Simplifying the Web Service Discovery Process

Nathalie Steinmetz, Mick Kerrigan, Holger Lausen,
Martin Tanler and Adina Sirbu

Semantic Technology Institute (STI) Innsbruck,
Universität Innsbruck, Austria
firstname.lastname@sti2.at

Abstract. One of the crucial reasons for adding semantic descriptions to Web services is to enable intelligent discovery, removing the need for a human to manually search and browse textual descriptions in repositories of services, like UDDI or ebXML. The Web Service Modeling Ontology (WSMO) provides a conceptual model within which the function of a Web service can be described in terms of formalized pre- and postconditions over the information space and assumptions and effects related to the real world; however WSMO is very flexible in the way in which the Semantic Web Service developer can use these elements to describe the functionality of a service. Thus a number of approaches for effectively describing the offered function of a Web service and the requirements of users, along with methods to compare them have surfaced in the last number of years, leaving developers unsure of which approach to use and if it is possible to combine them. In this paper we introduce a framework within which these different approaches can be combined and present some new tools that can be used with this framework by the Semantic Web Service developer.

1 Introduction

Web services are quickly becoming the standard for B2B integration. They have machine-processable annotations that are well structured (using XML) and describe how to interface with these services. However these annotations are purely syntactic and not machine-understandable, thus large amounts of human effort are required to build Service Oriented Architectures (SOA). Semantic Web Services are the extension of Ontologies to describe Web services such that critical, currently human intensive, activities in the process of using Web services can be totally or partially automated, reducing the amount of human effort needed to develop an application using a Service Oriented Architecture. One of these core activities is the process of finding services that can fulfill the requirements of an end user, referred to as the process of Web service discovery, which involves matching the description of a users functional requirements against the descriptions of the functionality provided by individual service providers. Automation of the process of discovering services enables Web service providers and Web service requesters to be truly decoupled in a Service Oriented Architecture as they need not know of each others existence prior to execution.

K. Belhajjame, M. d'Aquin, P. Haase and P. Missier (Eds.): SeMMA 2008
CEUR Workshop Proceedings, ISSN 1613-0073, online at CEUR-WS.org/Vol-346/

The Web Service Modeling Ontology (WSMO)[6] and the Web Service Modeling Language (WSML)[14] provide a conceptual model and formal language within which Web services and user requirements can be captured and semantically described as WSMO Web Services and Goals. In WSMO the functional requirements of end users and the functionality offered by a given providers Web service are described in terms of a WSMO Capability, by specifying conditions on the state of the world that must exist for execution of the service to be possible and conditions on the state of the world that are guaranteed to hold after execution of the service. Thus the process of discovery in a WSMO sense involves matching the requested capability from the end users goal with the provided capabilities of known provider Web Services.

WSMO is very flexible in the way Web service and goal descriptions can be written leading to the creation of a number of different discovery approaches, with these approaches differing in terms of the level of detail in which Web Services and Goals need to be described and each having different associated computational properties, precision and recall, and the amount of effort required to create descriptions. These different approaches enable the support of a wide range of applications with different requirements, from those that require high precision of results to those that require high efficency; however the availability of multiple approaches does cause problems for those wishing to semantically describe their services or create goal descriptions. Developers have become unsure and confused of how to create their descriptions of Web Services and Goals. They are unsure of whether different approaches are compatible with one another and if a given Semantic Execution Environment is capable of supporting their requirements or not.

In the paper we present an overview of the discovery approaches that have become available in the last number of years and the discovery engines that implement them, providing an easy to use framework, that can be deployed within a Semantic Execution Environment or stand alone within an application, which integrates the different discovery engines together and provides a single generic entrypoint for accessing this functionality. We also present some tool enhancements to the Web Service Modeling Toolkit (WSMT), which is an Integrated Development Environment for Semantic Web Services, that can be used by the Semantic Web Service developer alongside this discovery framework. We close the paper with a look at some related work, an evaluation of our approach and a description of some future work and conclusions.

2 Background

In order to describe Semantic Web Services a conceptual model is required. The Web Service Modeling Ontology (WSMO)[6] is such a conceptual model and provides four top level elements, namely Ontologies, Web Services, Goals and Mediators. Ontologies are the basis for the other descriptions by providing the terminology that they use. WSMO Web Services provide a semantic description of both the function of a service, in terms of a Capability, and the mechanism for

interacting with it, in terms of an Interface. A WSMO goal allows for the requirements of the requester to be semantically described. Finally, WSMO Mediators provide a means to resolve heterogeneity issues that inevitably occur between the other elements due to the open and distributed nature of the Web. The Web Service Modeling Language (WSML)[14] is a formalization of the WSMO ontology, providing a language framework within which the properties of Semantic Web Services can be described. There are five language variants, based on Description Logic and Logic Programming. Each language variant provides different levels of logical expressiveness[14]. These variants are: WSML-Core, WSML-DL, WSML-Flight, WSML-Rule and WSML-Full.

