, for example, when integrating two systems that utilize different semantics to refer to entities that are conceptually identical. Such semantic differences must be resolved manually, a costly process that is unavoidably error-prone.

Trying to solve these problems, the Semantic Web provides an infrastructure for concepts to be specified formally, so relations between different concepts can be reasoned automatically, providing a basis for automated semantic reconciliation processes. Several standards have emerged to support the Semantic Web vision, including Web Ontology Language (OWL) [8] and DARPA Agent Markup Language (DAML+OIL) [5]. These languages are designed to enable computer applications to semantically process the information that formerly could be interpreted only by humans.

In order to search for Web services and use them, a software agent needs a computer-interpretable de-

scription of the service and means to access it. OWL-S [1], which is a framework for containing and sharing these descriptions, provides the following three essential types of knowledge about a service.

1. The **Service Profile** describes the properties of a service, such as its functionality, which is necessary for its automatic discovery, and its set of inputs, outputs, preconditions and effects (abbreviated IOPE).
2. The **Process Model** defines the control flow of the service, using its IOPEs and a set of control constructs and descriptions of each process that takes part in the flow. OWL-S has three types of processes: atomic processes, which are executed by single communication protocol calls, simple (undivided) processes, and composite processes, which comprise other processes.
3. The **Service Grounding** ontology connects the process model to communication-level protocols, such as the message description of Web Services Description Language (WSDL) [4].

Even though OWL-S had gained a considerable momentum in the Semantic Web community, several shortcomings prevent OWL-S from being more widely adopted by the software engineering industry. The main problem is that OWL-S is relatively complex and inaccessible for humans [21]. The separation between the definition of the service and the semantic meta-information requires users to master several languages and carry out translations among them. Moreover, the lack of suitable complexity management mechanisms makes the readability of the domain scripts difficult. Exacerbating this situation is the need for different types of notation in disparate models that must be used to define various aspects of a Web service, requiring consistency and integrity checking that is typical of systems that suffer from the model multiplicity problem [18].

2.2 OWL-S Modeling Methods

In an attempt to overcome the complexity of OWL-S, two approaches to creating OWL-S scripts have been developed: automatic generation and modeling. Automatic generation relies on tools that create OWL-S descriptions from existing concrete artifacts, such as programming code or WSDL specifications [4]. Employing bottom-up development, this approach is implemented, for example, in the WSDL2OWL-S tool [16], which provides a partial translation from a WSDL specification of a Web service to an OWL-S

script. The OWL-S modeling approach, in contrast, supports top-down development using a formal modeling language. Narayanan and McIlraith have used Petri nets to compose Web services [14]. While Petri nets are sufficient for modeling the dynamics and control flows of OWL-S modules, they are limited in describing static aspects of systems, like module allocations and their dependencies and structural relations [22]. Furthermore, lacking a refinement mechanism that is essential for describing large-scale systems, Petri nets are not directly scalable and are hence not widespread across the software engineering community.

Several OWL-S modeling methods rely on Unified Modeling Language (UML) [12], the standard object-oriented modeling language. Bose et. al. [3] have suggested using UML for modeling the OWL-S Service Profile and Process Model ontologies. Class diagrams represent the structure of the top level OWL-S specification, i.e., the entities of the Service Profile. Since OWL-S specifications include also the Process Model ontology, additional dynamic UML diagrams are needed to specify these behavioral aspects. The need to use several diagram types necessitates the development of a consistency management and protection mechanism that must be employed while the multiple-view UML model is mapped onto the single formal OWL-S specification.

DUET [7] is a software tool that implements an OWL-S modeling solution based on UML activity diagrams. Each process is represented by an activity, while composite processes contain descendant activities. Inputs and preconditions are modeled as entry actions that occur when the activity is executed. Outputs and effects are modeled as exit actions that are executed when the activity ends. This modeling framework ignores several elements of the OWL-S Process Model ontology, such as conditions and conditional outputs. Another drawback of this solution is that it is limited only to the Process Model ontology, neglecting the OWL-S Service Profile and Service Grounding ontologies.

