

ACE-GIS

Adaptable and Composable E-commerce and Geographic Information Services

IST-2002-37724



Semantics-bases Service Discovery Assistant User Guide

Date:	2004-08-27
Author(s):	Florian Probst, Kean Huat Soon, Patrick Maué
Distribution:	
WP:	ALL
Version:	1.1
Keywords:	Semantic-based service doiscovery
Description:	User Guide



Composability and dynamic adaptability in software, systems and services

**Contents**

SeDA - Semantics-based Service Discovery Assistant	2
Functionality and Benefits	2
How to use SeDA	3
Step 1 - Select Domain of Interest	3
Step 2 - Specify Focus of Discovery Process	4
Step 3 - Browse merged Application and Domain Ontologies	5
Step 4 - Retrieve Information	6
Architecture	7
WSDL Extension	10
Parsing UDDI, WSDL and OWL	11
ACE-GIS context	12
Assumptions made	12
Building the Composite Service	12
1.1 Semantics-based Service Discovery Assistant	2

Component Tool	Prerequisites	Downloads / Installations
Semantics-based Service Discovery Assistant (SeDA)	SeDA requires JAVA 1.4.2 Installing SeDA will also install Apache SOAP 2.2	SeDA is an Open Source product. Download SeDA from: http://musil.uni-muenster.de/SeDA Download the SeDA installation guide from: http://musil.uni-muenster.de/SeDA

SeDA - Semantics-based Service Discovery Assistant

Composition of web services based on currently available descriptions such as WSDL is error-prone because the meaning (or semantics) of the labels used in these syntactic descriptions is unclear. In the previous phase of the project, three problem types have been identified as resulting from semantically heterogeneous service descriptions.

SeDA facilitates the discovery of semantic heterogeneities between web service descriptions and thus helps to avoid the problem of composing web services in a way that may lead to unintended results. In this chapter the use of SeDA (see ACE-GIS tutorial), its functionality and architecture is described. The ontologies described in chapter 3 were utilized to develop and test the prototype.

Functionality and Benefits

Organized in the form of a Wizard, the Semantics-Based Service Discovery Assistant helps users to align



their own understanding of a term with that of service developers.

Guiding the user through a fixed number of steps, the discovery tool offers the following benefits:

1. It creates a merged ontology, containing terms that are:
 - Used in the domain that the user has selected;
 - Used in application ontologies describing either service output or service input
2. It provides the user with the ability to:
 - Browse through the newly created ontology, which is merged from ontologies according to the user's selection, and
 - Select the concept of interest.
3. It returns information about of the service that references its WSDL-terms to the concept identified by the user.

How to use SeDA

The tool provides a wizard user interface, which guides the user through four steps. Within these four steps the user's requirements (input or output) are semantically aligned with the capabilities of the available services.

Step 1 – Select Domain of Interest

In step one the user is presented with a list of domain ontologies. Each of the domain ontologies comes with a short description to give you some orientation (figure 3). Selecting a domain can be understood as selecting the area of interest on which the discovery process will be focused. By selecting a domain, you select the context-domain in which your search will take place. Each different domain will have its own '*understanding*', depending on the context for which it has been defined. E.g. if you are looking for information about *wind speed* at a certain location, selecting the *meteorology* domain is appropriate. If you are looking for a service providing a gas plume calculation, selecting either the domain of *chemistry* or the domain of *geographic entities* may be appropriate. However, you always search in a certain context in which the service will be used, and thus will be able to distinguish whether the domain of *chemistry* or *geographic entities* is appropriate. In both domains, services about gas plumes may be provided, but with very different modelling approaches. There are two reasons for selection of a domain.

1. A domain is characterised by the use of specialised vocabulary. For example, some terms may be used exclusively by the members of that domain; some terms may have a special meaning in that domain compared to others.
2. Selecting a domain, reduced number of potentially suitable services on offer.

The number of ontologies available depends on how many domain ontologies are registered with the UDDI.

