Towards Universal Web Service Clients

A. Eberhart
School of Information Technology, International University in Germany
Bruchsal, Germany

Abstract

In this paper we propose the Web Service Description Framework (WSDF), a suite of tools for the semantic annotation and invocation of side effect free Web Services. WSDF bases on established standards from both the Web Services and the Semantic Web communities. We developed a software package on top of Apache AXIS and a relational database server that allows an application to invoke a service without any prior knowledge. The sole prerequisite is a shared ontology defining major domain concepts.

1. INTRODUCTION

After the web's initial phase of being a medium for convenient reading and publishing static information, the popularity of web applications has grown enormously. Today, there hardly is a service or a good that is not available online. Nevertheless, almost all of these services are geared towards human interaction. The electronic data interchange (EDI) community had quite some success in standardizing message formats for application integration; it is however impossible to develop a lightweight standard that serves a variety of application domains. Therefore, EDI solutions are typically very specific to a certain industry.

In 1998, the XML specification laid the foundation for more large-scale solutions by defining a generic syntax for semi-structured data. Even though XML is a very low level specification, the wide support in all kinds of software solutions allows for much simpler and faster development of B2B or EAI software. Currently, existing EDI standards are being mapped to the XML world. EbXML is one of the more prominent examples of this trend. Microsoft's BizTalk server also incorporates this approach. It acts as a message exchange hub that is able to map different message formats into others. Internally, all incoming files are converted to XML in order to be able to convert them using XSLT.

Apart from XML as being a data representation format, Web Services are the other major application area. Remote procedure calls basing on XML data encoding and standard Internet transport protocols are believed to be the silver bullet for lifting the web to the next level: The programmatic exchange of information between technically and organizationally completely heterogeneous systems. This is a point where traditional middleware like DCOM and CORBA failed already at a technical level. Incompatible solutions between different vendors and a lack of universally available infrastructure like naming and security services and protocols, made a wide-scale adoption impossible. Therefore, these middleware solutions are typically found in implementations of a distributed system, which is deployed under the supervision of a single team of developers. This situation was disadvantageous not only for customers, but also for core technology providers since not too much money can be made in the middleware market. This is one of the main reasons for the previously unseen cooperation on XML middleware interoperability. The goal is to compete on the enterprise application level instead.

The Web Services standards stack of SOAP, WSDL, and UDDI allows for business partners' services being located and invoked at runtime. A supply chain management system, for instance, can easily register a new trading partner in a local UDDI repository. From there, the overall system can obtain the service location and query prices or the availability of items in stock. The seamless integration of Web Service standards into development tools, most notably Apache AXIS and Microsoft Visual Studio .NET, also allow for rapid product development.

Despite the recent advances on technical interoperability, it is still a dream to invoke remote services in a completely automated fashion. In the supply chain management example above, there needs to be some sort of standardization body that specifies the interfaces for querying prices and availability. These standards will

then be represented by UDDI tModels. A new supplier's services can only be invoked, if they follow the overall system's tModel specification. Commonly, this specification is given as a WSDL interface description. The other common usage pattern is to use the UDDI registry as a design time service repository. A developer can manually search for services, read the documentation, generate proxies, and write code to invoke the services. Neither of the two approaches allows for a completely automatic service invocation without a priori knowledge about a certain tModel or a human writing custom code for unknown tModels.

Obviously what is needed, is a more detailed description of services that goes beyond the simple method signatures that can be found in WSDL specifications. It is necessary to capture what a service does on a conceptual level. UDDI addresses this issue to some extent by allowing to classify services and companies according to standard industry, service, and geographic taxonomies such as UNSPEC. In this paper we extend this idea by applying some of the concepts and ideas from the Semantic Web community. Standard mark-up languages like the Resource Description Framework Schema (RDFS) or the DARPA Agent Mark-up Language (DAML) allow the specification of ontologies, which can be viewed as a powerful extension to taxonomies. We show how in a first step, ontological terms can be used to provide a detailed semantic description of service parameters and return types and how this information can be used to determine ad-hoc Web Service calling sequences.