2.3 Object-Process Methodology (OPM)














Object-Process Methodology (OPM) [9] is a holistic, integrated approach to the study and development of systems in general and information systems in particular. The basic premise of the OPM paradigm is that objects and processes are two types of equally important classes of things. Objects are (physical or informatical) things that exist, while processes are

things that transform objects. In most interesting and challenging systems structure and behavior are highly intertwined and hard to separate. Meeting this modeling challenge, stateful objects and processes that transform them describe the function, structure and behavior aspects of the modeled system within a single framework in a domain-independent manner without highlighting one aspect at the expense of suppressing another.

Contrary to the object-oriented approach, processes in OPM can stand alone, allowing intuitive modeling of the system's behavior that involves several object classes, possibly cutting across the system's structure. Processes are connected to the involved objects through procedural links, which are classified into enabling links, transformation links, condition links, and event links. The same and only diagram type provides also for modeling the system's structure, including the fundamental aggregation, generalization, exhibition, and classification relations alongside any other user-defined structural relation.

Two semantically equivalent modalities, one graphic and the other textual, jointly express the same OPM model. A set of inter-related Object-Process Diagrams (OPDs) constitute the graphical, visual OPM formalism. Each OPM element is denoted in an OPD by a graphic symbol, and the OPD syntax specifies correct and consistent ways by which entities can be linked. Table 1 lists the fundamental OPM elements along with their symbols and semantics. The Object-Process Language (OPL) is the textual counterpart modality of the graphical OPD set. As a dual-purpose language, OPL is oriented towards humans as well as machines. Catering to human needs, OPL is designed as a constrained subset of English, which serves domain experts engaged in analyzing and designing a system (or a Web service). Every OPD construct is expressed by a semantically equivalent OPL sentence or phrase. Designed also for machine interpretation, OPL provides a solid basis for automatically generating the designed application. This dual representation of OPM increases the processing capability of humans according to Mayer's theory [13].

Table 1: Main OPM elements, their symbols and semantics

Element Name	Symbol	Semantics
Object		A thing that has the potential of unconditional existence
Process		A pattern of transformation that objects undergo
Environmental thing		An environmental (external) thing (object or process) which communicates with the system
Characterization		A relation representing that a thing (object or process) exhibits another thing
Generalization		A relation denoting the fact that a thing generalizes a set of specialized things
Aggregation		A relation which denotes that a thing consists of other things
General structural relationship		A general association between things
Instrument link		A link indicating that a process requires an (input) object for its execution
Effect link		A link indicating that a process changes an object
Result/ Consumption link		A link indicating that a process creates/consumes an object
Invocation link		A link indicating that a process activates (invokes) another process
Condition link		A link representing a condition required for a process execution. While an enabling link has a "wait until" meaning, a condition link has an "if" meaning
Agent link		A link indicating that an external agent is required for the process execution

OPM also exhibits three complexity management mechanisms which enable refinement/abstraction of OPM models: (1) *unfolding/folding* is used for refining/abstracting the structural hierarchy of a thing; (2) *in-zooming/out-zooming* exposes/hides the inner details of a thing within its frame; and (3) *state expressing/suppressing* exposes/hides the states of an object. Using flexible combinations of these mechanisms, OPM enables specifying a system to any desired level of detail without losing legibility and comprehension of the resulting specification.

OPM is supported by the OPCAT [11] (Object-Process CASE Tool) modeling environment¹. It has also been applied and tested in various domains, including Web applications [19] and Semantic Web Information [10]. In this paper, we further extend OPM to support the modeling of semantic Web services. For future reference, this extension is called OPM/S.

¹ OPCAT can be downloaded from: <http://www.objectprocess.org>

3. The OPM/S Framework

The OPM/S modeling framework wraps OWL-S, reflecting the characteristics and features of semantic Web services. Contrary to OWL-S, which requires using three different types of ontologies with overlapping concepts, OPM/S employs its single frame of reference that can be presented at different abstraction levels. The combination of the single OPM view with its bimodal presentation increases the accessibility and usability of the OPM/S modeling framework to humans as well as machines (e.g., code generators and automatic translators).

The OWL-S ontologies are mapped to OPM/S as follows. The **Service Profile** is represented in OPM/S as the top level Object-Process Diagram (called the System Diagram, SD) along with its corresponding Object-Process Language paragraph. The **Process Model** is expressed by zooming into the top-level specification to expose the process structure and flows. From the top-level processes and downwards in the process containment hierarchy of OWL-S, each composite process is in-zoomed in OPM/S to reveal its subprocesses and their IOPE sets. The **Service Grounding** ontology is expressed by the interfaces of the atomic processes, which are unfolded at the deepest level of the process hierarchy. Table 2 summarizes the mapping between each of the three OWL-S ontologies and the corresponding OPM/S concepts.