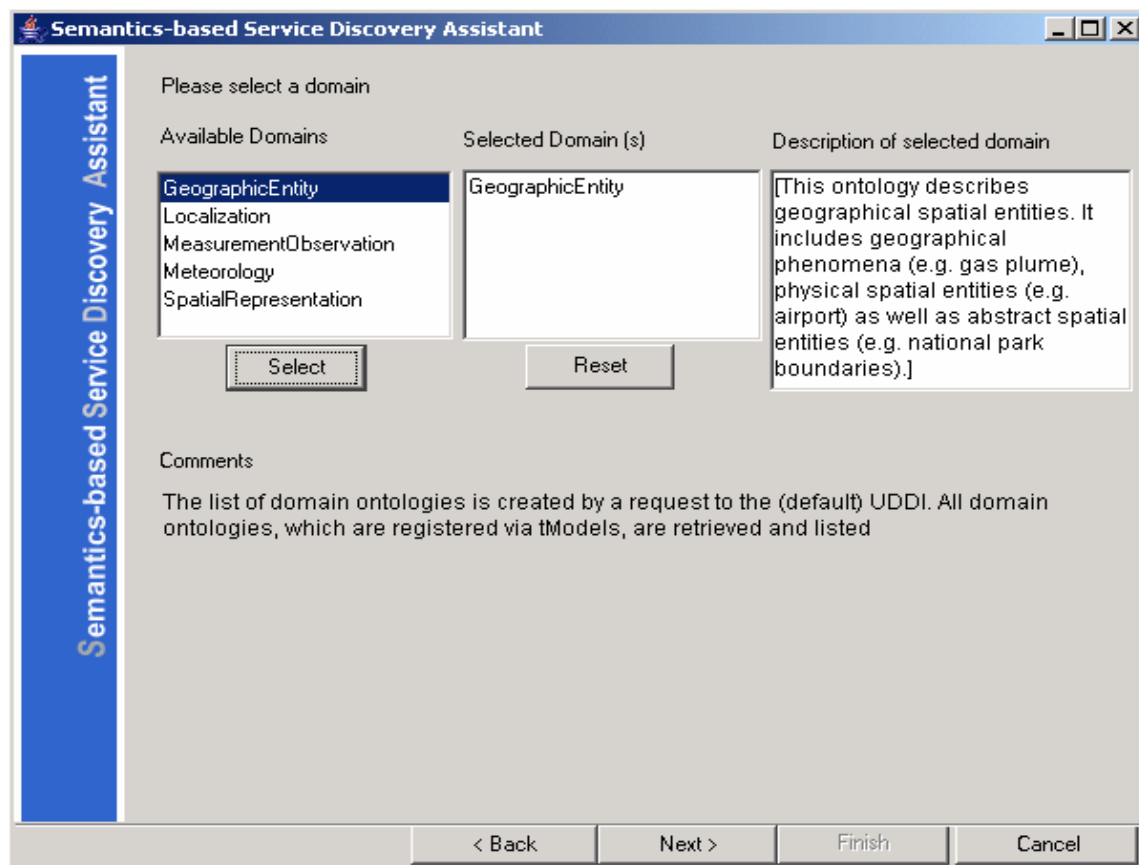


Figure 3: The user selects the domain, which is identified by the user as setting the context for the service discovery.

Step 2 – Specify Focus of Discovery Process

Each atomic service has an input data type and an output data type for the operation it offers. The meaning of the data type a service requires as input or the data type it provides as output are described in its applications ontology. Depending on the approach chosen to build a composite service, there are two ways to identify a service (figure 4). The user can either specify that the discovery process should focus on the data type which the service accepts as input or on the data type which is returned as result (output).

Similar to step one, the specification whether input concepts or output concepts are in the focus of the discovery process reduces the amount inappropriate search results. This decision is needed to enable selection of the appropriate application ontologies, and later to mark the terms used to describe the input (output) appropriately in order to increase the readability of the provided information.

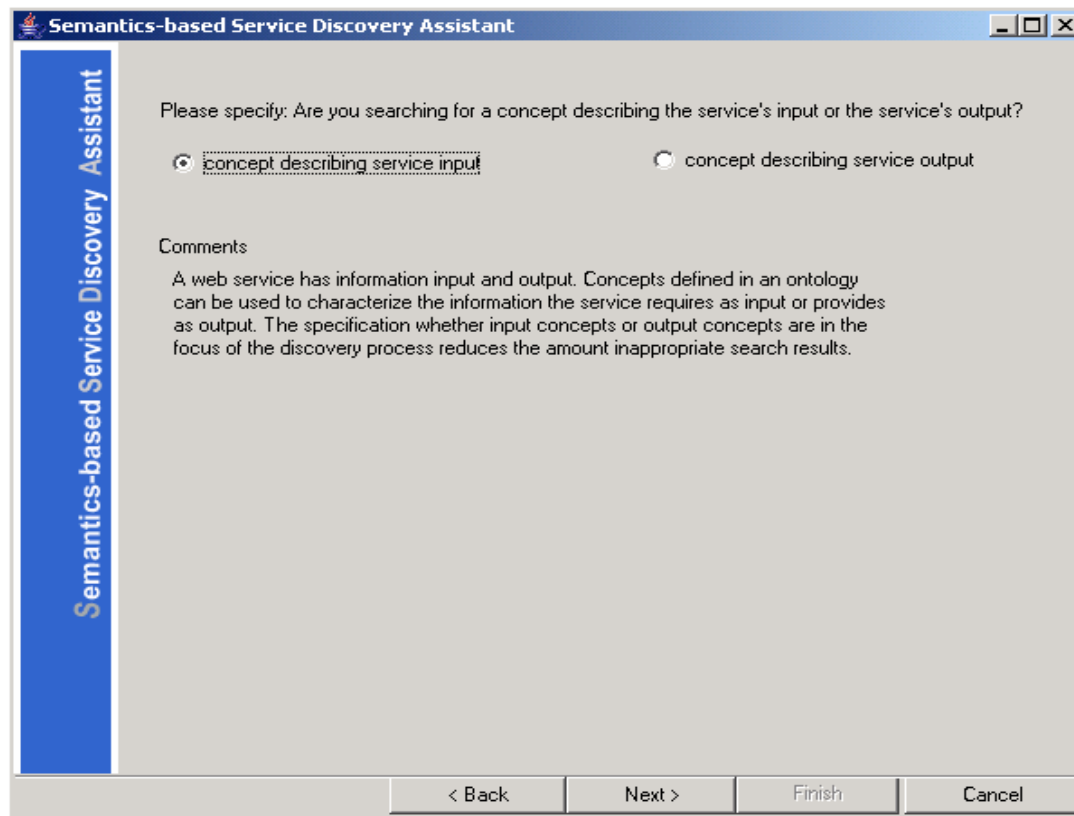


Figure 4: The user specifies whether the focus of the discovery process is on the information requested by the service (input) or provided by the service (output).

