

A Process and a Tool for Creating Service Descriptions based on DAML-S*

Michael Klein and Birgitta König-Ries

Institute for Program Structures and Data Organization
Universität Karlsruhe
D-76128 Karlsruhe, Germany
{kleinm,koenig}@ipd.uni-karlsruhe.de
<http://www.ipd.uni-karlsruhe.de/DIANE/en>

Abstract. In distributed environments, collaboration is often achieved with the help of services. To enable automatic service trading, semantically expressive, automatically comparable, flexible, and editable service descriptions are needed. Our analysis shows that only ontology-based service descriptions like DAML-S offer the necessary expressiveness and flexibility. Unfortunately, up to now, DAML-S offers just a generic framework and lacks support for creating and editing appropriate service descriptions for specific classes of services. Therefore, in this paper, we present an approach to improving the usability of DAML-S. The two main building blocks of our approach are a process and a tool. The process guides through the steps necessary to create adequate service descriptions by introducing a layered ontology of services. The graphical tool, DINST, implements these ideas, thus offering a comfortable way to edit service descriptions.

1 Introduction

In the last few years, services have become the major basis for collaboration in distributed environments. In such environments, on the one hand, the members provide their resources as services and on the other hand, use the functionalities offered by other members to complement their applications. In many service oriented systems, the components are loosely coupled. Examples are the internet-based web services or services in networks with higher dynamics like peer-to-peer or ad hoc networks. In these cases, like on a marketplace, service offers have to be explicitly described and published, whereas service requests need to be paraphrased and compared with the offer descriptions. For this reason, what is needed, is a dedicated service description language.

Obviously, service descriptions can be done in various levels of detail. They range from simple keyword-based to highly complex ontology-based approaches and many different languages have been proposed in the last years. However, for

* This work is partially funded by the Deutsche Forschungsgemeinschaft (DFG) within SPP 1140 [1].

a good service description language, several requirements have to be fulfilled, which are partially competitive:

Semantically expressive. In general, the service description has to be expressive enough so that other applications can understand the functionality of the service without human help. Therefore, it is crucial to include the functional semantics of the service into the description. It describes, how the service transforms inputs into outputs and which side effects it produces. Moreover, with an increasingly complex service description it becomes less and less likely that a service request and a service description match exactly. Therefore, the language has to be able to describe services on various levels of abstraction, which allows to find similar services. To enhance the expressiveness when searching for services, it should be possible to add complex conditions to a request description.

Automatically comparable. Service description should enable an automatic comparison, which can be done without the help of an additional human analysis. Thus, service descriptions need to have a certain structure.

Flexible. Contradictory to a structured language, flexibility is another requirement. Generally, the service description language should not be a fixed template for the following reasons: (1) The functionality of the described services (i.e. the inputs and outputs, their relationship as well as the side effects) are too diverse to be adequately expressed with one unique and common template. (2) Services operate on data from various domains. To express them appropriately, the description template should not be fixed, but yield the possibility to be adapted to different fields of knowledge by configuration. (3) When describing a service offer or request, the user is frequently confronted with elements she does not know in advance, whereas additional elements cannot be inserted into the template reasonably. A flexible description language would help to solve these problems.

Editable. The service description must have the ability to be created, read and edited by human users. As services are especially interesting in mobile environments, this needs to be easy enough to be done on mobile devices with limited display and input facilities.

In the following, we will analyze existing service description languages in terms of these requirements (Section 2). It turns out that ontology-based description languages like DAML-S [2] are well suited for these demands. However, DAML-S only offers a very generic framework which cannot be used directly without further work. Therefore, in this paper, we enhance the usability of DAML-S by presenting a process for creating a service description and DINST, a graphical Java-based tool that implements the presented process. Both were developed in our research project DIANE [3], which aims at enabling semantic service trading in ad-hoc-networks. Section 3 contains the description of the process and illustrates it using the example of information services, a typical and important service category, while Section 4 describes DINST.