At the heart of any Semantically Enabled Service-oriented Architecture, made up of Semantic Web Services, there needs to be a number of services that provide the core functionality needed to bind requesters and providers together dynamically at runtime and to resolve any heterogeneity issues that may exist between them. These services together are termed a Semantic Execution Environment (SEE) and include services for *discovering*, *composing*, *ranking*, *selecting*, *mediating*, and *invoking* Web services in order to meet the requirements of the end user. There are currently two Semantic Execution Environments for WSMO, namely the Web Service Execution Environment (WSMX)[7] and IRSIII[2]. Ongoing work in the OASIS Semantic Execution Environment Technical Committee[1] aims to provide a standard reference architecture for SEEs.

The purpose of developing Semantic Web Services is to totally or partially automate activities that occur in the process of using Web services and one of the most important activities that needs to be automated is the process of finding services that can fulfill the end users goal. This *discovery* step is performed by matching the end users goal description with the set of known Semantic Web Services in a Semantic Execution Environment. This matching involves the comparison of elements from the capability in the goal with elements from the capabilities of the Web Services. A WSMO capability is described in terms of conditions on the state of the world that must exist for execution of the service to be possible and conditions on the state of the world that are guaranteed to hold after execution of the service. WSMO makes a distinction between the state of the information space, i.e. concerning the state of the inputs and outputs of the service, and the state of the real world. Thus a capability is broken up into four main elements, with *preconditions* and *postconditions* making statements about the information space, and *assumptions* and *effects* making statements about the real world. A capability has also a set of shared variables that can be used across preconditions, postconditions, assumptions and effects.

The following example shows two small, matching goal and Web service descriptions in WSMO. They are narrowed down to the necessary, using only postconditions in their capabilities. The Web service offers information about train trips in Austria, and the goal seeks information about trips from Innsbruck to Vienna:

[1] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=semantic-ex

```
webService tripsInEurope
     capability tripsInEuropeCapability
     sharedVariables ?x
     postcondition tripsInEuropePost definedBy
          ?x memberOf TrainTrip and
          forall ?from (?x[from hasValue ?from]
          implies ?from memberOf PlaceInAustria)) and
          forall ?to (?x[to hasValue ?to]
          implies ?to memberOf PlaceInAustria)).

goal tripIbkToVienna
     capability tripIbkToViennaCapability
     sharedVariables ?x
     postcondition tripIbkToVienna definedBy
          ?x memberOf Trip and
          forall ?from (?x[from hasValue ?from]
          implies ?from memberOf PlaceInInnsbruck) and
          forall ?to (?x[to hasValue ?to]
          implies ?to memberOf PlaceInVienna).
```

It is possible to use different levels of abstraction when describing Web Services and Goals with WSMO. Thus we can consider Web Services and Goals as simple abstract objects with properties, using just postconditions within the capability descriptions to specify the targeted or delivered output of a goal or Web service. Alternatively we can describe Web services and goals in a much more fine-grained manner, using conditions on service input and output, assumptions and effects, and taking into account states of the world that represent the world before and after the execution of a service. These different levels of abstraction can be accounted to different approaches to Web service discovery, both in general and in WSMO in particular. Below we describe the three main WSMO discovery approaches that have appeared in the last number of years:

- **Keyword-Based Discovery[10]** is a basic approach to the discovery of Web services, built on the concepts of simple term matching and using standard Information Retrieval methods. Thus a set of keywords from the query is matched with the keywords contained in the service descriptions. Such keyword-based techniques are limited due to the ambiguities of natural language and the lack of semantics; However they have the major advantage of being able to easily scale to a large number of services and can utilize mature keyword matching technologies.
- **"Lightweight" Set-Based Discovery[9]** uses service descriptions that describe the output of the service in an abstract way, taking only the postconditions and effects of services and goals into account, and not considering any of the inputs to the services, i.e. the preconditions and assumptions. Thus the Web service and the goal are described by sets of objects and a match between them is determined if the sets of objects are interrelated, i.e. there is some set-theoretic relationship between the goal set and the Web

Service set. The most basic set-theoretic relationships that are consider in this discovery approach are:

- Set Equality: $SET_{\mathcal{GOAL}} = SET_{\mathcal{WS}}$
- Plugin Relation: the goal description is a subset of the Web Service description - $SET_{\mathcal{GOAL}} \subseteq SET_{\mathcal{WS}}$
- Subsume Relation: the Web service description is a subset of the goal description - $SET_{\mathcal{WS}} \subseteq SET_{\mathcal{GOAL}}$
- Intersection Relation: there exists some common elements between the goal and Web service descriptions - $SET_{\mathcal{GOAL}} \cap SET_{\mathcal{WS}} \neq \emptyset$

– **"Heavyweight" Discovery[11]** is based on richer semantic descriptions than the "lightweight" approach, taking into account the relationship between preconditions, postconditions, assumptions, and effects. Thus this approach allows for the properties of the states before and after Web service Execution to be considered. Using this approach the Semantic Web service developer can clearly specify constraints on the service input, while the developer of a goal description can describe the relationship desired between the input data he will provide and the output the service should provide.

As has already been mentioned the availability of these different approaches to describing services and user requirements semantically has resulted in a number of implementations of each of these approaches being created. In the next section we introduce a framework that brings these different approaches, and their associated implementations, together in order to make these discovery engines more accessible and to reduce the complexity of integration service discovery into an application.

3 The Discovery Framework

In the previous section we introduced three different discovery approaches. At this time there are five different discovery engines which implement these three approaches to discovery. The *Keyword-Based Discovery*[10] engine takes as input a query made up of a set of keywords. The engine matches the keywords from the query to the keywords contained in the service descriptions. Two discovery engines use the *"Lightweight" Set-Based Discovery*[9] approach, one being based on Description Logic and the other on Logic Programming. There currently also exist two different engines, both Logic Programming based, using the *"Heavyweight" Discovery*[11] approach. They differ in that they pursue different strategies in using pre- and postconditions and the relationship between them. It is certain that new approaches to Web service discovery and engines implementing these approaches will appear in the near future.

To ease the use of these different discovery engines by the developer, we think that it is vital to integrate them together into one framework and to provide a single, common, interface to the developer. Furthermore we want to try to support him in choosing which engine is the right one for his individual discovery needs. Therefore it is important that the different discovery engines

can be used together in a Semantic Execution Environment (SEE) like WSMX or IRSIII to effectively provide discovery functionality within a Semantically Enabled Service-oriented Architecture. Thus we have designed and implemented a discovery framework that can integrate different discovery approaches together, providing a single generic interface for discovery within a SEE and providing each integrated engine with access to a registry of Web Services.

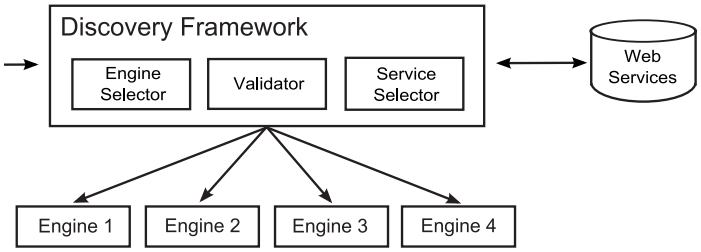

Fig. 1. The Discovery Framework

As described in the previous section the different discovery approaches require different levels of abstraction when describing Web Services and Goals. Obviously they lead to different service descriptions, in the sense of their formal and logical nature. This means that the effort needed to create WSMO Web Services and Goals differs depending on the targeted discovery approach. Providing simple textual description or keywords of services is quite easy, while the description of simple abstract concepts is more difficult, but still manageable. The specification of state-based service descriptions however gets significantly more difficult and calls for a skilled expert to provide the descriptions. Along with differences in effort needed, approaches can differ in terms of their accuracy and effectiveness. Simple textual descriptions may lead to a rather high recall, but with a low precision. Simple abstract concept descriptions may lead to a lower recall, but with a relative increase in precision. The very fine-grained approach, with states corresponding to the world and with reflected pre- and postconditions with the relations in between, leads to a very high precision with recall decreasing, as it becomes harder to actually find a match, as a service will need to fulfill a number of exact requirements to be able to match with a goal. [8] also discusses this gain vs. pain in choosing a discovery approach that leads to a targeted precision and recall. Furthermore the different discovery approaches also have very different computational properties. As service descriptions become more fine grained the complexity of the associated discovery algorithms increases leading to less desirable computational properties.

Due to the features of the integrated discovery engines, the framework supports a wide range of application scenarios from those that need high efficiency to those requiring high precision in the discovery results. The following Section 3.1 provides an overview of the methods that we introduced to allow a flexible framework configuration and Section 3.2 explains how we support the developer in choosing the "right" discovery approach by validating given Web Services and Goals.

