Web Services based on PROLOG and XML

Bernd D. Heumesser[1], Andreas Ludwig[1], and Dietmar Seipel[2]

[1] University of Tübingen, Wilhelm Schickard Institute for Computer Science
Sand 13, D – 72076 Tübingen, Germany
{heumesser,ludwig}@informatik.uni-tuebingen.de

[2] University of Würzburg, Department of Computer Science
Am Hubland, D – 97074 Würzburg, Germany
seipel@informatik.uni-wuerzburg.de

Abstract. This paper describes how the deductive power of PROLOG can be made available across the Internet using standardized Web service technologies. This facilitates the use of PROLOG as a component of distributed information systems and in many new application scenarios. Since a lot of information available on the Internet is nowadays XML based and since Web service technologies use XML based encodings, it is both necessary and useful to be able to process XML documents in PROLOG itself. To make this possible, a new package for SWI–PROLOG called X2P is introduced, making available to PROLOG many of the XML processing facilities of the Libxml2 library, which is a very up–to–date and efficient implementation of most of the current XML related standards.

1 Motivation and Overview

The World Wide Web as we know it today, was initially designed as a platform for information sharing. The core Web technologies, i.e. HTTP, HTML, Web servers and Web browsers, enable the exchange of information in the form of documents.

Web browsers soon became standard tools and it was realized that they can also be used as universal clients to information systems, if these information systems expose their user interface using HTML documents. Most modern information systems use such a Web based architecture. Tools like application servers simplify the development of Web based information systems at the server side.

The development and deployment of distributed information systems, which integrate applications that are distributed on an Intranet or even on the Internet, is still a lot more difficult. The need for such an (enterprise) application integration over the Web was initially generated mainly by business to business (B2B) applications. However, there are a lot of interesting application scenarios aside from B2B.

While the traditional Web is concerned with the interaction between applications and humans, Web services technologies and standards aim at taking the Web one step further by enabling interaction between applications and thereby facilitating application integration.

Web services are still at an early stage of development, but they certainly have the potential to solve many interoperability problems and are likely to play an important role as a means of implementing distributed information systems in the near future.

We believe that making applications based on the deductive power of PROLOG available across the Internet is very useful and promising, because PROLOG is a very widely used logic programming language, which at the same time enables rule–based programming and rapid prototyping of applications. By adopting the new Web services technologies and standards to facilitate the use of PROLOG based applications across the Internet, we enable the widest range of platforms to integrate such applications.

All Web services standards use XML based document encodings and not all of those documents are transparently handled by the Web services toolkits or middleware. Furthermore, a lot of information is made available on the Internet in the form of XML documents. In the context of Web services based on PROLOG, this illustrates that it is both necessary and useful to be able to process XML documents in PROLOG programs.

The rest of the paper is structured as follows: Section 2 describes how XML document processing can be handled in PROLOG using a new package for SWI–PROLOG called X2P, which is compared with another standard package. Web services, their underlying technologies, standards and tools are the topic of Section 3, while Section 4 shows how in practice all those tools and libraries together can be used to make available Web service based on PROLOG. Following this, the next section discusses some application scenarios, and the paper concludes with a summary and an outlook on future developments in Section 6.

2 XML Document Processing in PROLOG

XML ([16]) provides a standard way to define the structure of documents that is suitable for automatic processing. This enables the development of generic tools that parse documents and extract their content as well as their structure. Restrictions on the structure of a document can be specified by Document Type Definitions (DTDs) or XML SCHEMAs (however, neither of these provide any semantic information). XML has been widely adopted as the foundation for data representation and formats on the Web. Many parsers and toolkits exist for different programming environments, which implement the XML related standards.

2.1 XML and PROLOG

Since every element in an XML document (except the root element) is nested into another element, we can consider XML documents term structures, which can be handled nicely by PROLOG, which can then be used to process those terms in a very compact and efficient way.

