HETEROGENEOUS INTEGRATION OF SERVICES INTO AN OPEN, STANDARDIZED WEB SERVICE
A Web Service-Based CSCW/L System

Thorsten Hampel, Jörg Halbsgut, Thomas Bopp
Heinz Nixdorf Institute
Computer Science
University of Paderborn
Fürstenallee 11, 33102 Paderborn, Germany

Keywords: web services, e-learning, sTeam, WSDL, SOAP, CSCW, CSCL

Abstract: There are currently a wide variety of services that are difficult or impossible to use because their interfaces, protocols and programming languages are either unknown or proprietary. In the future, this problem will be compounded by the growing range of services available, especially in the area of e-learning, and not least by the increasing number of service consumers (clients) and the resulting heterogeneity in terms of applications and protocols. The web service architecture presented in this paper uses the successfully applied open-source sTeam system to illustrate how arbitrary services can be integrated into a heterogeneous web service. A flexible service structure of this kind is designed to create standardized interfaces allowing new web-based interoperability.

1 INTRODUCTION

The rapidly growing number of web-based services offers the promise of an ever increasing variety of potential applications. Unfortunately, this promise cannot be kept because unknown or proprietary interfaces, protocols and programming languages make such services difficult or even impossible to use. The growing number of different applications and protocols on the client side seriously compounds the problems caused by this lack of interoperability.

A flexible and integrated service structure with standardized interfaces is needed if the various existing heterogeneous systems are to continue to be accessible to a wide range of users with highly diverse application systems. Such an open service infrastructure ensures high scalability with respect to future service requirements and, where possible, global use of the services offered.

The interoperability of heterogeneous services is especially necessary in the area of cooperative knowledge organization. The available systems, which differ in terms of the quality of their knowledge organization or the scope of the information considered, should not continue to exist separately from one another. Integrating these services into a single system opens up completely new use options. This is illustrated by the following scenario.

The search for a specific document using a keyword is a service already provided by various data sources. Such a search must, however, be conducted in each of these data sources separately. In the worst-case scenario, it also involves using different applications or devices. Integrating such search services will enable an arbitrary client to make a single, simple search request to the overall system. This will yield a search result that includes all the results of existing data sources.

For a number of years now, work has been under way in Paderborn to design and test various architectures for cooperative knowledge organization and e-learning (e.g. Hampel & Bopp, 2003). Developing new forms of web-based interoperability and standardization is a specific prerequisite for creating an open-source architecture capable of integrating a variety of application concepts. Conceptually, our efforts to build such an infrastructure are based on the idea of cooperative knowledge spaces (cf. Hampel & Keil-Slawik, 2003).

Cooperative virtual knowledge spaces combine synchronous and asynchronous forms of cooperation for administering hypermedia documents. Users

Lüken M. (2004).
INTRODUCING AN OPERATIONAL AND TECHONOLOGICAL E-COMMERCE FRAMEWORK FOR EUROPEAN SMES - Designing regional and interegional e-commerce zones for SMEs in four Less Favoured European Regions (LFRs) based on Request Based Virtual Organisation (RBVO) concept.
In *Proceedings of the Sixth International Conference on Enterprise Information Systems*, pages 182-189
DOI: 10.5220/0002632101820189
Copyright © SciTePress

(learners) meet in virtual knowledge areas, where they can use web-based facilities to store and actively process documents by obtaining shared views of them, by exchanging, arranging, mutually annotating and linking them. This form of open and cooperative handling of material is supported by authentication processes such as user groups and access authorization.

Here, the principle of self-administration allows specific knowledge structures to be created for groups and individual users and enables virtual communities to be built on a self-organized basis. Cooperative knowledge spaces are currently provided by the open-source sTeam system.

The sTeam system's architecture has already been presented at a number of conferences (cf. Hampel & Keil-Slawik, 2003).

The functionality offered by sTeam can be used with different interfaces. These include various standard protocols that have been implemented on the server side within the sTeam kernel. E-mail protocols (e.g. POP3, IMAP, SMTP) and file-transfer protocols (e.g. FTP, WebDAV) are supported. In addition, the COAL protocol (cf. sTeam, 2003) allows communication with the sTeam server. This enables, for instance, methods to be directly executed on sTeam objects. The implementation of the protocol is available in the languages Pike and Java.

An Internet browser can also be used to access sTeam via a web interface, the available data objects being presented to the user in graphical or text form.

