An Approach For Describing/Discovering Services and For Adapting Them to the Needs of Users in Distributed Systems

Oussama Kassem Zein and Yvon Kermarrec
ENST Bretagne, Dept. IASC
Technopôle Brest Iroise, BP 832, 29285 Brest Cedex, France
Email : {Oussama.Zein,Yvon.Kermarrec}@enst-bretagne.fr
Tel : 33 2 29 00 12 85 Fax : 33 2 29 00 10 30

Abstract

In an earlier context, we have defined a metadata model to describe the static properties of a service. We have developed and implemented tools which constitute a trader based on ontologies and knowledge representation in order to discover a service based on its static properties. In this paper, we propose an extension of these tools to address the dynamic properties of a service (its behavior). We also present our approach for adapting services to the needs of different users. For this purpose, we define for each user a profile. These profiles will be taken into account in the searching tools defined in our trader to filter the service queries and to provide to the client its appropriate services.

KEYWORDS: service description, service discovery, user profile, ontology, trader.

Introduction and Context

The trader (ISO 1992), (OMG 1997) plays a major role in distributed systems since it enables to link clients to servers. It is an advanced directory service which allows services to be discovered at run-time via an attribute based (or yellow page) style of search.

A service provider (Exporter) may advertise a service offer and a service consumer (Importer) can inquire in order to discover a service needed. In our current investigation of service description and discovery, we have used the approach chosen for the trader or the mediator we have developed and which relies on the architecture described in (Zein & Kermarrec 2002) to discover services.

Once a service is discovered, the client needs information about how it can invoke the service operations. Therefore, it must discover the behavior of the service which determines the protocol of interaction between the service operations. We have selected SDL (ITU-T 2001) and Interface Automata (Alfaro & Henzinger 2001) as supports for describing the behavior of a service by an automaton. This automaton describes a service as a function that provides outputs from given inputs. The client can thus query a service based on the data flows by indicating the input and the expected output. This combination of search allows clients to discover a service by querying its dynamic and static characteristics.

In this paper, we also present our approach for adapting services to the needs of different users. For this purpose, we try to define a profile for each user (client). It contains desired values by the client of some properties of a service like its location. When a client calls the trader to query services and if there are numerous offers, we will use the client profile to filter the returned offers and to make the request more selective. This enables the client to get the services that are compatible with its preferences.

This paper is organized as follows: Section 2 describes how knowledge representation and ontologies are used in our current implementation. Section 3 illustrates how a service behavior can be represented by an automaton. Section 4 presents the tools that we have designed and implemented to manage the ontology, to index and query the service properties. Section 5 presents the description of a user profile and how we use it in the searching tools. Finally, we present the perspectives of our work and conclude in section 6.

Knowledge Representation and Ontologies

Many definitions of ontologies have been proposed and there is a growing interest for knowledge management. Gruber gives the following definition (Gruber 1993): "An ontology is a formal, explicit specification of a shared conceptualization of a domain of interest". The term is borrowed from philosophy, where an ontology is a "systematic account of existence". As the importer and the exporter interact, it is necessary to share the same vocabularies and the same concepts.

The role of an ontology in the knowledge engineering process is to facilitate the construction of a domain model. It provides a vocabulary of terms and relations with which to model the domain.

We can represent an ontology by a graph whose nodes are concepts and whose arcs represent relationships or associations between the concepts.

Among the most important of these relationships is inheritance, which is a powerful abstraction for sharing properties among classes while preserving their differences (Huhns & Singh 1997). Each subclass inherits the features of its superclass, adding other features to its own. This relationship provides flexible ways to find services closest to our request.

When we query information from a concept S in an ontology and this information is not available, the inheritance relation allows us to navigate in the graph representing this ontology and to query information from the concepts that inherit S. The result of this recursive search is therefore the closest to the starting point (see section 2.1). The issue is therefore to design properly the ontology concepts and relationships and to take into account its future usage.

In our approach, we have used ontologies in order to index and store the service characteristics (properties) by concepts. Each concept is composed of a set of attributes. Clients can discover a service by querying their characteristics from the ontology by using a logic language like Frame-Logic (Kifer, Lausan, & Wu 1995). This language will be presented in the next section.

Logic Language

The structure of the ontologies relies to the logic of the first order. The languages available which allow us to query information from the ontologies are those of the logic. We have selected an existing language, so called F-Logic (Frame-Logic) as our support engine. It is an object oriented descriptive language which aims at defining concepts, entities and relationships between these concepts and entities. Its main achievement is to integrate conceptual modeling constructs (classes, attributes, inheritance, axioms, ...) into a logical framework.

