

A Universal Service-Semantics Description Language

Ajay Bansal, Srividya Kona, Luke Simon,
Ajay Mallya, and Gopal Gupta
Department of Computer Science
University of Texas at Dallas
Richardson, TX 75083

Thomas D. Hite
Metalect Corp
2400 Dallas Parkway
Plano, TX 75093

Abstract

For web-services to become practical, an infrastructure needs to be supported that allows users and applications to discover, deploy, compose, and synthesize services automatically. This automation can take place only if a formal description of the web-services is available. In this paper we present an infrastructure using USDL (Universal Service-Semantics Description Language), a language for formally describing the semantics of web-services. USDL is based on the Web Ontology Language (OWL) and employs WordNet as a common basis for understanding the meaning of services. USDL can be regarded as formal service documentation that will allow sophisticated conceptual modeling and searching of available web-services, automated service composition, and other forms of automated service integration. A theory of safe service substitution for USDL is presented and proved sound and complete. The rationale behind the design of USDL along with its formal specification in OWL is presented with examples. We also compare USDL with other approaches like OWL-S and WSML and show that USDL is complementary to these approaches.

1. Introduction

A web-service is a program available on a web-site that “effects some action or change” in the world (i.e., causes a side-effect). Examples of such side-effects include a web-base being updated because of a plane reservation made over the Internet, a device being controlled, etc. The next milestone in the Web’s evolution is making *services* ubiquitously available. As automation increases, these web-services will be accessed directly by the applications themselves rather than by humans. In this context, a web-service can be regarded as a “programmatically interface” that makes application

to application communication possible. An infrastructure that allows users to discover, deploy, synthesize and compose services automatically needs to be supported in order to make web-services more practical.

To make services ubiquitously available we need a semantics-based approach such that applications can reason about a service’s capability to a level of detail that permits their discovery, deployment, composition and synthesis. Several efforts are underway to build such an infrastructure. These efforts include approaches based on the semantic web (such as OWL-S [5]) as well as those based on XML, such as Web Services Description Language (WSDL [7]). Approaches such as WSDL are purely syntactic in nature, that is, they merely specify the format of the service. In this paper we present an approach that is based on semantics. Our approach can be regarded as providing semantics to WSDL statements. We present the design of a language called Universal Service-Semantics Description Language (USDL) which service developers can use to specify formal semantics of web-services [14]. Thus, if WSDL can be regarded as a language for formally specifying the syntax of web services, USDL can be regarded as a language for formally specifying their semantics. USDL can be thought of as *formal service documentation* that will allow sophisticated conceptual modeling and searching of available web-services, automated composition, and other forms of automated service integration. For example, the WSDL syntax and USDL semantics of web services can be published in a directory which applications can access to automatically discover services. That is, given a formal description of the context in which a service is needed, the service(s) that will precisely fulfill that need can be determined. The directory can then be searched for the exact service, or two or more services that can be composed to synthesize the required service, etc.

To provide formal semantics, a common denominator must be agreed upon that everybody can use as a

basis of understanding the meaning of services. This common conceptual ground must also be somewhat coarse-grained so as to be tractable for use by both engineers and computers. That is, semantics of services should not be given in terms of low-level concepts such as Turing machines, first-order logic and their variants, since service description, discovery, and synthesis then become tasks that are practically intractable and theoretically undecidable. Additionally, the semantics should be given at a conceptual level that captures common real world concepts. Furthermore, it is too impractical to expect disparate companies to standardize on application (or domain) specific ontologies to formally define semantics of web-services, and instead a common universal ontology must be agreed upon with additional constructors. Also, application specific ontologies will be an impediment to automatic discovery of services since the application developer will have to be aware of the specific ontology that has been used to describe the semantics of the service in order to frame the query that will search for the service. The danger is that the service may not be defined using the particular domain specific ontology that the application developer uses to frame the query, however, it may be defined using some other domain specific ontology, and so the application developer will be prevented from discovering the service even though it exists. These reasons make an ontology based on OWL WordNet [2, 8] a suitable candidate for a universal ontology of atomic concepts upon which arbitrary meets and joins can be added in order to gain tractable flexibility.

We describe the meaning of conceptual modeling and how it could be obtained via a common universal ontology based on WordNet in the next section. Section 3, gives a brief overview of how USDL attempts to semantically describe web-services. In section 4, we discuss precisely how a WSDL document can be prescribed meaning in terms of WordNet ontology. Section 5 gives a complete USDL annotation for a Book-Buying service. Automatic discovery and composition of web-services using USDL is discussed in section 6. In section 7 we explore some of the theoretical aspects of service description in USDL. Comparison of USDL with other approaches like OWL-S and WSML is discussed in section 8. Finally, conclusions and future work are addressed in the last section.

2. A Universal Ontology

To describe service semantics, we should agree on a common ground to model our concepts. We can describe what any given web-service does from first principles using approaches based on logic. This is the

approach taken by frameworks such as dependent type systems and programming logics prevalent in the field of software verification where a “formal understanding” of the service/software is needed in order to verify it. However, such solutions are both low-level, tedious, and undecidable to be of practical use. Instead, we are interested in modeling higher-level concepts. That is, we are more interested in answering questions such as, what does a service do from the end user’s or service integrator’s perspective, as opposed to the far more difficult questions, such as, what does the service do from a computational view? We care more about real world concepts such as “customer”, “bank account”, and “flight itinerary” as opposed to the data structures and algorithms used by a service to model these concepts. The distinction is subtle, but is a distinction of granularity as well as a distinction of scope.