1.1 Shortcomings of the Existing System

All the above-mentioned protocols offer a standardized but highly specialized form of communication with the sTeam server. The familiar e-mail protocols can be used only with the specified e-mail applications, and FTP programs are needed to transfer data via the FTP interface. It is, however, conceivable that application developers might wish to access e-mails in sTeam from their own software, which was developed for a specific application area. To make this possible, the e-mail protocols would have to be implemented on the client side. Such an implementation, though awkward, is possible. If, however, the application is to access objects in sTeam in order to manipulate them, this is not possible using the above-mentioned standard protocols. The only available alternative is to use the COAL interface. Here, however, recourse must be had to the existing Java API, meaning that the application is again tied to a single programming language, or application developers must reimplement COAL in the language they use (e.g. C#). In most cases, though, the additional effort this involves is unjustifiable. Also, the advantages that might come with using a development environment (IDE) can no longer be exploited because such environments do not support COAL.

Nor is COAL an unambiguously defined description language – a fact that gives rise to two disadvantages: it is not possible to automatically connect a client to sTeam or to generate client classes using an IDE, and COAL fails to give any information about the service functionality provided.

The sTeam server does have a partially modular design, i.e. modules written in Pike can be added to extend the functionality. But it cannot be extended to include external programs. The sTeam kernel cannot access concurrently available programs in other languages in order to make use of them.

Given these shortcomings, sTeam can be described, in summary, as a relatively closed, monolithic system with a proprietary interface.

The shortcomings of the Paderborn sTeam environment apply to practically all available CSCW and CSCL systems. Their architecture, mostly consisting of monolithic servers, makes them difficult to integrate flexibly into web services. This means that such systems have problems taking into account all possible needs and use constellations in the interfaces and functionality they provide. This dilemma can only be resolved by integrating different services and applications.

However, for the reasons mentioned above, it has not yet proved possible to combine sTeam with other services or applications to create an integrated overall knowledge management system. The following scenarios attempt to show, by reference to the sTeam system, why such integration is needed and give initial instances of the use of the web service architecture presented in this paper.

1.2 Scenarios and Requirements

Using cooperative knowledge management, sTeam enables different types of documents to be administered. These include text documents, which may be annotated, or e-mails. For instance, a user engaged in writing a research paper wishes to make a search using a keyword in his/her own and in other sTeam documents. But the search is to be extended to external texts and web pages as well as to the university's literature database. The results of the search should then be available in the user's own C# application, running on his/her Windows laptop, for use as references in the paper being written. Finally, it should be possible to annotate selected references and make them available for subsequent work or to other sTeam users.

The system needed to perform these tasks must be able not only to conduct a search on sTeam but, parallel to this, to access other services, e.g. an Internet and library search. On completion of the search, the user/client application should be supplied with a normalized search result that is compatible with the C# application. The system should also be capable of integrating potentially annotated search results in sTeam in the form of structured data.

Time-scheduling and calendar functions are also important in a system employed by several users cooperatively, e.g. to work jointly on an object for a specific deadline using the whiteboard provided by sTeam, or for a real meeting at an appointed time. In most cases, however, users are unwilling to depart from their traditional time-scheduling applications because these are also used in other contexts. Changing from one time-scheduling system to another would be time-consuming and lead to synchronization errors. It is therefore a good idea to enable the time-scheduling system currently employed by the user to access the cooperative system and use it to make appointments with other users.

sTeam does not currently support time scheduling that may have to be synchronized with other users. If the sTeam kernel is extended to include calendar and time-scheduling functions, standardized interfaces must be provided to ensure the interoperability with arbitrary time-scheduling systems. The same applies in cases where, parallel to sTeam, the knowledge management system is extended to include a centralized time-scheduling system.

The above scenario is an example of how existing cooperative systems like sTeam could also benefit from cooperation with other services, e.g. a time-scheduling system. This would enable the scope of the required functionality within a system, which is basically already available, to be drastically reduced.

Consideration of these scenarios yields a wide range of requirements that must be met by an integrated overall system consisting of different services for cooperative knowledge organization and e-learning.

A key issue here is the standardized integration into the system of various client types that differ in terms of the programming language used, the transmission protocols available, performance, hardware resources and usability. A mobile phone running a Java application has nothing in common with a C# application on a desktop computer. But in both cases the same services should be usable from the applications. These services, in turn, should be part of a complex service structure with diverse functionalities.