3.1 The Framework Configuration

An important facet of the discovery framework is the fact that it is highly configurable. When designing the framework we considered that the developer of a goal description should not have to write a special configuration file, or similar, to use the framework. Thus the goal developer can simply add his configuration wishes to the nonfunctional properties of the Goals that he submits to the framework. Similarly the developer of Semantic Web Services simply specifies which approach the given Web service description is compliant with.

The configuration that can be made in the goal is made up of three parts, namely the discovery engine, the match type and the pre-filter mechanism. Firstly the goal developer can specify which discovery engine should be used by adding the *discovery#discoveryStrategy*[2] Non Functional Property (NFP) to the goal description. The framework extracts this NFP when the goal is submitted, initializing the correct engine and registers the available Web service descriptions that are compliant with this engine. The framework will choose the most appropriate engines if no specific engine is specified within the goal, this is performed by analyzing the goal and establishing which engines it is compliant with (see Section 3.2). The compliant Web Services are chosen in a similar manner, either the Web service developer specifies, in the NFP, the discovery engine(s) he supports or the framework analyzes the Web service and chooses whether they fit to the goal's targeted engines or not.

Secondly the goal developer can specify which target match type, e.g. only exact match, or range of match types, e.g. "exact → plugin", or "exact → plugin → subsume", are desired by specifying the *discovery#typeOfMatch* NFP in the goal. The framework extracts this NFP when the goal is submitted and ensures that the targeted discovery engine can deliver these match types. If no match type has been targeted by the user, the framework returns all types of matches and indicates the detected match type in the nonfunctional properties of the resulting Web services. It is important to note that not all discovery engines support all types of matches (e.g. the *intersection* type of match is currently only supported by the Description Logic based *"Lightweight" Set-Based Discovery* engine). This means that the framework needs to handle "inconsistent" configuration wishes, either due to inconsistencies in the NFPs provided by the user, or resulting from the analysis of the goal by the framework. If a developer targets a discovery engine and a match type in the NFP of his goal that are not compatible, the framework disregards the targeted match type and returns all type of matches allowed by the chosen discovery engines. Furthermore the framework indicates the configuration error in the nonfunctional properties of the resulting Web services.

Finally the goal developer can choose the pre-filtering mechanism that should be employed by specifying the *discovery#preFilter* NFP in the goal. A keyword-based discovery engine can be used as a pre-filter to any of the other engines, in order to reduce the thousands of available Web service descriptions in the registry

[2] discovery : http://wiki.wsmx.org/index.php?title=DiscoveryOntology#

down to a more manageable number. Combining, for example, a "Heavyweight" Discovery approach with a Keyword-based approach increases the performance of the "Heavyweight" approach while improving the precision of the keyword-based approach.

3.2 Validation

Choosing a discovery approach to use for a given application scenario involves trading off the computational properties, the achievable accuracy and the effort needed to create service descriptions. The choice of discovery engine to use to achieve this discovery approach is based upon the choice of WSML Variant made by the developer. As can be seen above, to effectively enable the configurability described it is necessary that it be possible to check the conformance of a given Web service or goal description against a given discovery engine. This "validation" ensures that only compliant Web Services are registered with a given discovery engine, improving performance of each discovery engine and also enabling the selection of potential discovery engines based on the conformance of the provided goal to these engines.

As already mentioned, the choice of WSML Variant allows Web service and goal descriptions to be written with different levels of logical expressiveness and in different styles, based on the underlying logic language, for example Logic Programming[15] or Description Logics[1]. When using the WSML[14] syntax to write down the semantic description of a WSMO Web service or goal it should be noted that the WSML syntax is broken down into two parts. The conceptual syntax is based on the structure of the WSMO conceptual model, and is independent from the underlying logic, shielding the developer from the particulars of the underlying language, while the logical expression syntax provides access to the full expressive power of the WSML variant chosen by the developer. Thus when describing services the conceptual objects, like Web Services, Goals and Capabilities are always written in the same way, but the content in the preconditions, postconditions, assumptions and effects will change depending on the WSML variant and the discovery approach pursued.

To be able to fulfill the needs resulting from the discovery framework configurability, we have introduced a validation functionality within the Discovery Framework that can be used in two ways. Firstly it supports the developer in writing correct service descriptions as it allows the validation of Web Services, Goals and Ontologies with regard to the WSML variant specified in the header of the WSML file. Secondly, and most importantly in this context, it allows the determination of the variant of a WSML description and thus the determination of which discovery engines can be used for a given goal.