In order to allow interoperability and machine-readability of our documents, a common conceptual ground must be agreed upon. The first step towards this common ground are standard languages such as WSDL and OWL. However, these do not go far enough, as for any given type of service there are numerous distinct representations in WSDL and for high-level concepts (e.g., a ternary predicate), there are numerous disparate representations in terms of OWL, representations that are distinct in terms of OWL’s formal semantics, yet equal in the actual concepts they model. This is known as the semantic aliasing problem: distinct syntactic representations with distinct formal semantics yet equal conceptual semantics. For the semantics to equate things that are conceptually equal, we need to standardize a sufficiently comprehensive set of atomic concepts, i.e., a universal ontology, along with a restricted set of connectives.

Industry specific ontologies along with OWL can also be used to formally describe web-services. This is the approach taken by the OWL-S language [5]. The problem with this approach is that it requires standardization and undue foresight. Standardization is a slow, bitter process, and industry specific ontologies would require this process to be iterated for each specific industry. Furthermore, reaching a industry specific standard ontology that is comprehensive and free of semantic aliasing is even more difficult. Undue foresight is required because many useful web services will address innovative applications and industries that don’t currently exist. Standardizing an ontology for travel and finances is easy, as these industries are well established, but new innovative services in new upcoming industries also need be ascribed formal meaning. A universal ontology will have no difficulty in describing such new services.

We need an ontology that is somewhat coarse-grained yet universal, and at a similar conceptual level to common real world concepts. Currently there is only one sufficiently comprehensive ontology that meets these criteria: WordNet [8]. As stated, part of the common ground involves standardized languages such as OWL. For this reason, WordNet cannot be used directly, and instead we make use of an encoding of WordNet as an OWL base ontology [2]. Using an OWL WordNet ontology allows for our solution to use a universal, complete, and tractable framework, which lacks the semantic aliasing problem, to which we map web service messages and operations. As long as this mapping is precise and sufficiently expressive, reasoning can be done within the realm of OWL by using automated inference systems (such as, one based on description logic), and we automatically reap the wealth of semantic information embodied in the OWL WordNet ontology that describes the relationships between ontological concepts, especially subsumption (hyponym) and equivalence (synonym) relationships.

3. USDL: An Overview

As mentioned earlier, USDL can be regarded as a language to formally specify the semantics of web-services. It is perhaps the first attempt to capture the semantics of web-services in a universal, yet decidable manner. It is quite distinct from previous approaches such as WSDL and OWL-S [5]. As mentioned earlier, WSDL only defines syntax of the service; USDL provides the missing semantic component. USDL can be thought of as a formal language for service documentation. Thus, instead of documenting the function of a service as comments in English, one can write USDL statements that describe the function of that service. USDL is quite distinct from OWL-S, which is designed for a similar purpose, and as we shall see the two are in fact complementary. OWL-S primarily describes the states that exist before and after the service and how a service is composed of other smaller sub-services (if any). Description of atomic services is left underspecified in OWL-S. They have to be specified using domain specific ontologies; in contrast, atomic services are completely specified in USDL, and USDL relies on a universal ontology (OWL WordNet Ontology) to specify the semantics of atomic services. USDL and OWL-S are complementary in that OWL-S's strength lies in describing the structure of composite services, i.e., how various atomic services are algorithmically combined to produce a new service, while USDL is good for fully describing atomic services. Thus, OWL-S can be used for describing the structure of composite services that

combine atomic services described using USDL.

USDL describes a service in terms of portType and messages, similar to WSDL. The semantics of a service is given using the OWL WordNet ontology: portType (operations provided by the service) and messages (operation parameters) are mapped to disjunctions of conjunctions of (possibly negated) concepts in the OWL WordNet ontology. The semantics is given in terms of how a service *affects* the external world. The present design of USDL assumes that each side-effect is one of following four operations: *create*, *update*, *delete*, or *find*. A generic *affects* side-effect is used when none of the four apply. An application that wishes to make use of a service automatically should be able to reason with WordNet atoms using the OWL WordNet ontology.

We also define the formal semantics of USDL. As stated earlier, the syntactic terms describing portType and messages are mapped to disjunctions of conjunctions of (possibly negated) OWL WordNet ontological terms which can be represented by points in the lattice obtained from the WordNet ontology with regards to the OWL subsumption relation. A service is then formally defined as a function, labeled with zero or more side-effects, between points in this lattice. The main contribution of our work is the design of a universal service-semantics description language (USDL), along with its formal semantics, and soundness and completeness proofs for a theory of safe service substitution.

4. Design of USDL

The design of USDL rests on two formal languages: Web Services Description Language (WSDL) [7] and Web Ontology Language (OWL) [6]. The Web Services Description Language (WSDL) [7], is used to give a syntactic description of the name and parameters of a service. The description is syntactic in the sense that it describes the formatting of services on a syntactic level of method signatures, but is incapable of describing what concepts are involved in a service and what a service actually does, i.e., the conceptual semantics of the service. Likewise, the Web Ontology Language (OWL) [6], was developed as an extension to the Resource Description Framework (RDF) [3], both standards are designed to allow formal conceptual modeling via logical ontologies, and these languages also allow for the markup of existing web resources with semantic information from the conceptual models. USDL employs WSDL and OWL in order to describe the syntax and semantics of web-services. WSDL is used to describe message formats, types, and method prototypes, while a specialized universal OWL ontology is used to formally describe what these messages and methods

mean, on a conceptual level.