SWI–PROLOG ([15]) offers a package called `Sgml2pl` which contains a SGML/-XML parser. This parser can parse a document from a file and tranform the content into a PROLOG data structure. The data structure used is a nested term of the functor `element` with three arguments: the name of an XML element, the list of its attributes and the content of this element. The parser also uses some other kinds of functors to represent other constructs in an XML document (e.g. entities, a DTD or processing instructions). The parsing process can optionally be controlled, e.g. to influence the treatment of spaces.

Libxml2 ([6]) is the XML parser and toolkit that was developed for the Gnome project ([4]), but it can also be used standalone and outside of the Gnome platform. This library of C functions implements XML parsers and toolkits for a number of existing standards related to XML, e.g. XPATH ([17]). Libxml2 contains functions to parse XML documents supporting validation against a (internal or external) DTD or an XML SCHEMA. It can also handle namespaces and different XML document encodings. The internal document representation follows the DOM interface. The library can also be used to evaluate XPATH expressions.

2.2 The X2P Package

SWI–PROLOG offers a *foreign language interface*, which can be used to combine code written in a foreign programming language like C with PROLOG programs. This interface can be used in two different ways: It provides data types and functions to implement PROLOG predicates in a foreign language, e.g. in C and hence may use other C functions and libraries. The C code is compiled to a shared object, and the predicates must be properly registered with SWI–PROLOG as so–called foreign predicates. When an foreign predicate is used, SWI–PROLOG calls the respective function from the shared object. On the other hand, the foreign language interface can be used to embed the PROLOG engine into a foreign language program. For instance, PROLOG goals can be called and evaluated from a C program.

We use the foreign language interface of SWI–PROLOG in both ways. First, to make some of the functionalities of Libxml2 available to SWI–PROLOG for efficient XML document processing. Secondly, to provide PROLOG based applications as Web services (see Section 4).

We have developed a package called X2P, which makes available through the foreign language interface some of the functionalities of Libxml2 within PROLOG. X2P is therefore implemented in C and PROLOG and consists of several foreign predicates that encapsulate Libxml2 functions and convert arguments from PROLOG to C and vice versa.

X2P offers predicates for reading, parsing and validating XML documents and for making their content available in PROLOG. There are options available to control some details of this processing, such as the handling of whitespaces, namespaces and entities. Furthermore, X2P allows for the evaluation of an XPATH expression on an XML document.

There are some differences between the X2P package and the SGML parser from SWI's Sgml2pl package that stem from the different concepts and history of the two packages. X2P offers the broader range of functionalities for XML documents and the more up–to–date support and implementation of XML related standards. Only with X2P documents can be validated with an XML SCHEMA or XPATH expressions can be evaluated. X2P can be easily extended to incorporate a current version of Libxml2 offering new functionalities. X2P offers many more detailed options, influencing the handling of entities, namespaces, comments or processing instructions. However, X2P is limited to XML documents, whereas Sgml2pl can parse all SGML based documents including HTML documents.

Document	SIGMOD	SIGMOD (Old version)	DBLP
Size	478 *KB*	704 *KB*	184 *MB*
Parsing time X2P	0.34 *s*	0.62 *s*	233.17 *s*
Parsing time `Sgml2pl`	0.32 *s*	0.67 *s*	NA

Fig. 1. Comparison of documents and parsing times

In practice it turned out, that in terms of parsing times `Sgml2pl` and X2P are comparable. However, for large documents X2P seems to be far more efficient: Even with the biggest possible trail stack for SWI (about 1 GB) `Sgml2pl` could not parse the XML document containing the data from the Digital Bibliography & Library Project (DBLP[1]), while X2P could in acceptable time. See Figure 1 for a more detailed comparison of runtimes for parsing the XML editions of DBLP and SIGMOD Record[2].

2.3 Field Notation and FNPATH

Instead of using term structures as the result of parsing an XML document, we map them to the so–called *field notation* (cf. [9]), which we use as the Document Object Model (DOM) for PROLOG. We represent an XML element, which can have attributes and nested elements, by a triple $\mathsf{T} : \mathsf{A} : \mathsf{C}$, where T is the name of the element, A is a list of attribute/value–pairs representing the attributes and their values and C represents the content of the element or the nested elements.