We indicate below the various logic expressions used by F-Logic:

- Subclassing: *C1::C2*, means that class C1 is a subclass of C2. This expression can be used to define that a class inherits from another.
- Instance of: *O:C*, means that O is an instance of class C.
- Attribute Declaration: *C1[A=>>C2]*, means that, for the instance of class C1 an attribute A is defined whose value must be an instance of C2.
- Attribute Value: *O[A->>V]* , means that the instance O has an attribute A with value V.
- Part-of: *O1 <: O2*, means that O1 is part of O2.

A fact is an instance of a class. For example:

S1:C[AA->>VV;BB->>VVV].

This fact indicates that S1 is an instance of the class C and has two valued attributes: AA with value VV and BB with value VVV.

We can define rules in F-Logic which can infer new knowledge, i.e., extending the object base intentionally. Their roles are to ensure the integrity of the data in the ontology and to derive new knowledge and relations between the concepts and the attributes that will be assigned in the ontology. Rules encode generic information of the form: whenever the precondition is satisfied, the conclusion must be verified. The precondition is called "rule body" and is formed of arbitrary logical formulas, which are combined by OR, NOT, AND, <-(=>), <->(<=>), FORALL (∀) and EXISTS (∃). Syntactically, the rule head is separated from the rule body by the symbol "<-" and every rule ends with a dot. By using this approach, we can benefit from all the power of the logic of first order. For example:

FORALL x,y x[son->>y] <- y:man[father->>x].

This rule means that for each instance y of "man" having x as a "father", x has y as a "son". It creates new knowledge from existing facts.

A query can be considered as a special kind of rule with an empty head.

FORALL x,y,z <- x:person and x[y->>z].
FORALL x,k <- x:k and k::person.

The first query returns all the triplets (x,y,z): x instance of the class "person", y attribute of x with value z. The second query returns all the instances of the concepts ("man" and "woman") that inherit "person". If for the first query there are no offer available, the client can use the second one to query the concepts that inherit "person". It provides the instances of ("man" and "woman") which are the closest to the starting query. This indicates that clients can benefit from the inheritance relationship in the queries tools to find the instances that are the closest to their required instances in intelligent ways.

Ontobroker as a Management Tool

Ontobroker (Fensel *et al*. 1998) is a result of a european research program. It is a complete system to manage ontologies. It consists of a query interface for formulating queries and an inference engine to derive new facts.

The inference engine is a software program which interacts with users and which processes the results from the rules and data in the knowledge base. The inputs of the inference engine consists of an ontology, collected facts and queries formulated in a logic language (e.g, F-Logic).

Ontobroker reads the rules and stores them in an internal database. It uses its inference engine to evaluate concepts, entities and relationships in its database and to compute an answer to the query submitted.

It provides a graphic interface for indexing and querying a service.

Service Behavior

The behavioral description of a service is critical since it provides the information enabling a client to use the service and to understand how its service invocation needs to be performed. This includes, for example, the correct sequences in which the operations of the service need to be invoked. The description of the service behavior is complementary to the description of its static properties: Once a client discovers a service by querying its static properties, it can query its behavior as a function or a relation between inputs and outputs and which sequence of operations to be invoked to get an output from a given input.

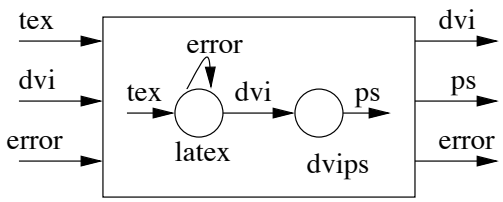

Figure 1: Example of an automaton describing a service behavior

Automaton Description SDL has been designed to specify and to describe the functional behavior of telecommunication systems. It describes a process behavior by an automaton which can then be exploited to validate the specification. Therefore, based on SDL and Interface Automata, we describe the service behavior via a finite state machine ("an automaton") that models the allowed operation sequences. Invocation of the service must satisfy these sequences. For example, a seller may require a buyer to log in before ordering something. Thus, we can describe valid sequences of operation invocation. The interactions between the service operations occur through message exchanges. These messages are the inputs and the outputs of the operations. We can describe the automaton representing the service behavior as: a black box which is a relation between its inputs and its outputs (external behavior) and as a white box which indicates the allowed interactions between the service operations to provide outputs from given inputs (internal behavior).

The automaton describing the service behavior is composed of a set of states and transitions between the states. Each service operation is defined by a state. For a current state, the successor state belongs to the set of its matching states: i.e., the output of the current state can be connected to the input of its next. Therefore, a transition connects a state to one of its valid next state. An interaction between operations can be performed only if it is defined in the automaton.