USDL can be regarded as the semantic counterpart to the syntactic WSDL description. WSDL documents contain two main constructs to which we want to ascribe conceptual meaning: messages and portType. These constructs are aggregates of service components which will be directly ascribed meaning. Messages consist of typed parts and portType consists of operations parameterized on messages. USDL defines OWL surrogates or proxies of these constructs in the form of classes, which have properties with values in the OWL WordNet ontology.

4.1 Concept

USDL defines a generic class called *Concept* which is used to define the semantics of parts of messages. Semantically, instances of *Concept* form a complete lattice, which will be covered in section 7.

```
<owl:Class rdf:ID="Concept">
  <rdfs:comment>
    Generic class of USDL Concept
  </rdfs:comment>
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#AtomicConcept"/>
    <owl:Class rdf:about="#InvertedConcept"/>
    <owl:Class rdf:about="#ConjunctiveConcept"/>
    <owl:Class rdf:about="#DisjunctiveConcept"/>
  </owl:unionOf>

  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasCondition"/>
      <owl:minCardinality
        rdf:datatype="&xsd;nonNegativeInteger">
        0
      </owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  ...
</owl:Class>

<owl:ObjectProperty rdf:ID="hasCondition">
  <rdfs:domain rdf:resource="#Concept"/>
  <rdfs:range rdf:resource="#Condition"/>
</owl:ObjectProperty>
```

The USDL *Concept* class denotes the top element of the lattice of conceptual objects constructed from the OWL WordNet ontology. For most purposes, message parts and other WSDL constructs will be mapped to a subclass of USDL *Concept* so that useful concepts can be modeled as set theoretic formulas of union, intersection, and negation of atomic concepts. These subclasses of *Concept* are *AtomicConcept*, *InvertedConcept*, *ConjunctiveConcept*, and *DisjunctiveConcept*.

4.1.1 Atomic Concept

AtomicConcept is the actual contact point between USDL and WordNet. This class acts as proxy for WordNet lexical entities.

```
<owl:Class rdf:about="#AtomicConcept">
  <rdfs:subClassOf rdf:resource="#Concept"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#isA"/>
      <owl:cardinality
        rdf:datatype="&xsd;nonNegativeInteger">
        1
      </owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>

  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#ofKind"/>
      <owl:minCardinality
        rdf:datatype="&xsd;nonNegativeInteger">
        0
      </owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  ...
</owl:Class>
```

The property cardinality restrictions require all USDL *AtomicConcepts* to have exactly one defining value for the *isA* property, and zero or more values for the *ofKind* property. An instance of *AtomicConcept* is considered to be equated with a WordNet lexical concept given by the *isA* property and classified by a lexical concept given by the optional *ofKind* property.

```
<owl:ObjectProperty rdf:ID="isA">
  <rdfs:domain rdf:resource="#AtomicConcept"/>
  <rdfs:range rdf:resource="#&wn;LexicalConcept"/>
  ...
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="ofKind">
  <rdfs:domain rdf:resource="#AtomicConcept"/>
  <rdfs:range rdf:resource="#&wn;LexicalConcept"/>
  ...
</owl:ObjectProperty>
```

4.1.2 Inverted Concept

In case of *InvertedConcept* the corresponding semantics are the complement of WordNet lexical entities.

```
<owl:Class rdf:about="#InvertedConcept">
  <rdfs:subClassOf rdf:resource="#Concept"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasConcept"/>
      <owl:cardinality
        rdf:datatype="&xsd;nonNegativeInteger">
```

```

    1
    </owl:cardinality>
  </owl:Restriction>
</rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty rdf:ID="hasConcept">
  <rdfs:domain rdf:resource="#Concept"/>
  <rdfs:range rdf:resource="#Concept"/>
</owl:ObjectProperty>

```

4.1.3 Conjunctive and Disjunctive Concept

The *ConjunctiveConcept* and *DisjunctiveConcept* respectively denote the intersection and union of USDL Concepts.

```

<owl:Class rdf:about="#ConjunctiveConcept">
  <rdfs:subClassOf rdf:resource="#Concept"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasConcept"/>
      <owl:minCardinality
        rdf:datatype="&xsd;nonNegativeInteger">
        2
      </owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="#DisjunctiveConcept">
  <rdfs:subClassOf rdf:resource="#Concept"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasConcept"/>
      <owl:minCardinality
        rdf:datatype="&xsd;nonNegativeInteger">
        2
      </owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

```

The property cardinality restrictions on *ConjunctiveConcept* and *DisjunctiveConcept* allow for n -ary intersections and unions (where $n \geq 2$) of USDL concepts. For generality, these concepts are either *AtomicConcepts*, *ConjunctiveConcepts*, *DisjunctiveConcepts*, or *InvertedConcepts*.

4.2 Affects

The *affects* property is specialized into four types of actions common to enterprise services: *creates*, *updates*, *deletes*, and *finds*.

```

<owl:ObjectProperty rdf:ID="affects">
  <rdfs:comment>
    Generic class of USDL Affects
  </rdfs:comment>
  <rdfs:domain rdf:resource="#Operation"/>

```

```

  <rdfs:range rdf:resource="#Concept"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#creates">
  <rdfs:subPropertyOf rdf:resource="#affects"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#updates">
  <rdfs:subPropertyOf rdf:resource="#affects"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#deletes">
  <rdfs:subPropertyOf rdf:resource="#affects"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#finds">
  <rdfs:subPropertyOf rdf:resource="#affects"/>
</owl:ObjectProperty>

```

Note that each of these specializations inherits the domain and range of the *affects* property. Most services can be described as a conjunction of these types of effects. For those services that cannot be described in terms of a combination of these specializations, the parent *affects* property can be used instead, or the property can be omitted entirely when the meaning of the operation parameter messages are enough for conceptual reasoning. The purpose of limiting the types of services as opposed to allowing the creation of new arbitrary side-effect types, for example via OWL-DL, is to: (i) make USDL more structured and therefore easier to create documents in, (ii) make USDL computationally more tractable for programs that process large volumes of USDL documents, and (iii) help prevent the semantic aliasing problem mentioned in section 2.