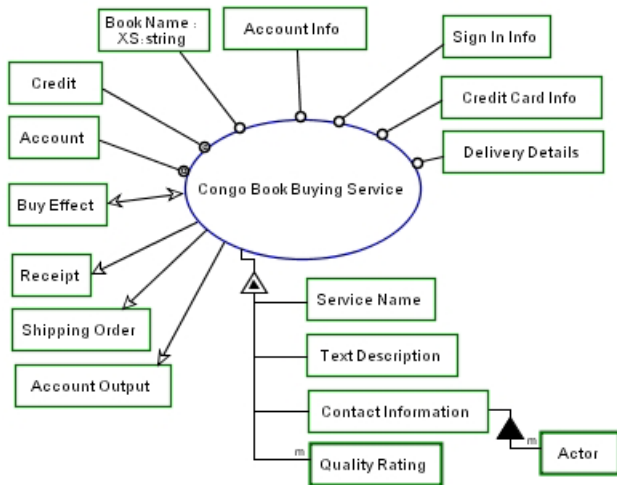
Table 2: Mapping OWL-S ontologies to OPM/S

OWL-S Ontology	OPM/S Representation	OPD or OPL Script Level
Service Profile	The highest level of the OPM/S model	0 (SD level)
Process Model	Each process is represented by an OPD and an OPL paragraph, which refine (zoom into) its subprocesses	$k ; 0 \leq k \leq N$ where N is the deepest level of the OPM/S model
Service Grounding	Unfolding of the deepest atomic process level	N

Two mechanisms are employed in order to support the dynamic and distributed nature of Web services: *meta-libraries* and *transparent reuse*. The rest of this section describes these mechanisms and explains their application to modeling the Service Profile and the Process Model in OPM/S. In order to demonstrate our approach, we use a running example of a book-buying service offered by the Web service provider Congo Inc. [6]. The OPM/S representation of the Service Grounding ontology is out of the scope of this paper.

3.1 The Service Profile Ontology

Figure 1 is an OPM/S model of the Service Profile ontology of the Congo Book Buying service. The top-level, main process in this model, **Congo Book Buying Service**, is surrounded by its inputs, outputs, pre-conditions, and effects, which are all objects. These objects are connected to the **Congo Book Buying Service** via procedural links that denote the role of each one of the objects in the top-level process.



Book Name is of type XS:string.

Congo Book Buying Service exhibits **Service Name**, **Text Description**, **Contact Information**, and many **Quality Ratings**.

Contact Information consists of many **Actors**.

Congo Book Buying Service occurs if **Account** exists and **Credit** exists.

Congo Book Buying Service requires **Book Name**, **Account Info**, **Sign In Info**, **Credit Card Info**, and **Delivery Details**.

Congo Book Buying Service affects **Buy Effect**.

Congo Book Buying Service yields **Receipt**, **Shipping Order**, and **Account Output**.

Figure 1: An OPM/S model of the Service Profile of the Congo Book Buying Service

As the condition links from the objects **Credit** and **Account** to the **Congo Book Buying Service** process denote, the existence of these two objects is a pre-condition for the execution of the **Congo Book Buying Service**. The corresponding OPL sentence, which is "**Congo Book Buying Service** occurs if **Account** exists and **Credit** exists", reinforces this condition semantics. Similarly, the objects **Book Name**, **Account Info**, **Sign In Info**, **Credit Card Info**, and **Delivery Details** are the process inputs, as the instrument links from each one of them to the **Congo Book Buying Service** process denote. The equivalent OPL sentence is "**Congo Book Buying Service** requires **Book Name**, **Account Info**, **Sign In Info**, **Credit Card Info**, and **Delivery Details**." The process outputs are **Receipt**, **Shipping Order**, and **Account**

Output. This is expressed by the result links in the OPD and by the corresponding result sentence in the OPL paragraph. The effect link between **Buy Effect** and the **Congo Book Buying Service** specifies that the process changes the object during its execution.