Besides supporting specialized standard protocols for e-mail and file transfer (POP3, WebDAV, ...), the desired system must also be able to meet changing requirements. On the server side, this includes the uncomplicated and swift implementation of desired functionalities and the extension and modification of existing services. On the client side, the services provided by the system should be easy to use. As in the case of the service provider, this involves extending existing applications or completely redesigning applications based on the given interfaces.

2 GENERAL SOLUTION

A web service (WS) is particularly well-suited for meeting the described requirements.

The required web service is a system of distributed service components, differing in terms of the hardware and software they use, which are made available to service consumers (clients) via the Internet using different protocols and message formats. To this end, the available services are encapsulated by an interface. This must be standardized and self-describing in order to achieve the greatest possible interoperability. Clients may take any form, e.g. a PC, a PDA, a software application or even a service provider itself. The client must merely connect to the WS's interface and a corresponding standardized message transfer format. The efficient development of client applications and a flexible service structure must also be ensured, with the option of integrating new services and adapting existing ones.

As part of a system with these features, sTeam is integrated with any other desired services to create a heterogeneous web service. Such an architecture is, of course, highly portable, which means that different CSCW/L systems could be integrated in the same way as the sTeam system.

The architecture's highly modular design also offers the advantage of load balancing, with the option of establishing individual services on different servers. "Outwardly", however, the web service always appears as a uniform, integrated service. This is illustrated in Figure 1.

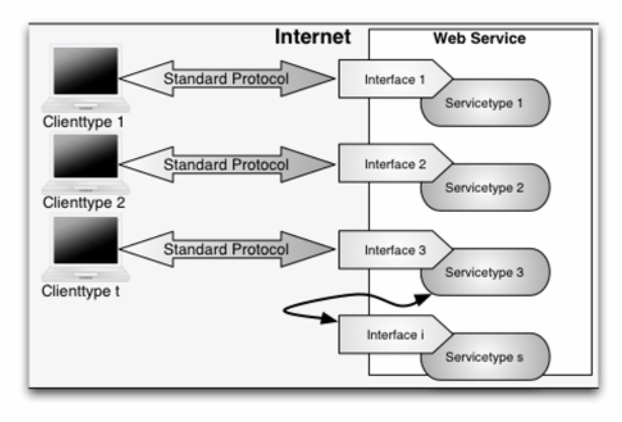

Figure 1: Webservice

3 ARCHITECTURE

The architecture presented here is designed to implement a web service for cooperative knowledge organization. It contains sTeam as a service kernel component. It is shown which concrete technologies are needed to implement this web service, taking into account the requirements mentioned in Section 2 (General Solution). It is also clearly demonstrated which fundamental design decisions must be taken based on the desired interface functionalities. Consideration is further given to important aspects such as security and interoperability. The aim is to present an architecture that serves as a model for the integratability of different existing web-based systems into a web service architecture.

3.1 Architecture Model

The web service can be broken down into several main components. These include the application server and service provider, responsible for the service logic. For communication purposes, the SOAP technologies (cf. W3C, 2000) and the corresponding protocols for accessing subservices are needed, in this case COAL for sTeam.

This reflects both a logical view of the system and its concrete technical implementation. The concrete example of the sTeam system is used below to demonstrate the conversion of a monolithic CSCW system into a web service architecture. With certain minor restrictions, this approach is clearly portable to a whole series of web-based system solutions.

3.1.1 Application server

The application server contains the service applications (service providers). It receives requests made to the web service from "outside", i.e. from client applications. These service requests cause the server to trigger the service providers with the desired functionality. Potential results are then returned to the requesting client.

Apache's open-source application- and web-server Tomcat is an obvious choice here. It is implemented in Java and offers various options for executing Java applications on the server side. In the architecture presented here, the service providers are implemented by servlets inside Tomcat.

3.1.2 Service providers

The logic of the individual service authorities is implemented inside the service providers. The entire web service can be built from an arbitrary number of service providers. This offers the advantage of extending the web service's functionality simply by adding a new service, without affecting existing components. The web service's design is thus entirely modular. Which functions are integrated into a single service provider is a matter of choice and remains a design decision (see also Section 0).

The number of service providers thus constitutes the directly usable "service layer". By contrast, there exists a further layer of "subservices" that is used by the service layer. sTeam is an example of a subservice. The individual subservices can be based on arbitrary technologies, but they must be encapsulated by a service provider in order to integrate them into the web service and thus make them usable. The service provider uses a suitable protocol to access the subservice.

The various services are implemented in Java because they are executed inside the Tomcat server as servlets (see Section 0).