Figure 1 shows an example of an automaton describing a service behavior: As a black box, this automaton has inputs ("tex", "dvi", "error") and outputs ("dvi", "ps", "error"). As a white box, it has two operations: "latex" and "dvips". The interactions between them are presented in figure 1.

A client can query an automaton as a function that provides outputs from given inputs and it can query the order of the operations to be invoked to get an expected output from a provided input.

The Tools Managing an Ontology

Description of the Service Characteristics

We first create an ontology describing the static properties of a service defined in (Dumas *et al.* 2001), (Merz, Wiethaut, & McConnel 1997) and (Swartout & Knight 1997) as example. These properties (service location, service provider name, etc) are the metadata that use the concepts of the domain to describe a service. We can benefit from the facilities and the structures provided by an ontology like the inheritance relationship to construct the metadata model for our service. If the service description is well structured in an ontology, then the queries processing can benefit from the underlying ontology properties and tools to provide powerful results. We present the following example which defines a metadata model of service description written in F-Logic:

```
Service [ name_provider =>> STRING;
          service_location =>> STRING;
          service_type =>> STRING;
          .
          .
          . ].
```

As indicated before, we represent the behavior of a service with an automaton. This automaton can be considered as a black box that for each input provides outputs (external behavior) as indicated in the previous section. It describes a service as a relation between its inputs and its outputs. For this purpose, we define in the ontology that we have created the following concept written in F-Logic:

```
Behavior [ service_type =>> STRING;
           input =>> STRING;
           output=>> STRING ].
```

We define another concept that describes the automaton as a white box which allows us to understand which operations and in which order we must invoke them to get a desired output from a given input (internal behavior). We can describe this concept in F-Logic with the following:

```
Automaton [ service_type =>> STRING;
            operation =>> STRING;
            input =>> STRING;
            output =>> STRING ].
```

By defining the above concepts, a client can query a service based on its static properties (Service concept), its behavior as a function that provides outputs from inputs (Behavior concept) and the correct order of the operations to be performed to provide an output from an input (Automaton Concept). When combining the description of the static and the dynamic (behavior) properties of a service, the indexation/search of a service can be addressed completely.

Implementation

In this section, we present the tools that allow a client to manage ontologies and examples that permit it to query/add a service through the ontology defined in the previous subsection.

Tools We have defined the tools that allow us to manage an ontology with the following interfaces:

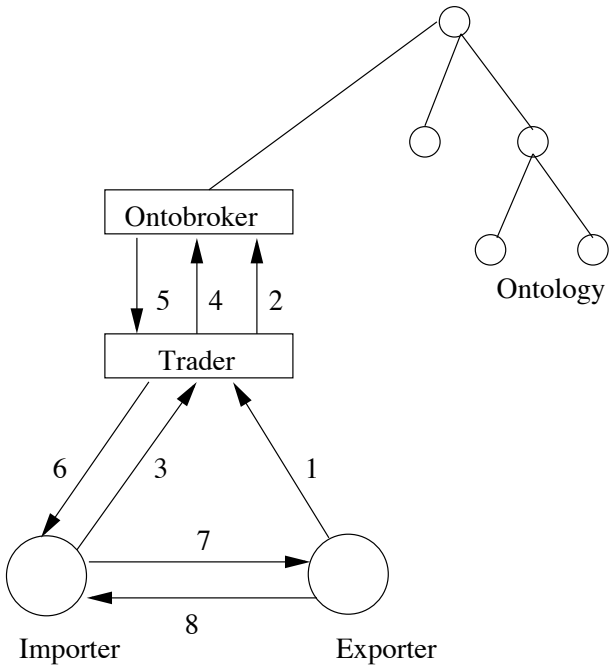

Figure 2: Interactions between the Trader, Ontobroker, Importer and Exporter

- Importing function: It is used by importers. It takes the query as the input, it makes a request to Ontobroker for the service required and returns the matching service offers to the importer.
- Exporting function: It is used by exporters. It allows a fact to be added to the ontology. A service offer is an instance of a service. It is described by a fact. To add it to the ontology, an exporter can call this function. Each service offer includes an object reference that is necessary for clients to invoke the service being advertised.
- A function is defined to remove a fact from an ontology.

Finally, we declare an object which implements all these functions and we start it on a server. This object is considered as a trader which allows a service to be discovered in an elaborate search. The clients and the servers use this object to contact Ontobroker and to add/retrieve services through the ontology.

