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).

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,

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

Figure 1: Conditional effects and outputs

- 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.

Split Construct A split composite process is consisted by several sub-processes that are being run concurrently. It simulates the usage of threads in a computer system. For the reasons of this simulation models we do not assume any other standards about how the actual processes are executed. There are any more specifications about waiting times, actual synchronization between the sub-processes due to limitation by this specific model of DAML-S. In order to solve this problem we create new constructs to facilitate the synchronization aspects. The Figure below shows how the two sub-processes work. When both

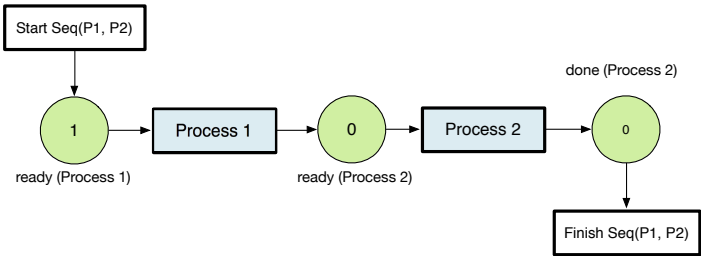

Figure 2: The sequence construct

processes are finished a transition fires and we are ready to end the Split(P1, P2) process.

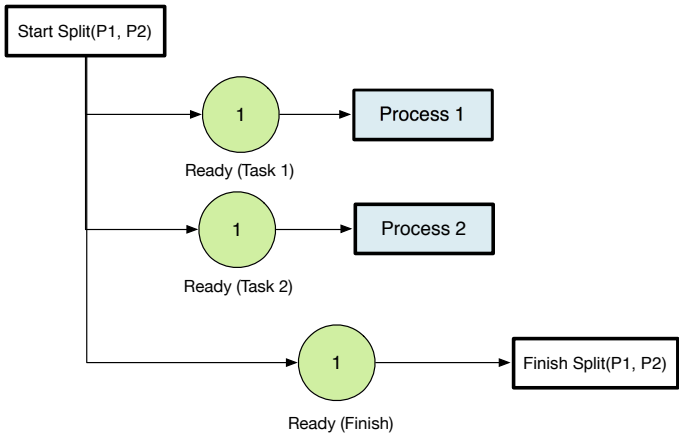

Figure 3: The Split construct

Split and Join Construct A Split and Join composite process consists of a number of processes being executed concurrently. This requires a state like a Synchronize in order for all the processes to finalize and be ready to end. This means that there is partial synchronization between the processes but the composite process can end only if all the sub-processes have been done. Regardless, this enables an optimization of sub-processes running together if they are not depended in each other. This can been shown from the below figure, the Process 1 and 2 are being executed when they are able. After the finalization of

execution of both processes we arrive at a state of Synchronization that enables us to finish the process.

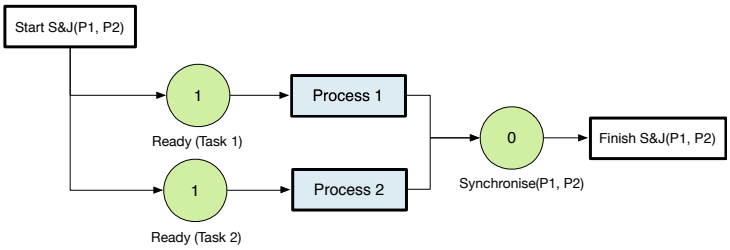

Figure 4: The Split and Join construct

Choice Construct A Choice composite processes consists of a number of sub-processes. When the time of execution comes a decision is taken and the process chooses a sub-process to execute. The chosen sub-process takes as a parameter a choice bag and is obligated to return another chosen bag. The cardinality of the bag can be expressed by the restriction of Choose(n) where n is $(0 < n < |bag|)$. DAML-S is limited in this way and is not able to separate each choice from the other alternatives from the bag of choices. As shown in the figure below Process 1 and 2 are both in the bag of choices. The DAML-S specification corresponds to both as both are being viable and valid possible choices. The selection and the execution of one of the sub-processes allows the model to transition to a state of Done(Choose) regardless the actual choices. This allows the finish of the choice construct composite process.

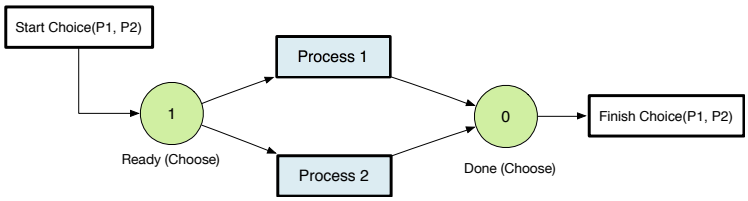

Figure 5: The Choice construct

The if-then-else Construct An if-then-else composite process is a simple construct that has a relation whose domain is a process and whose range is a binary value. The evolution of the processes can be affected by the evalua-

tion of the world state along with one or more corresponding test actions. This allows the choice of possible sub-processes in relation with the satisfaction of each condition. This is achieved by using DAML-S. DAML-S has a property called *conditionValue* which is type of boolean. The actual execution of each branch depends on the value of this boolean value. The figure shown below depicts that exact construct. The is the choice between a Process Then and a Process Else. In order the system to choose what path is followed it is necessary to read and examine the property Ready(if-then-else). In case the Condition is false the lower path is followed, in the other case that condition is true then we execute sub-process Process Then. Either way when the corresponding process has been executed and terminated we arrive at a state of Done(Then or Else) that allows us to finish the composite process of if-then-else.