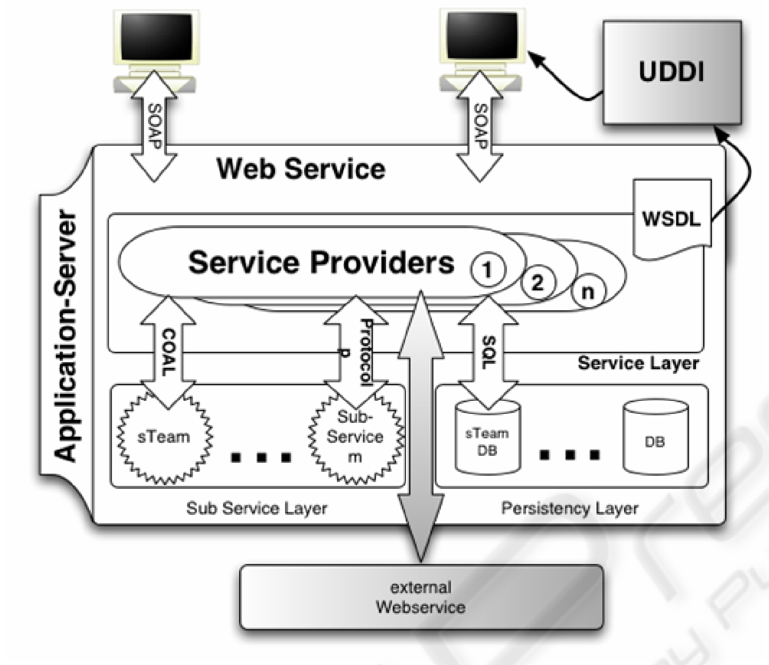

Figure 2: Web Service Architecture

3.1.3 sTeam as a subservice of the service providers

In this architecture, the sTeam server functions as a subservice of the service providers. This means that the functions of sTeam are available to the service providers via the COAL protocol. Outwardly, then, sTeam is completely encapsulated by one or more providers. The extent to which sTeam is used depends on the logic of the individual services. The possibilities range from the service making no use at all of sTeam to the work being completely outsourced to the sTeam server. In this case, the service provider would merely be used to "pass on" the service requests. Section 0 takes a systematic look at the different ways of distributing and shifting the service logic over the two above-mentioned layers. By integrating an existing system like sTeam as a subservice of a service provider, all the sTeam system's functionality can be used in combination with arbitrary other services. Already developed and tested parts of the sTeam system can then be used via access layers encapsulated by the web service.

3.1.4 Persistence layer

Another web service component is the database needed by sTeam. This could also be used jointly by the service layer and other subservices. It is, of course, possible to install additional databases.

3.1.5 Communication protocols

The SOAP protocol is always used for communication between the clients and the web service/service providers. By contrast, a service provider uses a suitable protocol to access a subservice. In the case of the subservice sTeam, this would be COAL.

SOAP is particularly well-suited for communication between the web service and the calling applications. As an XML protocol, SOAP enables remote methods to be called – in this case the functions of the service providers – irrespective of the programming language in which the client application or the service has been programmed. This is a basic advantage in comparison to other technologies for web services, e.g. Java RMI or CORBA. SOAP is specified in (W3C, 2000) and realized in various open-source or freely available implementations for a large number of programming languages.

An example of an API that supports the implementation of web services based on SOAP and WSDL is Apache Axis (Apache Software Foundation, 2003). This API can be used to generate JAVA methods and classes from a WSDL document. The generation and sending of messages conforming to SOAP is encapsulated by the API's JAVA methods, making it very easy to do and promising to help avoid programming errors. In addition, it is possible to automatically generate a WSDL document from an existing server implementation or JAVA-interface and data-type definition.

When working with WSDL and SOAP, efficiency can be improved by using so-called "web service toolkits". Examples of such products can be found in (Borland 2003, Systinet 2003, IBM 2003 and Sun Microsystems 2003). They can be integrated into commonly used development environments and allow the generation of WSDL documents and the corresponding client and server implementations based on SOAP within the usual IDE, e.g. Netbeans, Eclipse or JBuilder. Some of these toolkits use well-known APIs, such as the Apache Axis mentioned above.

The COAL protocol serves as an interface between sTeam and the service providers implemented in Java (see also Section 0). In the sTeam environment, there is a corresponding API that implements the COAL protocol, thus providing easy-to-use methods in Java. This API must be made available to the implementation of the service providers as a class library.