Consider the following fragment of the DBLP XML document:

```
<article mdate="2002-12-04" key="journals/cacm/Codd70">
  <author>E. F. Codd</author>
  <title>A Relational Model of Data for Large Shared Data Banks.</title>
  <pages>377-387</pages>
  <year>1970</year>
  <volume>13</volume>
  <journal>CACM</journal>
  <number>6</number> ...
</article>
```

Let us take a look at the predicate, which parses an XML document and transforms it into field notation. Its implementation indicates how PROLOG and `Libxml2` work together:

```
xmlfile_to_fn(XMLFile, FN, Options) :-
   process_options(Options, Options1)
   new_parser(Parser),
   set_parser_options(Parser, Options1, Options2),
   parse_xml_doc(Parser, [ source(XMLFile), document(FN) | Options2 ]),
   free_parser(Parser).
```

[1] `http://dblp.uni-trier.de/xml/`
[2] `http://www.acm.org/sigmod/record/xml/`

First, some options influencing the parsing are preprocessed. The foreign predicate `new_parser` uses `Libxml2` to initialize a new parser object. The PROLOG predicate `set_parser_options` calls another foreign predicate to pass the parsing options to the parser object created before. The actual parsing process is initiated by the predicate `parse_xml_doc`. This predicate calls the foreign parsing routine of `Libxml2` and transforms the resulting parse tree into field notation. Finally, `free_parser` frees the resources allocated by the parsing process.

The resulting field notation representation of the document fragment is shown below:

```
article:[mdate:'2002-12-04', key:'journals/cacm/Codd70']:[
   author:[]:['E. F. Codd'],
   title:[]:['A Relational Model of Data for Large Shared Data Banks.'],
   pages:[]:['377-387'],
   year:[]:['1970'],
   volume:[]:['13'],
   journal:[]:['CACM'],
   number:[]:['6'], ... ]
```

All the content is enclosed in single quotes, since we transform the data contained in elements into atoms.

Based on this field notation we use a powerful and flexible *query language* called FNPATH, which has been introduced in [9]. A detailed description of the field notation and FNPATH can be found there. The FNPATH language allows to address, select and change any part of an XML document. In terms of functionality is FNPATH comparable to XPATH, but it is much more appropriate for using in PROLOG.

3 Web Services

The term Web services ([1]) is not always used with the same meaning. Often, in a very generic meaning, a Web service is simply seen as an application accessible over the Web. We want to use the more specific and restrictive definition given by the W3C's Web Service Architecture Working Group ([12]) defining a Web service as *"a software system identified by a URI, whose public interfaces and bindings are defined and described using* XML. *Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using* XML *based messages conveyed by Internet protocols"*.

Web services are still at a very early stage of development and some say that they currently are not more than yet another attempt to master the complexity of enterprise application integration.

But if the ongoing standardization effort succeeds and Web services standards become as widely adopted as the Web technologies already have, Web services could become the basis for a seamless and almost completely automated infrastructure for enterprise application integration, because the use of standard technologies reduces heterogeneity drastically.

So the Web services activities try to bridge the gap between what the Web already provides (originally for the interaction between humans and applications, e.g. HTTP as a standard interaction protocol and XML as a standard data format) and what application integration still requires (e.g. interface definition languages, name and directory services, transaction protocols and so on).

Web services assume that some functionality (performed by internal systems) will be exposed as a service and made discoverable and accessible for applications (not humans) through the Web in a controlled manner. Today, three components and proposed standards are the core of Web services, all of them covering different aspects of application integration over the Web: The Simple Object Access Protocol (SOAP) as a way to communicate, the Web Services Description Language (WSDL) as a way to describe services and the Universal Description, Discovery, and Integration (UDDI) project as a name and directory server. Besides these fundamental specifications, which have gone quite some way in terms of standardization and are already implemented and used in practice, some other more high–level concepts are being developed: coordination protocols (concerned for example with transactions) and Web service composition or flow languages (concerned with the composition of Web services clients and services into complex business processes). However, those specifications are at a very early stage of standardization and are still changing rapidly.

