

Beyond service discovery and composition

Oussama Kassem Zein

Yvon Kermarrec

ENST Bretagne, Technopôle Brest Iroise

BP 832, 29285 Brest Cedex, France

{Oussama.Zein,Yvon.Kermarrec}@enst-bretagne.fr

Abstract

In this paper, we present our current work and approach on service discovery and composition and their applications in two domains: e-learning and distributed objects. We show how the use of ontology, knowledge representation techniques, help us to achieve our goal and open new perspectives for end users and also service providers.

KEYWORDS: service discovery, description and composition - composite web services - ontology.

1 Introduction and Context

Service discovery has always been a major concern for distributed systems. Name services and their variations appear in most of the middleware and distributed platforms. For example, a trader 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. And this service is even more critical in systems where services (or resources) are in large number, are made available any time and may disappear.

We showed in [1] [2] that the service de-

scription plays a central role and that it was necessary to take into account various levels of description in order to retrieve the service and to access it. Once a service is properly described, it can then be indexed and users (either human or software) can look for it, and then access it. We developed a trader and integrated it as an OMG CORBA component in the ORBACUS platform and then as a JMS component.

Once a service or object is found, the user must also determine how to access it or invoke its methods. This constitutes the second stage of our work. For consistency reason, we considered the interface and behavior of the service (or object) as part of the description. Therefore, the search facilities return information on how a service can be invoked and accessed. This extra level of description shows powerful results since a user (human or software) can then call any service, which is described. This constitutes the initial stage for service composition and we proposed several models for combining services.

When we associated, service description, service directories and search facilities, and service composition, we achieved a powerful combination and reached an interesting goal. In the first section of this paper, we shall present the underlying models we have se-

lected. In the second section, we shall describe the various levels we proposed for describing a service: a set of attributes and properties, but also an interface and behavior description that can be used when accessing the service. In the third section we present the search facilities we propose and the underlying model for composing services when a search is unsuccessful. In the fourth section, we present the applications of our approach in two different domains. The conclusion will raise issues and present our future work.

2 Ontologies and knowledge representation

Many definitions of ontologies have been proposed and there is a growing interest for knowledge management. We indicate a definition given by Gruber [3]: “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.

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.

In our approach, we have used ontologies as a means to express a domain model and also a consistent service description. Initially, our concern was the development of a trading service. For this purpose, it was necessary to describe services and objects so that users can find them. The underlying use of ontologies helped us to explicit the vocabularies used and also develop a generic architecture: the domain ontology which is specific to the application is considered as a parameter.

We can represent an ontology by a graph whose nodes are concepts and whose arcs rep-

resent relationships or associations between the concepts. Among the most important of these relationships is inheritance, which is powerful abstraction for sharing properties among classes while preserving their differences [4]. Each subclass inherits the features of its superclass, adding other features to its own. Inheritance is transitive across an arbitrary number of levels. 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.

Inheritance plays a central role in our context since it makes it possible to add flexibility in a search operation. When navigating in the ontology through its inheritance relationships, we can present to the user ways to make his/her request more selective (when the results of the search operation are numerous) or less selective (i.e.; to relax some of his/her constraints when the search returns no result).

We have used OntoBroker as the tool to manage ontologies: it provides a query interface for formulating queries and an inference engine to derive new facts. OntoBroker provides also storage facilities which make it usable in our context.

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 [5].

The main issue is therefore to design properly the ontology concepts and relationships and to take into account its future usage by various kinds of users. We shall present parts of the ontologies in the last section of the document.

3 Describing a service

In this section, we present a metadata model to describe a service. We build an ontology [6] which is constituted of concepts to index and store the service properties. Service description (and indexing) is central and the model should be designed with care with the help of end users.

3.1 Static Properties of a Service

Many different approaches for describing, managing and providing services have been developed over the years [7], [8], [9]. Nevertheless, a clear understanding and consensus about what constitutes a service has not been reached. We have used DAML-S as starting point to identify a set of characteristics of a services (provider, pricing, payment, quality of service, type of service, etc.) which can be useful in different contexts.

3.2 Dynamic Properties of a Service

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.

SDL [10] 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. There-

fore, based on SDL [10] and Interface Automata [11], 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.

3.3 Description of a Service Interface

As each service has an interface, we define a concept that describes the service interface. This concept is considered as an attribute of a service. It contains the descriptions of all the operations, the exceptions, the attributes, the type definitions and the constant definitions of the service. As the number of these latter attributes vary, we define each of them with a list. For example, each element of the operations list contains a list of parameters, specific attributes (type of return, number of

parameters, and so on) and a reference to the next element of the list (recursive). Thus, the Operation concept is composed of the following attributes:

```
Operation[ name ==>> STRING;  
           nbr_parameters ==>> INTEGER;  
           parameter ==>> Parameter;  
           return_type ==>> STRING;  
           operation ==>> Operation ].
```

Each element of the parameters list includes a name, a type (integer, string, etc), a position and a mode that indicates the direction in which the parameter is being passed during a dynamic invocation (input 'in', output 'out', input/output 'inout') and a reference to the next element of the list.

```
Parameter[ name ==>> STRING;  
           type ==>> STRING;  
           position ==>> INTEGER;  
           mode ==>> STRING;  
           parameter ==>> Parameter ].
```

This interface description is very similar to WSDL and IDL of CORBA and this guarantees compatibility with existing models.. For CORBA, the interface repository can be used to invoke dynamically a service, for example by using the DII (Dynamic Invocation Interface) of CORBA. This interface allows clients to build dynamically operation requests for any interface that has been stored in the interface repository and invoke any operation of this interface at run-time. By using this interface, clients are not restricted to use the services that were defined at the time the client was compiled. The interface description that we have defined allows clients to get the necessary information to invoke dynamically a service.