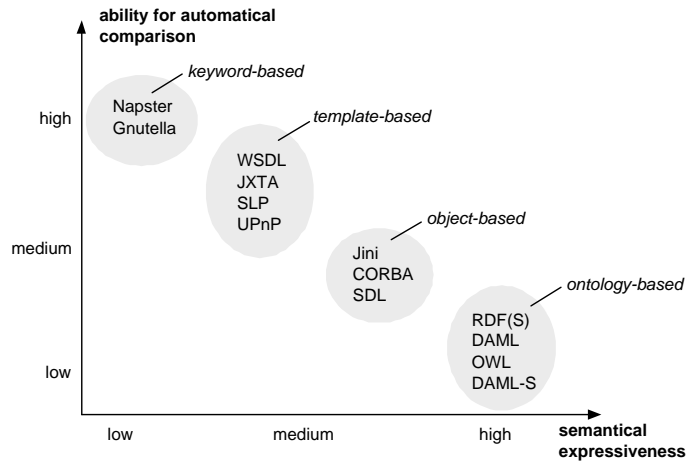


Fig. 1. Categorization of service description approaches by semantical expressiveness and ability for automatic comparison.

2 Existing Approaches

In this section, we examine existing approaches for service description languages and categorize them along the first two requirements: semantical expressiveness and the ability for automatic comparison. As a result, we can distinguish four clusters of description languages: keyword-based, template-based, object-based and ontology-based descriptions (see Figure 1).

The simplest form of service description is a *keyword-based approach* that tries to approximate the semantics of a service by providing some open keywords or a human-readable text. A typical example for this can be found in file-sharing networks like Gnutella [4] or Napster [5], where each participant offers its files as several “file downloading services”, which are described by one or a few keywords only – namely the filename of the offered document. In general, these techniques suffer from a low semantic expressiveness (resulting in low values for precision and recall), which often requires the human user to manually examine the numerous results in order to find the desired files. However, the comparison of two service descriptions is very easy and can be performed automatically and efficiently.

More sophisticated approaches as used in WSDL [6], ebXML [7], JXTA-Search [8], SLP [9] or UPnP [10] rely on a *template-based description*. They enhance the expressiveness by dividing the description into several properties that can be filled with concrete keywords or values. Very similar to that are *object-based* descriptions, which additionally allow to use object references to link objects in a service description. Typical representatives are the description languages used in Sun’s Jini [11] and CORBA [12]. In both, often only the non-functional aspects as well as the inputs and outputs of the service can be

described. The functional semantics, i.e. what the service does, must be derived from name, textual description and parameters. In general, precision and recall are increased, but humans are still necessary to analyze the found results. Therefore, in most cases, an automatic computer-to-computer interaction is not possible. Nonetheless, comparison is still easy to medium and can be performed quickly in most cases.

Ontology-based service descriptions promise far higher values for precision and recall. This results from their increased expressiveness stemming from the introduction of rich ontologies that make use of many different property types. Besides generic ontology languages like RDF [13], DAML [14] and OWL [15], the most prominent ontology-based description language especially for services is DAML-S [2]. DAML-S achieves a large expressiveness by providing an upper ontology for all types of services. This ontology proposes a rough structure for a general service description by defining common properties for non-functional as well as functional attributes (like input, output, precondition, and effect). Unfortunately, DAML-S is extremely generic as most of its attributes' types are left open, which leads to difficulties when comparing two descriptions.

When examining our requirements from the previous section, we come to the conclusion that only ontology-based description languages provide enough expressiveness for an automatic service trading. However, up to now, much has to be done to fulfill the other requirements:

- *Comparison* of two ontology-based descriptions is very difficult, as their structure is constrained just for the first one or two levels. This results from the fact that most of the parameters can be built up arbitrarily because no type constraints prescribe their structure.
- Although the extreme generality enables a high *flexibility*, a usage of such a description is only reasonable, if there are standardizations to structure the relationships between the parameters as well as the use of different categories and domains.
- *Editing* such service descriptions is very tedious as the files have to match some given flexible service ontology. It is possible to use an ordinary text or XML editor, but they are not designed for the language's specialities and offer no directed support for it.

As a result, in this paper, we will take DAML-S as a starting point for a service description, but enhance its use by presenting a process and a tool, which help to fulfill the requirements from the previous section.

3 A Process for Creating a Service Description based on DAML-S

In this section, we present a process for creating DAML-S 0.9 based service descriptions. It consists of four steps (see Figure 2): (1) acquiring the upper service ontology, (2) defining the service category, (3) inserting domain ontologies, and

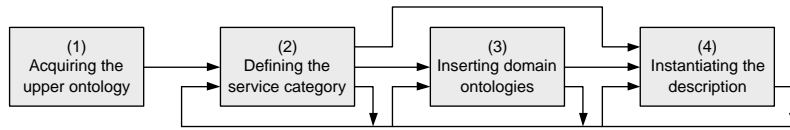


Fig. 2. A process for creating an ontology-based service description.