The validation is performed over two different description aspects of the specified Goals and Web Services, namely the "structures" (e.g. precondition, postcondition) used in the capabilities and the logical expressions used within these structures. Firstly, the Web service and goal descriptions may use different capability "structures" i.e. preconditions, postconditions, effects, assumptions and

shared variables, the *"Heavyweight" Discovery* engine for example is the only discovery approach to take into account the preconditions of service descriptions. Secondly the logical expressions defined within the capability descriptions need to comply to the WSML variant that is supported by the discovery engine, i.e. WSML-Core, WSML-DL, WSML-Flight or WSML-Rule. Thirdly, certain discovery engines only accept restricted logical expressions, e.g. descriptions may not be allowed to contain constraints, implications, inverse implications, equivalence implications, logic programming rules, negation or possibly disjunctions. This depends on which features the used WSML variant allows, e.g. constraints are not allowed in WSML-Core and WSML-DL. Finally, it is important to ensure that the ontologies imported by the goal or Web service descriptions and used within the logical expressions conform to the WSML variant supported by the discovery engine in question. This validation involves checking whether both the conceptual and the logical expression definitions within the ontology are compliant to the WSML variant supported by the targeted discovery engines.

Let us have a look at the example goal that we introduced in Section 2:

```
goal tripIbkToVienna
     capability tripIbkToViennaCapability
     sharedVariables ?x
     postcondition tripIbkToVienna definedBy
          ?x memberOf Trip and
          forall ?from (?x[from hasValue ?from]
          implies ?from memberOf PlaceInInnsbruck) and
          forall ?to (?x[to hasValue ?to]
          implies ?to memberOf PlaceInVienna).
```

By looking at the "structures" used in this description we see that we cannot use the *"Heavyweight" Discovery* engine for finding matching services, as this goal does not describe any preconditions or assumptions. When validating the logical expression used in the capability, we find out that it is only compliant to WSML-DL. This leads to the conclusion that with this goal we can only use the *"Lightweight" Set-Based Discovery* based on Description Logic.

As will be seen in the next section this validation support can be reused to aid the developers of Semantic Web Services and Goals in building valid and compliant descriptions at design time.

4 Supporting the Semantic Web Service Developer

The Web Services Modeling Toolkit (WSMT)[12][13] is an Integrated Development Environment (IDE) for Semantic Web Services implemented in the Eclipse framework. The WSMT aims to support the developer through the full development cycle of Semantic Web Services, developed through the WSMO paradigm, in order to improve the productivity of the developer and to ensure the quality of the Semantic Web Services produced. The tools provided in the WSMT are seamlessly integrated with one another and thanks to the Eclipse framework can

be integrated with other toolkits like the Java Development Toolkit (JDT)[3] or the Web Tools Platform (WTP)[4] so that the developer can create his Java code, Web services and Semantic Web Services side by side in one application.

Over the last three years the WSMT has provided text-based, form-based and graph-based editors and visualizers allowing the engineer to create and manage WSMO descriptions through the WSML language. It provides validation support for ensuring that the WSMO Ontologies, Web Services, Goals and Mediators created by the developer are syntactically correct and that the Ontologies conform to the WSML variant specified. A number of reasoners for the Description Logic and Logic Programming variants of the WSML Language are embedded within the WSMT enabling the developer to execute sample queries over the semantic descriptions created to ensure they behave as expected. Developers can also created Ontology Unit Tests, where competency questions are encoded as tests, to ensure that ontologies still behave as expected as they evolve. The WSMT has an Eclipse Perspective dedicated to the interaction with Semantic Execution Environments (SEE), such as WSMX and IRSIII, through which developers can manage their connections to SEEs, browse the content stored on these SEEs, store or retrieve WSMO descriptions to or from a SEE, and invoke any of the entry points they expose, for example the entry point for achieving a goal.

However as mentioned before, the fact that WSMO is very flexible in terms of how a developer can specify Web Services and Goals means that the WSMT, while providing generic support for creating WSMO Web service and goal descriptions, has in the past been weak in terms of providing useful tools to the developer for the process of building Web service and goal descriptions. While the initial work described in [13] provided one of the discovery engines that implemented the "Lightweight" Set-Based discovery approach to the developer such that the Goals and Web Services produced by the engineer could be tested against one another, this discovery engine was not available in any Semantic Execution Environment and was only usable by those using this specific discovery approach.

To further support the Semantic Web Service developer through the development cycle we have extended the WSMT in two crucial ways. Firstly, as described for Ontologies in our previous work in [12], the developer can waste huge amounts of time trying to debug errors related to small syntactical or modeling mistakes. Up until now there has been no validation support for Web Services and Goals available in the WSMT, due to the ambiguity regarding what should actually be present within a precondition, postcondition, assumption and effect. The WSMT now reuses the validation support within the discovery framework, providing the Semantic Web Service developer with immediate feedback within the development environment regarding not only syntactical mistakes but also the conformance of the Web service or goal being created with the target discovery engine as specified with the *discovery#discoveryStrategy* Non Functional Property. This feedback is delivered to the developer via the same approaches

[3] http://www.eclipse.org/jdt/
[4] http://www.eclipse.org/webtools/

as when he builds WSMO Ontologies, namely via annotations within editors, in a list view in the Eclipse Problems view and via graphical annotations on those files with errors in the WSML Navigator.

