CS565
Simulation, Verification and Composition of Web Services

Michalis Athanasakis
michath@csd.uoc.gr

Munasic Priom
priom_pires@hotmail.com

Kostas Plelis
plelis@csd.uoc.gr

Semantic Web

Introduction

Semantic Web currently is contained in more than four million Webdomains based on a survey done in 2013. But what is exactly the Semantic Web? The Semantic Web is an extension of the web though standards defined by the World Wide Web Consortium (W3C). These standards promote common data formats and exchange protocols on the Web, most fundamentally the Resource Description Framework (RDF). According to the W3C, "The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries". The term was coined by Tim Berners-Lee for a web of data that can be processed by machines. While its critics have questioned its feasibility, proponents argue that applications in industry, biology and human sciences research have already proven the validity of the original concept.

Standards

As we said before, the standardization of Semantic Web in the context of Web 3.0 that was defined by Tim Berners-Lee is under the care of W3C. When we talk about standardization we mean the standardization of the formats and the technologies that enable the Semantic Web. These technologies include:

- Resource Description Framework (RDF), a general method for describing information
- RDF Schema
- Simple Knowledge Organization System (SKOS)
- SPARQL, an RDF query language

- Notation3 (N3), designed with human-readability in mind
- Turtle (Terse RDF Triple Language)
- Web Ontology Language (OWL), a family of knowledge representation languages
- Rule Interchange Format (RIF), a framework of web rule language dialects supporting rule interchange on the Web

Applications

All these standards enabled us to enhance the usability and the usefulness of the Web and its interconnected resources through.

- Servers which expose existing data systems using the RDF and SPARQL standards. Many converters to RDF exist from different applications. Relational databases are an important source. The semantic web server attaches to the existing system without affecting its operation.
- Documents "marked up" with semantic information (an extension of the HTML <meta> tags used in today's Web pages to supply information for Web search engines using web crawlers). This could be machine-understandable information about the human-understandable content of the document (such as the creator, title, description, etc.) or it could be purely metadata representing a set of facts (such as resources and services elsewhere on the site).
- Common metadata vocabularies (ontologies) and maps between vocabularies that allow document creators to know how to mark up their documents so that agents can use the information in the supplied meta-data.
- Automated agents to perform tasks for users of the semantic web using this data
- Web-based services (often with agents of their own) to supply information specifically to agents, for example, a Trust service that an agent could ask if some online store has a history of poor service or spamming.

Take notice of the last bullet because we are going to analyze Web-based services much more and all the challenges its simulation, verification and composition of these services might bring. But before that a quick introduction about what exactly a web service is.

Web Services

Introduction

A Web service is a method of communication between two electronic devices over a network. It is a software function provided at a network address over

the Web with the service always on as in the concept of utility computing. The W3C defines a Web service generally as:-

> a software system designed to support inter operable
> machine-to-machine interaction over a network.

Web services have automated tools to help generate them. There are approached both bottom-up and top-down oriented. Web services can exist without the necessary presence of the semantic web. Though, combining both the technologies provided by the web services and the semantic web it can give a major advantage and enrich the experience of the users.

Semantic Web Services

Semantic Web Services, like conventional web services, are the server end of a clientserver system for machine-to-machine interaction via the World Wide Web. Semantic services are a component of the semantic web because they use markup which makes data machine-readable in a detailed and sophisticated way (as compared with human-readable HTML which is usually not easily "understood" by computer programs).

The reasons that we were not satisfied with the already existing services is that we have yet problems to address that could be solved using the principles of the semantic web.

The mainstream XML standards for inter operation of web services specify only syntactic interoperability, not the semantic meaning of messages. For example, Web Services Description Language (WSDL) can specify the operations available through a web service and the structure of data sent and received but cannot specify semantic meaning of the data or semantic constraints on the data. This requires programmers to reach specific agreements on the interaction of web services and makes automatic web service composition difficult.

Semantic web services are built around universal standards for the interchange of semantic data, which makes it easy for programmers to combine data from different sources and services without losing meaning.