(4) instantiating the description. In the first three steps, a layered service ontology is constructed, which is taken as a template in the last step. Although this process seems strictly sequential, in most cases, the steps will be traversed iteratively, e.g. parts of the service description could already be instantiated before additional domain ontologies are included. In the following subsections, the steps are explained in more detail. We will exemplify them by constructing a concrete example description for a printing service.

3.1 Acquiring the Upper Service Ontology: DAML-S

When describing a service, we start with the upper service ontology DAML-S (see Figure 4a). Generally, this upper ontology defines the basic structure of a service description, i.e. services have to be presented in three aspects: the description of the functionality in a black-box view as **Profile**, the operating sequence in a glass-box view as **Model**, and the technical access path to the service as **Grounding** (see Figure 3). As we are interested in a description for service trading, in the following, we will concentrate on the service profile. Its properties can be divided into two main groups: functional and non-functional parameters. Non-functional properties provide meta-information about the service, i.e. its name, its offerer, quality parameters and so on. The functional properties try to capture what the service does (i.e. its functional semantics). In DAML-S, this is achieved by the four attributes **input** and **output** (describing control data entering and leaving the correctly working service) as well as **precondition** and **effect** (describing the state of the affected part of the world before and after successful service execution). To gain high usability and flexibility, the range type of these functional parameters is **ParameterDescription**, which is very generic as it is not restricted any further.

Notice that besides DAML-S other upper service ontologies could have been used. However, it is important that this ontology is unique and commonly ac-

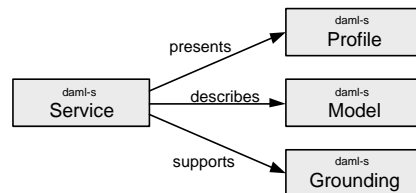


Fig. 3. Basic structure of a service description as proposed in DAML-S v0.9

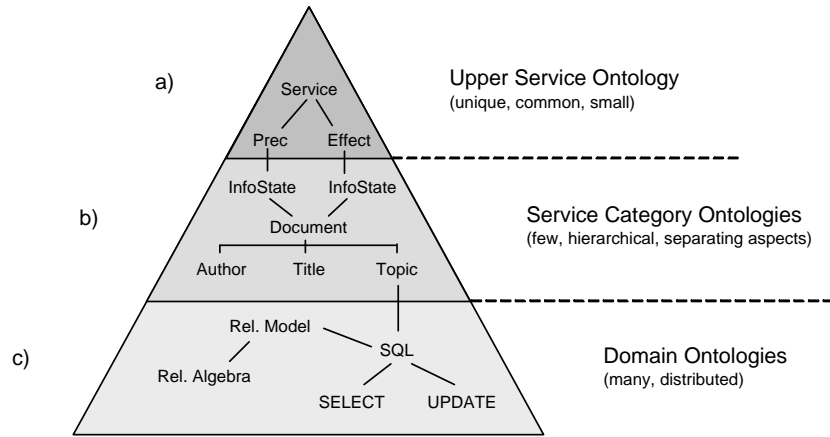


Fig. 4. Layered service ontology. On top, one unique, small, and commonly accepted upper service ontology can be found describing the general structure of a service description. It is specialized by one of a few service categories, which mainly defines and fans out the types of the functional parameters. At the bottom, one or more ontologies (taken from a large distributed pool) set up the domain specific vocabulary of the description.

cepted in the community that wants to trade services. Furthermore, such an ontology usually is rather small and generic.

For describing our example printing service, we just have to acquire the upper ontology in this step. No further action is needed.

3.2 Defining the Service Category

The middle layer of our service ontology describes different service categories (Figure 4b). This layer is needed because the classes of functionalities offered by services span an incredibly wide range. A service can be anything from the possibility to download a document, to the selling of shoes, to a complex computation. Obviously, the service descriptions for these services will need to address widely varying parameters and will thus be very different from one another.