4.3 Conditions and Constraints

Services may have some external conditions (pre-conditions and post-conditions) specified on the input or output parameters. *Condition* class is used to describe all such constraints. Conditions are represented as conjunction or disjunction of binary predicates. Predicate is a trait or aspect of the resource being described.

```

<owl:Class rdf:ID="Condition">
  <rdfs:comment>
    Generic class of USDL Condition
  </rdfs:comment>
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class
      rdf:about="#AtomicCondition"/>
    <owl:Class
      rdf:about="#ConjunctiveCondition"/>
    <owl:Class
      rdf:about="#DisjunctiveCondition"/>
  </owl:unionOf>
  ...

```

```

</owl:Class>
<owl:Class rdf:about="#AtomicCondition">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasConcept"/>
      <owl:cardinality
        rdf:datatype="&xsd;nonNegativeInteger">
        1
      </owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#onPart"/>
      <owl:cardinality
        rdf:datatype="&xsd;nonNegativeInteger">
        1
      </owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty
        rdf:resource="#hasValue"/>
      <owl:maxCardinality
        rdf:datatype="&xsd;nonNegativeInteger">
        1
      </owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

```

A condition has exactly one value for the *onPart* property and at most one value for the *hasValue* property, each of which is of type USDL *Concept*.

```

<owl:ObjectProperty rdf:ID="onPart">
  <rdfs:domain rdf:resource="#AtomicCondition"/>
  <rdfs:range rdf:resource="#Concept"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasValue">
  <rdfs:domain rdf:resource="#AtomicCondition"/>
  <rdfs:range rdf:resource="#Concept"/>
</owl:ObjectProperty>

```

4.3.1 Conjunctive and Disjunctive Conditions

The *ConjunctiveCondition* and *DisjunctiveCondition* respectively denote the conjunction and disjunction of USDL *Conditions*.

```

<owl:Class rdf:about="#ConjunctiveCondition">
  <rdfs:subClassOf rdf:resource="#Condition"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasCondition"/>
      <owl:minCardinality
        rdf:datatype="&xsd;nonNegativeInteger">
        2

```

```

      </owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#DisjunctiveCondition">
  <rdfs:subClassOf rdf:resource="#Condition"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasCondition"/>
      <owl:minCardinality
        rdf:datatype="&xsd;nonNegativeInteger">
        2
      </owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:ObjectProperty rdf:ID="hasCondition">
  <rdfs:domain rdf:resource="#Concept"/>
  <rdfs:range rdf:resource="#Condition"/>
</owl:ObjectProperty>

```

The property cardinality restrictions on *ConjunctiveCondition* and *DisjunctiveCondition* allow for n -ary conjunctions and disjunctions (where $n \geq 2$) of USDL conditions. In general any n -ary condition can be written as a combination of conjunctions and disjunctions of binary conditions.

4.4 Messages

Services communicate by exchanging messages. As mentioned, messages are simple tuples of actual data, called parts. Take for example, a flight reservation service similar to the SAP ABAP Workbench Interface Repository for flight reservations [4], which makes use of the following message.

```

<message name="&flight;ReserveFlight_Request">
  <part name="&flight;CustomerName" type="xsd:string">
  <part name="&flight;FlightNumber" type="xsd:string">
  <part name="&flight;DepartureDate" type="xsd:date">
  ...
</message>

```

The USDL surrogate for a WSDL message is the *Message* class, which is a composite entity with zero or more parts. Note that for generality, messages are allowed to contain zero parts.

```

<owl:Class rdf:about="#Message">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasPart"/>
      <owl:minCardinality
        rdf:datatype="&xsd;nonNegativeInteger">
        0
      </owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

```

Each part of a message is simply a USDL *Concept*, as defined by the *hasPart* property. Semantically messages are treated as tuples of concepts.

```
<owl:ObjectProperty rdf:ID="hasPart">
  <rdfs:domain rdf:resource="#Message"/>
  <rdfs:range rdf:resource="#Concept"/>
</owl:ObjectProperty>
```

Continuing our example flight reservation service, the *Itinerary* message is given semantics using USDL as follows, where `&wn;customer` and `&wn;name` are valid XML references to WordNet lexical concepts.

```
<Message rdf:about=
  "%flight;ReserveFlight_Request">
  <hasPart>
    <AtomicConcept
      rdf:about="%flight;CustomerName">
      <isA rdf:resource="%&wn;name"/>
      <ofKind>
        <AtomicConcept>
          <isA rdf:resource="%&wn;customer"/>
        </AtomicConcept>
      </ofKind>
    </AtomicConcept>
  </hasPart>
  <hasPart>
    <AtomicConcept
      rdf:about="%flight;FlightNumber">
      <isA rdf:resource="%&wn;number"/>
      <ofKind>
        <AtomicConcept>
          <isA rdf:resource="%&wn;flight"/>
        </AtomicConcept>
      </ofKind>
    </AtomicConcept>
  </hasPart>
  ...
</Message>
```

4.5 PortType

A service consists of portType, which is a collection of procedures or operations that are parametric on messages. Our example flight reservation service might contain a *portType* definition for a flight reservation service that takes as input an itinerary and outputs a reservation receipt.

```
<portType name="%flight;Flight_Reservation">
  <operation name="%flight;ReserveFlight">
    <input message="%flight;ReserveFlight_Request"/>
    <output message="%flight;ReserveFlight_Response"/>
  </operation>
</portType>
```