Currently, Semantic Web services platform use Web Ontology Language(OWL) to allow data and service providers to emantically describe their resources using third-party ontologies like Simple Semantic Web Architecture and Protocol (SSWAP) But the creation and usage of OWL is based on other languages like DARPA Agent Markup Language (DAML) or Ontology Inference Layer (OIL). DAML and OIL were combined and DAML+OIL was created.

DAML-S

Introduction

The target is to find ways and technologies that will enable us to describe, simulate, automatically compose, test and verify Web service compositions. For this

to work it is necessary to have a way to describe these Semantic Web services. We come to the conclusion that we are in need of a semantic markup language able to describe the content and the capabilities of these web services. The state of the art include languages like OWL we mentioned before. We are going to focus on the DARPA agent markup language (DAML). After the creation of DAML+OIL we improved and created DAML-S. DAML-S is markup language based on DAML+OIL that was created to specifically describe services. Correspondingly, as DAML was replaced with OWL, so DAML-S was also superseded by OWL-S.

DAML+OIL

DAML+OIL is an AI-inspired description logic-based language for describing taxonomic information. The DAML+OIL is based on top of XML and RDF(S), both defined in the Semantic Web standards. Essentially it is a language with well-defined semantics and a set of language constructs. These include classes, subclasses and properties with domains and ranges. All these can be used to describe a Web domain.

DAML-S

DAML-S is DAML+OIL ontology for Web services. It is also developed under the auspices of the DAML program. There are few different releases as expected. The DAML-S ontology is able to describe a set of classes and properties, specific to the description of Web services. It is separated in parts, the upper ontology of DAML-S is created by a service profiler used for describing service advertisements, another process model to enable the description of actual programs that execute each service and a service grounding for describing the transport-level messaging information associated with the actual execution of the program.

DAML-S Examples

DAML-S process model is able to describe two different types of processes, *atomic* and *composite*. Also it allows the creation a type of *simple* process, which is a description of a view or abstraction of a atomic or composite process to which expands.

```
<daml:Classrdf:ID="Process">
        <daml:unionOfrdf:parseType="daml:collection">
        <daml:Classrdf:about="#AtomicProcess"/>
        <daml:Classrdf:about="#SimpleProcess"/>
        <daml:Classrdf:about="#CompositeProcess"/>
    </daml:unionOf>
</daml:Class>
```

An atomic process is a non-decomposable Web-accessible program. It is executed in a single http call and returns a response. An example of a atomic

process is the `LocateBook` that either returns information about the book or nothing.

```
<daml:Classrdf:ID="LocateBook">
    <rdfs:subClassOf rdf:resource="&process;#AtomicProcess"/>
</daml:Class>
```

On the other hand a composite process is composed of other composite or atomic processes. These composing processes are connected using control constructs, such as *sequence, if-then-else, while, fork etc.* An example of a composite process is the `Find-n-Buy` service that contains the LocateBook along with some other services.

```
<daml:Classrdf:ID="CompositeProcess">
    <daml:intersectionOf rdf:parseType="daml:collection">
        <daml:Classrdf:about="#Process"/>
            <daml:Restrictiondaml:minCardinality="1">
                <daml:onPropertyrdf:resource="#composedOf"/>
            </daml:Restriction>
    </daml:intersectionOf>
</daml:Class>
```

Also, each process is able to has a set of properties, called parameters and can be either be input or output. In the case of the LocateBook an input parameter might be something like the name of the book.

```
<rdf:Propertyrdf:ID="bookName">
    <rdfs:subPropertyOf rdf:resource="&process;#input"/>
    <rdfs:domainrdf:resource="#LocateBook"/>
    <rdfs:rangerdf:resource="&xsd;#string"/>2
</rdf:Property>
```