The service category layer of the ontology is our approach to group the space of services into different classes. Each service category is characterized by possible types of the function parameters **input**, **output**, **precondition**, and **effect**. Moreover, these types are not simple data or enumeration types, but in most cases complex graphs of classes. Therefore, as a guideline, these types should be defined by class hierarchies or by aspect separation, which helps to reduce their complexity and enhance their readability. In short, a service category ontology consists of two parts: (1) type restrictions for the functional parameters as well as (2) hierarchical, aspect separating ontologies for these types.

Notice that the service categories themselves can build a hierarchy. This results from the possibility to vary the generality of the types that are used

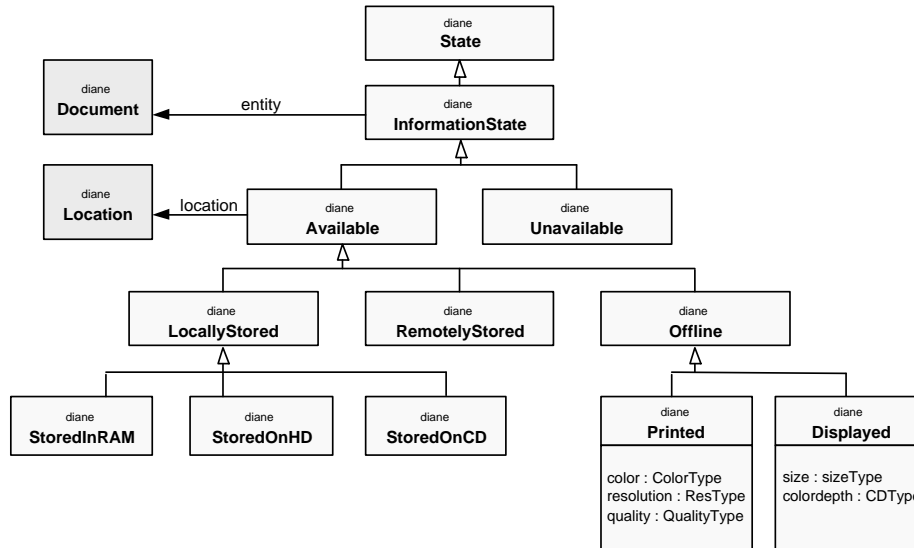


Fig. 5. Example ontology for states of documents. Instances of it are used to describe the precondition and effect (=postcondition) of an information service.

to specify the functional parameters of each category. However, we expect the existence of only a few different top level service categories, and at most a few dozen specialized service categories.

For our example printing service, we need one very important service category: the category of information services. We define an information service as a service that changes the state of a piece of information, i.e. a document. Typical states for documents are depicted in the hierarchical state ontology of Figure 5. Generally, a document can be available (which means that the user can access the document) or not. We distinguish three degrees of availability: the document is locally stored on the user’s device (e.g. in the RAM, on hard disk, or on CD), the document is remotely stored in the network, but its location is known and accessible to the user, or the document is not stored electronically, but available in an offline manner (e.g. printed or simply displayed on a screen). The description of a document itself can be structured in an aspect separating manner as described in [16].

As information services change the state of documents, typical information services are our printing service, which transforms the state from **LocallyAvailable** to **Printed**, a downloading service, which transforms the state from **Unavailable** to **StoredOnHD**, or a scanning service, which transforms the state from **Printed** to **LocallyAvailable**. To generalize, the typical structure of an information service can be found in Figure 6. **Input** and **output** are used for controlling information only, not for the documents itself (i.e. our printing service does not *output* the printed document, but possibly the time when the printing will be finished).

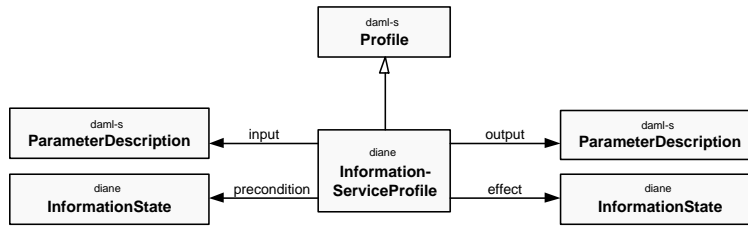


Fig. 6. Profile for an information service. `input` and `output` are describing controlling information as parameters, `precondition` and `effect` the state of the (affected) world before and after service execution. Note that this is not conform to DAML-S 0.9 as it demands the range of `precondition` and `effect` to be `ParameterDescription`, too.

