SEMANTIC LOOKUP IN SERVICE-ORIENTED ARCHITECTURES

UWE ZDUN

Department of Information Systems, Vienna University of Economics, Austria
E-mail: zdun@acm.org

Lookup of services is an important issues in many distributed systems. This paper deals with lookup in service-oriented architectures, such as Web services, P2P systems, GRIDs, or spontaneous networks. Service-oriented architectures impose specific requirements onto the lookup service, for instance regarding the runtime extensibility of lookup models, runtime extensibility of lookup queries, construction of complex lookup queries, scalability, and fault tolerance. These requirements are not well solved by existing lookup approaches. We propose a semantic lookup service using Semantic Web ontologies, expressed in RDF. Query scripts are sent from the client to the server and are interpreted at server side using the RDF repository. We also present a safe, scalable, and efficient architecture for defining and querying lookup information using this lookup service concept.

1 Introduction

In a many distributed systems, servers offer services to clients. Time and again, services get added or removed. Clients need to know which services are currently offered by which server application, running at which location. To allow for efficient, inexpensive, and timely publication of services, lookup services [1,2] are used by most distributed object middleware systems. The basic lookup architecture is used at many other network layers as well.

In a lookup service, servers can bind a (symbolic) name and/or a set of properties to distributed objects or other distributed resources. The lookup service can lookup these objects by name or by property. Usually it provides the location of the service and other information how to access the service, such as the remote interface or protocols supported. Clients need not know the location of servers but only the location of the lookup service – which can be provided by bootstrapping mechanisms, such as initial references or broadcast messages. All other required location information is retrieved via the lookup service. The most simple kind of lookup services are naming services that just bind a symbolic name to a resource location.

Today, we see many new requirements for lookup services arising that are due to new kinds of applications for these services and new lookup requirements – especially in so-called service-oriented architectures (SOA). For instance, Web service frameworks and other emerging kinds of middleware, such as P2P systems, GRIDs, or spontaneous networks, require highly dynamic lookup repositories with services that can connect, disconnect, move, and change their properties dynamically. In many existing lookup implementations, however, the properties and relationships of elements in the lookup service are too simple or hard to extend.

Sometimes lookup queries need to be construct or modified at runtime as well – for instance, to construct complex lookup queries based on previous lookup results. This can be done by issuing a number of simple queries to a remote query API of the lookup service and performing computations with the query results on client side. However, for complex queries, this requires a number of consecutive queries, wasting network bandwidth and performance. Other challenges for lookup service in SOAs are that the more the architecture depends on lookups the higher the requirements for scalability, resource management, or tolerance of partial failures get.

Traditional lookup services (see Section 2 for a discussion of this related work) need to be extended to support these specific requirements of a SOA. As we point out in Section 2, some of the these requirements are already supported by some of the existing approaches. Yet none of these lookup approaches provides a solution to all the challenges named above.

This paper proposes a semantic lookup service as a solution to these challenges. The basic idea is to make the lookup service itself an extensible Semantic Web repository and make it programmable at runtime – both from client and server side. We offer the Redland RDF store [3] remotely in which we can define resources and resource relationships according to different Semantic Web ontologies, such as RDFS [4], OWL [5], Dublin Core [6], or LOM [7]. In contrast to most existing solutions in the Semantic Web area, we do not use a logic reasoner, but use an imperative, object-oriented scripting language for construction and interpretation of queries. The major goal of this approach is to keep queries simple for programmers who are used to the imperative, object-oriented model of programming. The queries are sent from client to server as scripts that are interpreted on server side. In other words, we apply a simple mobile code approach for query construction. We ensure safety of the interpretation of the mobile code by using a dedicated sandbox interpreter for lookup query interpretation.

In this paper, we first discuss related work to explain the background of this work and problems in other approach regarding SOAs in Section 2. Then we describe the basic architecture of the semantic lookup service concept in Section 3. In Section 4 we first explain the basic lookup and query model, and next we explain how to extend ontologies and queries. Finally, we present a brief case study in Section 5, and then we conclude.

2 Related Work

Naming services are applied at many layers of distributed computing for binding (i.e. associating) names with objects. These objects include hosts, network cards, distributed objects, and many others. For instance, the address resolution protocol (ARP) [8] is used to map IP addresses to physical addresses. At the network protocol layers, TCP/IP, for instance, either uses static, flat host files or the hierarchical Domain Name System (DNS) [9] for mapping internet addresses (e.g. in URLs) to IP addresses. Whereas host files and ARP only provide simple resolutions, DNS is distributed and scalable. But it is not, in general, extensible. Also, its update strategy is designed for rare changes only.