The rest of the paper is organized as follows: The next section provides a brief overview of semantic web technology. Sections three and four cover our Web Service Description Framework and its actual implementation using the OntoSQL toolkit and Axis. The final two sections wrap up the paper with remarks about future work and a summary.

2. BACKGROUND

Recently, the Semantic Web initiative has gotten a lot of attention in the research world. This section briefly introduces and explains the necessary standards and concepts that our research bases on.

2.1 A whirlwind tour of RDF

The Resource Description Framework (RDF) is a framework for metadata and the most basic mark-up language in the context of the Semantic Web. The core idea is that any concept, no matter how abstract, can be treated as a "resource", and hence identified by a URI. A person could be denoted by her or his homepage. When talking about Ora Lassila, we might use the resource <http://www.lassila.org>; a desk in some office might be referenced via the company's inventory list as <http://xyz.com/inventory#K4622-ERF>. In the RDF terminology, these things are called resources. It is then possible to make statements about the resources. If Joe is Peter's brother, we could state this as the following subject, predicate, object triple:

Subject: <http://www.mit.edu/~joe/>

Predicate: <http://www.cogsci.princeton.edu/~wn/concept#isBrotherOf>

Object: <http://www.mit.edu/~peter/>

In this case, subject and object are resources. Therefore, we end up with a directed labeled graph with resources as the graph nodes. Note that the `isBrotherOf` predicate is a URI pointing to a version of the Princeton Wordnet lexical database project. In Wordnet, a concept brother can be found. Note that other meanings of brother, such as monk, can be specified by different URIs. Consequently, the pre- and postfix denotes the relationship of being someone's blood brother. Even though this is somewhat clumsy, there is a specific reason for choosing Wordnet as the predicate namespace. Due to the wide acceptance and popularity of Wordnet, we can assume that our statement can be correctly interpreted by many agents. Consider another example, the statement that Joe lives in Boston:

Subject: <http://www.mit.edu/~joe/>

Predicate: <http://www.schema.org/rdf/livesIn>

Object: Boston

The first difference is that the predicate comes from another namespace. Secondly, the object is a simple string or literal, rather than another resource. If another statement also uses the string "Boston" as its object, it would be up to the application to decide, if the city of Boston, or maybe a project with codename Boston is meant. For further aspects of RDF / RDF Schema (<http://www.w3.org/RDF>) and information on DAML

(<http://www.daml.org>), RuleML (<http://www.dfki.de/ruleml>), and the Semantic Web in general (<http://www.semanticweb.org>) can be found on these websites.

RDF is serialized using XML syntax. Like any XML document, RDF might only be a stream of bytes traveling from one application to another via the network. RDF can also be stored in static files, separately or, if statements about an HTML website are being made, within the head tag of the website.

2.2 RDF Schema

RDF Schema (RDFS) takes the logical next step and provides a type system for RDF graphs. It allows to define a class inheritance hierarchy, with multiple inheritance being explicitly supported. Following the principle that everything can be represented by a URI, they also identify classes. The subject, predicate, object triple notation is also used in the RDFS context. The following statements, for instance, declare 1) the class student as such, 2) the class student as a subclass of person and 3) identify Joe as an instance of the class student:

Subject: <http://www.mit.edu/types#Student>
 Predicate: <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
 Object: <http://www.w3.org/2000/01/rdf-schema#Class>

Subject: <http://www.mit.edu/types#Student>
 Predicate: <http://www.w3.org/2000/01/rdf-schema#subClassOf>
 Object: <http://www.schema.org/rdfs/types#Person>

Subject: <http://www.mit.edu/~joe/>
 Predicate: <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
 Object: <http://www.mit.edu/types#Student>

The strength of this mechanism is obviously the ability to cross-reference other schemas. In the example, a more specific schema used by MIT bases on a general schema from a higher-level entity.