`Precondition` and `effect` denote the state of the information before and after successful service execution. Note that this structure is not conform to DAML-S directly as preconditions and effects have to point to `ParameterDescriptions`. One possibility to solve this problem could be the declaration of the topmost class `State` as subclass of `ParameterDescription` resulting in a syntactically correct but semantically problematic solution. Therefore, we propose to substitute the range of `precondition` and `effect` to some sort of `State` which would help to more clearly describe the “change of the world” resulting from the service.

3.3 Inserting Domain Ontologies

The lowest layer of the ontology describes different application domains (see Figure 4c). Consider, for instance, a service offering access to documents on relational database systems and a user looking for information on SQL. Quite probably, the service would be able to provide the user with the desired information. However, for a computer to be able to determine that this is the case, it needs the information that SQL is a subtopic of relational database systems. Such information can be formalized by domain specific ontologies. Typically, such ontologies consist of three parts: (a) a general part (or TBox in description logics) describing types and possible relationships (like `Topic` and `isSubtopicOf`) and (b) a concrete world description (or ABox) by listing concrete instances and their relationship (e.g., `SQL` is a subtopic of `RelationalModel`), and (c) a domain specific comparison function.

There exists a potentially large number of these domain specific ontologies, ranging from databases, to pediatrics, to locations. It is not expected that everybody knows all these ontologies. Rather, whenever a service is offered or searched for, the appropriate ontology needs to be acquired. This could be achieved by special expert services offering access to these domain ontologies. Information about which ontology was used is part of the service description. Notice that in general, instances of domain ontologies can only be usefully compared with the appropriate domain specific matching functions.

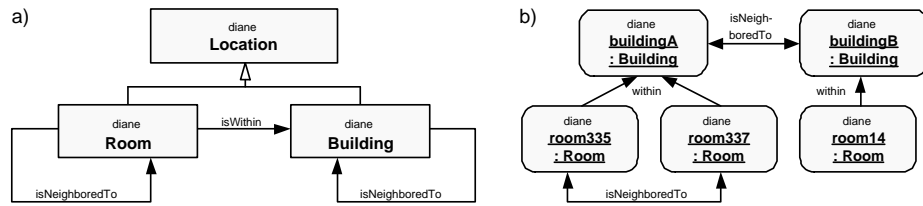


Fig. 7. Example domain ontology for locations.

To describe at which location the printout of our example service will be available, we use the domain ontology describing locations on an university campus from Figure 7. Its general part (a) differentiates Locations into Buildings and Rooms and allows a `isNeighboredTo` and a `within` relation. With that, the situation of the real world is described in the second part (b). The domain specific comparison function (not shown) could for example use the `isNeighboredTo` and `within` relation to calculate a “hop” distance between arbitrary Locations.

3.4 Instantiating the Layered Service Ontology

To obtain the concrete service description, the layered service ontology created in the first three steps has to be instantiated. Therefore, appropriate (mostly

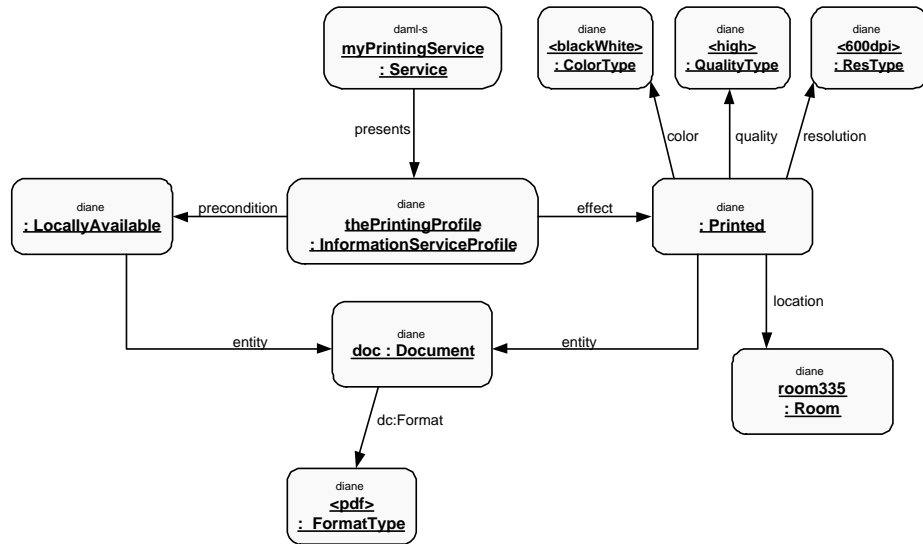


Fig. 8. Typical description of a printing service. Basically, it transforms the state of a document from `LocallyAvailable` to `Printed`. Note that `precondition` and `effect` are not independent, but connected through the Document of interest. Moreover, all parameter values are instantiations of enumeration types to enhance the expressiveness.