The service itself exhibits the attributes **Service Name**, **Text Description**, **Contact Information**, and several **Quality Ratings**. The **Contact Information** section can contain many points of contact, each represented by an **Actor**. The **Quality Ratings** are matrices that express various quality measurement of the service. Multiple matrices can characterize the same service. The thick contours of **Quality Rating** and **Actor** indicate that further refinement of the objects exists, i.e., in our case they are unfolded in separate diagrams (not shown here due to space limitations) of their internal structure.

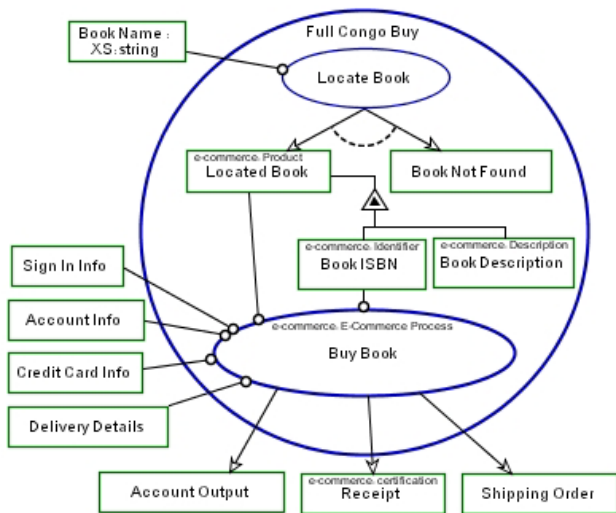
3.2 The Process Model Ontology

The **Congo Book Buying Service** is refined through zooming into two sub-processes, **Full Congo Buy** and **Express Congo Buy**, each describing different type of the **Congo Book Buying Service**. **Express Congo Buy**, which is an atomic process (not shown here), is a "one shot" service for buying a book with Congo, Inc. Figure 3 refines the composite process **Full Congo Buy** as a two-step buying procedure, in which the book is first located (**Locate Book**) and then bought (**Buy Book**).

For comparison purposes, the OWL-S specification of the **Full Congo Buy** process is listed in Figure 3. The XML statements contain the process type and its control flows. As noted, an OWL-S process can be of type composite, simple, or atomic. OPM/S, in contrast, defines a single concept—a process class, the complexity of which is determined by its position in the process hierarchy of the OPM model. OPM/S processes that are refined into subprocesses are mapped to OWL-S composite processes. A refined OPM/S process, such as **Buy Book**, is denoted by the fact that the contour of its ellipse, which is originally thin, becomes thick. Processes that are not in-zoomed (and hence remain with thin contours) are mapped to either atomic or simple OWL-S processes. Atomic processes differ from simple ones in that the former contain folded binding information (expressed in the service grounding ontology), while the latter do not.

Another difference between OPM/S and OWL-S is the way they treat control flows. OWL-S enables explicit selection of a control flow construct for defining

an ordering or conditional execution of sub-processes. Examples of these constructs include Sequence, Split, Split+Join, Choice, Unordered, Condition, If-Then-Else, Iterate, Repeat-While, and Repeat-Until. OPM/S, on the other hand, utilizes the limited set of basic elements supplied by OPM, namely procedural links, to specify control flows. Process sequences are denoted by the relative vertical position of the sub-processes, taking into account that the time line flows from the top of the diagram downwards. In Figure 3, for example, the process **Buy Book** is executed after **Locate Book**. Two processes with the same vertical position are executed in parallel or as alternatives. An invocation link, which specializes a procedural link, can override this convention [17].



Book Name is of type XS:string.
Full Congo Buy zooms into **Locate Book** and **Buy Book**, as well as **Book Not Found** and **Located Book**.
Located Book plays the role of **Product** of **e-commerce**.
Located Book exhibits **Book ISBN** and **Book Description**
Book ISBN plays the role of **Identifier** of **e-commerce**.
Book Description plays the role of **Description** **e-commerce**.
Locate Book requires **Book Name**.
Locate Book yields either **Located Book** or **Book Not Found**.
Buy Book plays the role of **E-Commerce Process** of **e-commerce**.
Buy Book occurs if **Located Book** exists.
Buy Book requires **Book ISBN**, **Credit Card Info**, **Sign In Info**, **Account Info**, and **Delivery Details**.
Buy Book yields **Receipt**, **Shipping Order**, and **Account Output**.
Receipt plays the role of **Confirmation** of **e-commerce**.