The USDL surrogate is defined as the class *portType* which contains zero or more *Operations* as values of the *hasOperation* property.

```
<owl:Class rdf:about="#portType">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasOperation"/>
      <owl:minCardinality
        rdf:datatype="xsd:nonNegativeInteger">
        0
      </owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty rdf:ID="hasOperation">
  <rdfs:domain rdf:resource="#portType"/>
  <rdfs:range rdf:resource="#Operation"/>
</owl:ObjectProperty>
```

As with the case of messages, portTypes are not directly assigned meaning via the OWL WordNet ontology. Instead the individual *Operations* are described by their side-effects via an *affects* property. Note that the parameters of an operation are already given meaning by ascribing meaning to the messages that constitute the parameters.

```
<owl:Class rdf:about="#Operation">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasInput"/>
      ...
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasOutput"/>
      ...
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#affects"/>
      ...
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

An operation can have multiple or no values for the *affects* property, all of which are of type USDL *Concept*, which is the target of the effect.

```
<owl:ObjectProperty rdf:ID="affects">
  <rdfs:domain rdf:resource="#Operation"/>
  <rdfs:range rdf:resource="#Concept"/>
</owl:ObjectProperty>
```

Finishing our flight reservation service example, we can describe the main side-effect of invoking the *ReserveFlight* operation in USDL as follows.

```
<portType rdf:about="%flight;Flight_Reservation">
  <hasOperation>
    <Operation rdf:about="%flight;ReserveFlight">
```

```

...
<creates>
  <AtomicConcept rdf:about="flight_reservation">
    <isA rdf:resource="&#x26;#x26;reservation"/>
    <ofKind>
      <AtomicConcept>
        <isA rdf:resource="&#x26;#x26;flight"/>
      </AtomicConcept>
    </ofKind>
  </AtomicConcept>
</creates>
</Operation>
</hasOperation>
</portType>

```

Again, it is not necessary to annotate the operation with regards to its input and output parameters, as these are already annotated by *Message* surrogates.

5. Semantic Description of a Service

This section shows an example syntactic description of a web-service using WSDL and its corresponding semantic description using USDL.

A simple Book-Buying Service: The service described here is a simplified book-buying service published in a web-service registry. This service can be treated as atomic: i.e., no interactions between buying and selling agents are required, apart from invocation of the service and receipt of its outputs by the buyer. Given certain inputs and pre-conditions, the service provides certain outputs and has specific effects.

This service takes in a *BookISBN*, *UserIdentifier*, and *Password* as input parameters. It has an input pre-condition that a *UserIdentifier* for the buyer must exist before invoking the service. It also has a global constraint that a valid credit card number for the buyer must exist. This service outputs an *OrderNumber* if the order was placed successfully. In case the book is not available, the output is an “*Out of Stock*” message.

5.1 WSDL definition

The following is WSDL definition of the service. This service provides a single operation called *Book-Buying*. The input and output messages are defined below. The conditions on the service cannot be described using WSDL.

```

<definitions>
...
  <portType name="BookBuying_Service">
    <operation name="BookBuying">
      <input message="BuyBook_Request"/>
      <output message="BuyBook_Response"/>

```

```

    </operation>
  </portType>

  <message name="BuyBook_Request">
    <part name="BookISBN" type="xsd:string"/>
    <part name="UserIdentifier" type="xsd:string"/>
    <part name="Password" type="xsd:string"/>
  </message>

  <message name="BuyBook_Response">
    <part name="OrderNumber/Availability"
      type="xsd:string"/>
  </message>
...
</definitions>

```

5.2 USDL annotation

The following is the complete USDL annotation corresponding to the above mentioned WSDL description. The input pre-condition and the global constraint on the service are also described semantically.

```

<definitions>
...
  <portType rdf:about="#BookBuying_Service">
    <hasOperation>
      <operation name="BuyBook">
        <input message="BuyBook_Request"/>
        <output message="BuyBook_Response"/>
        <creates>
          <AtomicConcept rdf:about="#BookOrder">
            <isA rdf:resource="&#x26;#x26;order"/>
            <ofKind>
              <AtomicConcept>
                <isA rdf:resource="&#x26;#x26;book"/>
              </AtomicConcept>
            </ofKind>
            <hasCondition>
              <Condition rdf:about="#exists">
                <AtomicConcept>
                  <isA rdf:resource="&#x26;#x26;exists"/>
                </AtomicConcept>
                <onPart rdf:about="#CreditCard">
                  <AtomicConcept>
                    <isA rdf:resource="&#x26;#x26;card"/>
                  <ofKind>
                    <AtomicConcept>
                      <isA rdf:resource="&#x26;#x26;credit"/>
                    </AtomicConcept>
                  </ofKind>
                </AtomicConcept>
              </onPart>
            </Condition>
          </hasCondition>
        </AtomicConcept>
      </creates>
    </operation>
  </hasOperation>
</portType>

  <Message rdf:about="#BuyBook_Request">

```

```

<hasPart>
  <AtomicConcept rdf:about="#BookISBN">
    <isA rdf:resource="&#x000A;identifier"/>
    <ofKind>
      <AtomicConcept>
        <isA rdf:resource="&#x000A;book"/>
      </AtomicConcept>
    </ofKind>
  </AtomicConcept>
</hasPart>