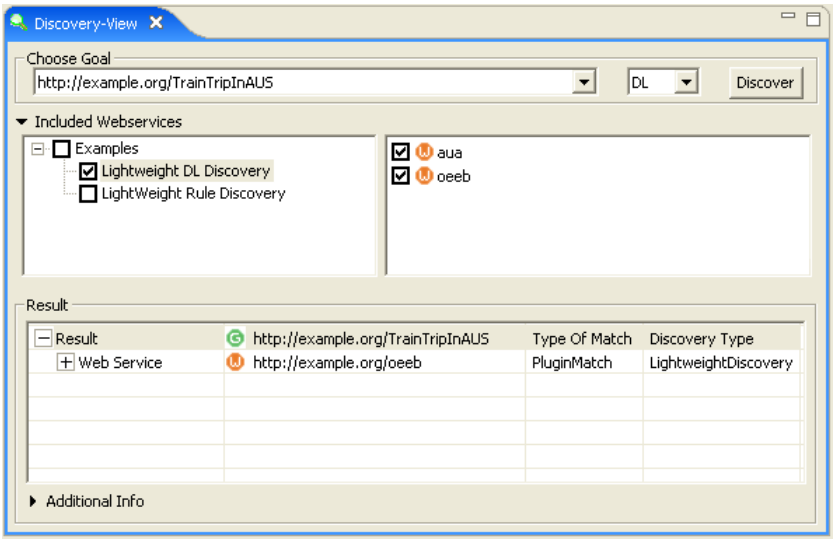

Fig. 2. The WSMT Discovery View

Secondly, the discovery view introduced in [13] has been extended such that the individual embedded discovery engine has been replaced with the discovery framework itself, as can be seen in figure 2. Now the developer can quickly test not only the conformance of his descriptions to all of the engines available through the discovery framework, but can also ensure that the relevant Web Services are discovered for the goal descriptions as expected. This testing can be performed without having to repeated deploy the Web service descriptions to a Semantic Execution Environment in order to perform testing, drastically reducing the duration of the testing cycle.

The importance of testing in the development cycle of Semantic Web Services cannot be emphasized enough. Considering a scenario where Amazon made a decision to provide their Web services as Semantic Web Services, they would also want to provide some sample goal descriptions to the community to ensure that those new to Semantic Technologies could find their services quickly and simply. Thus the developer in Amazon trusted with the development of Web service and goal descriptions for this new effort needs to be confident that these sample Goals do indeed match the Amazon services as expected. Considering a similar scenario when a new competitor service wishes to semantically describe their services, it would be very important for the developers of these new Web service description to ensure that the Amazon sample Goals will discover the Web services he is annotating, such that all Amazon customers will automatically discover these services and allow his company to effectively compete with Amazon.

It is also important that the developer has visibility of the matching between Goals and Web Services as the descriptions of both evolve. Therefore a unit testing tool for discovery called DUnit, as can be seen in figure 3 was implemented and added to the WSMT. The DUnit View compliments the Ontology Unit Testing (OUnit) and Mapping Unit Testing (MUnit) Views, providing the developer with the ability to specify a goal, a set of Web Services and the degree to which each should match. The DUnit View invokes the underlying discovery framework for each of the tests at the click of a button and gives feedback to the developer regarding the successful execution (or not) of the tests selected.

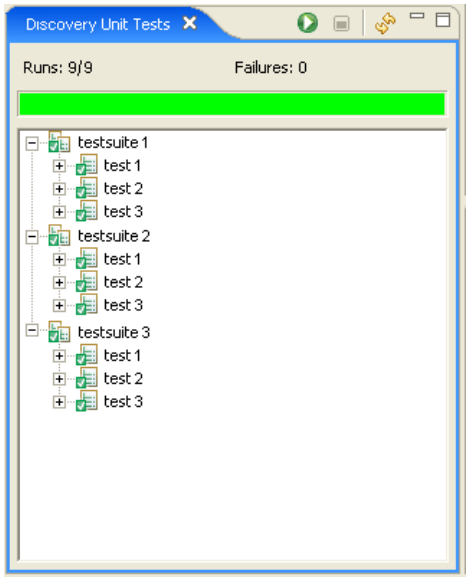

Fig. 3. The Discovery Unit Testing (DUnit) View

With these new additions to the WSMT the developer is now not only supported in the process of creating and maintaining Web service and goal descriptions, whether by textual, form or graphical means, but also receives detailed validation information regarding the conformance of his descriptions to the WSML variant selected and to the chosen discovery engine. The developer can also at the click of a button test the result of discovery with a single goal or can execute a batch load of discovery unit tests via the DUnit View.