Figure 2: An OPM/S Process Model of the Full Congo Buy Service

```
<owl:Class rdf:ID="FullCongoBuy">
  <rdf:subClassOf
    rdf:resource="http://www.daml.org/services/owl-
s/0.9/Process.owl#CompositeProcess" />
  <rdf:subClassOf>
    <owl:Restriction>
      <owl:onProperty
        rdf:resource="http://www.daml.org/services/owl-
s/0.9/Process.owl#composedOf" />
      <owl:allValuesFrom>
        <owl:Class>
          <owl:intersectionOf rdf:parseType="Collection">
            <owl:Class
              rdf:about="http://www.daml.org/services/owl-
s/0.9/Process.owl#Sequence" />
            <owl:Restriction>
              <owl:onProperty
                rdf:resource="http://www.daml.org/services/owl-
s/0.9/Process.owl#components" />
              <owl:allValuesFrom>
                <owl:Class>
                  <process:listOfInstancesOf
                    rdf:parseType="Collection">
                      <owl:Class rdf:about="#LocateBook" />
                      <owl:Class rdf:about="#CongoBuyBook" />
                    </process:listOfInstancesOf>
                  </owl:Class>
                </owl:allValuesFrom>
              </owl:Restriction>
            </owl:intersectionOf>
          </owl:Class>
        </owl:allValuesFrom>
      </owl:Restriction>
    </owl:intersectionOf>
  </owl:Class>
```

Figure 3: OWL-S specification of the Full Congo Buy service

Conditional flow constructs, such as if-then-else or case statements, are specified in OPM/S by setting a decision object (which is Boolean for binary decisions). Each state of this object is linked via a condition link to the process that needs to happen if the object is at that state. In Figure 2, for example, **Buy Book** is executed only if **Located Book** exists, i.e., only if the book was found. OPM/S also supports modeling of iterative control flows (such as loops) by combining condition links to express the halting conditions and an invocation link to enable the iterations. Adding an invocation link in Figure 2 from **Buy Book** to the complete **Full Congo Buy** specifies that this process is executed until the requested book is not found.

3.3 Consistency Maintenance of OPM/S Models

OPM/S models are not only readable and comprehensible; they are also consistent across the various abstraction levels of the same Web service. The OPM consistency rules [9] require that the IOPE set of a process is either identical at any two consecutive refinement levels of the process or is refined in the deeper level. For example, **Delivery Details**, which is connected in Figure 1 to the **Congo Book Buying Service** via an instrument (input) link, remains con-

nected through an instrument link to **Buy Book** of **Full Congo Buy**, a sub-process of **Congo Book Buying Service**, in Figure 2. These rules are enforced by OPCAT. Such consistency checks are difficult to enforce in OWL-S, especially since they require verification that involves two different ontologies: the Service Profile and the Process Model.

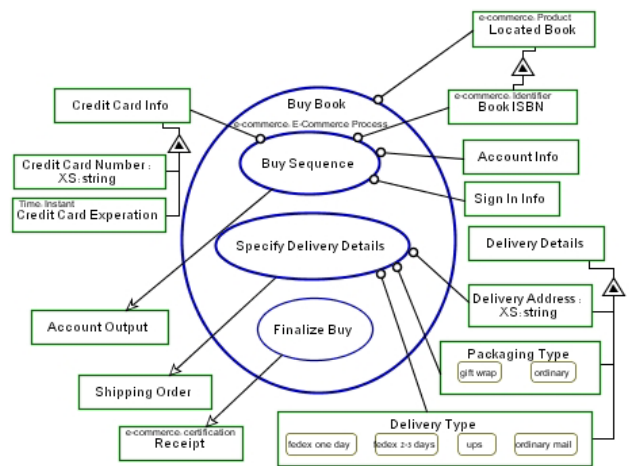
The consistency of an OPM/S specification is conserved also within the same Process Model displayed at various refinement levels. For example, in Figure 4, the **Buy Book** composite process is zoomed into, exposing its **Buy Sequence**, **Specify Delivery Details**, and **Finalize Buy** sub-processes. **Buy Sequence**, in turn, is also a composite process, responsible for handling shopping carts, customers, and accounts. **Specify Delivery Details** and **Finalize Buy**, on the other hand, are atomic processes. The single instrument link from **Delivery Details** to **Buy Book** in Figure 2 is refined in Figure 4 into three instrument links from **Delivery Address**, **Packaging Type**, and **Delivery Type** (which are attributes of **Delivery Details**), to **Specify Delivery Details**. Changing one of these links to an effect link will automatically change the link between **Delivery Details** and **Buy Book** in the higher-level specification (shown in Figure 2) to an effect link, denoting that **Buy Book** somehow changes **Delivery Details**.

