Software Research Lab
Institut fr Computerwissenschaften
Universitt Salzburg
Jakob-Haringer-Str. 2
A-5020 Salzburg
Austria

Interaction of Device-Independent User Interfaces with Web Services

Guido Menkhaus Guido.Menkhaus@uni-konstanz.de
Wolfgang Pree pree@acm.org
Philipp Baumeister baumeister@fmi.uni-konstanz.de
Uli Deichsel deichsel@fmi.uni-konstanz.de

TR-C048 April 1, 2002

The Web has evolved from a set of mere static pages to a means for offering applications through dynamic Web sites. Web services extend this paradigm by providing application logic via standardized Internet technology. A Web service consists essentially of an interface for accessing black-box functionality over the Internet. A Web service can be used by end users as well as by other applications or Web services. If an adequate set of Web services will be available, it should be possible to create Web-based applications by composing the appropriate Web services, no matter where they are located or how they are protected.
Another trend has to be considered to cope with the requirements of Web applications in the future: a variety of devices with different display and processing resources will offer access to the Web, ranging from cell phones to various handheld devices. Thus, we see an urgent need to support these different devices no matter which Web services underlie a particular application. The paper presents the Multiple-User-Interfaces-Single Application logic (MUSA) architecture and system as an attempt to address this issue.

Interaction of Device-Independent User Interfaces with Web Services

Guido Menkhaus
Software Research Lab, University of Salzburg, A-5020 Salzburg, Austria, and
University of Constance, D-78457 Constance, Germany,
email: guido.menkhaus@uni-konstanz.de, menkhaus@SoftwareResearch.net
Tel: +49 7531 88 2188
Fax: +49 7531 88 3577

Wolfgang Pree
Software Research Lab, University of Salzburg, A-5020 Salzburg, Austria,
email: pree@SoftwareResearch.net

Philipp Baumeister
University of Constance, D-78457 Constance, Germany,
email: philipp.baumeister@uni-konstanz.de

Uli Deichsel
University of Constance, D-78457 Constance, Germany,
email: uli.deichsel@uni-konstanz.de

Interaction of Device-Independent User Interfaces with Web Services

Guido Menkhaus[1,2], Wolfgang Pree[2], Philipp Baumeister[1], Uli Deichsel[1]

[1] *Software Research Lab, University of Salzburg, A-5020 Salzburg, Austria, email: {lastname}@SoftwareResearch.net*
[2] *University of Constance, D-78457 Constance, Germany, {firstname.lastname}@uni-konstanz.de*

Abstract. The Web has evolved from a set of mere static pages to a means for offering applications through dynamic Web sites. Web services extend this paradigm by providing application logic via standardized Internet technology. A Web service consists essentially of an interface for accessing black-box functionality over the Internet. A Web service can be used by end users as well as by other applications or Web services. If an adequate set of Web services will be available, it should be possible to create Web-based applications by composing the appropriate Web services, no matter where they are located or how they are protected.

Another trend has to be considered to cope with the requirements of Web applications in the future: a variety of devices with different display and processing resources will offer access to the Web, ranging from cell phones to various handheld devices. Thus, we see an urgent need to support these different devices no matter which Web services underlie a particular application. The paper presents the Multiple-User-Interfaces-Single Application logic (MUSA) architecture and system as an attempt to address this issue.

1. Introduction

Web services allow for a fast and smooth creation of Web-based applications across traditional hardware and software barriers. They enable organizations to create applications and services, which run distributed on several servers crossing administrative domains protected by firewalls. In the ideal scenario, the composition and integration of Web services would make it possible to offer complete services without ever having to develop any part of it [14]. The advent of Web services enables service providers to come a step closer to the vision of complete dynamic and distributed Internet applications. At the same time, the trend of Web access is drifting away from the desktop PC as the principal device to access services and information on the Internet to consumer devices such as mobile phones, handheld computers, and a wide spectrum of Personal Digital Assistants (PDAs). The objective of service providers is the creation of services that are device independent, to avoid fragmentation of the Web space into spaces that are solely accessible with specific type of devices, and make sure that the web related technologies supporting various kinds of devices can interoperate with each other or with the existing web as much as possible [12]. Applications will need to be able to reconfigure their interfaces to take advantage of whatever modalities are available on the user's platform.