5 Related Work

In this paper we align, in a configurable manner, several discovery approaches that can be used with the WSMO conceptual model. Beside WSMO, there are other approaches to semantically describe Web Services, for example OWL-S and SAWSDL. OWL-S[4] is an OWL based Web service ontology that can be used to semantically describe Web services. Similarly to WSMO it does support different discovery approaches, for example keyword-based discovery, discovery

based on simple concept categorization, or discovery based on matching pre- and postconditions. To our knowledge at this time, there have been no efforts in OWL-S to align different approaches to discovery. SAWSDL[5] is another approach to semantic annotation of Web services and the only official W3C technology recommendation for Semantic Web services at this point in time. Nevertheless SAWSDL does not support the definition of service pre- and post-conditions, which severely limits the annotation possibilities for the functional behavior of Web services. As a direct consequence of this it follows that SAWSDL can only be used with the simplest of discovery techniques, e.g. using keywords or on concept categorizations. As, e.g., [9] and [11] have shown, preconditions and, especially, postconditions are essential for a more precise service discovery. Recent works in [16] tries to combine the two approaches of OWL-S and SAWSDL. The early nature of SAWSDL means that only one approach to discovery exists and thus a need to align different approaches is unnecessary at this point. It is envisioned that as SAWSDL becomes more popular such a need will arise.

From the perspective of tool support there is only one other tool for creating Semantic Web Services through the WSMO paradigm, namely WSMO Studio[3]. WSMO Studio provides a broader feature set than the WSMT in that it covers not only WSMO, but also functionality for modeling semantic business processes and SAWSDL descriptions; however the WSMT provides more functionality for WSMO and WSML covering functionality in all phases of the life cycle of Semantic Web Services. Recently an extension to WSMO Studio was provide with a Quality of Service based discovery engine allowing Goals and Web Services to be matched based on Quality of Service metrics.

6 Evaluation

In this paper we have introduced our Semantic Web Service Discovery framework that integrates and manages different discovery engines, as well as its integration into the Web Service Modeling Toolkit. Due to the lack of competing tools in this area we have been unable to do any direct comparison of our tool; However we are aware that the integration of the current existing discovery engine implementations into the framework does not affect the performance of these engines. This is because the discovery step itself is performed in exactly the same way as if the engine was called directly by the developer. What the framework adds is a simple lookup of the non-functional properties of the Web services and goals and then the execution of a validation process to validate the given services and ontologies and to eventually find out which discovery engine is the right one for a given Goal. So far no reasoning is done by the framework itself and it is important in the future to ensure, if reasoning is added to the framework, that the performance of the reasoners while in the framework does not deteriorate.

The main evaluation we actually can do is in terms of usability of the different discovery approaches. Looking at the use case of a developer that has written a goal and would like to discover fitting services. Without the discovery framework that this paper introduces he first must manually integrate each

of the discovery engines that supports his goal into his application. With the discovery framework all he needs to do is 'set a flag' on his goal, that is add a simple non-functional property to his goal. The case is even worse if the developer does not know which engines would actually support his goal. In this case he would first need to compare the WSML variant and expressivity used in the logical expressions of his goal's capability with the WSML variant and expressivity supported by the single available engines. Our framework allows him to simply omit the choice of specific targeted engines, allowing the framework to validate his goal description and automatically add the right discovery engines to the discovery process. Another aspect is that the developer is, without our discovery framework and its configuration abilities, not able to target a specific type of match; and would need to implement a software component that would analyze the output of each of the discovery engines integrated into his application. This use cases shows the added-value of the discovery framework that we introduce and demonstrates the actual simplification of the whole Web Service Discovery Process that has occurred through the introduction of this framework.

In terms of the WSMT the developer will see a dramatic improvement in the confidence he can have in the Web service and goal descriptions that he creates. The ability to quickly and easily test the descriptions he creates in the integrated environment he is using as he creates them enables him to be sure that what he is creating is correct and the ability to create unit tests that can be executed as ontologies, service and goal descriptions evolve ensure that he is sure that what he has previously created is correct.