3.1 SOAP, WSDL, and UDDI

The Simple Object Access Protocol (SOAP, [10]) is the communication protocol for Web Services. The SOAP specification describes in detail a message format, i.e. how information can be packaged into an XML document, a set of conventions for using SOAP messages to implement different interaction patterns (among others the traditional RPC interaction pattern), a set of processing rules that each entity that processes SOAP messages must comply to, and how SOAP messages should be transported on top of HTTP or SMTP. Bindings to other transport protocols can also be defined, but currently HTTP is the most commonly used transport protocol.

SOAP exchanges information using messages. These messages are used as an envelope where the application encloses whatever information needs to be sent. In essence, there are two different interaction styles: document–style and RPC–style. When using document–style interaction, two interacting applications have to agree upon the structure of XML documents exchanged between them, which are then transported from one application to another in SOAP messages. For RPC–style interaction on the other hand, two interacting applications have to agree upon the RPC method signature. The SOAP specification then governs how XML documents representing the request with input parameters and the response with output parameters have to be constructed. This task is typically hidden by the SOAP middleware.

We use gSOAP (see Section 3.2) as a SOAP middleware to provide standalone Web services that make the functionality of PROLOG applications available on the Internet. gSOAP handles the SOAP messaging transparently, so that we are not really concerned with the exchange of messages, the handling of message envelopes or the underlying transport protocol HTTP etc. This leaves us with the task of providing the implementation for the functionality that is to be exposed as a Web service. This implementation differs depending on which interaction style is actually used for the Web service: for

RPC–style interaction, we have to provide a function that receives some input parameters (of simple types) from the SOAP middleware and returns the output parameters, while for document–style interaction the implementation must be able to process an XML document as input and produce another XML document as output.

Both interaction styles can be used in conjunction with PROLOG based Web services, but in particular Web services using a document–style interaction make it necessary to be able to process XML documents using PROLOG.

The Web Services Description Language (WSDL, [14] and [13]), is an XML based language that is used both as an advanced form of interface definition language and to describe several aspects of a service that are unique to Web services. This includes the transport protocol (e.g. HTTP) to use when invoking the service and the address where the service can be requested (e.g. an URL when using HTTP as the transport protocol).

The Universal Description, Discovery, and Integration (UDDI) specification ([11]) describes how to organize information about Web services and how to build registries where such information can be published (by service providers) and queried (by client service requesters). UDDI is currently not used in conjunction with the PROLOG based Web services, although it could of course be used to register the available Web services at the UDDI registry.

3.2 gSOAP

gSOAP ([3]) is a Web services middleware toolkit for C and C++ developed by Robert van Engelen at the Florida State University. The gSOAP compiler tools offer full SOAP interoperability using a simple API which relieves the user from the burden of SOAP details and thereby ease the development of Web services and client application in C and/or C++. The included WSDL parser automates server and client application development and also generates WSDL documents to publish the Web services. gSOAP is available for almost any platform, has shown to be very fast and efficient, and is very well maintained and up–to–date in terms of supporting the latest standards.

4 Putting it all Together

Figure 2 shows how all the technologies and concepts presented so far work together to provide for PROLOG based Web services.

The implementation of a PROLOG based Web service starts with the specification of the function that will be exposed as a service. This is done by supplying a C/C++ header file containing the definitions of data types and a function prototype. The gSOAP compiler generates a C/C++ stub from this header file and a corresponding WSDL document. The stub can then be used to implement the Web service's functionality. In our case, the implementation is merely a C wrapper function, which uses the foreign language interface of SWI–PROLOG to embed the PROLOG engine into the application. For this purpose, a so–called *saved state* of SWI–PROLOG is used, where the predicates of the X2P package already have been consulted and can be used immediately. These predicates again use the foreign language interface to access functions of the `Libxml2` library, which leads to the inclusion of this library as a shared object. The top layer