3.1.6 WSDL

The officially recognized standard for describing web services, WSDL (specified in W3C, 2003), can be used to unambiguously structure and define the functionality of the web service. This is necessary, among other things, to enable the application developer or the development environment to implement the SOAP calls in conformity with the given web service interface. Which parameters are needed for a successful call and which data might be used as return values is defined unequivocally within the WSDL document.

Figure 2 shows the relations between the main components of the web service.

3.2 Design Decisions

The web service with the above-described architecture is able to offer a wide variety of services and thus different functions. We focus below on three different functions: abstract functions encapsulating complex logic inside the service provider, basic functions that provide the basic functionality of subservices or protocols, and service functions – in this case sTeam service functions offering special web service functions implemented within a subservice.

3.2.1 Abstract functions

The web service provides a service function, which is realized within the service provider by implementing complex logic. The service may contain a range of functions and can, in order to accomplish the service's task, access other subservices, e.g. sTeam, or even extreme services.

Towards a possible client, the complex implementation is encapsulated and a single method call is provided.

In the field of e-learning, an abstract web service function "conduct literature research" is conceivable, which would search for literature based on a certain topic or keyword. The user is offered only this one operation, its logic being implemented inside the service provider.

3.2.2 Basic functions

The client can be provided with different basic functions – in the case of sTeam, using the COAL interface. The service provider can implement this functionality by simply calling the desired sTeam methods and returning the result to the client – no special logic is needed here.

Such web service functions would be very powerful when accessing the sTeam server directly. But this would be at odds with the initial idea of creating an easy-to-use interface. Indeed, sTeam's entire functionality would be available directly in the used programming language. Even proprietary protocols such as COAL can be encapsulated via a web service in standardized protocols. The service provider must simply map the respective protocol to the appropriate SOAP call.

3.2.3 sTeam service functions

If necessary, a complex service function is implemented inside the sTeam server and made available via a service provider. As in the previous section, the main logic was moved to the subservice. This is useful in cases where many sTeam accesses are needed to perform the web service's task. Otherwise, numerous repeated calls via the COAL interface would mean large performance losses. It is a good idea, then, when searching for certain sTeam users to allow the whole task to be performed by the sTeam server, with only the final result being returned by the service provider to the calling client. Otherwise, if the search were left to the service provider, a large number of individual calls to the sTeam server might be necessary.

Depending on the use case, there are different options for distributing and shifting the service logic over the "service provider" and "subservice" layers. This is also possible later on, e.g. if new service providers are added.

3.3 Basic Implementation Steps

The following basic steps are needed to practically implement the described web service architecture. In concrete applications, they may be slightly modified or used in a different order.

a) Making the relevant design decisions (described in Section 0)
b) Providing the necessary infrastructure
   Basically, they include an application server and a SOAP implementation, e.g. Tomcat and Apache AXIS (see also Section 0)
c) Implementing the service logic
d) Creating the WSDL file(s) as a "by-product" of the implemented service logic
e) Deployment and publication of the web service by integrating the service providers' implementations and the WSDL file into the application server. In addition, publication of the WSDL file on a UDDI server makes the web service globally accessible (for further details, see OASIS UDDI, 2003).

3.4 Security

The web service functionality should not be universally accessible. It must be possible to allow only authenticated users access to the web service and to assign different access rights to members of this group. These options are precisely defined in WS Security, specified by Microsoft and IBM (Microsoft & IBM, 2002). WSDL or a corresponding implementation can be extended, e.g. to include authentication functions.

In addition, the already existing security mechanisms of SOAP's underlying transport protocols, e.g. HTTPS, can be used. This means that transmitted messages would be encoded for third parties.

3.5 Interoperability

Unfortunately, some of the various implementations of SOAP are not interoperable because of different interpretations of the specification. For instance, it may be the case that a client implemented in C# is unable to communicate with a web service in Java because the serialization of complex data types is incompatible.

With the specification WS-I Basic Profile (WS-I, 2003), the Web Services Interoperability Organization (WS-I) is attempting to make the implementations of web services and clients compatible.

3.6 Data Synchronization

Another aspect must be considered when implementing and using web services. It is fundamentally impossible to work with object references. If, for instance, a data object is returned to a client as a result of a function call, it is always a copy of the object. Changes therefore only affect the copy and not the original object within the web service. When changed objects are returned to the web service, data synchronization is necessary. At present, this is still left to the developer on both the server and client side. Programming-language-independent solutions are already available in SyncML (SyncML Initiative, 2002).