Inputs can be either mandatory or optional. On the other hand, output are usually optional. This is important, in the case of locating and buying a book it is obligatory to provide any type of information about the book but it is optional for an output of type of price, since there is a possibility the book is not found or not currently available. We define this as a conditional class that both describes a condition and an output based on that specific condition.

```
<rdf:Property rdf:ID="output">
    <rdfs:domain rdf:resource="#parameter"/>
    <rdfs:range rdf:resource="#ConditionalOutput"/>
</rdf:Property>
    <daml:Class rdf:ID="ConditionalOutput">
    <daml:subClassOf rdf:resource="http://www.daml.org/...#Thing"/>
    </daml:Class>
    <rdf:Property rdf:ID="coCondition">
    <rdfs:comment>
        The condition of the conditional output.
```

```
    </rdfs:comment>
    <rdfs:domain rdf:resource="#ConditionalOutput"/>
    <rdfs:range rdf:resource="#Condition"/>
</rdf:Property>

<rdf:Property rdf:ID="coOutput">
    <rdfs:comment>
        The output of the conditional output.
    </rdfs:comment>
    <rdfs:domain rdf:resource="#ConditionalOutput"/>
    <rdfs:rangerdf:resource="http://www.daml.org/...#Thing"/>
</rdf:Property>
```

This offers the ability to the program or function metaphor to be utilized in order to perform real actions in order to create a service. Essentially preconditions for every process is that the process agent has to know in order to execute the process. For example, to execute LocateBook() properly you must execute BookStoreHasBook(bookName). Many web services that are utilized as programs in the web have only preconditions. We analyze further only the abstract preconditions since real physical devices might have more such as bandwidth resources and limited memory.

DAML-S Semantic & Situation Calculus

DAML+OIL provides many necessary tools to describe the content and the capabilities of the web. The problem is that DAML+OIL is not able to restrict DAML-S to all and only intended interpretations. The target here is to describe a semantics for that portion of DAML-S that is really relevant to this work, simulation, verification and composition of web services.

The target here is to describe the actual DAML-S in a more expressive language , such as first-order logic. DAML-S is a process modeling language there is a way to transform it to a Process Specification Language(PSL). PSL is a process specification ontology that can be described in a situation calculus, a first-order logic language for dynamic systems.

Situation Calculus

The situation calculus language we are going to use is for representing dynamically changing worlds in which all of the changes are the direct results of actions performed by an agent. On the hand, Situations are sequences of actions evolving from a specific situation, called S0. So, if a is an action and s is a situation then the result of performing a in a s situation is *do(a,s)*. For example, we can have Own(bookName,s). Finally, Poss(a,s) is a specific fluent expressing that an action a is possible to perform in a situation s.

Also the notation of Knows(ϕ ,s) denotes that formula f is a known in a situation s. For example, Knows(Owns("On the road",s)). As a logical step we have Kwhether(ϕ ,s) = Knows(ϕ ,s) Knows($\neg\phi$,s).

$$Poss(a,s) \wedge \gamma_F^+(x,a,s) \rightarrow F(x,do(a,s))$$
$$Poss(a,s) \wedge \gamma_F^-(x,a,s) \rightarrow \neg F(x,do(a,s))$$

Figure 1: Conditional effects and outputs

Using all this we can create conditional effects and outputs. The conditional effects of an atomic process are represented in the situation calculus as both positive and negative effect axioms of the following form:

To sum up, the complete situation calculus axiomatization of a DAML-S description includes the sets of axioms described above,

- successor state axioms, Dss
- action precondition axioms, Dap
- foundational axioms of the situation calculus,
- axioms describing the initial situation, DS0
- unique names for actions, Duna
- domain closure axioms for actions, Ddca

Semantics

After describing the necessary semantics to the relevant subset of the DAML-S we can talk more about the actual running of web services. We are using this representation in order to provide to special-purpose machines for task we want to execute. More to the point, we use Petri Nets. Petri Nets are a distributed operational semantics of processes. There are other options available but only Petri Nets is capable of performing quantitative analysis. The usage of Petri Nets also provides the ability to perform computational semantics, it is easy to implement and can address offline tasks such as Web service composition and online execution tasks such as deadlock determination resource satisfaction, and quantitative performance analysis.

There are tradeoffs by using Petri Nets. We have to consider their natural representation of change and and concurrency, it gives us the ability to construct a distributed and executable operational semantics of Web services. Also, much research has been done that offers the advantage of a lengthy literature that can support the usage of Petri Nets in cases like this. Finally we have to consider the ability of Petri Nets to deal with resources, something that is very important in real web services.