It will be increasingly important for service provider to appropriate means for scalable and flexible presentation logic and for easy integration and composition of Web services to form a new entity. The paper presents the MUSA system and the description language EG-XML. EG-XML allows the description of service logic that enables users via a UI and other Web services to access a Web application. Service providers are put into the position to rapidly develop Web applications by composing Web services.

The remainder of the paper is structured as follows: The motivation of the work is presented in Section 2. Section 3 describes the architecture of the MUSA system. The Web service infrastructure and its application in the work is discussed in Section 4. Results are presented in Section 5. Section 6 briefly presents related work and Section 7 concludes the paper with a discussion of future work.

2. Motivation

In recent years, a wide range of mobile computing devices has emerged. This has led to a diversification of markup languages for different types of devices [6]. Each class of target devices comes with its typical browser and has its specific markup language. Among the languages are HTML, XHMTL, DHTML (the combination of HTML, CSS and XSL style sheets, the Document Object Model, and scripting), XML-based User Interface Language (XUL) for traditional desktop applications; the Wireless Markup Language (WML) for mobile devices such as cell phones; and the Voice Markup Language (VoxML) for voice-enabled gadgets without display such as conventional telephones. Each language aims at a specific platform and is optimized for supporting it. That is, each approach establishes a 1 to 1-relation between the User Interface (UI) description and the target platform. The explosion of the variety of gadgets for Internet access with widely differing properties with regard to UI capacity directed research on architectures that allow the creation of a single description to serve a multitude of platforms or to make existing descriptions available to device categories other than the originally intended class of devices [2, 8, 9]. We argue that the introduction of an abstract description of a UI is an essential component that facilitates the development of UIs for a variety of computing devices.

Web services reinforce the trend of distributed applications, where service providers offer applications composed from a set of services, which are developed and run by one or more different organizations. The service provider integrates application-specific Web services to implement the service logic. The service facade exposes the service logic to a user who interacts with the service logic via a UI or to other Web service (Figure 1).

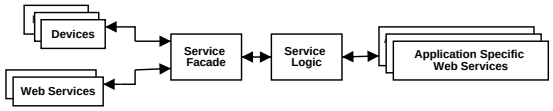

Figure 1: Architecture of a Web application composed of Web services.

3. MUSA

The MUSA project concentrates on device independent UI description and on the support of the integration and composition of Web services. The objective is a reduction of development time, cost, as well as improved maintainability and flexibility. Figure 2 illustrates the high-level architecture of MUSA. The system is conceptually split into four tiers and employs an event-driven design.

Client. The client environment represents the first tier. The client is represented either by a device with a UI or by a Web service. In case of a client with a UI, no service data is installed on the client side and the client communicates via wireline or wireless Internet with the service. Typically, the client is some sort of browser, but could also be a device with no visualization capacity such as a telephone. If the client is a Web service, it communicates with a Web service gateway, which itself is implemented as a Web service.

Request Processor and Client Gateways. The request processor deals with the client's request. The communication between the services and the client passes through the request processor. It forwards the communication stream to the MUSA core system, e.g. to the EG-XML Interpreter SPU. The request processor is the link between the MUSA core system and the client and converts the client requests into events that are used throughout the MUSA architecture.

The short message service (SMS) gateway is a module that is logically located between the client and the request processor. It receives short messages from the client. The short messages have a predefined syntax and semantic and the gateway converts them into events that are forwarded to the request processor. The request processor receives the request and returns the answer to the SMS gateway, which in turn transfers the result to the client.

The Web service gateway implements the same behavior as the SMS gateway but for Web service clients. It acts as the interface of the overall Web application. A Web Service interface is defined strictly in terms of the messages the Web Service accepts and generates [7]. The incoming messages are dispatched as events to the MUSA system. The MUSA system translates the outgoing events into response messages and forwards them to the client.

MUSA Core System. The MUSA core system consists of specialized processing units (SPU), which reflect the separation of concerns of the system.

1. **Event graph interpreter SPU.** The event-graph interpreter mediates between the client interaction and the interaction with the Web services. The event-graph interpreter contains the description of the service logic in EG-XML and handles the event processing. The incoming events from the request processor are sent to individual event handlers that contain the necessary information to properly respond to the input event. In response to the event processing the system generates outgoing events for each incoming event, which are further processed in the MUSA system. The request processor sends the processed outgoing events to the appropriate client.