Besides class and instance definitions, RDFS also allows for the definition of predicates. The example above uses predicates from the given W3C namespaces. The examples of the RDF section above, however, referenced predicates like `livesIn`, which are undefined so far. In RDFS, the signature of predicates is specified using the domain and range keywords, which are of course again predicates defined in the W3C namespace. For the examples above, the signatures could be `livesIn(Person, Literal)` and `brotherOf(MalePerson, Person)`. Note that `literal` denotes any kind of string, i.e. no URI.

2.3 Ontology

The term ontology is frequently used in the Semantic Web community. It has been redefined as "a formal specification of a conceptualization" [1]. Ontologies are believed to be the silver bullet to achieve true interoperability among intelligent agents [2]. The most prominent ontology mark-up language is the DARPA Agent Mark-up Language (DAML). DAML bases on description logics, however its core is similar to RDFS. Therefore, as other researches have done before [3], we base our ontology description on RDFS. From the previous section's introduction, it becomes clear that an ontology has similarities to database schemas or UML class definitions [4, 5]. The main point, however, is not to simply defining classes or data types, but to represent a common understanding of things that exist, and the relationships between them. In ontology engineering, strong emphasis is put on the ties to natural language, rather than data normalization for instance.

Outside of AI and Semantic Web communities, ontologies are often viewed as purely academic and not really relevant to problems such as enterprise application integration. It is important to note that ontologies do have a large overlap with the various standardization efforts such as RosettaNet or ebXML. Both need to define a standardized vocabulary for a domain. For instance one can leverage the broad shared understanding of the RosettaNet concept of a buyer and define one's own ontology class by extending the resource <http://rosettanet.org/roles/Buyer>.

2.4 Rule Mark-up Language

The rule or logic layer sits on top of the RDF data and the RDFS / DAML ontology layers. Even though no W3C

recommendation specifically deals with these aspects, we believe it is crucial for making Semantic Web technology useful. It is the rules that encode common knowledge about an ontology's concepts and actually allow for computations to happen. A simple example would be a rule stating that people living in an US city also live in the US itself. For humans, this is a trivial conclusion to make, however, it is not trivial at all for a computer program. The simple rule `livesIn(?x ?z) <- livesIn(?x, ?y) and partOf(?y, ?z)`.

One of the most promising candidates for rule mark-up standards is the RuleML [6] language, developed at the German institute for artificial intelligence. We are currently working on syntax for cleanly linking the underlying terms of the ontology to the rules. Currently, this is done by simply reusing the RDFS predicates also as rule predicates.

The next section will introduce a simple ontology for Web Services and describe how additionally tagging a method with some rules allows for the automatic invocation and composition of simple Web Services.

3. WEB SERVICE DESCRIPTION FRAMEWORK

The idea of semantically describing Web Services is not new. Within the DAML project, the DAML-Service (DAML-S) group defined an Ontology for describing complex Web Services as well as business processes. Their work provides a vocabulary for marking up services [7]. In Europe, a large research group has started working on the Web Service Modeling Framework (WSMF) [8]. It proposes a complex mediator architecture for brokering between different data models and service invocation styles. Both projects propose good concepts and ideas, however, due to the extremely complex nature of the problem and the wide scope of both projects, the results and suggestions are currently mostly limited to how Services should be marked-up. The actual service consumption is often left out. McIlraith et. al. provide an approach where user profiles can be fed into a ConGolog program which locates and executes services [9]. Another paper providing very concrete and helpful ideas on information retrieval was written by Denker et. al. [10], however, the service aspect is comparatively vague.

With the Web Service Description Framework (WSDF) we take a different approach by reducing the scope to calling function-like services without side effects and omitting the problem of business process integration with its workflow aspects. Our philosophy is to provide proof of concept that is applicable today on a smaller scope.