Petri Nets

Since we have constructed a situation calculus theory using the DAML-S we can use these models and utilize the Petri Nets to perform our tasks, simulation,

evaluation and automatic composition. Before continuing in order to define the basic principles that define Petri Nets let us give a short introduction.

Introduction

A Petri Net is a bipartite graph containing places (drawn as circles) and transitions (drawn as rectangles). Places hold tokens and represent predicates about the world state or internal state. Transitions are the active component. When all of the places pointing into a transition contain an adequate number of tokens (usually 1) the transition is enabled and may fire, removing its input tokens and depositing a new set of tokens in its output places.

The important features of Petri Nets is the ability to model events and states in a distributed environment. To improve the usage of the system we have includes formalisms like typed arcs, hierarchical control, durative transitions, parameterization, typed (individual) tokens and stochasticity.

Mapping

This part describes our automatic model construction, simulation and analysis of DAML-S markups using the theory of Petri Nets.

A Petri Net (PN) is an algebraic structure (P, T, I, O) composed of:

- finite set of places, P = p1, p2, ... pn
- finite set of transitions, T = t1, t2, ... tm
- Transition Input Function, I. I maps each transition ti to a multiset of P
- Transition Output Function, O. O maps each transition ti to a multiset of P

Definition 1 (Petri Nets) A Petri Net (PN) is an algebraic structure (P, T, I, O) composed of:

- finite set of places, P = p1, p2, ... pn
- finite set of transitions, T = t1, t2, ... tm
- Transition Input Function, I. I maps each transition ti to a multiset of P
- Transition Output Function, O. O maps each transition ti to a multiset of P

Definition 2 (Markings/Tokens/Initial marking) A marking in a Petri Net PN(P, T, I, O) is a function μ, that maps every place into a natural number.

Definition 3 (Enabled/Fireable transitions at marking μ) At a given marking μ, if for any ti ϵ T, μ(p) $\geq$ #[p,I(ti)], $\forall$ p ϵ P, then ti is said to be enabled by the marking μ.

Definition 4 (Transition firing/Occurrence sequence) The firing of any enabled transition, ti, at marking μ, causes the change of the marking μ to a new marking μ' as follows: $\forall$ p ϵ P, μ'(p) = μ(p) - #[p, I(ti)] + #[p, O(ti)]. Where: #[p, I(ti)] and #[p, O(ti)], denotes, the number of occurrences of place p in the multiset I(ti) and in the multiset O(ti) respectively. A sequence of firings (t1 ...tn) that take an initial marking μ0 to a new marking N is called an occurrence sequence.

Graphical representation The algebraic structure of a Petri Net PN(P, T, I, O) may be represented graphically. In this graphical representation, a Petri Net will be represented by a bipartite graph, where: every place will be represented by a circle; every transition will be represented by a rectangle ; the function I will be represented by directed arcs linking every p $\in$ I(ti) to the transition ti. These arcs are called input arcs to the transition ti; and the function O will be represented by directed arcs linking each transition ti to every p $\in$ O(ti). Analogously with the input arcs, these arcs are called output arcs to the transition ti.

Following we provide some examples on how several types of Distributed OPErational (DOPE) Semantics for the DAML-S Composite Process Constructs would look based on their functionality.

Sequence has a list of components that are sub-processes that specify the body. The semantics of the sequence is the execution of several sub-processes with a predefined way. We assume that all the preconditions are satisfied by the execution of each sub-process such as Process 1 and Process 2 so we can finally end the sequence.

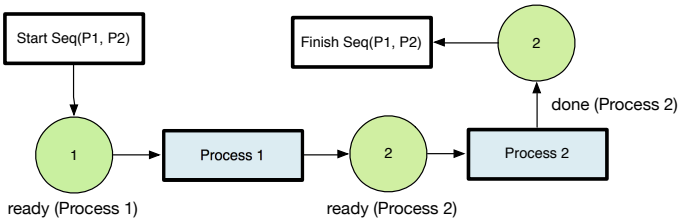

Figure 2: The sequence construct