Similar naming services are also provided by distributed object frameworks. For instance, the CORBA Naming Service [10] allows server developers to bind symbolic names with CORBA objects, and clients can lookup the objects by name. In the CORBA Naming Service, each host is given a globally unique ID, and hosts are organized into hierarchical namespaces.

A problem of all kinds of Naming Services is that they cannot store extra information about the objects. For network information, the Network Information Service (NIS) and NIS+ provide centralized control over a variety of network information. NIS and NIS+ store information about workstation names and addresses, users, the network itself, and network services.

Extra information is also provided by various kinds of directory services. The basic tasks of a directory service are grouping of names, adding and removing information, getting information using queries (e.g. using wildcards), and setting access modes (read/write/execute) for directory entries. Example directory services are X.500 and LDAP. LDAP is more simple and more efficient than X.500 directories accessed over DAP, and it can work with more simple directories than X.500. The Java Naming and Directory Interface (JNDI) is a similar Java-only solution.

Extra information about distributed objects can be provided in CORBA using the Property Service [10] (for creation and manipulation of dynamic name/value pairs) or the Trading Service [10]. The Trading Service provides a kind of "yellow pages" style open directory that supports complex lookup queries.

In the area of Web services, UDDI [11] provides a directory for Web services. Besides location and interface information about the Web services, UDDI stores business information about the service and the organization providing it. UDDI has not gained wide acceptance yet, mainly because its design is based on a Web service vision of widely available Internet-based Web services, with searches and frequently changing connections the norm. In practice, however, Web services are used for quite different tasks, such as enterprise application integration, middleware integration, or business-to-business communication. These tasks usually require a more simple and more controlled lookup service that can be tailored to the particular problem domain.

All directory and trading services enrich the information about objects with extra information, yet they are not generally extensible with unanticipated kinds of query processing, more complex information ontologies, runtime service extensions, service scalability, automatic cleanup for unused resources, or fault tolerance.

Different approaches provide some domain-specific extensions to solve these problems for a particular domain or goal. For instance, Tari and Craske extend the CORBA trading service with a query routing mechanism that uses dynamic environment information, such as a hit factor, to leverage the service's scalability [12]. Maffeis extends the CORBA naming service with some measures to tolerate partial failure [13]. In particular, active replication of naming context objects (in object groups) and reliable multicast is used. These and similar approaches extend the service but the extension is hard wired, and cannot be designed by the user. Thus the user cannot make similar extensions to the service.

Lookup in ad hoc or spontaneous networking environments, such as Jini [14] or P2P frameworks like JXTA [15], solve scalability problem by service federation, and they cleanup unused resource with leasing. Peers in the network can announce services and lookup services with meta-data at the lookup service.

An alternative solution to dedicated lookup services is offered by many semantics-based metadata services, such as Triple [16], Ontobroker [17], or Fact [18]. These use logic reasoners to express the ontology and queries. As queries are formulated by clients, the server developer does not have to foresee all possible kinds of queries. However, there are a number of problems with these approaches in practical, distributed software engineering projects. Logical languages must be learned by the software developers. Reasoning can be relatively slow compared to evaluations in imperative mainstream programming languages. Facts, rules, and queries are not separated; thus a malicious client can potentially compromise the fact and rule base; a way to avoid that is to analyze the query semantically – which is difficult and slow. Finally, when a large number of rules are defined, debugging these rules can turn out to be difficult.

In summary, there are many approaches solving particular problems motivated in Section 1. Yet, a simple and efficient lookup service approach for distributed object frameworks – with special focus on the challenges in SOAs – is still missing.

3 A Semantic Lookup Service Infrastructure

Typical implementations of lookup services, such as naming or trading services, can be queried using a name (or a set of properties). Then, the service searches in a table (or an other data structure) for matching elements. As a result of the lookup, the absolute object reference [2] of a remote object is returned, or other location information such as an interface description [2] like a WSDL file [19].

As explained before we want to allow for more advanced lookup functions than this simple lookup functionality as well – examples are ontology-based queries, behavioral extension, query extension, load-balancing, and fault tolerance. This requires a high flexibility regarding inputs, processing, and outputs of the lookup service.