3.1 Semantics of Parameters and Return Types

Consider the following example: Let's assume we need to find out the courses a student is currently enrolled in. The following service is applicable:

```
public String[] getCurrentCourses(String studentID)
```

This illustrates the first challenge. A WSDL specification only provides us with the raw data types, string and string array in this case. The fact that the parameter actually denotes a student ID that is given out by the university's registrar and that course IDs identify the courses taught at the university is completely beyond the scope of WSDL. Solely the information that both kinds of IDs happen to have the same data type string is given. Of course one could wrap the simple types in custom data types such as `StudentIDType`. Nevertheless, a client is still left guessing and needs to linguistically analyze the type's name. The WSDL message types look like this:

```
<message name="getCurrentCoursesRequest">
  <part name="studentID" type="xsd:string">
</message>
```

At this point, we use an ontology about universities to supplement the WSDL specification with conceptual information. The ontology will provide `courseID` and `studentID` properties linking the instances to the literal values. The WSDF then simply adds another type information to each message part. In the example above, `studentID` would not only be a string, but also a literal, which is connected to the ontology's student class.

3.2 Semantics of the Method

Let's assume the require student ID is not known. Instead, we have the user's first name, last name, and

birthday. Before the `getCurrentCourses` method can be invoked, the following service must be called in order to obtain the student ID:

```
public String getStudentID(String fn, String ln, Date bd)
```

The same argument about the parameters and the return type applies here. However, we want to point out another issue. Obviously the parameters supplied must belong to the same person, as well as the student ID which is returned, is the student ID of the same person. Again, this is quite intuitive for a programmer reading the interface, but quite hard for a program to figure out. In WSDF we use rules to describe this behaviour. We define an ontological classes `WebService` and `WebServiceCall`. A `WebServiceCall` has parameters and results, the `WebService` class carries relevant technical information about an attached stub for instance. The following rule captures the method's semantics. Note that the use of the variable `S` on both sides of the rule ensures that the parameters and results belong to the same student.

```
studentID(?S, ?I) <- returnValueOf(?I, ?WSC) and
    hasParameter(?WSC, ?FN, ?LN, ?BD) and
    firstName(?S, ?FN) and
    lastName(?S, ?LN) and
    birthday(?S, ?BD)
```

The rule already describes how returned values should be interpreted. In a logical sense, the new fact `studentID(S, I)` is asserted upon `I` being returned. The next section will provide insight into what is actually happening in a running system in this case.

In principle, a service can be invoked once all parameters are available. The following second rule specifies, that the Web Service instance `getStudentID` can be called once the three arguments from the same student are available.

```
callable(?getStudentID) <- firstName(?S, ?FN) and
    lastName(?S, ?LN) and
    birthday(?S, ?BD)
```

3.3 When to invoke a Service?

Of course not any service that theoretically could be invoked should actually be invoked. The client application can use the following two approaches. In the goal-driven approach, we use backtracking to determine a calling sequence, which will answer the goal query. In the previous example, the goal could be to find out whether Joe is enrolled in IT200. This can be answered by calling the `getCurrentCourses` service. Having established this, the new sub goal becomes finding out a student ID. This process is repeated until a subsequent sub goal is fulfilled by the current knowledge state.

The second approach is more heuristic in nature. Here, a service is called if it looks promising. In the example, the rationale might be to try to find out as much information about the user as possible. Therefore, both services would also be called.

3.4 WSDF Syntax

The WSDF syntax complements existing WSDL information about a service. This implies that the Web Service can of course still be invoked in the traditional way. The WSDF file contains several links to the parameter and method descriptions. The rules are given in RuleML syntax.

```
<?xml version="1.0"?>
<wsdf xmlns="http://www.i-u.de/schools/eberhart/wsdf"
    xmlns:ruleml="http://www.dfki.de/ruleml"
    targetWSDL="http://www.mit.edu/services/CourseInfo.wsdl">
  <type part="studentID" message="getCurrentCoursesRequest"
    ontology="http://www.mit.edu/properties#studentID" />
  ...
  <method name="getCurrentCourses">
```

```

<semantics>
  <ruleml:imp>
    <ruleml:head>
      <ruleml:atom>
        <ruleml:rel>
          http://www.mit.edu/properties#studentID
        </ruleml:rel>
        <ruleml:var>S</ruleml:var>
        <ruleml:var>I</ruleml:var>
      </ruleml:atom>
    </ruleml:head>
    <ruleml:body>
      <ruleml:and>
        <ruleml:atom>
          <ruleml:rel>
            http://www.i-u.de/schools/eberhart/wsdf#returnValueOf
          </ruleml:rel>
          <ruleml:var>S</ruleml:var>
          <ruleml:var>I</ruleml:var>
        </ruleml:atom>
        ...
      </ruleml:and>
    </ruleml:body>
    ...
  </ruleml:imp>
</semantics>
...
</method>
</wsdf>