Step 3 – Browse merged Application and Domain Ontologies

The Semantics-based Service Discovery Assistant searches in the UDDI for all Operation tModels which reference either their input concept or output concept to the domain ontology the user specified in step 1. Since the Operation tModel is linked to the WSDL file describing the service to which the operations belongs to, the application ontology which provides the semantics for that service (and its operation(s)) is identified.

All application ontologies that are found during this process are imported and merged with the user selected domain ontologies. This results in a *concept hierarchy*, (figure 5) containing only the relevant concepts from the domain ontology needed to define the application ontology concepts describing either operation input or output. Such reduction of complexity appears to be crucial if one considers a large number of ontology to exist.

The user's task in step 3 is to browse the newly created ontology and identify concepts of interest. By clicking on any of the concepts (highlighting), the right hand windowpane displays that concept's definition based on DL (description logic) statements. This enables the user to learn about the meaning of a particular term in a particular service. The user learns about the differences between the services, discover semantic heterogeneities and thus avoid errors.

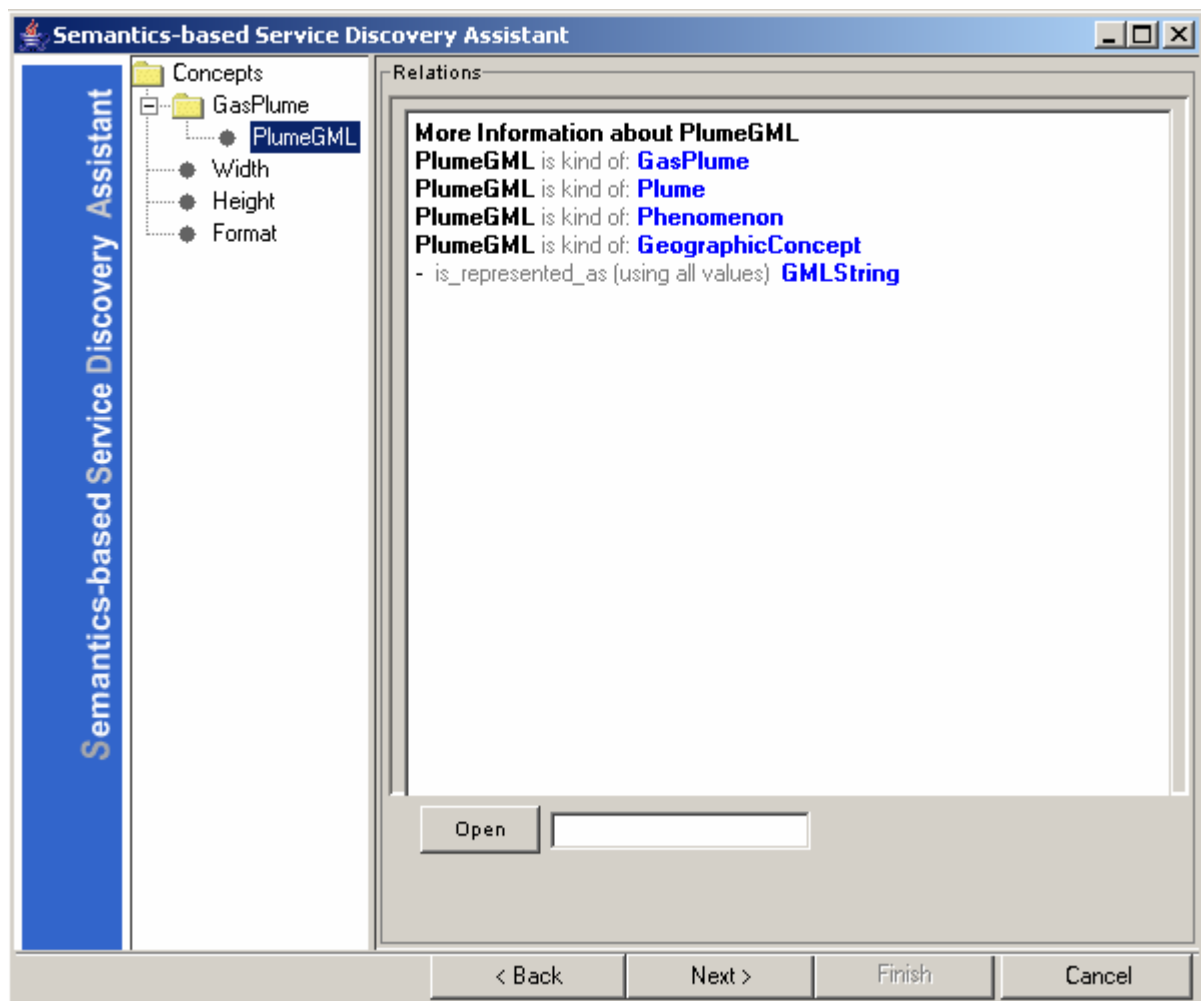


Figure 5: The domain and application ontologies selected in the previous steps are merged.

Step 4 - Retrieve Information

In step 3 the user selects a concept that describes the input (or output) information he requests. The Semantics-based Discovery Assistant uses the WSDL file of a service registered in the UDDI to import the application ontology of that service. Therefore it is possible to trace back which service makes use of the user-selected concept. Step 4 retrieves from the UDDI all available information about that service.

The user is provided with the information needed to invoke the service (figure 6).