3.4 Synthesis on the Metadata Model Proposed

By defining the above concepts, a client can query a service based on its static properties, 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) and its interface which permits the service to be invoked dynamically. When combining the description of the static, the dynamic (behavior) properties and the interface of a service, the indexing/search of a service can be addressed completely. The combining of these three levels of service description and the use of a dynamic invocation of services operations, we can compose services at run-time to create novel services which provide novel functionalities.

4 Browsing and accessing a service

Services and objects are stored and indexed in ontologies ; the domain models are also described in ontologies and can be accessed. Services and objects can therefore be searched with a query written in the F-logic language. The search engine will return the matching service offers to the client, which, in turn, can invoke the service once its interface or behavior is obtained, tune the request in order to extend it if no match exists, or scope it if the results of the request are too numerous.

Our main contribution is the association of knowledge representation, distributed systems (with web services and distributed objects), and user needs. The approach we developed present 3 major benefits:

- **A multi dimensional level request.** The search query can reference one or several of the description levels (static description, interface and behavior descriptions).

The query engine goes beyond the well known trading service since it can provide also information on how to invoke the service. The use of the interface automaton makes it possible to query the behavior in a rather simple way.

- **Service composition is made available.** We have a service description which is based on interface automata and combination of automata is a well known domain. Our approach for service composition is also simple and relies on the connexion of corresponding flows. This initial work should be continued in order to integrate other description models (e.g.; WSFL).
- **Towards automatic search and composition.** The tools we have developed target a human use and also a software agent usage. The benefits of discovering and composing for software agents are very high: services can be combined and the resulting services can also be described easily. Their behavior is the combination of the interface automata and parts of their properties can be extracted from its components.

5 Applications

The final paper will present 2 application domains where our approach has been used. The first one relates to distributed objects in a dynamic system. We indicate how our approach has been used to develop a flexible trader which can connect clients and servers in a distributed system.

The second application we have is within a EU sponsored project devoted to e-learning: Kaleidoscope which integrates the activities of around 60 research teams. Our main objective is to make available a set of services and doc-

uments that are to be shared within the community of researchers.

For both applications, we have proposed a complete metadata model which includes specific properties, interface and behavior descriptions.

6 Conclusion and perspectives

In this paper, we have presented our approach for indexing, searching and composing web services and distributed objects. We have combined an integrated approach where 3 distinct description levels provide information on the service (with static properties similar to DAML-S, for distributed objects), interface and behavior.

The user, either human or software, can combine the 3 levels in order to tune his/her request and the underlying nature of the architecture (which rely on ontologies) makes it possible to extend the request or to reduce its scope in a flexible way.

The approach has been applied to two domains: to distributed objects and to e-learning services. We believe that our approach can be adapted to other contexts since ontologies are parameters of our tools. The user should then provide a specific domain ontology to deal with a new context.

We have identified 3 possible future directions of our work:

- The definition and implementation of various facilitators and mediators. The trader is only one of the communication service which can ease the interactions between a client and a server. Additional facilities can be made available in our context and yet provide powerful tools for the users.
- The second extension consists in extending the description in order to integrate

WSFL for the behavior description. We also believe that the transition from interface automata is limited.

- The third perspective relates to the definition of a user model and a description of the environment. For example, the user model would provide valuable information on the user and carry implicit values when searching the service repositories. We believe, that our architecture can be extended to deal with these 2 additional description levels.

References

- [1] O. Kassem Zein and Y. Kermarrec. An approach for describing, discovering services and for adapting them to the needs of users in distributed systems. In K Sycarra and T Payne, editors, *In the proceedings of AAAI Spring Symposium on Semantic Web Services*, Stanford, California, March 2004.
- [2] O. K. Zein and Y. Kermarrec. An Approach for Describing and Querying Service Behavior in Distributed Systems. In *The IEEE international Symposium on Applications and the Internet*, pages 136–139, Trento, Italy, January 2005.
- [3] T. R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 1993.
- [4] M. Huhns and P. Singh. Ontologies For Agents. *IEEE, Internet Computing Journal*, December 1997.
- [5] M. Kifer, G. Lausan, and J. Wu. Logical foundations of object-oriented and frame-based language. *Journal of the ACM*, 42(4) : 741 - 843, 1995.
- [6] O. K. Zein and Y. Kermarrec. An Elaborate and Flexible Trader Based on Ontologies. In *The IFIP WG6.7. Workshop and EUNICE Summer School on Adaptable Networks and Teleservices*, pages 103–108, Trondheim, Norway, September 2002.
- [7] M. Dumas, J. O’Sullivan, M. Heravizadeh, D. Edmond, and A. Hofstede. Towards A Semantic Framework for Service Description. In *Proc. of the 9th Int. Conf on Database Semantics, Hong-Kong*, April 2001.
- [8] M. Merz, M. Witthaut, and S. McConnel. Catalogue and Service Architecture. <http://osm-www.informatik.uni-hamburg.de/osm-www/public/docs.OSM D8>, 1997.
- [9] W. Ng, G. Yan, and E. Lim. Heterogeneous product description in electronic commerce. *ACM SIGeCom Exchanges*, 1(1):7-13, 2000.
- [10] ITU-T Recommendation Z.100. Specification and Description Language (SDL).
- [11] L. De Alfaro and T. A. Henzinger. Interface Automata. In *Proceedings of the Ninth Annual Symposium on Foundations of Software Engineering*. ACM Press, 2001.