2. **Encoding transformer SPU.** The transformer SPU transforms and maps outgoing events of the event graph to an appropriate presentation form. If the client is a device with a GUI, the events are mapped to those concrete UI elements, which are able to implement and trigger the specified events. The SPU applies a transformation on the event graph depending on the client's profile. Figure 2 shows four transformers: a HTML, a WML, a SMS, and a Web services transformer.

3. **Presentation model integrator SPU.** The integrator SPU models the overall presentation layout of the events, which are transmitted in the course of the current interaction between the user and the application. The integration of the events into a presentation model (PM) is particularly important for HTML, since the user has to receive nicely designed HTML pages. The UI designer creates the PM and submits them in the PM repository. For Web service clients, this SPU is not applicable.

Service Proxy. The service logic is the body of code for which the MUSA system provides the service facade. The Web services that implement the service logic are accessible via service proxies, which connect the MUSA system to other Web services. The Web services have no knowledge of the event-based interaction structure or the presentation issues of the service data. They are designed independently from these

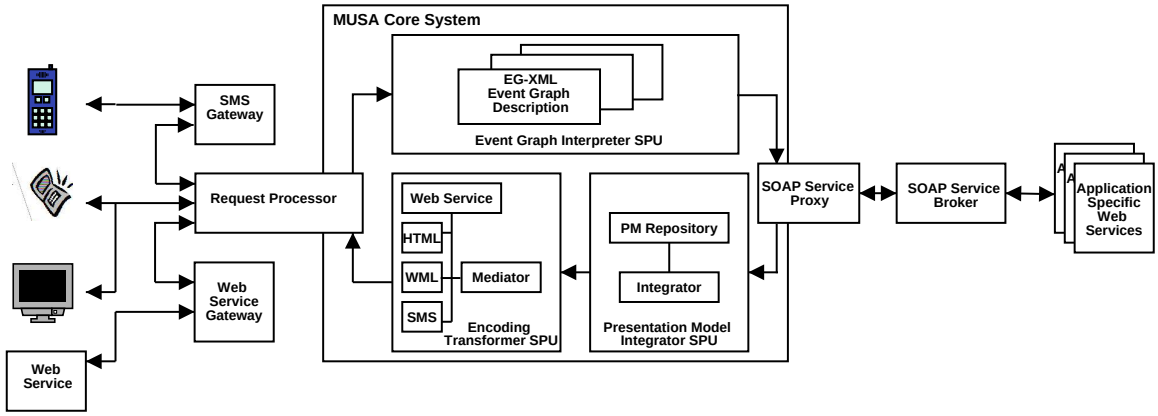

Figure 2: High-level architecture of MUSA system.

issues. Section 4 describes the communication mechanism between the description of the service logic in EG-XML and the Web services. The next section presents the description of the service logic in EG-XML.

The MUSA system allows the description of a service in a client-independent way and enables the integration of Web services from different sources. MUSA describes an abstract service model in EG-XML that can be accessed by Web services and a wide range of different consumer devices. On the one hand, the MUSA focuses on the delivery of service content to different clients and on the other hand on the support of the integration of different Web services. The main three areas addressed by MUSA are:

1. Semantic preparation of the service logic,
2. Communication between the service logic and client,
3. Communication between the description of the service logic and the Web services, which implement the service logic.

The semantic preparation of the service logic deals with the capability to adequately associate semantic objects, carrying application related information, with concrete UI interaction objects in case of clients, which use a device to access the application. For Web service clients, a Web service proxy, which is an intermediary between the service logic description and the client Web service, fulfills this task. The Web service proxy itself is implemented as a Web service. In an idealized model, semantic objects represent data and functions as name/value pairs. The semantic objects have to be communicated, i.e., transferred between the service logic and the client.

The appropriate infrastructure needs to be available to bind the service logic to the separate clients with their specific interaction capability. Since the clients may be very heterogeneous ranging from Web services, standard Internet browsers to cellular telephones communicating via Short Message Service (SMS), a standard event mechanism is put in place to allow unified communication across the MUSA system. The interaction between the client and the service logic is implemented using an event-driven approach [13]. There are two forms of interaction. One involving the Web services implementing the service logic and the other, which is resolved without interaction with Web services.

1. **Local Interaction.** Local interaction does not involve communication with Web services that implement the service logic. Events representing local interaction do not involve interaction with the Web services. The event's target is solely the glue part of the service logic implemented with EG-XML and it is processed exclusively by it. E.g., the navigation of the UI by a client accessing the service via a consumer device like a cellular telephone.