Figure 2 shows all the interactions in our trader. We launch the Ontobroker server by indicating the port number, the host name, the ontology that we have defined (section 4.1) as target and the rules used to initialize this ontology. This makes it accessible from another machine in the distributed system. The steps are as follows:

1. The exporter sends to the trader a fact written in the F-Logic language which constitutes a service offer including an object reference and properties of the service.
2. The trader contacts Ontobroker to add the service offer to the ontology.
3. The importer requests the trader a service offer by specifying the desired service properties in the form of the F-Logic language.
4. The trader calls Ontobroker to query the service offers needed.
5. Ontobroker searches the services requested from the ontology and it sends them to the trader.
6. The trader returns to the importer the results with their references.
7. The importer can interact with the exporter of these service offers by invoking the operations available.
8. The exporter replies to the importer's service request.

This trader provides more powerful results in the searching tools than the standard trader (like ODP trader (ISO 1992) or OMG CORBA trader (OMG 1997)). The standard trader stores the service offers in its databases. Clients can query services from the trader by using SQL or a boolean request. The query facilities are limit and rigid. When a client requests a service offer and if there is no matching service, the trader indicates to the client that the service requested is not available and there are no possible matches. Maybe, there are services closest to the service requested by the client, but the trader has not the ability to inform the client. This major limit is due to the structure of the databases. The service queries are rigid due to the language used like a boolean language. The Client can not use (∀, rules, facts, =>, etc) which enrich the query tools. Therefore, the standard trader lacks flexibility.

Our trader exceeds all these limits of the standard trader. When the client requests a service offers and if this service is not available, our trader provides to the client the possibility of querying from the concept that inherit from the concept of the service requested (see section 2.1). These concepts allow the client to get the services that are the closest to its starting service. This phenomenon provides a more powerful results in the service queries than the standard trader. This is due to the structure of the ontology and its relationships like inheritance. When we using a logic language like F-Logic, the service queries become more powerful. We can define rules which enrich the query tools and infer new knowledge and we use logic expressions (∀, rules, facts, =>, etc). The client can benefit from all the power of the logic of first order which is more flexible than SQL, boolean language, etc.

Examples We give a few examples of the queries used by the importing function that we have defined to query a service from the ontology described in section 4.1.

- Static properties:
  FORALL x <- x:Service and x[name_provider->>"Service Transformation"] and x[service_type->> "tex_transf"].

  The result of this query is all the instance x of "Service" having "Service Transformation" as a name of provider

and “tex_transf” as a type of service.

- Dynamic properties:
  We can extend the request by querying the behavior of this service which has “tex_transf” as a type of service. We query the service as a relation between inputs and outputs (*Behavior* concept). We indicate the desired outputs from given inputs as criteria in the request.

  FORALL x <- EXISTS y x:Service and x[name_provider->>“Service Transformation”] and x[service_type->>“tex_transf”] and y:Behavior and y[input->>“tex”] and y[output->>“ps”] and y[service_type->>“tex_transf”].

  For each service, we can query which operations need to be invoked to obtain a desired output from a given input (*Automaton* concept).

  We indicate examples of facts used by the exporting function that we have defined to add a service offer to the current ontology:

  ser:Service[name_provider->>“Service Tr- ansformation”; type_service ->>“tex_transf”].

  This fact allows us to add the service offer “ser” having “Service Transformation” as a name of provider and “tex_transf” as a type of service.

  behav:Behavior[input->>“tex”;output->> “ps”; type_service ->>“tex_transf”].

  This fact allows us to add in the knowledge base the instance “behav” of “Behavior” having “tex_transf” as a type of service, taking an input a “tex” file and providing an output a “ps” file.

User Profile

For adapting services to the needs of different users, so-called “user profile” are frequently defined and used. User profiles are collections of information and assumptions about behaviors and preferences of individual users: they are used in the services adaptation and personalization process. A user profile helps to adapt services to the user needs. It holds the user requirements: it can contain critical information about the user like its location, its language, its experiences in the domain, its preferences, etc.

When a client calls the trader to query services from an ontology (like this described in the previous section) and if there are numerous offers, we use its profile as an addition criterion to extend its request. This allows the client to filter the results and to get the services that satisfy its profile and its requirements.

For example, a client creates a profile which includes as preferences its desired specialities of restaurants (for example, the french specialities). It requests all the instances of a restaurant service having “Paris” as a service location by the following query:

FORALL x <- x:restaurant and x[service_ location ->> “Paris”].

If for this query the client gets a high number of answers, we use its profile to filter the results and to search its appropriate services. The filter is applied by adding the extra criteria of the profile:

FORALL x <- x:restaurant and x[service_ location ->> “Paris”] and x[speciality ->> “French”].