rather specific) classes are chosen from the ontology, transformed to instances and connected to other instances with valid property instances. Moreover, XML schema datatypes like `xsd:string` or `xsd:integer` are filled with concrete values, whereas enumeration types are instantiated by picking one predefined instance from a set.

An example of a printing service instance is depicted in Figure 8. The service changes the state of an arbitrary pdf document from `LocallyAvailable` to `Printed` in room 335 with the parameters `black/white`, `high quality` and `600 dpi`. Two characteristics of the description have to be mentioned in detail:

- Precondition and effect are not independent, but connected via the entity-property of their states pointing to the same document instance. In more complex service descriptions, the effect and the output may also be dependent from the input.
- All values for describing the characteristics of the printout and the document are instances of predefined enumeration types like `ColorType` or `FormatType`. The chosen instance is denoted within angle brackets like `<600dpi>`. Avoiding instances of generic data types from XML schema (like `xsd:string` or `xsd:decimal`), we increase the expressiveness and comparability of the description as the properties are restricted to values that uniquely correspond to characteristics from the real world.

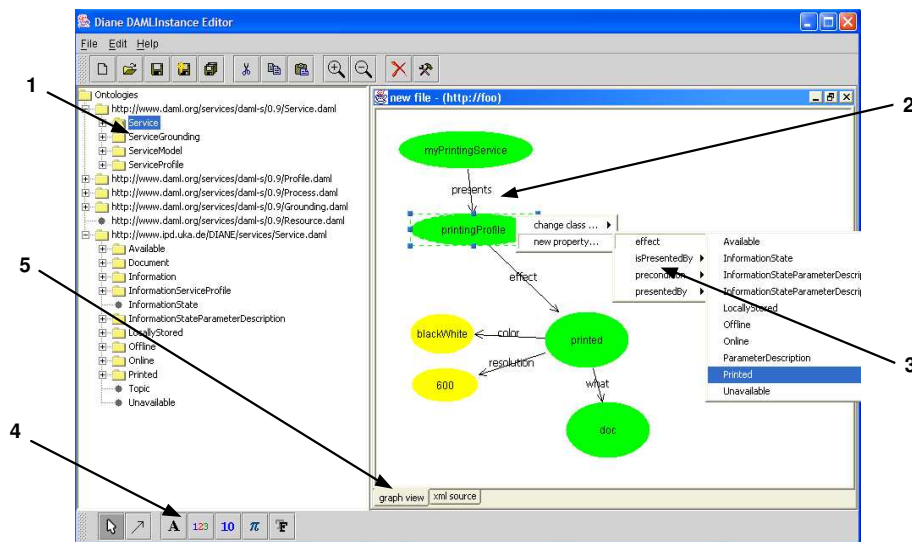


Fig. 9. Screenshot of DINST - a tool for instantiating DAML-S based service descriptions: (1) Area for loading ontologies, (2) graphical representation of the description, (3) context sensitive operations to extend and modify the description, (4) buttons to add datatype instances, (5) switch to change between graphical and XML view.

3.5 Evaluation

In this section, we analyze whether the proposed process helps DAML-S to fulfill the requirements identified in Section 2. Generally, descriptions that are complying with the process are well *comparable*: the overall structure of the description is fixedly given by the upper ontology whereas the category ontology structures the details of the functional parameters. Moreover, although the used domain vocabularies are more or less unstructured, they can be compared with the domain specific matchers that come with each ontology. On the other hand, *flexibility* is not restricted too much, as arbitrary (but standardized) categories and domains can be used. Furthermore, elements can be added and omitted optionally which is possible thanks to the RDF basis. However, this can lead to penalties when matching. In the next section, we will introduce a graphical tool that eases *editing*.