2. **Global Interaction.** Global interaction is represented by events that involve interaction with the Web service, that are integrated into the service logic. This type of interaction offers the client to submit an event, which is transformed into a call of the API of a Web service. The application processes the event and returns data, which are further processed in the MUSA system and presented to the client.

The presentation layer determines how the data of a service are presented to the client. If the client has a UI, information on how to map content to concrete UI objects is handled by the local layout. The global layout considers high-level aspects of layout management and determines the position of concrete UI objects to each other. A Web service is a special case of a client having no UI.

The communication between the MUSA system and the Web service is done using the Simple Object Access Protocol (SOAP). The service description in EG-XML mediates between the events dispatched by the client and the Web services implementing the overall service logic. EG-XML delivers a mechanism that supports developers to integrate Web services by offering a standard way of communication with a Web service. The

communication is based on the assumption that events carry semantic objects representing data and functions as name/value pairs.

3.1. Event Graph XML

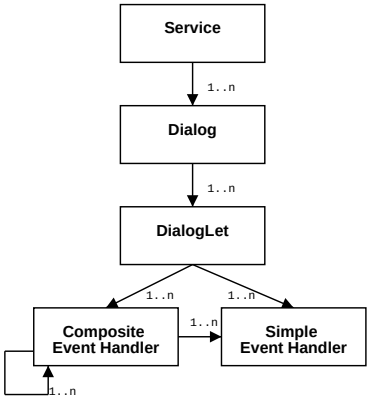

Figure 3: Structure of the event graph.

The event graph is at the core of the MUSA system. The introduction of the event graph follows the idea of the reactive constraint graph described in [1] and the abstract depiction hierarchy presented in [10]. The event graph implements an interaction-oriented service description. It is an abstract description of a service logic (a service facade to the set of Web services), which is available for service access to a wide range of clients. The basic building blocks of the event graphs represent specific event handlers. The event handlers receive events from the client dispatched by the request processor and emit events in response to the event processing. In case of a client with a UI, outgoing events are assigned and eventually mapped in the transformer SPU to concrete UI elements that are able to trigger the events. The UI elements trigger the event either on display of the UI elements or in response to user interaction. The introduction of the event graph as a high-level abstraction of the service logic/client interaction allows rapid development of services. This concept facilitates the implementation and design of services. A set of hierarchical structures that help to divide a service into smaller parts further promotes the development (Figure 3).

1. **Simple Event Handler.** An event handler is an abstract interaction object. It contains the necessary information on how to handle the event or to delegate the processing of the event and its associated data.
2. **Composite Event Handler.** An event handler is composite if it is composed of other event handlers.
3. **Dialoglet.** A dialoglet consists of a number of events, which belong to a group logically and semantically.
4. **Dialog.** A dialog is designed to represent a task or a sub-task of a specific Web-based service. A dialog contains one or more dialoglets.
5. **Service.** A service is composed of a sequence of dialogs.

The objective of the concept of the event graph is to structure the service design by using the abstractions listed above. In practice, each of these play an important role in service design. By providing the event graph for describing these abstractions, the vocabulary of a designer's informal design practices can be matched. This makes it easy for a designer to map its vocabulary to the abstractions, both in terms of formalizing an informal specification and in terms of communicating the results to other stakeholders.

4. Integration in the Web Service Environment

The service logic is implemented using the event graph. The event graph allows the composition of a set of Web service. The event graph interpreter SPU integrates the Web services by communicating with them using SOAP. SOAP is a XML based protocol that transmits messages and method calls in a distributed computing environment. It describes a message format for the application-to-application communication for distributed applications. A SOAP method call, which calls a method in a distant object, can be divided in a call and a response message.

Unlike current component standard, Web Services are not accessed via object-model-specific protocols, such as the distributed Component Object Model (DCOM), Remote Method Invocation (RMI), or Internet Inter-ORB Protocol (IIOP) [7]. Web Services are usually accessed with standard Internet technologies, such as Hypertext Transfer Protocol (HTTP) and Extensible Markup Language (XML). These characteristics guarantee interoperability, which forms the basis of method calls across over hardware and software boundaries.

If a set of Web services is offered over the Internet by means of SOAP, the question arises, whether to assign an own URL to each service, to each instance of a services, or whether to use only a single URL for all services, and to add suitable arguments in the SOAP method call, which identify a specific service. The following variations are at disposal:

1. A communication end point for each service,
2. An end point per service (all instances of a services share the same communication end point),
3. A single communication end point for all services.