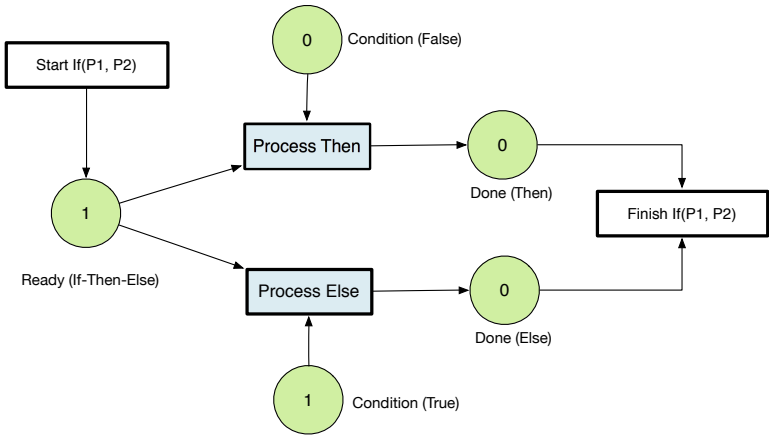

Figure 6: The If-Then-Else construct

The repeat-condition Construct DAML-S has construct to offer repeat-while and repeat-until composite processes. There are differences between what each construct offers to the agent. In the case of Repeat-while we are using a class called ControlConstruct with properties like whileCondition, which has a type of repeat. Until this stage there are no restrictions or obligations this construct to be asynchronous(without the need of prioritized interrupts) or synchronous with specific policies that implement polling or busy-waiting strategies. On the other hand Repeat-Until takes the ControlConstruct and does it more specific to adjusts to its needs. We have properties such as untilCondition(range if type of Condition) and another untilProcess that has as a type of Repeat. In the particular case the figure below shows the concept of Repeat-Until. In this particular case the Process is executed after all the preconditions are satisfied. After the Process is Done there is a condition check in order to find out if it is necessary to do one more iteration. If the iteration

is chosen then we check all the preconditions again and we execute the process once more before the next check. In the case that the condition is checked then we are able to Finish the composite repeat-condition process. The Repeat-while semantic is analogous except that input places for the DAML-S conditionValue (the Condition=true and Condition=false nodes are reversed.)

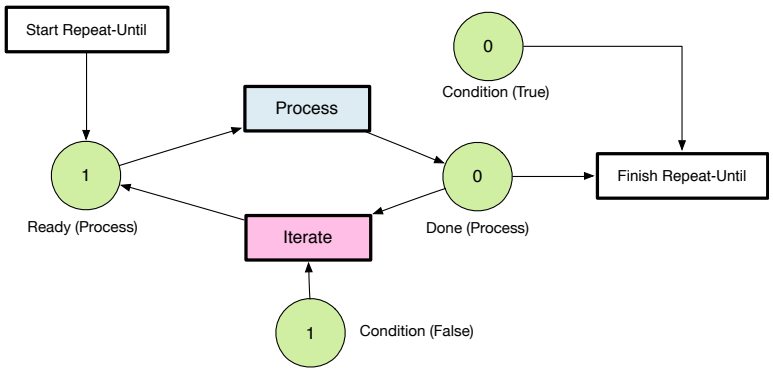

Figure 7: The repeat-condition construct

Analysis of Web Services using Petri Net

After all this introduction of languages, tools and calculus in order to portray a web services let us visit the subject on how we analyze all these services. Right now the whole web is full of web services. The type of these web services ranged from web services manually created by provides of web content, by 3rd party web service providers, or by an automated tool.

It is essential that we have a way to perform an analysis on these web services. Features like correctness, effectiveness, safety and efficiency of composite services is very important in order to be able to facilitate a reliable automation in the world of Web services. In this section we provide a variety of computational analysis tools. In order to construct these tools we use languages such as custom made version of DAML-S based on the Petri Net representation.

This enables us the ability to perform the following activities:

Simulation It is essential in order to analyze a web service to test how a Web service can be executed in real life circumstances under different conditions.

Validation You can simulate a web service but is necessary to be able to determine if the running results are as expected. In order to find out if a running service is running as expected it is necessary to perform a validation.

Verification During the design and the composition of a service a certain amount of properties are being set. Verification offers us a way to check if the values of these properties match the expected ones. This ensures safety etc along with potentially being an indicator on what exact part you can improve of the web service.

Composition A large amount of web services it is possible to generate them using automated tools in order to compose them. The only we have to do is to set a specified goal and a set of properties that has to be matched and respected.