3.4 OPM/S Typing Methods

There are three kinds of OPM/S typing methods: basic types, roles, and enumerations. These are demonstrated in Figure 4. Basic types includes all the primitive types, such as xs:int and xs:string, and user-defined types. The type of **Credit Card Number** and **Delivery Address** in Figure 4, for example, is XS:string.

Roles denote the ontological context, according to a given meta-library. **Credit Card Expiration**, for example, is declared as an Instant from the Time ontology, which is represented by an OWL meta-library. Role names are recorded in the upper-left corner of an object box or a process ellipse.

Enumerations are specified in OPM/S by states. States define the set of possible situations an object can be at or legal values an object can assume. At any point in time an object is in exactly one of its states or in transition between states. States can change only through the occurrence of a process. In the graphical OPD notation, states are denoted as rounded-corner rectangles inside an object. **Packaging Type** in Figure 4, for example, can be **gift-wrap** or **ordinary**.



Buy Book plays the role of **E-Commerce Process** of **e-commerce**.

Credit Card Info exhibits **Credit Card Number** and **Credit Card Expiration**.

Credit Card Number is of type XS:string.

Credit Card Expiration plays the role of **Instant** of **time**.

Delivery Details exhibits **Delivery Address**, **Packaging Type**, and **Delivery Type**.

Delivery Address is of type XS:string.

Packaging Type can be **gift wrap** or **ordinary**.

Delivery Type can be **fedex one day**, **fedex 2-3 days**, **ups**, or **ordinary mail**.

Book ISBN plays the role of **Identifier** of **e-commerce**.

Located Book plays the role of **Product** of **e-commerce**.

Located Book exhibits **Book ISBN**.

Buy Book requires **Located Book**.

Buy Book zooms into **Buy Sequence**, **Specify Delivery Details**, and **Finalize Buy**.

Buy Sequence requires **Sign In Info**, **Credit Card Info**, **Account Info**, and **Book ISBN**.

Buy Sequence yields **Account Output**.

Specify Delivery Details requires **Delivery Type**, **Delivery Address**, and **Packaging Type**.

Specify Delivery Details yields **Shipping Order**.

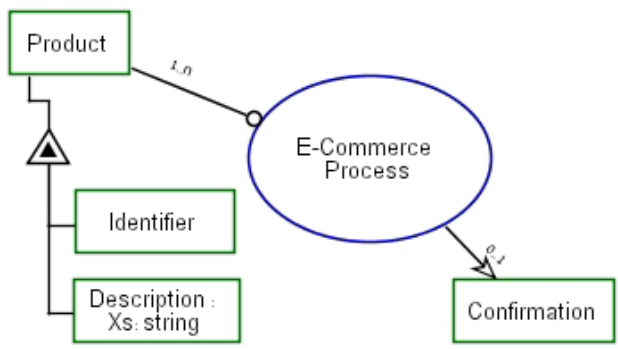
Finalize Buy yields **Receipt**.

Receipt plays the role of **Confirmation** of **e-commerce**.

Figure 4: An OPM/S model of the Process Model of Buy Book

3.5 Meta-Libraries

Meta-libraries are used to specify and utilize ontologies and domain knowledge in a dynamic and distributed environment. A meta-library captures domain knowledge as an OPM model or as an OWL specification. For instance, a meta-library concerning e-commerce will contain concepts such as product, customer, order, invoice, and so fourth. It will also contain static and dynamic constraints among these concepts, such as how many products can be ordered in a single order, should the paying certification precede the supplying process, etc.



Product exhibits **Identifier** and **Description**.
Description is of type Xs:string.
E-Commerce Process requires many **Products**.
E-Commerce Process yields an optional **Confirmation**.

Figure 5: An E-Commerce meta-library

Figure 5 is an OPM model of a simple e-commerce meta-library. According to this e-commerce ontology, **Product** exhibits two attributes, **Identifier** and **Description**. This e-commerce meta-library also introduces dynamic constraints on the **E-Commerce Process**: the process requires one or more **Products** and yields an optional **Confirmation**.