The use of a single URL per service appears semantically correct. However, this means that clients, which access a Web service, must implement and use a specific interface for each service. At first, this appears to be no problem. Yet, many applications use a multiplicity of interfaces, and thus their management becomes unwieldy. In order to master the management and use of interfaces for a multitude of Web services and to

handle calls to Web services transparently, we decided to implement a single communication end point for all Web services of a administrative domain. The result of this consideration is the design of an architecture based on the structural design patterns Proxy, Facade [5] and on the Broker architecture pattern [3]. These patterns help to represent a variety of structures by a central abstraction.

4.1. The SOAP Infrastructure

The main concern is to enable the implementation of a Web service method call on the client-side as uniformly as possible and thus to keep the writing of services as comfortable as possible. A SOAP broker, which offers a uniform interface by means of the Facade pattern for all services, manages the services, for which it offers the uniform interface. A client-side proxy adopts the SOAP based communication with the Web service (Figure 4).

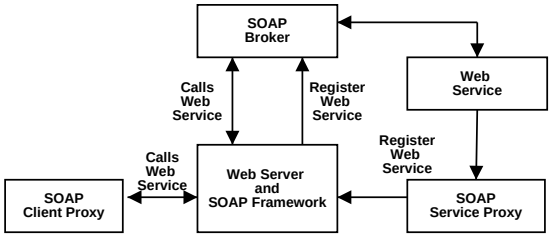

Figure 4: SOAP proxy and broker infrastructure.

The SOAP broker is a Web service that registers with the SOAP framework. Services register themselves with the SOAP broker over the SOAP service proxy. Clients use services, which have registered before with the SOAP broker. The SOAP broker implements a facade, which offers a uniform interface. The following paragraph describes in detail the SOAP infrastructure.

4.2. SOAP Client Proxy

On the client side, the client communicates over a SOAP client proxy with the service. The Proxy pattern serves as means to hide from the client the communication details with the communication end point [3]. Tools exist already, which create based on the Web Service Description Language (WSDL) a Web service as a proxy. The proxy created from WSDL acts as a local representative for the distant Web services. This solution becomes inefficient and the administration of the proxies unwieldy if a large number of proxies is used. Instead, the Facade pattern offers a uniform interface for a set of Web services. This results in a single proxy offering a single interface for a set of services. Fewer interfaces must be managed.

4.3. SOAP Broker

The SOAP broker implements the Facade pattern. The SOAP broker is the only service, which is registered with the SOAP frameworks of the Web Server. It represents the only communication end point for all Web services. Unlike the SOAP broker, all Web services register with the SOAP broker and not with the SOAP framework. Client applications call a Web service via the SOAP broker. The SOAP broker transfers the call to the target Web service.

The application of the Facade pattern results in a significant reduction of the interfaces. A disadvantage of this solution is the loss flexibility. The SOAP broker represents a facade for a set of services. Since the facade offers a uniform interface, this may lead to a loss of flexibility. The consequence is that a service must adapt its interface to comply with the SOAP broker facade. The advantages are various. With the help of the Facade pattern a uniform interface is offered for a set of services. This simplifies the use of Web services. Additionally, the Facade pattern reduces dependency between client and Web service. It decouples the use from its implementation so that both can vary independently. The Broker pattern is used to implement distributed software systems with decoupled components [3]. The SOAP broker is responsible for the coordination of communication with the SOAP client and manages the registered services. Clients specify the service, which they would like to use in the method call as parameter. The SOAP broker passes the call of the clients to the specified Web service and transmits the response to the client.

5. Results

As a sample application scenario, we have implemented Web Office; a set of services related to the well-known office application domain. The objective is to offer typical office tasks such as email, address books, personal documents, etc. on the Web so that they are easily accessible and manageable by a multitude of Internet clients in a heterogeneous environment. Our application relies on Web service technology where each task is implemented as a Web service. We used Sun Microsystem's Web Services Developer Pack (WSDP) as underlying infrastructure for developing the Web services, where SOAP serves as middleware for Web service communication and Apache Tomcat as Web server to deploy services on the Internet. The client side of our application is a Web front end that represents the online office system using Servlet technology for the mediation between services and client interaction.

5.1. The Web Office Application

The Web Office application is service-oriented and relies on the composition of distributed services using EG-XML. The functionality of our office application consists of five services.