There are two simple alternative solutions. First, we can use a semantics-based metadata services, such as Triple [16], Ontobroker [17], or Fact [18], that provides more powerful metadata definitions and queries. However, as explained above, this approach is not well suited for all application areas, because the solutions might get hard to understand and/or hard to debug. A second alternative is to use an ordinary naming or trading service, and perform the additional computation on client side. The benefit of this solution is that clients can formulate customized queries in an imperative language. This is easy to use for developers used to such languages, but as a liability for complex queries a lot of data or a lot or queries might have to be performed. For example, the complex query "get all resources that are connected using the property 'sub-class-of' transitively and have the property X" would yield multiple atomic queries across the network. This results in a high use of network bandwidth and is relatively slow. In extreme cases the complete ontology and all resources needs to be transmitted to the client side.

As a solution to these problems we combine the basic ideas of these two approaches using mobile code that is sent from the client to the server, and an interpreter on server side for interpreting the lookup queries. In particular, this architecture consists of the following components:

- *RDF Store:* All metadata about the services is stored in an RDF store. RDF [20] supports semantic metadata about Web resources described in some ontology or schema. For instance RDF-Schema [4] and OWL [5] support general relationships about resources, like "subclass of". Developers can also use RDF ontologies from other domains; for instance, in an e-learning system probably an ontology for learning materials will be used, such as LOM [7]. We use the Redland RDF store [3]. It can store the data in an in-memory storage or in an external database.

- *Script Interpreter:* The RDF store is accessed using an XOTcl interpreter. XOTcl [21] is an object-oriented extension of the language Tcl [22]. The interpreter can evaluate imperative scripts on server side. Two interfaces to the RDF store are offered: one can change the data in the store, the other one is used for querying only. As explained below, queries are executed in a safe sub-interpreter that can read the data from the RDF store. Each script interpreter runs in its own thread of control to make the architecture scalable and efficient.

- *Remote Interface:* A remote query interface is offered so that remote clients can send scripts to the lookup service. The lookup service evaluates the scripts using its embedded script interpreter. We use scripts as mobile code instead of offering the XOTcl Redland API remotely, because this way clients can perform complex queries without having to send multiple invocations across the network. Also, this remote script interface makes the queries extensible at runtime using the scripting language.

Figure 1 shows how these components are assembled to a remote service architecture, and how it is used by services and clients for lookups. The XOTcl interpreter is embedded in the service component, and offers two remote interfaces in the Web service framework: one for server applications to announce services in the lookup service, and one for clients to query the service framework. Inside of the lookup service, the Redland RDF store is embedded which stores the actual metadata as RDF graphs. An object-oriented XOTcl interface to the Redland store is provided. Scripts can use this interface for their queries. Potentially, domain-specific applications can add domain-specific interfaces on top of the generic interface to ease application development (see Section 4 below).

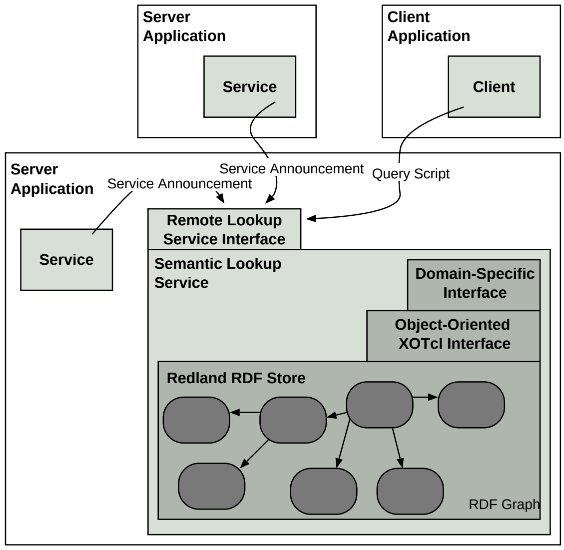

Figure 1. Architecture Overview of the Semantic Lookup Service

As a Tcl extension XOTcl can be embedded in multiple host languages, including Tcl, C, and C++. Thus the service can potentially be used with a number of different Web service frameworks; just the remote interfaces need to be re-written to port the lookup service to other frameworks or languages.

In our lookup model, code is executed within the embedded XOTcl interpreter. Standard Tcl provides operation system calls, and other unsafe commands that might be exploited by a malicious clients. From within the interpreter, it should only be allowed to access to commands offered by the object-oriented interface to the Redland store, as well as basic Tcl/XOTcl commands.