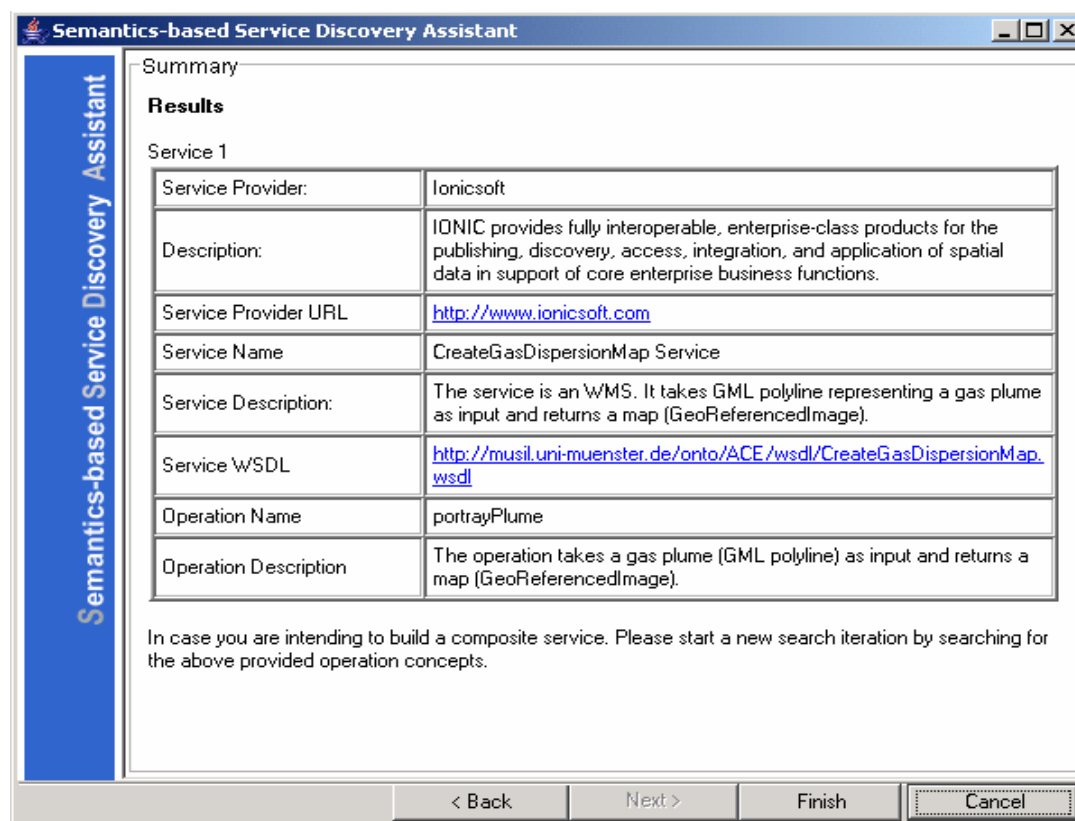


Figure 6: From the UDDI, the information for service invocation is retrieved and presented.

Step 4 concludes the search for a service. In the case of building a composite, the search is repeated. If the backward chaining approach is used, the input concept from the discovered service is now used as output concept for the next service to be discovered. If the forward chaining approach is used, the output concept of the discovered service is used as the required input concept of the next service to be discovered.

Architecture

The architecture of SeDA focuses on incorporating existing standards. For registering services, we use and extend the Universal Description, Discovery and Integration (UDDI) protocol. Services are described using the Web Service Description Language (WSDL) (provided by partners). Domain and application ontologies are built using the Web Ontology Language (OWL). Figure 7 shows how the different components of SeDA collaborate.

Enhanced use of UDDI Registry

The Universal Description, Discovery and Integration (UDDI) protocol is currently one of the major building blocks required for web service registration and discovery. UDDI provides a standard interoperable platform that enables companies and applications to dynamically find and use web services over the Internet (www.uddi.org). We employ a UDDI registry as basis for semantically enhanced service discovery. However, several changes to the standard procedure of registering a service were made. Instead of using tModels to describe web services as atomic units, we introduced *Operation tModels* in order to perform searches for service operations directly. This results in registering several Operation tModels if the service offers several operations. Additionally we introduced *Domain Ontology tModels* as a means of registering domain ontologies in a UDDI registry. In the following the functionality of a tModel in general as well as the specific functionalities of the newly introduced Operation - and Domain Ontology tModels is described