Performance analysis Performance is always in a service running in the web. Essentially what we want is to be able to evaluate the ability of a service to meet certain requirements and more specially regarding throughput times, service levels and the level of resource utilization.

All of the above activities must be implemented in tools in order to provide sophisticated automated performance analysis. There are many possible implementation of these models and constructs and we present some of them below that have utilized the DAML-S language we described before.

Simulation and Validation

The simulation of a Petri Net is simple and straightforward. In a same pattern, validation can be done by essentially running interactive simulation on hypothetical cases. These test cases can either be manually created and evaluated or they can be parts of tests suites that have a designated role of evaluating such services. More specifically tests are being fed to the system in order to check if the output is and the expected effects are correct relative with the Petri Net representation.

In the case of verification, composition and performance analysis more advanced techniques are required that also have already been developed in Petri Net. We require techniques that use linear algebra and can be used in order to verify many properties such as place invariants, transitions invariants and possible non-reachability to certain parts of the services. We can also use coverability graph analysis, model checking and common reduction techniques to analyze dynamic behavior of a Petri Net. Finally, in order to do performance analysis and simulation we can use Markov-chain analysis.

Verification

In the previous sector we talked about what tools could be possibly be used in order to analyze and perform other tasks in a Petri Net. In this section we present more information about the task of verification and the challenges that creates.

Currently the are three major problems with the task of *verification*. We emphasize in the problems of reachability, liveness and the existence of deadlocks. As you can easily understand there is a number of embedded devices and other possible targets the meaning of safe operation is becoming very important and can not be avoided. Try to imagine the meaning of safety in the context of Web services, we must be able to verify that a composite service upholds a certain amount of a safety for each constraint. For example, we must make sure that when a financial transaction takes place using a Web service, very common in nowadays, the charge happens only once and the bought products are being send off only the corresponding times there were bought. In the same logic we are evaluating the verification of safety constraints, the detection of deadlocks and that the actual automated composition of these Web services is being done in terms of reachability.

In order to do that we try to define those principles in the same way we did in the prior work presented in this work.

Reachability A marking M is reachable if it is the marking reached by some occurrence sequence (Definition 4). Given a marking M of N, the set of reachable markings of the net (P; T; F; M) (i.e., the net obtained by replacing the initial marking M0 by M. Notice that the empty sequence is an occurrence sequence and that it reaches the initial marking M0. The reachability problem for a net N is the problem of deciding for a given marking M of N if it is reachable. With the above knowledge we can try and define the meaning of *safety*.

Safety of a distributed system is defined as lack of reachability to an unsafe state

Safety of Web Service Compositions Let S be a Web service composition with associated net (P;T;F;M). Let ϕ be a safety constraint, and let marking M encode the negation (i.e., the violation) of the safety constraint ϕ. Then a Web service composition S is safe with respect to ϕ iff there is no occurrence sequence of the net of S that reaches M.

In a similar way we define the task of generating a composition of Web services to achieve a goal as the problem of finding an occurrence sequence that reaches the marking depicting the user's desired final state. In order to do that we can use a composition of atomic services to achieve the final target state.

Automated Composition of Web Services Let A be a set of atomic Web services and let N=(P;T;F;M) be the net that depicts the behavior of all the services in A. Further, let ϕ represent the users goal, and let M be the marking that depicts this goal in N. Then a1;a2,...;an is a sequential composition of atomic services that achieves user goal ϕ iff a1;a2,...;an is an occurrence sequence in the reachability analysis of M in N. In the case of the composition of a Web service we consider it as an individual service treated as an atomic process. Also, it is important to remember that a smart agent given an agent goal, a service description and our process semantics can produce the optimal composition of using partial service executions.

The idea of using the feature of automated composition for the Web services is analogous of an AI to design systems where we have all the necessary informations from the beginning. Finally, it is essential to remember that in most real life and practical Web services the possible compositions are very branchy and offer many options, but on the final resulting composition tend to be short and the most efficient.

Complexity of DAML-S services tasks

Below we present the complexity of each DAML-S approach. This is important in the case the agent must decide what subset of DAML-S is going to use in order to perform a certain process.

DAML-S subset	Reachability	Deadlock
DAML-S \ Iterate & Choice	Polynomial	Polynomial
DAML-S \ Iterate & Condition	NP-Complete	Constant time
DAML-S \ Iterate	NP-Complete	Polynomial time
DAML-S 0.5	P-Space Complete	P-Space Complete
DAML-S + Resources	Exp- Space-Time-hard [21]	Exp-Space-Time-hard[21]

Figure 8: Tractability results for DAML-S subsets

Possible Implementation

Based on the principles presented during this report we can also present an implementation of this kind of work. There is a research paper called *Simulation, Verification and Automated Composition of Web Services* from Srini Narayanan and Sheila A. McIlraith that provides an actual implementation of what we

presented using the same languages and tools.

More specifically, they utilize a DAML-S interpreter that is able to translate the DAML-S markup into a simulation and modeling environment called KarmaSIM. The KarmaSIM allows the interactive simulation and also supports the various verification and performance analysis techniques that were presented earlier.