```

Note that the rules consist of predicates from both the target ontology that is used to describe the services as well as the WSDF mini-ontology on Web Services, return types, etc.

3.5 Java2wsdf and prolog2ruleml

Like any XML dialect, WSDF and RuleML are not meant to be edited manually. Following java2wsdl's example of generating a service's WSDL description directly from the public methods, we developed the java2wsdf tool. It constructs the WSDF markup from information the programmer places in the javadoc formatted comments preceding a method which will be exposed as a Web Service:

```

/**
 * Returns the studentID of the student with a given
 * name, firstname, and birthday
 *
 * @param fn The student's first name
 * wsdf:ontotype="http://www.mit.edu/types#firstname"
 *
 * ...
 * @return The list of courses takes by the student
 * wsdf:ontotype="http://www.mit.edu/types#studentID"
 * @rules callable(?getStudentID) <- firstName(?S, ?FN) and
 * lastName(?S, ?LN) and
 * birthday(?S, ? BD)
 *
 * ...
 */
public String studentID getStudentID(String fn, String ln, Date bd)

```

Since the naming conventions of java2wsdl are known, the conversion is fairly straightforward. The implementation bases on a doclet which can read the values of the existing tags in order to extract the ontological tagging as well as the custom tags like @rules. The rules themselves are denoted in a prolog-like

syntax which allows :- or <- as the rule symbol and interprets both commas and the "and" keyword as logical conjunctions. The actual prolog2ruleml (<http://www.i-u.de/schools/eberhart/prolog2ruleml/>) parser which is then called from java2wsdf was constructed with the JavaCC compiler compiler.

4. ONTOSQL

In the previous section, we described how WSDL information is enhanced by additional ontological typing information as well as rules, encoded in standard mark-up, that capture the service semantics. We will now demonstrate, how this information can be converted to code in a similar way as WSDL is converted to a client stub.

We use our OntoSQL [11] framework (<http://www.i-u.de/schools/eberhart/ontosql/>), a cross compiler that maps RDFS and RuleML into SQL views and triggers and effectively allows us to use a relational database service like SQL Server 2000 as a deductive database engine.

4.1 Storing the Facts

We chose a very straightforward approach for storing the RDF graph data. A table is created for every predicate or graph label found. OntoSQL can read this information directly from the RDF Schema file specifying the ontology to be mapped.

```
create table <PredicateName>Fact
(
  subject varchar(255),
  object varchar(255),
  primary key(subject, object)
)
```

A tuple (S,O) in the table PFact then represents the RDF triple (S,P,O). Note that the choice of the composite primary key avoids duplicate entries in the respective predicate tables. Every subject and object refers to a resource. The resource's type is stored in the typeFact table, which is implicitly created. We do not require referential integrity of subjects and objects to subjects in the typeFact table since a resource with no type information is treated as the most general type RDF-Resource.