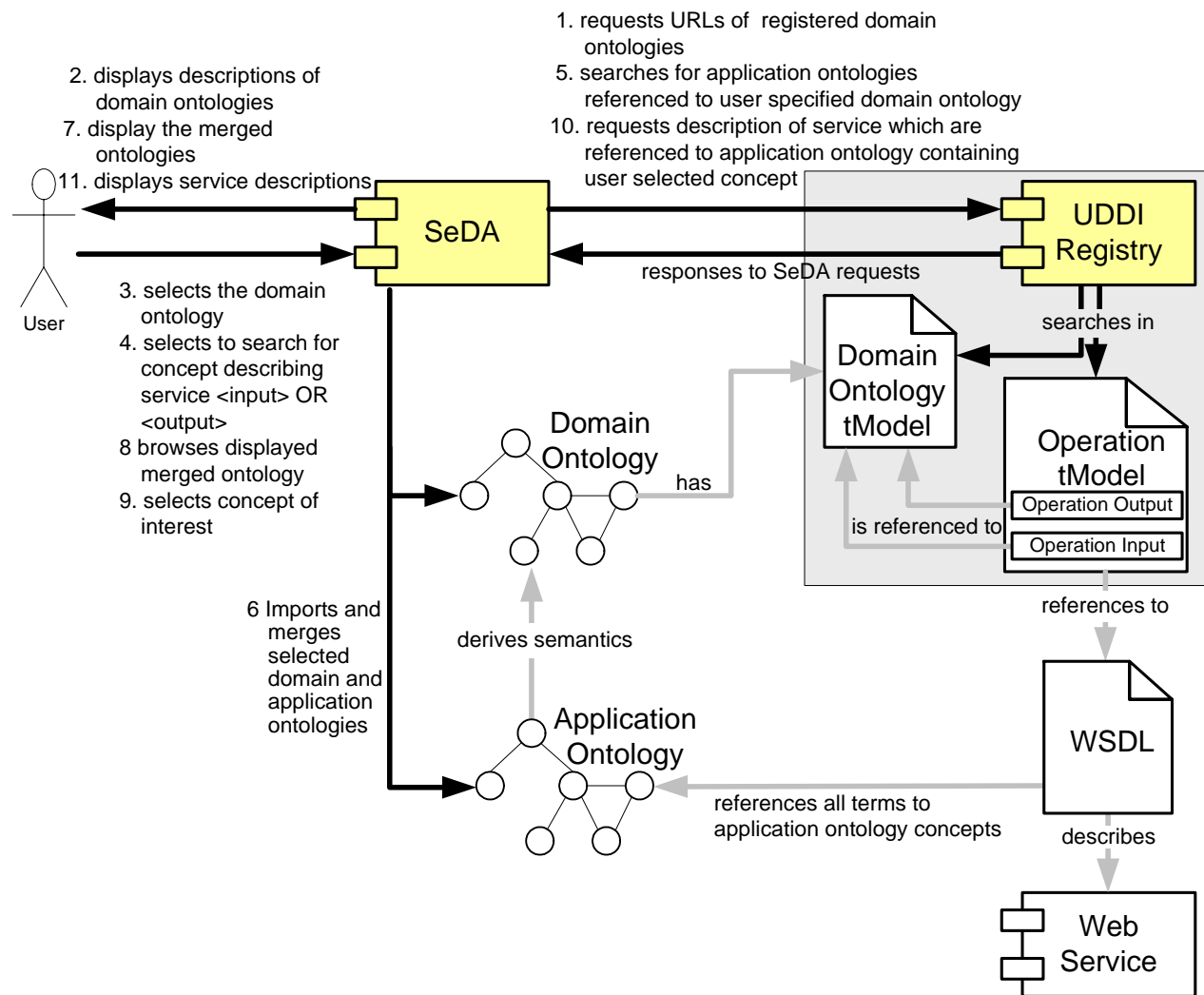


Figure 7: Collaboration of the SeDA components.

tModels provide the ability to describe compliance with a specification, a concept, or a shared design (www.uddi.org). TModels have various uses in the UDDI registry. We are interested here in the use of tModels to represent technical specification of the operations of the web services and domain ontologies. Each tModel should consist of a *name*, a *description*, one or several *identification scheme*, which contain a *keyname* and a *keyvalue*, and an *overviewURL* that identifies the location of a document describing the service e.g. a WSDL file.

Domain Ontology tModels are introduced to allow the registration of domain ontologies at the UDDI registry in analogy to registering a web service.

Each Domain Ontology tModel contains *domain ontology name*, *domain ontology description*, and *domain ontology URL* (as *overviewURL*) and a unique *tModelKey* used for identification. The *domain ontology name* and *description* provide the name of the domain ontology (e.g. meteorology) and a natural language description of the content to facilitate the user to select the appropriate ontology

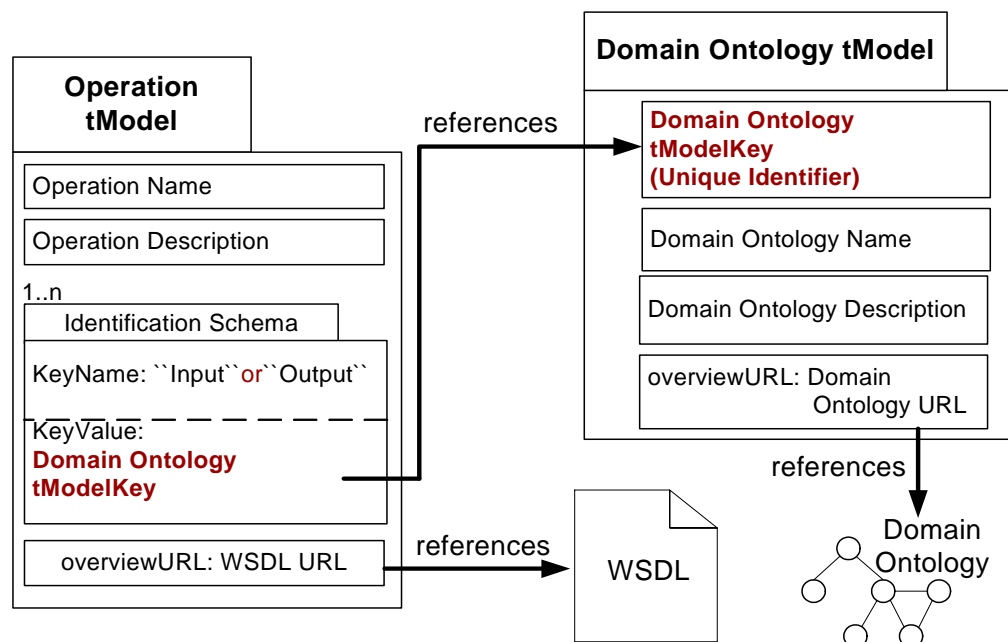


Figure 8: Connection between tModels, WSDL and Domain Ontology

Operation tModels, in contrast to common tModels, employ one or several identification schema in order to install a link between the terms describing that operation in the WSDL file and domain ontologies from which these terms derive their semantics. Since each operation has an input message and an output message, at least two terms need to be semantically annotated. If the two messages are described by terms that are linked to different domains, two identification schemas are installed in the Operation tModel. One of the two identification schemas has the keyname *input*; the other has the keyname *output*. Their keyvalues are the unique tModelKeys of the Domain Ontology tModels. The Domain Ontology tModels in turn allow identifying the location of the domain ontologies (see figure 8).