This query allows the client to obtain its matching services which satisfy its profile. We consider that the user profile contains properties with their default values. When we extend the service query to include the properties contained in the user profile and if some properties have values in the user request different from those in the user profile, we use their values which are required in the user request as priority to search the services. This is possible by using “or” between the properties existed in the user profile in the extended request:

initial_request and [p1->>v1 or p2->>v2 or p3->>v3 or ...].

Where p1, p2, p3 ... the properties contained in the user profile with their values v1, v2, v3 ... respectively.

In addition, we have used the score to order the results before returning them to the client. Each property in the user profile has a value and a preference. The sum of these preferences is equal to 1. For each property in the user profile, we calculate its partial score by the following expression:

$$V_i = \frac{A_i}{\max_i A_i} P_i$$

Where V_i is the partial score of the property “i”, A_i its number of occurrences in the results (returned offers) and P_i its preference. We count the number of occurences of a property “i” in the returned offers if only its value coincides with that in the user profile.

Each offer has a score which is the sum of the partial scores of its properties included in the user profile such as their values coincide with those which are in the user profile. Our trader returns the offers to the client ordered by the score.

Conclusions and Perspectives

In this paper, we have presented our current related work in the indexing and searching of a service. For this latter operation, we have proposed a metadata model for describing a service. The current model we have proposed consists

of several concepts which describe the static properties of a service like (its name of provider, its location, etc) and its dynamic properties (its behavior). We have implemented this model with ontologies, consisting of concepts representing the characteristics of a service. Flexibility is introduced since the ontology and its content can be changed and adapted to usage. This approach is a powerful one since it provides a complete description of a service. When a client discovers a service offer by querying its static properties, it can obtain information about the service behavior enabling it to use the service discovered and to understand how the invocation of the service operations needs to be processed.

We have presented tools we designed and developed: they are based on ontologies and knowledge representation. They allow an exporter to advertise a service offer and a client to discover a service by querying its static and dynamic properties in distributed systems. By using ontologies and their relationships like inheritance and a logic language, the service discovery is more flexible. When we request a specific service and if none exists, the inheritance relationship allows us to navigate in the ontology and to query information from the concepts that inherit the concept of the service requested. These concepts enable us to get the services that are the "closest" to the service desired. This requires that the ontology is designed with care and user intention in mind.

We have presented a model of user profile for adapting services to the needs of different users. It includes information about the user and its preferences. When the client requests a service from the trader and if there are a high number of answers, we use its profile to adapt the obtained offers to its requirements. This allows the client to get its interested and desired services.

As perspectives, we will extend the metadata model to describe wireless and mobile services and to adapt the trader that we have presented and the user profile that we have defined to be compatible with this metadata.

References

Alfaro, L. D., and Henzinger, T. A. 2001. Interface Automata. In *Proceedings of the Ninth Annual Symposium on Foundations of Software Engineering*. ACM Press.

Dumas, M.; O'Sullivan, J.; Heravizadeh, M.; Edmond, D.; and Hofstede, A. 2001. Towards a semantic framework for service description. *In Proc. of the 9th Int. Conf on Database Semantics, Hong-Kong*.

Fensel, D.; Decker, S.; Erdmann, M.; and Stude, R. 1998. Ontobroker : The very high idea. *In Proceedings of the 11th International Flairs Conference (FLAIRS-98), Sanibal Island, Florida, USA*.

Gruber, T. R. 1993. A Translation Approach to Portable Ontology Specifications. *Knowledege Acquisition, 5 : 199-220*.

Huhns, M., and Singh, P. 1997. Ontologies For Agents. *IEEE, Internet Computing Journal*.

ISO. 1992. *ISO/IEC JTC1/SC21/WG7/N743. Working Document on Topic 9.1- ODP Trader*.

ITU-T. 2001. ITU-T Recommendation Z.100. Specification and Description Language (SDL).

Kifer, M.; Lausan, G.; and Wu, J. 1995. Logical foundations of object-oriented and frame-based language. *Journal of the ACM, 42(4) : 741 - 843*.

Merz, M.; Witthaut, M.; and McConnel, S. 1997. Catalogue and Service Architecture. http://osm-www.informatik.uni-hamburg.de/osm-www/public/docs. *OSM D8*.

OMG. 1997. *CORBAservices : Common Object Services Specification. OMG Document*.

Swartout, W., and Knight, R. 1997. Toward distributed use of large-scale ontologies. In Spring Symposium Series on Ontological Engeneering. *pages 33-40, Stanford,AAA1 Press*.

Zein, O. K., and Kermarrec, Y. 2002. An Elaborate and Flexible Trader Based on Ontologies. *IFIP WG6.7. Workshop and EUNICE Summer School on Adaptable Networks and Teleservices*.