4.2 Rules

Using relational databases as deduction engines is a well-established practice in the database community. OntoSQL implements the ideas presented in [12]. Every predicate is converted into an SQL view as follows. Assume rule-1 through rule-n contain the same predicate in their rule heads:

```
create view [PredicateName] as
  select subject, '[PredicateName]', object
    from [PredicateName]Fact
  union
  select clause for rule-1
  union
  ...
  union
  select clause for rule-n
```

The first query selects the known base facts from the respective table. These definitely must be included in the result. The remaining components of the overall union query can be obtained from the derivation rules. The individual rules are translated to SQL queries as outlined in the following example. Consider the rule $a(?X, ?Y) \leftarrow b(?X, ?Y) \text{ and } c(?Y, \text{"const"})$. The conjunction is translated into the equijoin of b and c on the variable Y . In turn, the columns to be selected are determined by the position of the variables in the rule head and body. Further conditions such as c 's object having to be equal to "const" appear as additional conditions in the query's where clause:

```
select b.subject, '[PredicateName]', b.object
```

```

from b, c
where b.object = c.subject and
      c.object = "const"

```

Note that *b* and *c* are again views, not the fact tables. This causes them to be executed if the select statement above is evaluated. Obviously recursive rule definitions will clash with a relational database's evaluation algorithm and an error will be raised when the view is to be created. OntoSQL solves this problem by simply computing the union of a series of self joins on the table. This works fine if the recursion depth is smaller than the maximum number of times the table is self-joined.

5. SYSTEM ARCHITECTURE AND INVOCATION SEQUENCE

We applied our WSDF tools in the context of a campus information system called SmartGuide [13]. Every participant is running a personal information agent, potentially on a mobile device, which displays information customized towards to user's preferences and current situation. The agent is pulling information from local programs such as the calendar application, but also accesses external sources such as weather data, train schedules, and the services offered by the university's information system.

5.1 The traditional approach

Even though the personal agents are rule based, i.e. not the traditional type of web service clients, the service invocation had to be performed much like in a client written in an imperative programming language. A procedural attachment in a rule head causes a regular method to be invoked, which then carries out the remote procedure call. The point is the following: Using a UDDI repository, it is definitely possible to perform multiple calls to different services implementing the same tModel or interface, however, any new service type introduced into the system requires a new procedure with its client code to be written by hand. A programmer needs to manually inspect the service API and a textual description before the code can be written. Since the number of services available today is fairly small and the few services tend to be easy to use, this is neither a very hard nor a very time consuming task. However, this approach does not scale if more and more services are becoming available, since the agents would constantly have to be changed in order to take advantage of these additions.

5.2 The WSDF-enabled application

Figure 1 shows the overall architecture of the new WSDF-enabled system. At runtime, the WSDL and WSDF descriptions are downloaded from the service providers. The necessary base is provided by the fact, that both the client application and the providers share the same RDFS ontology. The OntoSQL toolkit and the WSDL compiler will then generate the SQL views and the Web Service stubs. The figure shows an example where client and server are both written in Java. Note that WSDF remains language-independent markup. Tools for other languages can easily be developed.

FIGURE 1: The overall system architecture using Java-based tools.

The actual invocation of the service works as follows. First, the application needs to determine which service to call. Using the SQL statement `select * from callable`, a list of Web Services where the required parameters are available is returned from the SQL view "callable". Heuristics or a backtracking algorithm to select the next service can use the WSDL type information. The input parameters are obtained from the database and the call is made. At the same time, a new `WebServiceCall` instance and the respective parameters used are inserted into the database. This is an important step for correctly interpreting the result. Once it arrives, a tuple is inserted which associates the result with the call. This will then cause the service semantic description rule's body to be true. At this point the rule head's values can explicitly be asserted and the Web Service call's information be archived or deleted from the active deductive state.

5.3 Ontology is the glue

The major step forward from the traditional to the WSDL approach is that no modification is necessary on the client side. The service implementer gets more responsibility and needs to formally specify a service using rules and ontological terms. Assuming the client shares the ontology, the rules are automatically transferred and installed on the client, enabling the ad-hoc service invocation.

6. FUTURE WORK AND SUMMARY

This paper describes our initial efforts on the Web Service Description Framework. We currently only cover the simplest case of calling methods that do not have side effects. The obvious next step is to try to extend WSDL to more complicated services. We need to explore if for those cases it is sufficient to describe the services using rules.