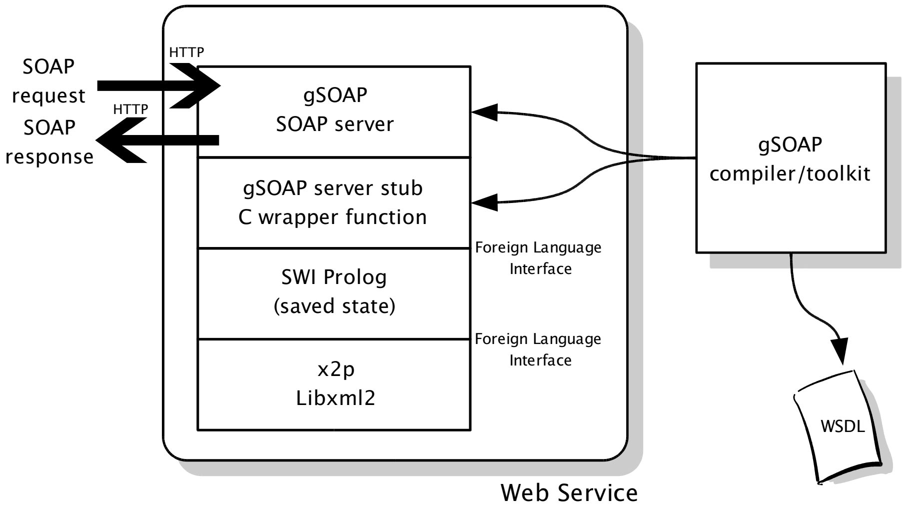

Fig. 2. Architecture of PROLOG based Web services

of the Web service application consists of a gSOAP runtime library acting as a SOAP server, which transparently handles the SOAP requests and responses.

The WSDL document describing the Web service can be used by a Web service client to locate and request the Web service (HTTP is used as the transport protocol). Upon such a request, depending on the interaction style used, the SOAP server extracts the input parameters (RPC–style) or an XML document (document–style) and passes them to the C function handling the request. In our case, this function uses a PROLOG engine to process the request and produce output parameters or another XML document as a result. This result is returned to the SOAP server, which packages it into a SOAP response that is sent back to the client.

5 Applications

The framework for PROLOG based Web services presented in the last section can be used in many different application scenarios. PROLOG itself can be used as an inference engine for many purposes, and with the advent of the Semantic Web this will gain even more importance. Furthermore, PROLOG enables rapid prototyping of heuristic approaches, especially together with the extensions for XML document processing. The Web services framework for PROLOG then allows us to quickly make such functionality available on the Web as standardized components.

As a simple example (due to space restrictions), consider a PROLOG based Web service which offers a fuzzy search on the SIGMOD Record XML document. The PROLOG program implementing this Web service receives through the gSOAP middleware and the wrapping C function an XML document (document–style interaction) containing the name of an author. This XML document is then parsed and the author's name is extracted. The SIGMOD Record document is also parsed and transformed into the field notation. The actual search for articles by the given author is fuzzy or fault–tolerant as it can deal (with respect to both the query document and the author names in the SIGMOD Record document) with different name formats, e.g. "firstname(s) lastname", "lastname, firstname(s)", firstnames abbreviated to initials and so on. This heuristic approach is well supported by PROLOG, the XML processing facilities and FNPATH resulting in a very compact and elegant program that is easy to maintain.

The main predicate of the search is shown below:

```
search_articles(Qname, Titles) :-
   data(sigmod, S),
   process_name(Qname, Query),
   findall( T,
      ( Article := S^'SigmodRecord'^issues^issue^articles^article,
        [Author] := Article^authors^author,
        process_name(Author, Aname),
        match_names(Query, Aname),
        T := Article^title ),
      Titles ).
```

Notice how the relevant parts of the SIGMOD Record XML document are addressed using FNPATH expressions. The predicate `process_name` does some preprocessing for the author name used as a query, while the fuzzy matching is done by the predicate `match_names`, which is a simple and compact rule–based algorithm. The predicate returns a list of article titles, which are then converted into an XML document and passed back through the layers of the Web services framework.