The reason why we need to introduce Operation tModels is that a service can provide several operations with two messages each, which could contain several parts. These parts finally need to be semantically annotated. Operation tModels provide the flexibility to account for such detailed reference to different domain ontologies. These references provide the possibility to quickly identify which input messages (or output messages respectively) refer to the domain ontology the user has identified as focus of his search. This increases the usability of the ontology browser since only terms describing input messages (or output messages respectively) are presented.

Figure 9 depicts the relationship between domain ontologies, operations, WSDL files, and application ontologies. It illustrates five of the six domain ontologies developed for the prototype, which have been registered as domain ontology tModel at our UDDI registry. The sixth domain ontology has no direct link to the terms of the registered operations; hence it is not showing up in the drawing.

The user starts a service discovery iteration by selecting one or more domain ontologies. Then the Operation tModels referenced to the selected domain ontologies are identified via the Domain Ontology tModels. The Operation tModels in turn identify the WSDL of the services. The WSDL files, providing syntactic descriptions of the services are semantically annotated via application ontologies. The WSDL extension described in 4.3.1 identifies the application ontologies.

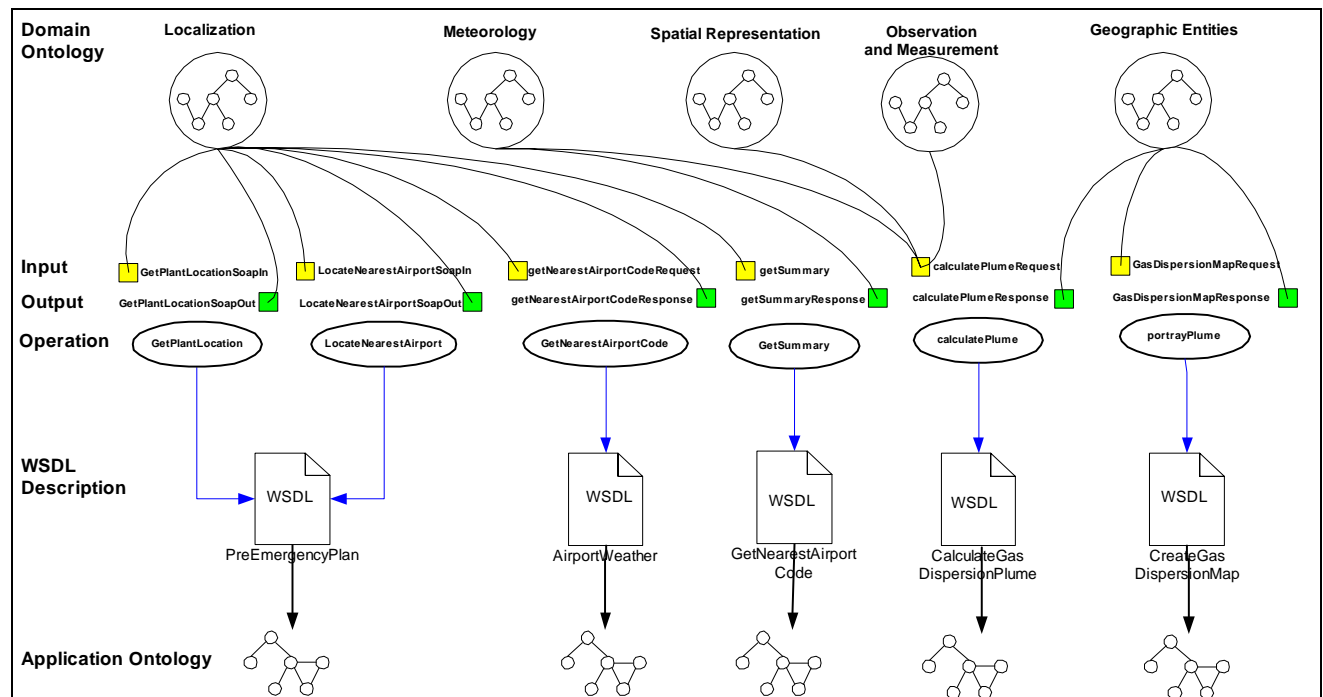


Figure 9: The relationship between domain ontologies, operations, WSDL files, and application ontologies employed in the prototype.