<hasPart>
  <AtomicConcept rdf:about="#UserIdentifier">
    <isA rdf:resource="&#x000A;identifier"/>
    <ofKind>
      <AtomicConcept>
        <isA rdf:resource="&#x000A;user"/>
      </AtomicConcept>
    </ofKind>
    <hasCondition>
      <Condition rdf:about="#exists">
        <AtomicConcept>
          <isA rdf:resource="&#x000A;exists"/>
        </AtomicConcept>
        <onPart rdf:resource="#UserIdentifier"/>
      </Condition>
    </hasCondition>
  </AtomicConcept>
</hasPart>

<hasPart>
  <AtomicConcept>
    <isA rdf:resource="&#x000A;Password"/>
  </AtomicConcept>
</hasPart>
</Message>

<Message rdf:about="#BuyBook_Response">
  <hasPart>
    <DisjunctiveConcept
      rdf:about="#OrderNumber/Availability">
      <hasConcept>
        <AtomicConcept rdf:about="#OrderNumber">
          <isA rdf:resource="&#x000A;number"/>
          <ofKind>
            <AtomicConcept>
              <isA rdf:resource="&#x000A;order"/>
            </AtomicConcept>
          </ofKind>
        </AtomicConcept>
      </hasConcept>
      <hasConcept>
        <InvertedConcept rdf:about="#NotAvailable">
          <AtomicConcept>
            <isA rdf:resource="&#x000A;available"/>
          </AtomicConcept>
        </InvertedConcept>
      </hasConcept>
    </DisjunctiveConcept>
  </hasPart>
</Message>
</definitions>

```

6. Service Discovery and Composition

Note that given a directory of services, a USDL description could be included for each service, making the service directly “semantically” searchable. However, we still need a query language to search this directory, i.e., we need a language to frame the requirements on the service that an application developer is seeking. Note that USDL itself could be such a query language. A USDL description of the desired service can be written, a query processor can then search the service directory to look for a “matching” service. For matching we could treat USDL descriptions as well as the USDL query as terms, and perhaps use some kind of extended unification to check for a match. This work is currently in progress [10].

With the USDL descriptions and query language in place, numerous applications become possible ranging from querying a database of services to rapid application development via automated integration tools and even real-time service composition [12]. Take our flight reservation service example. Assume that somebody wants to find a travel reservation service and that they query a USDL database containing general purpose flight reservation services, bus reservation services, etc. One could then form a USDL query consisting of a description of a travel reservation service and the database could respond with a set of travel reservation services whether it be via flight, bus, or some other means of travel. This flexibility of generalization and specialization is gained from USDL’s subsumption relation covered in section 7.

Furthermore, with a pool of USDL services at one’s disposal, rapid application development (RAD) tools could be used to aid a systems integrator with the task of creating composite services, i.e., services consisting of the composition of already existing services. The service designer could use such a RAD tool by describing the desired service via a USDL document, and then the tool would query the pool of services for composable sets of services that can be used to accomplish the task as well as automatically generate boilerplate code for managing the composite service, as well as menial inter-service data format conversions and other glue. Of course these additional RAD steps would require other technologies already being investigated [13].

At present, we presume only four specific operations (find, create, delete, update); this set of basic operations may be extended as more experience is gained with USDL. In practice, most services, however, deal with manipulating databases and for such services these four operations suffice. As stated, one of the reasons for limiting the side-effect types is to safe-

guard against the semantic aliasing problem described in section 2. This is also one of the main reasons for restricting the combining forms in USDL to conjunction, disjunction, and negated atoms. As discussed next, this allows USDL descriptions to be put into a type of disjunctive normal form (DNF), from which a sound and complete notion of subsumption is created.

7. Theory of Substitution of Services

Next, we will investigate the theoretical aspects of USDL. This involves concepts from set theory, lattice theory, and type theory. From a systems integration perspective, an engineer is interested in finding (discovering) a service that accomplishes some necessary task. Of course, such a service may not be present in any service directory. In such a case the discovery software should return a set of services that can be used in a context expecting a service that meets that description (of course, this set may be empty). To find services that can be substituted for a given service that is not present in the directory, we need to develop a theory of *service substitutability*. We develop such a theory in this section, and show its soundness and completeness. Our theory relates service substitutability to WordNet’s subsumption relation.

In order to develop this theory, we must first formally define constructs such as USDL-described concepts and services, which we will also call concepts and services for short. While it is possible to work directly with the XML USDL syntax, doing so is cumbersome and so we will instead opt for set theoretic notation.

Definition 1. Let Ω be the set of WordNet entities and \leq_{Ω} be the OWL subsumption relation on Ω .

Definition 2. Let Θ be the least set of concepts s.t.:

1. $x \in \Omega$ implies $x, \neg x \in \Theta$
2. $x, y \in \Theta$ implies $x \cup y, x \cap y \in \Theta$

Hence Θ is simply the set of USDL concepts.

Definition 3. Let $\Gamma = \{(L, E) \mid L \in \Psi, E \in \Theta\}$ be the set of USDL side-effects, where $\Psi = \{\text{creates, updates, deletes, finds}\}$, L is the affect type and E is the affected object.

Definition 4. For any set S , let

$$S^* = \{\epsilon \mid \epsilon \notin S\} \cup S \cup \{(x, y) \mid x \in S, y \in S^*\}$$

be the set of lists over S . Let $\Sigma = \{(A, I, O) \mid A \in 2^{\Gamma}, I \in \Theta^*, O \in \Theta^*\}$ be the set of USDL service descriptions, where A is the set of side-effects, I is the list of input parameters, and O is the list of output parameters of a particular service.

Now that the formal definitions of concept and service descriptions are given, we would like to define a subsumption relation \leq_{Σ} over Σ so that we can reason about substitutability of services, but this will, in turn, require a subsumption relation \leq_{Θ} over Θ . The proof of correctness of \leq_{Σ} is covered by the principle of safe substitution below.

Definition 5. Assuming without loss of generality that all concepts are expressed in DNF, let \leq_{Θ} be the ordering relation defined on concepts such that:

1. $x \leq_{\Theta} y$ for $x, y \in \Omega$ and $x \leq_{\Omega} y$
2. $\neg x \leq_{\Theta} \neg y$ if and only if $y \leq_{\Omega} x$ for $x, y \in \Omega$