4 RELATED WORK

There are currently a number of Internet services that have been made available via web services not only for Internet browsers but also for various other applications. These include the Google search engine (www.google.de) and Amazon (www.amazon.de). Both of these web services are described by corresponding WSDL files. In addition, APIs are available that implement, in languages such as Java or C#, corresponding classes for calling methods via SOAP. For instance, the SOAP call "doGoogleSearch" can be used to conduct a search with given keywords. A similar search exists for Amazon, where users have the additional option of ordering goods.

5 SUMMARY AND OUTLOOK

This paper shows how easy it is to generate a modern web service architecture to integrate existing systems. By using various standardized communication protocols like SOAP and a Web Service Description Language (WSDL), parts of an existing application (in this case the sTeam system) can be made available for other web services and any other type of application, thus enabling a flexible web-based service infrastructure to be built.

The advantages of such an approach are obvious. Elaborately developed and tested web-based systems can be neatly integrated, allowing a sustainable infrastructure to be built. And web-based interfaces make the interoperability of different systems possible for the first time – which fits in particularly well with open-source approaches like the sTeam system.

This concept could also conceivably be used for building or supporting a peer-to-peer network. On the one hand, a web service could make it possible to find a P2P partner. And on the other, a client might also be a service provider, resulting in a network of highly diverse services that could be used on a peer-to-peer basis. The process of finding such – possibly widely distributed – services could be supported by UDDI. Another alternative, in small networks, might be a variant of the Zeroconf Protocol (IETF, 2003), which would have to be extended to include self-description capabilities. This would make it possible to automatically integrate services provided by the individual peers in an Intranet.

REFERENCES

Apache Software Foundation: WebServices – Axis, 2003. URL: http://ws.apache.org/axis/ [31.10.2003]

Borland: Web Services – Solutions for Web Services, 2003. URL: http://www.borland.com/webservices [31.10.2003]

Hampel, T., Bopp, T.: Combining Web-Based Document Management and Event-Based Systems -Integrating MUDs and MOOs With DMS to Form a Cooperative Knowledge Space, Fifth International Conference on Enterprise Information Systems, April 23-26, 2003, Angers, France, 219-223.

Hampel, T., Keil-Slawik, R.: sTeam: Structuring Information in a Team – Distributed Knowledge Management in Cooperative Learning Environments. In: *ACM Journal of Educational Resources in Computing* 1(2) 2002, 1–27.

Hampel, T., Keil-Slawik, R.: Experience With Teaching and Learning in Cooperative Knowledge Areas., Proceedings of The Twelfth International World Wide Web Conference, 20-24 May 2003, Budapest, Hungary, on CD-ROM, 1-8.

IBM: WebSphere SDK for Web Services (WSDK) Version 5.1, 2003. http://www-106.ibm.com/ developerworks/webservices/wsdk/ [31.10.2003]

IETF: Zero Configuration Networking (Zeroconf), 2003. URL: http://www.zeroconf.org [31.10.2003]

Microsoft, IBM: Web Services Security (WS-Security), 2002. URL: http://www-106.ibm.com/developerworks/ webservices/library/ws-secure/ [31.10.2003]

Oasis: Universal Description, Discovery and Integration (UDDI) of Web Services – About UDDI, 2003. URL: http://www.uddi.org/about.html [31.10.2003]

Universität Paderborn: sTeam – cooperative learning, 2003. URL: http://steam.upb.de [31.10.2003]

W3C: Simple Object Access Protocol (SOAP) 1.1, 2000. URL: http://www.w3.org/TR/SOAP/ [31.10.2003]

W3C: Web Services Description Language (WSDL) Version 1.2 Part 1: Core Language, 2003. URL: http://www.w3.org/TR/wsdl12 [31.10.2003]

WS-I: Basic Profile Version 1.0a – Final Specification, 2003. URL: http://www.ws-i.org/Profiles/Basic/2003-08/BasicProfile-1.0a.HTML [31.10.2003]

Sun Microsystems: Java Web Services Developer Pack 1.2, 2003. http://java.sun.com/webservices/ webservicespack.html [31.10.2003]

SyncML Initiative: SyncML - Data Synchronization and Device Management, 2002. URL: http://www.openmobilealliance.org/syncml, [31.10.2003]

Systinet: WASP Developer, 2003. URL: http://www.systinet.com/products/wasp_developer [31.10.2003]