4 A Tool for DAML-S Instantiation: DINST

Manually writing DAML-S based service descriptions in the style of the presented process is a tedious task. Thus, we have developed the tool that supports the process from Section 3 in a purely graphical manner. A screenshot of DINST is depicted in Figure 9. In Area 1, different DAML ontologies can be loaded. Typically, the editor starts with the DAML-S ontology and the user loads own category and domain ontologies. Ontologies are represented as folders that contain the classes of the ontology when opened. Classes themselves are folders containing possible properties for this class. To create an instance of a class, it is possible to drag and drop a class from Area 1 to Area 2. By right-clicking on an instance, the user can choose from two options (Area 3): (1) changing the class of the instance to a subclass from a list, which is often necessary when a class has to be specialized because of an ontology that has been loaded afterwards (e.g., a general instance of `State` is refined to a special state like `Printed` after the information service category has been loaded.) (2) enlarging the instance graph by adding a property and a correct typed range instance to the current node selectable from a list (e.g. right-clicking on an information service profile instance yields the possibility to add a property instance of effect and an instance of a state like `Printed`). For reasons of flexibility, it is also possible to add property and datatype instances that are not represented in the ontology by using the tools from Area 4. Finally, Area 5 provides a switch for changing between graphical and XML view, which helps to get an idea of the final result. To summarize, DINST offers the functionality to edit a layered service description in a purely graphical manner. In principle, it can also be used to edit the other parts of the description, namely the `ServiceModel` and the `ServiceGrouping`.

5 Conclusion and Future Work

In this paper, we have presented a process for creating a DAML-S based service description. This process sets up a layered service ontology consisting of

an upper service ontology, an ontology for the service category, and several domain ontologies, which is instantiated to a concrete service description at the same time. A typical service category for information services and an example of an appropriate service instance have been presented. To describe the functional semantics of information services, we introduced a state ontology dealing with states of documents and proposed to change the range of `ServiceProfile`'s `precondition` and `effect` properties in DAML-S from `ParameterDescription` to some sort of state. Finally, we presented DINST, a Java-based tool that supports the user when creating service descriptions by offering a graphical interface which leads him through the layering and instantiating process.

Currently, we are in the process of developing ontologies for further service categories beyond information services. We are also implementing and describing a number of example services. With that, we hope to gain more insights into possible dependencies between inputs, outputs, preconditions and effects.

Acknowledgement

We would like to thank Ting Zheng for the implementation of DINST.

References

1. Deutsche Forschungsgesellschaft (DFG): Schwerpunktprogramm 1140: "Basissoftware für selbstorganisierende Infrastrukturen für vernetzte mobile Geräte". (<http://www.tm.uka.de/forschung/SPP1140/>)
2. Defense Advanced Research Projects Agency: DARPA agents markup language - services (DAML-S). (<http://www.daml.org/services/>)
3. Institute for Program Structures and Data Organization, Universität Karlsruhe: DIANE project. (<http://www.ipd.uni-karlsruhe.de/DIANE/en>)
4. Gnutelliums: Gnutella. (<http://gnutella.wego.com>)
5. Napster: Protocol specification. (<http://opennap.sourceforge.net/napster.txt>)
6. World Wide Web Consortium: Web service description language (WSDL). (<http://www.w3.org/TR/wsdl>)
7. OASIS, UN/CEFACT: ebXML. (<http://www.ebxml.org/>)
8. Waterhouse, S.: JXTA search: Distributed search for distributed networks. Sun Microsystems Whitepaper - <http://search.jxta.org/JXTAsearch.pdf> (2001)
9. Network Working Group: Service location protocol - RFC 2165. <http://www.ietf.org/rfc/rfc2165.txt> (1997)
10. Microsoft Corp.: Universal plug and play. (<http://www.upnp.org>)
11. Sun Microsystems: Jini. (<http://www.jini.org/>)
12. CORBA: Trading object service specification. (<http://cgi.omg.org/docs/formal/00-06-27.pdf>)
13. World Wide Web Consortium: Resource description framework (RDF). (<http://www.w3.org/RDF/>)
14. Defense Advanced Research Projects Agency (DARPA): DARPA agent markup language (DAML). (<http://www.daml.org/>)
15. World Wide Web Consortium: Web ontology language (OWL). (<http://www.w3.org/TR/owl-ref/>)
16. König-Ries, B., Klein, M.: Information services to support e-learning in ad-hoc networks. In: First International Workshop on Wireless Information Systems (WIS2002). (2002) 13–24