When defining a Web service, one or more meta-libraries are imported by selecting a local file or a URI, or by searching through UDDI registries. After the meta-library is imported, the Web service model dynamically references it. The references are refreshed each time a design session is initiated, ensuring that every change in the meta-library would be reflected to all the Web service models that import it.

Each element in a Web service model can be assigned to one or more roles each of which corresponds to an element of the meta-library. **Buy Book** in Figure 2, for example, plays the role of an **E-Commerce Process**. Hence, it requires **Products**, **Located Books** in this case, and yields a **Receipt**, which plays the role of a **Certification**. **Buy Book** also requires additional information (**Credit Card Info**, **Sign In Info**, **Account Info**, and **Delivery Details**) and yields **Shipping Order** and **Account Output**. Furthermore, the **Located Book** (input) object fulfills the structural constraints of a **Product**, enforces by the e-commerce meta-library: it exhibits **Book ISBN** (the product identifier) and **Book Description** (the product description). A special algorithm (not described in this paper) validates that the Web service model fulfills the meta-library (static and dynamic) constraints.

3.6 Transparent Reuse

The interoperability of semantic Web services is supported by applying OPM's transparent reuse method [20]. Transparent reuse enables dynamic bindings of system models. Each model is saved separately and can be modified throughout the entire development lifecycle, enabling the percolation of its most updated version to all the models that reuse it. This way, transparent reuse supports development of a Web service from external services that can be further enhanced and developed after they were integrated.

When applying OPM's transparent reuse method, elements from the reused service, which are symbolized as environmental elements ("stubs"), are bound to concrete (systemic) elements in the target Web service, using generalization relations. The environmental elements of the reused Web service cannot be edited in the target model, and are loaded from their sources (local files or Internet addresses) each time a design session is initiated. This way, any alteration in the reused Web service model will be reflected in all the target Web services that reuse it.

As an example to the transparent reuse method, consider a case in which **Finalize Buy** from Figure 4 is not specified in the **Congo Book Buying Service**. In this case, **Finalize Buy** would be marked as an environmental and, correspondingly, **Receipt** would be marked as an environmental object (in all levels of the OPM model), enabling the reuse of external services for finalizing a buy. An example of such a service can be the **FedEx Service**, whose interface includes delivery details (as an input) and a receipt (as an output) in special formats. When binding the **FedEx Service** to the **Congo Book Buying Service** (as a **Finalize Buy** process), the integrated Web service uses FedEx in order to deliver books from the Congo provider.

A supervision mechanism is employed in order to detect changes in Web services that might affect other Web services. The designer receives a list of warnings regarding broken bindings between Web services, as well as suggestions to alternative substitutions for the broken bindings.

4. CONCLUSIONS

The dynamic nature of the Web and its heterogeneous information formats require addressing semantic issues when searching and developing Web-based systems, such as Web services. Using OWL-S for specifying semantic Web service holds great opportunities for automating Web service discovery, invocation,

composition, interoperation, and execution monitoring. However, current OWL-S modeling and engineering methods need to be improved in order to make them accessible and useful for humans as well. This work has proposed OPM, which is a general system engineering method, as a convenient, intuitive, yet formal language for wrapping and expressing OWL-S specifications. A small set of about a dozen OPD symbols, along with a semi natural language, Object-Process Language (OPL), simplify the definition of Web services for humans. Ontologies and domain knowledge are handled in the OPM/S modeling framework using meta-libraries which are dynamically imported and shared by multiple Web services. In order to support interoperability between Web services, OPM's transparent reuse method is adopted as a way to integrate Web services. Transparent reuse enables ongoing changes in the integrated models, while alerting the designer about influences of the changes on other Web services.

The usage of objects and processes in OPM along with its three built-in complexity mechanisms enable modeling the Service Profile, Process Model, and Service Grounding ontologies of a Web Service in a single framework, using a single (bimodal) modeling language. The complexity management mechanisms also ensure that a Web service model is internally consistent.

OPM/S expressiveness should be further improved to support all types of OWL-S logical expressions, such as complex branches. The research plans also include implementing an OWL-S translator, which will enable bi-directional conversion of OPM/S models to OWL-S specifications and vice versa. In addition, development guidelines will be defined for specifying semantic Web services in OPM/S and OWL-S.