An approach like this is very promising when used in conjunction with serveral different XML documents from the same domain (e.g. the DBLP and SIGMOD Record documents) or in general for information fusion or information integration tasks.

6 Summary and Outlook

We have presented how the ability to efficiently process XML documents in PROLOG together with the gSOAP Web services toolkit and middleware make it possible to expose the functionalities of PROLOG programs on the Internet as Web services. Due to the thorough standardization of Web services technologies, this enables generic clients to autonomously use such Web services.

Until now, there is very few related work on the topic of combining PROLOG and Web services: In [2], Chen et al. describe the architecture of an intelligent agent that integrates concepts from the Semantic Web with Web Services. This agent system uses SWI–PROLOG as its inference engine for processing semantic information extracted from DAML+OIL documents and SOAP to access other Web services, but it only includes a Web services client and does not provide PROLOG based applications directly as Web services.

Web services technologies are rapidly being adopted by the industry. They are very likely to become the dominant platform for implementing distributed information systems. Just like Web browsers became the universal client for the interaction of humans with information systems, generic Web services clients will be used to discover and autonomously access information systems. For example, major Database Management Systems like IBM's DB2 offer extensions ([8]) to support Web services both as a provider (i.e. exposing relational data through a Web service) and as a requester (i.e. invoking Web services from within SQL statements using user defined functions). This is also an interesting perspective for our project, because it makes the deductive power of PROLOG available to Database Management Systems.

We think that PROLOG based Web services can be valuable components of many distributed information systems, for example in an information broker ([7]) to support complex information gathering and integration strategies or to control a mulit–agent system ([5]).

References

1. G. Alonso, F. Casati, H. Kuno, V. Machiraju. *Web Services: Concepts, Architectures and Applications*. Springer, 2004.
2. Y. Chen, W. Hsu, P. Hung. *Towards Web Automation by Integrating Semantic Web and Web Service*. Proc. of the 12th Intl. World Wide Web Conference, 2003.
3. R. van Engelen. *The gSOAP toolkit*. Florida State University, `http://www.cs.fsu.edu/~engelen/soap.html`
4. *GNU Network Object Model Environment*, `http://www.gnome.org`
5. B. Heumesser, R. Schimkat. *Deduction on* XML *Documents: A Case Study*. Proc. of the 14th Intl. Conf. on Applications of PROLOG (INAP), 2001.
6. D. Veillard. `Libxml2`. `http://www.xmlsoft.org`
7. A. Ludwig, U. Güntzer. *An Information Brokering Framework*. Proc. of the 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI), 2003.
8. S. Malaika, C. J. Nelin, R. Qu, B. Reinwald, D. C. Wolfson. *DB2 and Web services*. IBM Systems Journal, 41 (4), 2002.
9. D. Seipel. *Processing* XML*–Documents in* PROLOG. Workshop on Logic Programming (WLP), 2002.
10. *Simple Object Access Protocol (*SOAP*) Version 1.2*, W3C Recommendation, `http://www.w3.org/TR/soap12-part0/`
11. *Universal Description, Discovery and Integration (*UDDI*) protocol*, OASIS Standards Consortium, `http://uddi.org/`
12. Web Services Architecture Working Group. *Web Services Architecture Requirements*, W3C Working Draft, `http://www.w3.org/TR/wsa-reqs`
13. *Web Services Description Language (*WSDL*) Version 2.0*, W3C Working Draft, `http://www.w3.org/TR/wsdl20/`
14. *Web Services Description Working Group*, `http://www.w3.org/2002/ws/desc/`
15. J. Wielenmaker. SWI–PROLOG *Reference Manual*, `http://www.swi-prolog.org/`
16. *Extensible Markup Language (*XML*) 1.1*, W3C Proposed Recommendation, `http://www.w3.org/XML/Core/#Publication`
17. XML *Path Language (*XPATH*) Version 1.0*, W3C Recommendation, `http://www.w3.org/TR/xpath`