1. **Authentification Web service.** The Authentification web service is responsible for the user administra-

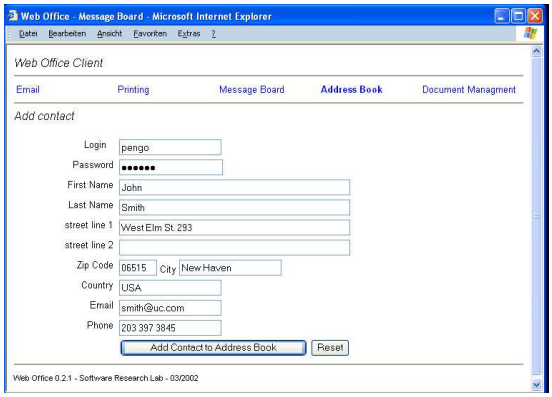

Figure 5: Address Book Web Service.

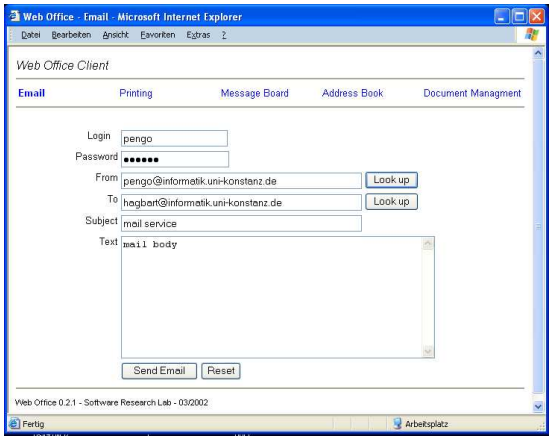

Figure 6: Email Web Service.

tion. New online office users register with the authentification server submitting username and password. This Web service is invoked by all other web services, which need to check for user permission and user data before any other transaction takes place.

2. **AddressBook Web service.** The AddressBook Web service offers the storage of addresses in a database on the server. It offers querying and updating of address data, consisting of name, street, city, email etc. An update or query of the address book is only possible after the Authentification service grants permission. (Figure 5)

3. **Email Web service.** The Email Web service allows users to write emails without having to configure their own email client. The service uses a SMTP server for the sending process. The recipient of the mail can be either a full email address, a name or the username of another online office user. If the Email Web service notices that the recipient is not a valid address, it will query the AddressBook Web service for the recipient's email address. The Authentification server will ask for the user's permissions in advance and serves as logging mechanism for security issues. (Figure 6)

4. **MessageBoard Web service.** The MessageBoard

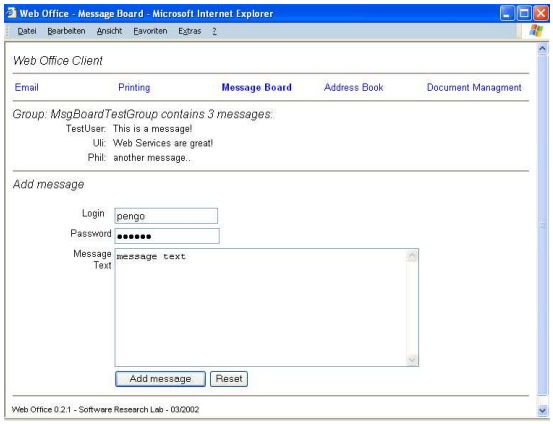

Figure 7: Message Board Web Service.

Web service enables users to post messages and read messages from other users. Messages are stored locally in a database on the server and can be either readable for all users or all users within the same group. (Figure 7)

5. **Printing Web service** The Printing Web service supports the printing of text documents in a LAN. This service also queries the Authentification service for user permissions. Since for printing a physical printer is required, dynamic discovery using registries will be available in future versions.

6. **DocumentManagement Web service** The DocumentManagement Web service enables the user to store documents and request them from a personal folder on the server. This way, documents are available from anywhere and on any device. The Authentification service checks the permissions.