3. $\bigcup_{\forall i} w_i \leq_{\Theta} \bigcup_{\forall j} x_j$ if and only if for all $w_j = \bigcap_{\forall k} y_k$ there exists some $x_j = \bigcap_{\forall l} z_l$ such that for every z_l there exists some $y_k \leq_{\Omega} z_l$.

Item 3 essentially extends the subsumption relation to concepts expressed in DNF.

Definition 6. Let $(A, I, O) \leq_{\Sigma} (A', I', O')$ if and only if $\forall (L, E) \in A. \exists (L, E') \in A'. E \leq_{\Theta^*} E'$ and $I' \leq_{\Theta^*} I$ and $O \leq_{\Theta^*} O'$, where \leq_{Θ^*} is the element-wise extension of \leq_{Θ} to lists of concepts.

Note that (Θ, \leq_{Θ}) and (Σ, \leq_{Σ}) respectively form a complete lattice. Given a description of a service $\sigma \in \Sigma$, we can now define the set of substitutables $C(\sigma)$, which in practice corresponds to a query against a database of services Σ , for services that satisfy description σ . Notice that the definition is contravariant for inputs and covariant for outputs. This contravariance is also seen in the field of type theory with regards to polymorphic subtyping, and will be covered in the proof of the principle of safe substitution below.

Definition 7. Let C be a function parametric over services, which maps Σ to a subset of services such that $C(\sigma) = \{\sigma' \mid \sigma' \leq_{\Sigma} \sigma\}$.

In order to be able to prove the soundness and completeness of C , we need a lemma known as the “Principle of Safe Substitution.”

Lemma 1. For any services $\sigma, \sigma' \in \Sigma$, if $\sigma \leq_{\Sigma} \sigma'$ then σ can safely be used in a context expecting service σ' .

Theorem 1. (Soundness of Substitutables) For any services σ and σ' , if service σ' can not be safely used in a context expecting service σ then $\sigma' \notin C(\sigma)$, that is the set of substitutables for a service does not contain any incompatible services.

Proof. Assume the existence of services σ and σ' such that σ' can not be safely used in a context expecting service σ . By the principle of safe substitution it is not

true that $\sigma' \leq_{\Sigma} \sigma$, and hence by the principle of extensionality, $\sigma' \notin C(\sigma)$. Therefore the set of substitutables only contains correct, i.e., compatible services. \square

Theorem 2. (*Completeness of Substitutables*) *For any service σ , if there exists a service σ' that can safely be used in a context expecting service σ then $\sigma' \in C(\sigma)$, that is the set of substitutables contains all services compatible with a given description.*

Proof. Assume there exists such a service σ' , then by the principle of safe substitution $\sigma' \leq_{\Sigma} \sigma$. So by the definition of $C(\sigma)$, $\sigma' \in C(\sigma)$, and therefore the set of substitutables is complete for arbitrary $\sigma \in \Sigma$. \square

The soundness and completeness theorems essentially state that the WordNet subsumption relation provides a strong enough foundation for safe service substitutability. Note that this theory is quite general, and is even applicable to services described using domain specific ontologies (as in OWL-S and WSML).

8. Comparison with OWL-S and WSML

In this section we present a comparison of USDL with other popular approaches such as OWL-S [5] and WSML [1]. Our goal is to identify the similarities and differences of USDL with these approaches. OWL-S is a service description language which attempts to address the problem of semantic description via a highly detailed service ontology. But OWL-S also allows for complicated combining forms, which seem to defeat the tractability and practicality of OWL-S. The focus in the design of OWL-S is to describe the structure of a service in terms of how it combines other sub-services (if any used). The description of atomic services in OWL-S is left underspecified [9]. OWL-S includes the tags *presents* to describe the *service profile*, and the tag *describedBy* to describe the *service model*. The profile describes the (possibly conditional) states that exist before and after the service is executed. The service model describes how the service is (algorithmically) constructed from other simpler services. What the service actually accomplishes has to be inferred from these two descriptions in OWL-S. Given that OWL-S uses complicated combining forms, inferring the task that a service actually performs is, in general, undecidable. In contrast, in USDL, what the service actually *does* is directly described (via the verb *affects* and its refinements *create*, *update*, *delete*, and *find*).

OWL-S recommends that atomic services be defined using domain specific ontologies. Thus, OWL-S needs users describing the services and users using the services to know, understand and agree on domain specific

ontologies in which the services are described. Hence, annotating services with OWL-S is a very time consuming, cumbersome, and invasive process. The complicated nature of OWL-S's combining forms, especially conditions and control constructs, seems to allow for the aforementioned semantic aliasing problem [9]. Other recent approaches such as WSMO, WSML, etc., suffer from the same limitation [1]. In contrast, USDL uses the universal WordNet ontology to solve this problem.

Note that USDL and OWL-S can be used together. A USDL description can be placed under the *describedBy* tag for atomic processes, while OWL-S can be used to compose atomic USDL services. Thus, USDL along with WordNet can be treated as the universal ontology that can make an OWL-S description complete. USDL documents can be used to describe the semantics of atomic services that OWL-S assumes will be described by domain specific ontologies and pointed to by the OWL-S *describedBy* tag. In this respect, USDL and OWL-S are complementary: USDL can be treated as an extension to OWL-S which makes OWL-S description easy to write and semantically more complete.

OWL-S can also be regarded as the composition language for USDL. If a new service can be built by composing a few already existing services, then this new service can be described in OWL-S using the USDL descriptions of the existing services. Next, this new service can be automatically generated from its OWL-S description. The control constructs like *Sequence* and *If-Then-Else* of OWL-S allows us to achieve this. Note once a composite service has been defined using OWL-S that uses atomic services described in USDL, a new USDL description must be written for this composite service (automatic generation of this description is currently being investigated [10]). This USDL description is the formal documentation of the new composite service and will make it automatically searchable once the new service is placed in the directory service. It also allows this composite service to be treated as an atomic service by some other application.

For example, the aforementioned *ReserveFlight* service which creates a flight reservation can be viewed as a composite process of first getting the flight details, then checking the flight availability and then booking the flight (creating the reservation). If we have these three atomic services namely *GetFlightDetails*, *CheckFlightAvailability* and *BookFlight* then we can create our *ReserveFlight* service by composing these three services in sequence using the OWL-S *Sequence* construct. The following is the OWL-S description of the composed *ReserveFlight* service.