WSDL Extension

Our goal of adding semantics to WSDL descriptions is to annotate those elements describing the information content provided by the service. We identified the need of referencing the tags `<ComplexType>`, `<element>` (within a `ComplexType`) and `<part>` (within a message) to the application ontology, in order to capture the semantics of the tag's name. A new attribute was introduced to reference these tag the application ontology. This attribute is called `<semRef>` and is defined in the following short XSD file (listing 1):

Listing 1: XSD file for defining the attribute needed to reference WSDL terms to application ontologies.

```
<xs:schema targetNamespace="http://musil.uni-muenster.de/onto/ACE/xsd"
xmlns="http://musil.uni-muenster.de/onto" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:annotation>
    <xs:documentation xml:lang="en">Schema for the semantic reference of WSDL Files. The
      attribute semRef allows to connect elements and complexTypes from a WSDL file
      with an application ontology (OWL).
    </xs:documentation>
  </xs:annotation>
  <xs:attribute name="semRef" type="xs:string" use="optional"/>
</xs:schema>
```

This attribute allows inserting the application ontology's URL in which the annotated element is explicitly described. Listing 2 gives an example of how the parts of the message `calculatePlumeRequest` of the `CalculateGasPlume` Service provided by `ionic` is referenced to the application ontology.

Listing 2: Example for WSDL parts referenced to an application ontology.

```
wsdl:message name="calculatePlumeRequest">
  <wsdl:part name="origin" type="tns1:Point" SeDA:semRef="http://musil.uni-
```



```

                                muenster.de/onto/ACE/A_CalPl.owl#Origin"/>
<wsdl:part name="windSpeed" type="xsd:float" SeDA:semRef="http://musil.uni-
                                muenster.de/onto/ACE/A_CalPl.owl#WindSpeed"/>
<wsdl:part name="windDirection" type="xsd:float" SeDA:semRef="http://musil.uni-
                                muenster.de/onto/ACE/A_CalPl.owl#WindDirection"/>
<wsdl:part name="emissionRate" type="xsd:float" SeDA:semRef="http://musil.uni-
                                muenster.de/onto/ACE/A_CalPl.owl#EmissionRate"/>
</wsdl:message>
```

Parsing UDDI, WSDL and OWL

We decided to use the Xerces (2) SAX parser. The increased performance compared to DOM parsers, justifies the more complex handling of the parser. First, the parser reads messages, message parts and types of the WSDL files under consideration. In the second step, the tool combines the parsed information and provides a list of items. Each item provides the following information, combined of the WSDL and UDDI parsing steps:

The WSDL parsing returns

- the location of the application ontology file (e.g. "http://musil.uni-muenster.de/onto/ACE/A_CalPl.owl")
- the *concept* within the application ontology, to which the parsed WSDL element references ("e.g. GasPlume")
- the name of the message part ("calculatePlumeResturn")
- the name of the message with this message part ("calculatePlumeResponse")
- the name of the operation containing this message ("calculatePlume")

After the previous three steps, information can be provided about

- the type of message to which the parsed WSDL element belongs to. The parsed element can either belong to an input message or an output message.

information returned by UDDI query

- the natural language description of the operation
- the name of the selected domain ontology. ("Meteorology")
- the location of the domain ontology ("http://musil.uni-muenster.de/onto/ACE/D_Met.owl")
- natural language description of the service and the service provider

With this data, the tool gathered sufficient information for the import of the application ontologies and a user-friendly presentation of the concepts.

The first step of the OWL parsing is to construct an Ontology model. To do this, the tool imports all application ontologies, whose URLs are given in the list of items returned by the WSDL parser. The imports are directly added to the model. The resulting ontology model includes therefore the concepts of the domain ontologies too, since those are imported internally by the application ontologies. The wizard does not present to the user all concepts of the model, but only the concepts resulted in of the WSDL parsing step. Each concept is added to the tree, as well as its super- and subconcepts.



The Jena2 Java Framework (3) is used for the parsing and handling of OWL files. Jena2 is originally a library, to simplify the work with files written in RDF (4), on which OWL builds. Since Jena2 provides an RDFS reasoner, it has capabilities to work directly with the ontology model. The Jena 2 Inference subsystem currently comprises inference engines structured as graph and includes a number of predefined reasoners (5). One of the important tasks of the reasoner is to check RDFS closure rules by translating each domain range, sub property and sub concept declarations into a single query rewrite rule. The inference engine also supports rule based RDF (using both forward and backward chaining methods). However since the Rule based approach is computationally expensive in large or complex ontologies Jena also provides a description logic reasoner interface. However, this reasoner is still under development and therefore not yet applicable.

In the SeDA tool, the right hand windowpane in step three (figure 5) presents the user information about relations of a concept which the user selected in the tree. The relations include taxonomic relations between the concepts, as well as non-taxonomic relations. These are mainly restrictions of the class properties. The information of the right field is drawn from the inference model, constructed by the RDFS reasoner. It allows displaying all statements where a class (concept) is used as subject of an RDF triple. The inference model improves the expressiveness of the ontology model and simplifies the analysis of the result.

ACE-GIS context

The Semantics-based Service Discovery Assistant offers improved UDDI (Universal Description, Discovery and Integration) based discovery of web services, and can therefore be of benefit to any current UDDI user.

More specifically within ACE-GIS, intended users of this tool are service developers who wish to build a composite service or service chain. These users do not need any deeper understanding of ontology engineering. However, current browsing of the ontology is done on a GUI that, at this time, still contains DL (description logic) statements and users would currently need to be able to read DL statements.

Assumptions made

To be practically applicable, services registered in a UDDI have to be described by application ontologies, which in turn draw their semantics from domain ontologies. We assume these ontologies to exist. For proof of concept, application and domain ontologies were developed within the pilot ACE-GIS Emergency Management System (EMS).

Building the Composite Service

When building a composite service, one practical approach is to start service discovery with the service that will deliver you the final result. In other words, discovering the last service in the chain first.

With this approach to building a composite service, the developer's search is focused on the atomic service's output, the discovery iteration process starting with the service that will be invoked last within the composite service.