5.2. Protocols and Future Work

In the case of Web services, all procedure calls and its parameters are translated into XML documents by SOAP serializers, which are sent back and forth between client and server using the HTTP protocol. In order to access a web service, a client needs knowledge about the service structure - that is, its parameters and return types. This information is exploited at compile time. This way of communicating with a web service is the ”static web service” usage. The client needs an a priori detailed knowledge about the web service to be used and the connection has to be hardcoded in the client's source. However, this may lead to errors once a web service's interface changes or when it is moved to another communication end point. In future versions of MUSA, the Web service interface may be dynamically discovered. This is especially interesting for the Printing Web service since printing depends on a physical printer, which might not be available in the user's current environment. Once the WSDL document of a Web service has been retrieved, information about the

services' endpoint, its namespace, its parameters, and return types are available and the client can start using the service. This type of web service usage is often described as "dynamic web services". In our office example, all services use the Authentication service. The Email service also may use the AddressBook service to retrieve email addresses from real persons' names. The binding between the services was implemented statically, because the dynamic discovery of the Authentication service each time it is needed would be too much of an overhead. Even if dynamic discovery of services was designed to be achieved purely machine-based, the quality of a service is very hard to assess by software only and the completely dynamic discovery without human interaction is less likely.

6. Related Work

The Web Services Flow Language (WSFL) [4] is an XML-based language for the description of Web Service compositions as part of a business process definition. It was designed by IBM to be part of the Web service technology framework and relies and complements existing technologies such as SOAP, WSDL, XMLP and UDDI. WSFL concentrates on the description how to compose the functionality provided by a set of Web services to implement a particular application and how a set of Web services interact with each other.

XLANG is a XML based language for automation of business processes based on web services. It focuses on the message exchange behavior [11].

MUSA and the event graph EG-XML were originally designed for device-independent service access. EG-XML was extended to support Web services. In contrast to XLANG and WSFL, EG-XML does not yet integrate dynamic discovery of Web services and relies on a simple proxy mechanism to use Web service. This may result in a loss of flexibility, but makes the language very simple and easy to use.

7. Conclusion

The technology is available that allows the development and deployment of Web services. However, service providers need mechanisms that allow workflow automation. We have presented the MUSA system, whose target is to enable service provider to integrate and compose Web services for device independent service access. The implementation of the Web Office application shows that the concept of the MUSA system is a first step towards a flexible and viable solution to Web application development.

Future work will be directed towards the integration of Web service registries into the MUSA system and dynamic exploration and automatic adaptation of Web services.

References

[1] T. Ball, P. Danielson, L. Jagadeesan, R. Jagadeesan, K. Laeufer, P. Mataga, and K. Rehor. Sisl: Several Interfaces, Single Logic. *"International Journal of Speech Technology"*, 3:93–108, June 2000.

[2] K.H. Britton, R.Case, A.Citron, R. Floyed, Y. Li, C. Seekamp, B. Topol, and K. Tracey. Transcoding. Extending e-business to new environments. *IBM Systems Journal*, 40(1):153–178, 2001.

[3] Frank Buschmann et al. *Pattern-Oriented Software Architecture: A System of Patterns. "*John Wiely and Sons", West Sussex, England, 1996.

[4] IBM Software Group Frank Leymann. Web Services Flow Language (WSFL 1.0), May 2001.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *"Design Patterns - Elements of Resuable Object-Oriented Software"*. "Addison Wesley", Reading, Mass., 1995.

[6] K.M. Goeschka and R. Smeikal. Interaction Markup Language - An Open Interface for Device Independent Interaction with E-Commerce Applications. In *Proceedings of the 34th Hawaii International Conference on Systems Sciences*. IEEE Computer Society, 2001.

[7] Mary Kirtland. A Platform for Web Services. *Microsoft Developer Network*, January 2001.

[8] G. Menkhaus. Architecture for client-independent web applications. In *Tools Europe*, pages 32–40, Zuerich, Switzerland, March 2001.

[9] Oracle. Oracle mobile online studio, developer's guide, 2001.

[10] Richard Taylor, Kari A. Nies, Gregory A. Bolcer, Craig A. Macfarlane, and Kenneth M. Anderson. Chiron-1: A software architecture for user interface development, maintenance, and run-time support. *ACM Transaction on Computer-Human Interaction*, 2(2):105–144, June 1995.

[11] Satish Thattle. XLANG - Web Service For Business Process Design, 2001.

[12] W3C. Device Independence Working Group Charter, 2001.

[13] K. Wang. An event driven model for dialogue systems. *Proceedings of the International Conference of Spoken Language Processing*, 2:393–396, 1998.

[14] Yoram Wind. The Challenge of Customerization in Financial Services. *Communications of the ACM*, 44(6), 2001.