```

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999
    /02/22-rdf-syntax-ns#"
  xmlns:process="http://www.daml.org/services
    /owl-s/1.0/Process.owl#">
<process:CompositeProcess rdf:ID="ReserveFlight">
  <process:composedOf>
  <process:Sequence>
    <process:components
      rdf:parseType="Collection">
    <process:AtomicProcess
      rdf:about="#GetFlightDetails"/>
    <process:AtomicProcess
      rdf:about="#CheckFlightAvailability"/>
    <process:AtomicProcess
      rdf:about="#BookFlight"/>
    </process:components>
  </process:Sequence>
</process:composedOf>
</process:CompositeProcess>
</rdf:RDF>

```

We can generate this composed *ReserveFlight* service automatically by using the USDL descriptions of the component services for discovering them from the existing services. Once we have the component services, the OWL-S description can be used to generate the new composed service.

Another related area of research involves message conversation constraints, also known as behavioral signatures [13]. Behavior signature models do not stray far from the explicit description of the lexical form of messages, they expect the messages to be lexically and semantically correct prior to verification via model checking. Hence behavior signatures deal with low-level functional implementation constraints, while USDL deals with higher-level real world concepts. However, USDL and behavioral signatures can be regarded as complementary concepts when taken in the context of real world service composition and both technologies are currently being used in the development of a commercial services integration tool [12].

9. Conclusions and Future Work

To reliably catalogue, search and compose services in a semi-automatic to fully-automatic manner we need standards to publish and document services. This requires language standards for specifying not just the syntax, i.e., prototypes of service procedures and messages, but it also necessitates a standard formal, yet high-level means for specifying the semantics of service procedures and messages. We have addressed these issues by defining a universal service-semantics description language, its semantics, and we have proved some useful properties about this language. The current version of USDL incorporates current standards in a way

to further aid markup of IT services by allowing constructs to be given meaning in terms of an OWL based WordNet ontology. This approach is more practical and tractable than other approaches because description documents are more easily created by humans and more easily processed by computers. USDL is currently being used to formally describe web-services related to emergency response functions [11]. Future work involves the application of USDL to formally describing commercial service repositories (for example SAP Interface Repository and services listed in UDDI), as well as to service discovery and rapid application development (RAD) in commercial environments [12]. Future work also includes developing tools that will allow automatic generation of new services based on combining USDL descriptions of existing atomic services. The interesting problem to be addressed is: can USDL description of such automatically generated services be also automatically generated?

References

- [1] A conceptual comparison between wsmo and owl-s. <http://www.wsmo.org/TR/d4/d4.1/v0.1>.
- [2] Ontology-based information management system, wordnet owl-ontology. <http://taurus.unine.ch/knowler/wordnet.html>.
- [3] Resource description framework. <http://www.w3.org/RDF>.
- [4] Sap interface repository. <http://ifr.sap.com/catalog/query.asp>.
- [5] Semantic markup for web services. <http://www.daml.org/services/owl-s/1.0/owl-s.html>.
- [6] Web ontology language reference. <http://www.w3.org/TR/owl-ref>.
- [7] Web services description language. <http://www.w3.org/TR/wsdl>.
- [8] Wordnet: a lexical database for the english language. <http://www.cogsci.princeton.edu/~wn>.
- [9] S. Balzer, T. Liebig, and M. Wagner. Pitfalls of owl-s - a practical semantic web use case. In *ICSOC*, 2004.
- [10] A. Bansal, S. Kona, L. Simon, A. Mallya, G. Gupta, and T. Hite. Automatic querying and composite services generation with usdl. Working paper, 2005.
- [11] A. Bansal, K. Patel, G. Gupta, B. Raghavachari, E. D. Harris, and J. C. Staves. Towards intelligent services: A case study in chemical emergency response. In *ICWS*, pages 751–758, 2005.
- [12] T. Hite. Service composition and ranking: A strategic overview. Internal Report, Metalect Inc., 2005.
- [13] R. Hull and J. Su. Tools for design of composite web services. In *SIGMOD*, 2004.
- [14] L. Simon, A. Bansal, A. Mallya, G. Gupta, and T. Hite. Towards a universal service description language. In *NWESP*, 2005.