Indeed, this may well be the only practical approach for the simple reason that the developer only knows what the final output of the composite service has to be and can therefore start the discovery iteration process by finding a suitable service that supplies the end-user's requirements as its output. Once this

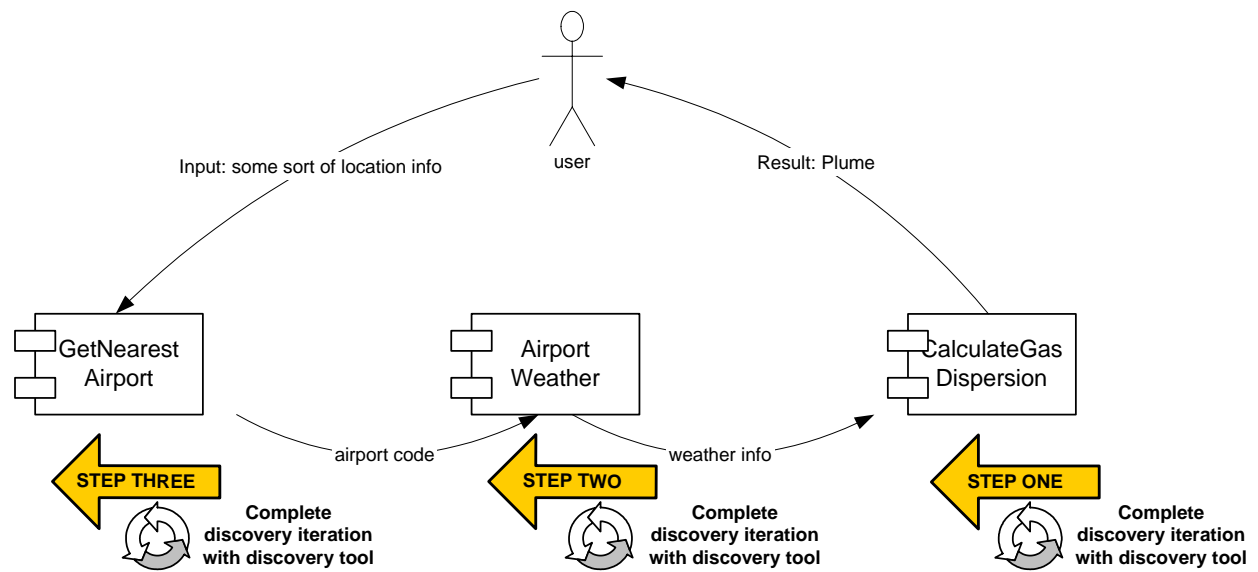


service has been found then its required inputs can be determined, and the search resumed based on this information, so working backwards. This backward chaining is continued until a service is found which requires an input the end-user is able to provide. E.g. a service requiring a street address as input may be acceptable for an end-user, whereas a service requiring the input of coordinates may be not acceptable.

Building a composite service by using a forward chaining approach is theoretically possible. The Semantic-based Service Discovery Assistant supports that approach. This approach has the advantage that the information required as first input, which in many cases will be a user input, can be chosen appropriately. This initial input, needed to invoke the composite service, can easily be provided by the user. The disadvantage of this approach is that both “ends” of the service chain are fixed. Closing the gap between potential start services (accepting input the user is able to provide) and potential end services (providing the information, the user requires) turns out to be difficult task, because at least one service has to accept exactly the provided input, and has to return exactly the required output.

Our Gas Dispersion example highlights the benefit of the backwards-chaining approach. We knew that the end-result of our composed service would provide a Fire Officer with a map displaying a gas plume resulting from an emergency at a chemical plant.

Although we knew that this plume would be calculated and mapped from various pieces of information, such as type and property of the leaking gas, rate of leakage, wind direction and wind speed, we could not predict at the outset which, if any, of the potential services that might eventually make up the final composed service would have compatible concepts of these pieces of information. By starting at the end (gas plume mapping), we could determine its exact input requirements and then select a penultimate service (gas plume calculation) whose outputs were compatible with the final service’s input requirements. The following diagram helps to illustrates the steps involved in this process:



- 3
- Search and find all services which deliver airport codes (identify those services with same or similar airport code concept as the AirportWeather service).
 - Learn about the input requested by these services.
 - Decide for one of the input concept. In our example „plantID“.
 - When building a composite service from the back end, it might be a good start to specify the first input in a rather broad way. E.G. „some sort of location information“ and then see what „bends and turns“ happen during service discovery. This method of service chaining will remain semi-automatic, since a human agent needs to judge whether the number of intermediate services is appropriate in relation to the overall goal of the composite service.

- 2
- Search and find all services which deliver weather information (identify those services with same or similar weather concept as the plume service).
 - Learn about the input requested by these services.
 - Decide for one of the input concept. In our example: „airport code“.

- 1
- Search and find all services which deliver plume information
 - Learn about the input requested by these services.
 - Decide for one of the input concept. In our example „weather information“.

Figure 1 Iterative backwards process for creating part of the composite Gas Dispersion service



Semantic Interoperability Tool

- In ACE-GIS, the semantic interoperability tool is intended to help developers with the task of building a composite service, by supporting the discovery of semantically appropriate Web services.

To facilitate the search for appropriate Web services, the WSDL service descriptions need to be tagged with concepts that are explained in *ontologies*. These concepts account for the meaning of the service's input and output data types. This tagging enables semantic (as opposed to purely syntactic) searches. It enables an appropriate chaining of the services, so that the semantics of the output from one service matches the semantics of the input required by another service.