We think a promising line of research would be to try to extend the framework from logic-based clients to clients written in imperative languages. Assuming that core data structures are semantically tagged, it should be possible to automatically generate code for a bridge which mediates between the different representations before and after invoking the service.

Furthermore, the relationship of our approach to workflow and process description languages like XLANG or WSFL and standardization efforts such as RosettaNet or ebXML is of interest. Finally, the syntax needs to be revisited. Since the languages used for WSDL, namely RuleML, WSDL, and RDFS, have very different notation styles, the current syntax can definitely be improved with respect to its clarity.

We presented our approach for the Web Service Description Framework, a semantic annotation of Web Services basing on the existing standards and languages WSDL, RDFS, and RuleML. Using a WSDL compiler and our OntoSQL RuleML to SQL converter, a client application is able to process the generic descriptions into running code. Web Services can therefore be invoked simply by agreeing on an RDFS ontology a priori. Even though we restrict ourselves to the simplest case of side effect free methods, this is a major step forward, since traditionally programmers must either be aware of a certain UDDI tModel or simply read a textual description and program corresponding client code. Compared to other approaches like WSMF or DAML-S, we provide a complete solution that not only specifies service mark-up, but also specifies how the client works.

REFERENCES

- [1] T. R. Gruber (1993) Towards Principles for the Design of Ontologies Used for Knowledge Sharing, In N. Guarino and R. Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Dordrecht, The Netherlands, 1993. Kluwer Academic Publishers.
- [2] D. Fensel (2001) *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce*, Springer-Verlag.
- [3] S. Staab, M. Erdmann, A. Maedche, and S. Decker (2000) An extensible approach for modeling ontologies in RDF(S), In *ECDL Workshop on the Semantic Web*.
- [4] A. Eberhart (2001) *OntoJava - applying mainstream technology to the Semantic Web*. International Conference on Electronic Commerce, Vienna, Austria, Workshop on Semantic Web-based E-Commerce and Rules Markup Languages.
- [5] S. Cranefield (2001) *UML and the Semantic Web*, In *Proceedings of the International Semantic Web Working Symposium (SWWS)*.
- [6] H. Boley, S. Tabet, and G. Wagner (2001) Design rationale of RuleML: A markup language for Semantic Web rules. *Proceedings of the Semantic Web Working Symposium (SWWS)*.
- [7] Anupriya Ankolekar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, Drew McDermott, David Martin, Sheila A. McIlraith, Srinu Narayanan, Massimo Paolucci, Terry Payne, Katia Sycara (2002) DAML-S: Web Service Description for the Semantic Web. in: Ian Horrocks & James Hendler (eds.): *The Semantic Web - ISWC 2002*, Lecture Notes in Computer Science 2342, pp.348-363, Springer Verlag
- [8] D. Fensel and C. Bussler (2002) WSMF in a nutshell, technical report.
- [9] S. McIlraith, T. Son, and H. Zeng (2001) Semantic Web Services. *IEEE Intelligent Systems*, 16(2).
- [10] G. Denker, J. R. Hobbs, D. Martin, S. Narayanan, and R. J. Waldinger (2001) Accessing information and services on the DAML-enabled web. In *Proc. Second Int'l Workshop Semantic Web (SemWeb'2001)*
- [11] A. Eberhart (2002) Automatic generation of Java/SQL based inference engines from RDF Schema and RuleML. in: Ian Horrocks & James Hendler (eds.): *The Semantic Web - ISWC 2002*, Lecture Notes in Computer Science 2342, pp.348-363, Springer Verlag
- [12] R. A. Elmasri and S. B. Navathe (1992) *Fundamentals of Database Systems*, chapter 24, pages 729-760. Addison-Wesley, second edition.
- [13] A. Eberhart (2002) SmartGuide: An intelligent information system basing on semantic web standards. In *Proc. of the International Conference on Artificial Intelligence - ICAI 2002*

© A. Eberhart 2002