We use the concept of safe interpreters, as introduced by Safe Tcl [23]. Safe interpreters isolate remote invocations and prevent the objects from using any of the unsafe features of the language (such operating systems calls or file access). The safe interpreter only has access to a restricted subset of the unsafe features using aliases and protected commands in the safe interpreter. These are the basic functionalities of the XOTcl/Tcl language and the Redland interfaces. This way the environment is protected from malicious actions in the embedded interpreter.

4 Lookup and Query Models

4.1 Basic Lookup and Query Model: Object-Oriented RDF Interface

Based on the infrastructure, described in Section 3, we can build the lookup and query models. In fact, the basic lookup model is very simple: it provides an object-oriented interface to Redland library (implemented in C) in a scalable and safe fashion remotely.

The basic model can be used for simple RDF graphs with absolute object references and properties and relationships. In the case of ordinary Web services, we simply announce the services with a service name, their WSDL interface description, and other information like the service's URL. More sophisticated descriptions might contain properties provided as domain-specific ontologies. For instance, information in the WSDL file (like operation names, supported protocols, etc.) can be reflected in the RDF graph as well – then this information can be included in queries.

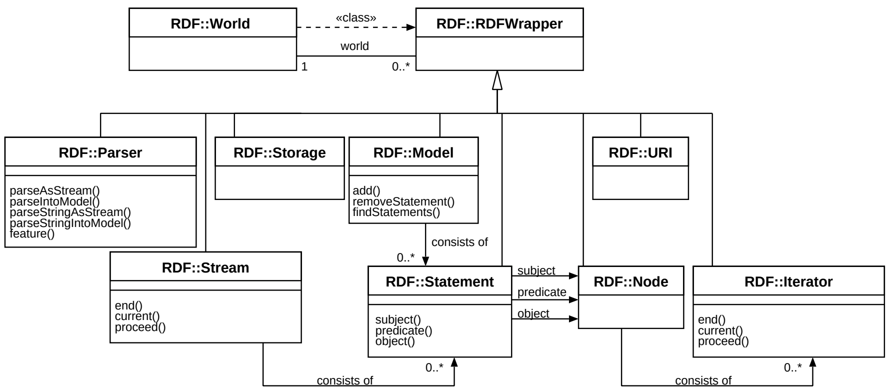

Figure 2. Object-Oriented RDF Store Interface

The class diagram in Figure 2 gives an overview of the object-oriented interface to the Redland RDF store library. All elements in one RDF store are managed by a "world" object. The world can either be filled by hand (programmatically) or by an XML parser. The RDF metadata is stored in an RDF model. The model consists of RDF statements (triples of RDF nodes for subject, predicate, and object). Some of these nodes are URIs, describing Web resources. Different storages exist in which the metadata can be stored, including the memory, files, and relational databases.

There are a number of ways to query the RDF store. Using the model class statements can be searched for some criteria. Such statements can be traversed using the stream class. Individual lists of nodes can be traversed using the iterator class.

Consider a simple example: we want to describe the "creator" of a Web service as a simple literal property. The following script defines the creator using a respective predicate from the dublin core ontology [6]:

```
Statement s1 "http://www.exampleservice.org/aService" \
    "http://purl.org/dc/elements/1.1/creator" \
    "Dave"
Statement s2 "http://www.exampleservice.org/anotherService" \
    "http://purl.org/dc/elements/1.1/creator" \
    "Jim"
Statement s3 "http://www.exampleservice.org/yetAnotherService" \
    "http://purl.org/dc/elements/1.1/creator" \
    "Jim"
rdfModel add s1
rdfModel add s2
rdfModel add s3
```

Figure 3. Example RDF Graph

This script can be sent by the Web service to the script interpreter of the lookup service. Here, three statements, each containing two URIs and a literal node, are created and are added to the default RDF model. The simple example model is shown in Figure 3. We can use this RDF model for queries later on. For instance, the following simple query example, produce a list of all the contents of all statements that have a creator (according to the dublin core ontology) and the value of the creator is "Jim":

```
set results ""
Statement queryStatement
queryStatement predicate "http://purl.org/dc/elements/1.1/creator"
queryStatement object "Jim"
set stream [m findStatements queryStatement]
while {![$stream end]} {
    lappend results [[$stream current] toString]
    $stream proceed
```

```
}
set results
```