REFERENCES

- [1] Ankolenkar, M. Burstein, T. C. Son, J. Hobbs, O. Lassila, D. Martin, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, and H. Zeng., OWL-S: Semantic Markup for Web Services, www.daml.org/services/, June 2002.
- [2] Berners-Lee, T., Hendler, J., Lassila, O., The Semantic Web, *Scientific American*, 284(5), 2001, pp. 34-43.
- [3] Bose, P., Kogut, P., Leung, Y.H., Woodward, H.M., Applying UML to Model Web Service Ontologies for the Semantic Web, *OMG Web Services Workshop*, 3 July 02.
- [4] Christensen, E., Curbera, F., Meredith, G., Weerawarana, S., Web Services Description Language (WSDL) 1.1, 15 March 2001, <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>.
- [5] Connolly, D., Harmelen, F., Horrocks, I., McGuinness, D. L., Stein, L. A., DAML+OIL Reference Description. W3C Note, 18 December 2001.
- [6] DAML.org, Congo Buy Specification, <http://www.daml.org/services/owl-s/0.9/CongoService.owl>
- [7] DARPA, CODIP Project, DUET: DAML UML Enhanced Tool, <http://grcnet.grci.com/maria/www/CodipSite>.
- [8] Dean, M., Connolly, D., Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., Stein, L. A., OWL Web Ontology Language 1.0 Reference, July 2002.
- [9] Dori, D., Object-Process Methodology - A Holistic Systems Paradigm, Springer Verlag, 2002.
- [10] Dori, D., Object-Process Methodology as a basis for the Visual Semantic Web. *Proc. 14th International Conference on Database and Expert Systems Applications (DEXA 2003)*, IEEE Computer Society Press, IEEE International Workshop on Web Semantics (WebS 2003), September 2003.
- [11] Dori, D. Reinhartz-Berger, I. and Sturm A. OPCAT – A Bimodal Case Tool for Object-Process Based System Development. *5th International Conference on Enterprise Information Systems (ICEIS 2003)*, pp. 286-291, 2003.
- [12] OMG, Unified Modeling Language 1.5 Specification. *OMG Document formal/03-03-01*, March 2003.
- [13] Mayer, R.E. *Multimedia Learning*. Cambridge University Press, 2001.
- [14] Narayanan, S. McIlraith, S., Analysis and simulation of Web services, *Computer Networks*, In Press, Uncorrected Proof, Available online 10 April 2003.
- [15] Noy, N. F., Sintek, M., Decker, S., Crubezy, M., Ferguson, R. W., Musen, M. A., Creating Semantic Web Contents with Protege-2000. *IEEE Intelligent Systems* 16(2), 2001, pp. 60-71, <http://protege.stanford.edu>.
- [16] Paolucci, M., Srinivasan, N., Sycara, K., Nishimura, T., Towards a Semantic Choreography of Web Services: from WSDL to DAML-S, *Proceed-*

- ings of the The First International Conference on Web Services (ICWS'03), June 2003.
- [17] Peleg, M, and Dori, D. Representing Control Flow Constructs in Object-Process Diagrams. *Journal of Object-Oriented Programming*, 11, 3, pp. 58-71, 1998.
- [18] Peleg, M, and Dori, D., The Model Multiplicity Problem: Experimenting with Real-Time Specification Methods. *IEEE Transaction on Software Engineering*, 26, 8, pp. 742-759, 2000.
- [19] Reinhartz-Berger, I., Dori, D., Katz, S., OPM/Web – Object-Process Methodology for Developing Web Applications. *Annals of Software Engineering*. 13, pp. 141–161, 2002.
- [20] Reinhartz-Berger, I., Dori, D., Katz, S., Open Reuse of Component Designs in OPM/Web, 26th annual international Computer Software and Applications Conference (COMPSAC'02), pp. 19-26, 2002.
- [21] Sabou M., Richards D., Splunter S. van, An experience report on using DAML-S, Workshop on E-Services and the Semantic Web (ESSW '03), The Twelfth International World Wide Web Conference. Budapest, Hungary, May 20, 2003.
- [22] Wirtz, G., Application of Petri Nets in Modelling Distributed Software Systems, MOCA01, Denmark, Aug. 2001