7 Conclusions and Future Work

In this paper we introduce a first attempt to aligning different WSMO discovery approaches in a configurable framework and to enhance the developer tools available in the Web Service Modeling Toolkit with additional functionality, made possible by this framework. In terms of future work, [9] mentions user intentions, related to the "lightweight" set-based discovery approach. Service descriptions, described as sets of objects, can be interpreted in different ways and are thus not semantically unique: A developer might want to express that either *all* elements contained in a goal or service description are requested or delivered, or that only *some* of these elements are requested or delivered. We want to take this user intentions into account in future versions of the discovery framework, enabling the user to specify more fine-grained match type wishes and at the same time getting more precise results.

In terms of the WSMT, future directions will include the creation of a new WSMO Web service and goal editor that includes discovery approach based User Interface plugins. With such a framework it will be possible to provide the developer with a simple form based editor into which the developer has fill only very simple information. From these forms complex logical expressions can be created within the underlying Web service or goal that conforms to the restrictions set in place by the underlying discovery approach.

8 Acknowledgements

The work is funded by the European Commission under the projects Knowledge Web, Musing, Salero, SEEMP, SemanticGov, Super, SHAPE, SWING and TripCom; by the FFG (Österreichische ForschungsFörderungsGeselleschaft mbH) under the projects Grisino, RW^2, SemNetMan, SeNSE, TSC, OnTourism.

References

1. F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider. *The Description Logic Handbook*. Cambridge University Press, 2003.
2. L. Cabral, J. Domingue, S. Galizia, A. Gugliotta, B. Norton, V. Tanasescu, and C. Pedrinaci. IRS-III: A Broker for Semantic Web Services Based Applications. In *Proc. of the 5th Intl. Semantic Web Conf. (ISWC2006)*, Athens, USA, 2006.
3. M. Dimitrov, A. Simov, M. Konstantinov, and V. Momtchev. WSMO Studio - a Semantic Web Services Modelling Environment for WSMO. In *Proc. of the 4th European Semantic Web Conf. (ESWC2007)*, Innsbruck, Austria, June 2007.
4. D. Martin (ed.). OWL-S: Semantic Markup for Web Services. W3C Member Submission 22 November 2004, 2004.
5. J. Farrell and H. Lausen. Semantic annotations for wsdl and xml schema. W3c member recommendation 28 august 2007, 2007. online: http://www.w3.org/TR/sawsdl/.
6. D. Fensel, H. Lausen, A. Polleres, J. de Bruijn, M. Stollberg, D. Roman, and J. Domingue. *Enabling Semantic Web Services – The Web Service Modeling Ontology*. Springer, 2006.
7. A. Haller, E. Cimpian, A. Mocan, E. Oren, and C. Bussler. WSMX - A Semantic Service-Oriented Architecture. In *Proc. of the Intl. Conf. on Web Services (ICWS2005)*, Orlando, USA, July 2005.
8. D. Hull, E. Zolin, A. Bovykin, I. Horrocks, U. Sattler, and R. Stevens. Deciding semantic matching of stateless services. In *Proc. of the 21st National Conf. on Artificial Intelligence (AAAI-06)*, Boston, USA, 2006.
9. U. Keller, R. Lara, H. Lausen, A. Polleres, and D. Fensel. Automatic location of services. In *Proc. of 2nd European Semantic Web Conf. (ESWC)*, 2005.
10. U. Keller, R. Lara, A. Polleres, I. Toma, M. Kifer, and D. Fensel. D5.1v0.1: WSMO Web Service Discovery". Technical report, DERI Innsbruck, 2005.
11. U. Keller, H. Lausen, and M. Stollberg. On the semantics of functional descriptions of web services. In *Proc. of 3rd European Semantic Web Conf. (ESWC)*, 2006.
12. M. Kerrigan, A. Mocan, M. Tanler, and W. Bliem. Creating Semantic Web Services with the Web Service Modeling Toolkit. In *Proc. of the workshop on Making Semantics Work For Business (MSWFB2007) at ESTC2007*, Vienna, May 2007.
13. M. Kerrigan, A. Mocan, M. Tanler, and D. Fensel. The Web Service Modeling Toolkit - An Integrated Development Environment for Semantic Web Services. In *Proc. of the 4th European Semantic Web Conf. (ESWC2007)*, Innsbruck, 2007.
14. H. Lausen, J. de Bruijn, A. Polleres, and D. Fensel. WSML - A Language Framework for Semantic Web Services. In *Proc. of the W3C Workshop on Rule Languages for Interoperability*, April 2005.
15. J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 2nd edition edition, 1987.
16. D. Martin, M. Paolucci, and M. Wagner. Towards semantic annotations of web services: Owl-s from the sawsdl perspective. In *Proc. of European Semantic Web Conf. (ESWC) workshop OWL-S: Experiences and Directions*, 2007.