If this query is sent by a client to the server with the above RDF model, the result is:

```
{{[http://www.exampleservice.org/anotherService],
  [http://purl.org/dc/elements/1.1/creator], [Jim]}}
{{[http://www.exampleservice.org/yetAnotherService],
  [http://purl.org/dc/elements/1.1/creator], [Jim]}}
```

4.2 Extending the Lookup and Query Model

As mentioned above, the lookup and query model can be extended using domain-specific ontologies. To do so, first, we need to extend the RDF store with statements defining the ontology. There are many predefined RDF ontologies for generic relationships, such as RDFS [4] or OWL [5], as well as ontologies for domain-specific tasks, such as dublin core for a broad range of purposes and business models [6] or LOM for learning resources [7]. Extending the store with ontology definitions is pretty easy, as the RDF definitions exist in the XML syntax of RDF [20] and can be loaded using Redland's parser. Of course, it is also possible to define new ontologies for the purposes of a particular domain. As a simple example, consider the XML definition of a "Class" in RDFS:

```
<rdfs:Class rdf:ID="Class">
  <rdfs:label xml:lang="en">Class</rdfs:label>
  <rdfs:label xml:lang="fr">Classe</rdfs:label>
  <rdfs:comment>The concept of Class</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#Resource"/>
</rdfs:Class>
```

Here, simply the label "Class" is introduced and annotated in different languages and with an explaining comment. The only semantic information is that a class is a special kind of resource.

The more important step for defining ontologies is to define the semantics of the ontology elements – that is, how they are interpreted. Optionally, we can provide a specific query interface for the ontology at the server side. Otherwise the client has to provide the interpretation in the query script. Just consider the dublin core example from the previous section again: here, the client knows what do to do with the information that someone is the creator of a Web service. Thus, if such queries are recurring, we can provide the implementation of query semantics on server side so that the client does not have to provide the query logic with each request, but can use the predefined functions. Such an interface must then be loaded into the server-side interpreter before the query happens.

Of course, this kind of server side extension is only needed for convenience. Any kind of query can also be defined on client side and be sent with each query – thus the query model is fully extensible without the need for changes in the server.

For instance, we could define a convenience method like the following to abstract the query in the example above:

```
Class DublinCoreQuery
DublinCoreQuery instproc findAllServicesbyCreator {creator} {
  set results ""
  Statement queryStatement
  queryStatement predicate "http://purl.org/dc/elements/1.1/creator"
  queryStatement object $creator
  set stream [m findStatements queryStatement]
  while {![$stream end]} {
      lappend results [[$stream current] toString]
      $stream proceed
  }
  return $results
}
```

Now the client can use more simple queries by using this server-provided, domain-specific query interface. For instance, the query for services created by "Jim" now looks as follows:

```
DublinCoreQuery q
q findAllServicesbyCreator Jim
```

5 Case Study: Lookup in Leela Remote Object Federations

Leela [24] is a federated model of remote objects. Within a federation, peers offers Web services (and possibly other kinds of services) to their peers, can connect spontaneously to other peers (and to the federation), and are equal to other peers. Each remote object can potentially be part of more than one federation as a peer, and each peer decides which services it provides to which federation. Certain peers in a federation can be able to access extra services that are not offered to other peers in this federation via its other federations.

In Leela, the semantic lookup service is used to find peers using metadata, exposed by the peers according to some ontology. Thus it enables loosely coupled services and simple self-adaptations for interface or version changes.

Leela has the goal to provide loosely coupled business services. To enable loose coupling we need to retrieve the invocation information dynamically – know at least the object ID, operation name, and operation signature. The lookup service is used to retrieve interface descriptions dynamically with both extensible kinds of metadata and queries.

Each peer provides semantic metadata about itself to its federation's lookup service. Peers can perform lookups in all lookup services of their federations. The federation provides metadata about all its peers, such as a list of absolute object references and object ids (the service names). Each peer adds information for its exported methods, their interfaces, and their activation strategy. Peers of a federation can read from and write to this metadata repository.

In summary, the semantic lookup service plays a central role in the Leela architecture as it enables loose coupling of services, spontaneous connections, and simple federation of services. In this architecture it is important to have a very simple lookup service that is easily extensible with both new kinds of lookup queries and ontologies – to foster independence of the peers. Finally, as many interactions rely on the lookup service, it is important that the service is still scalable and efficient. This combination of properties was not provided by one of the existing lookup approaches (see Section 2) – for this reason we have designed the semantic lookup service described in this paper.

6 Conclusion

In this paper we have described an approach to lookup of services in a service-oriented architecture (SOA) using Semantic Web ontologies. Thus, we are able to describe the services with rich metadata, instead of simple properties only. The metadata is dynamically extensible with domain-specific ontologies and query code at server side. At client side we provide dynamic extensibility of queries. A mobile code approach reduces the use of network bandwidth and thereby enhances efficiency. Nonetheless the approach is safe due the use of the dedicated sandbox interpreter. The interpreter approach is also scalable due to the use of one thread per interpreter. As shown in the Leela architecture, the lookup service itself can be federated as well, to ensure scalability within a larger SOA. Our approach is novel in so far that none of the earlier approaches supports this combination of properties and thereby resolves the challenges motivated in Section 1 fully. As a future work, we plan to support a more powerful query engine on top of Redland, provide more domain-specific ontoloies, and combine the semantic lookup service with other Web service frameworks.

References

1. M. Kircher and P. Jain. Lookup pattern. In *Proceedings of EuroPlop 2000*, Irsee, Germany, July 2000.
2. M. Voelter, M. Kircher, and U. Zdun. Remoting patterns. To be published by J. Wiley and Sons Ltd. in Wiley's pattern series in 2004, 2004.
3. D. Beckett. Redland RDF Application Framework. http://www.redland.opensource.ac.uk/, 2004.
4. D. Brickley and R. V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. http://www.w3.org/TR/2004/REC-rdf-schema-20040210/, 2004.
5. D. L. McGuinness and F. van Harmelen. Web Ontology Language (OWL). http://www.w3.org/TR/2004/REC-owl-features-20040210/, 2004.
6. Dublin Core Metadata Initiative. Dublin core. http://dublincore.org/, 2004.

7. IEEE WG 12. Learning object metadata. http://ltsc.ieee.org/wg12/, 2004.
8. D. C. Plummer. Ethernet address resolution protocol. RFC 826, 1982.
9. P. Mockapetris. Domain names – concepts and facilities. RFC 1034, 1987.
10. Object Management Group. Common request broker architecture (corba). http://www.omg.org/technology/documents/corba_spec_catalog.htm, 2004.
11. OASIS. UDDI. http://www.uddi.org/, 2004.
12. Zahir Tari and Gregory Craske. A query propagation approach to improve CORBA trading service scalability. In *International Conference on Distributed Computing Systems*, pages 504–511, Taiwan, 2000.
13. Silvano Maffeis. A fault-tolerant CORBA name server. In *Symposium on Reliable Distributed Systems*, pages 188–197, Niagara-on-the-Lake, Ontario, Canada, 1996.
14. K. Arnold, A. Wollrath, B. O'Sullivan, R. Scheifler, and J. Wald. *The Jini Specification.* Addison-Wesley, 1999.
15. Project JXTA. JXTA. http://jxta.org, 2004.
16. M. Sintek and S. Decker. Triple–a query, inference, and transformation language for the semantic web. In *Proceedings of the First International Semantic Web Conference (ISWC)*, Sardinia, June 2002.
17. S. Decker, M. Erdmann, D. Fensel, and R. Studer. Ontobroker: Ontology based access to distributed and semi-structured information. In R. Meersman, editor, *Semantic Issues in Multimedia Systems. Proceedings of DS-8*, pages 351–369. Kluwer Academic Publisher, 1999.
18. I. Horrocks. The FaCT System. http://www.cs.man.ac.uk/∼horrocks/FaCT/, 2001.
19. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (WSDL) 1.1. http://www.w3.org/TR/wsdl, 2001.
20. W3C. Resource Description Framework (RDF). http://www.w3.org/RDF/, 2004.
21. G. Neumann and U. Zdun. XOTcl, an object-oriented scripting language. In *Proceedings of Tcl2k: The 7th USENIX Tcl/Tk Conference*, Austin, Texas, USA, February 2000.
22. J. K. Ousterhout. Tcl: An embeddable command language. In *Proc. of the 1990 Winter USENIX Conference*, January 1990.
23. J. K. Ousterhout, J. Y. Levy, and B. B. Welch. The safe-tcl security model. In G. Vigna, editor, *Mobile Agents and Security.* Springer-Verlag, 1998.
24. Uwe Zdun. Loosely coupled web services in remote object federations. In *Proceedings of the Fourth International Conference on Web Engineering (ICWE'04)*, Munich, Germany, July 2004.