Towards Context-Aware Adaptable Web Services

Markus Keidl
Universität Passau
94030 Passau, Germany
keidl@db.fmi.uni-passau.de

Alfons Kemper
Universität Passau
94030 Passau, Germany
kemper@db.fmi.uni-passau.de

ABSTRACT

In this paper, we present a context framework that facilitates the development and deployment of context-aware adaptable Web services. Web services are provided with context information about clients that may be utilized to provide a personalized behavior. Context is extensible with new types of information at any time without any changes to the underlying infrastructure. Context processing is done by Web services, context plugins, or context services. Context plugins and context services pre- and post-process Web service messages based on the available context information. Both are essential for automatic context processing and automatic adaption of Web services to new context types without the necessity to adjust the Web services themselves. We implemented the context framework within the ServiceGlobe system, our open and distributed Web service platform.

Categories and Subject Descriptors

H.m [**Information Systems**]: Miscellaneous; H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Web-based services*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Distributed systems, Information networks*

General Terms

Design, Human Factors

Keywords

Web services, information services, context, service platform, extensibility, automatic context processing, extensible framework

1. INTRODUCTION

Today, consumers have several ways to access information services on the Internet, e.g., browsers on desktop computers, PDAs, or cell phones. As the trend to an increasing number of ubiquitous, connected devices—called pervasive computing—continues to grow, the heterogeneity of client capabilities and the number of methods for accessing information services also increases. Nevertheless, consumers expect Web services to be accessible from all of these devices in a similar fashion. They also expect that Web services are aware of their current environment, e.g., the type of device they are using, their preferences, or their location. Generally, this kind of information is called *context*.

More precisely, in our work context constitutes information about clients and their environment that may be used by Web services to provide clients with a customized and personalized behavior. So, context contains, e.g., a consumer's name, address and current location, the type of client device (hard- and software) the consumer is using, or all kinds of preferences regarding the communication, the format of the Web services' replies, or—in case of information services—the maximum volume of data that should be returned. Web services use such context information to adjust their internal control flow as well as content and format of their replies.

In this paper, we present a context framework that facilitates the development and deployment of context-aware adaptable Web services. The framework consists of two main parts: a distributed infrastructure, which transmits context between clients and Web services and manages the context processing, and the context types, which are the supported types of context information and which are extensible at any time.

The actual context processing is done by three components: Web services themselves, context plugins, or context services. Context plugins and context services are provided by the context framework, and they pre- and post-process Web service messages based on available context information. Both components are essential for automatic context processing, i.e., for processing context without the support of the original Web services, and automatic adaption of Web services to new context types.

Context plugins are basically Java objects implementing a dedicated interface. They are loaded by the service platform during startup, and they support locally executed Web services, i.e., a context plugin cannot be used if it is not locally available. Context services, on the other hand, are Web services (implementing a special interface defined using the WSDL standard) and they might be available anywhere on the Internet. They provide similar functionality as context plugins, but need not be locally available.

Our context framework has several advantages: The set of context types is extensible at any time without any changes to the underlying infrastructure. By adding appropriate context services and/or context plugins, new context types are used instantly and automatically. If using these new context types is achieved by pre- and post-processing Web service messages, the implementations of Web services need not be adjusted. That way, Web services may even utilize context information that was unknown at their development time.

We implemented the context framework within the ServiceGlobe system [22, 19, 20], our open and distributed Web service platform. Information about the current implementation status of the context framework is given in Section 4.5. The context framework as well as ServiceGlobe are fully implemented in Java Release 2 and are based on standards like XML, SOAP, UDDI, and WSDL. ServiceGlobe supports mobile code, i.e., Web services can be distributed on demand and instantiated at runtime at arbitrary Internet servers

WWW2004, May 17–20, 2004, New York, New York, USA.
.

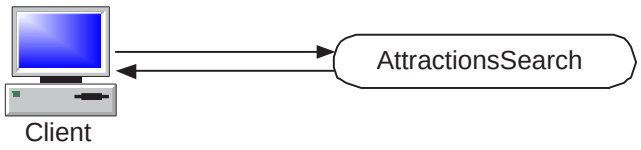

Figure 1: Example Scenario: No Context Processing

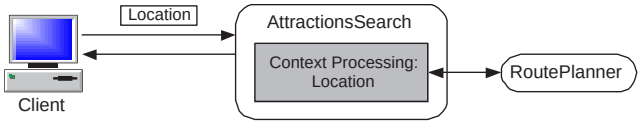

Figure 2: Example Scenario: Internal Context Processing

participating in the ServiceGlobe federation. Also, it offers all standard functionality of a service platform like SOAP/XML communication, a transaction system, and a security system [37].

The remainder of this paper is structured as follows: Section 2 presents a motivating scenario that is used as an example throughout this paper. In Section 3, we present a short introduction into Web service standards that are important for our work. In Section 4, our context framework is described. Several types of context information available in our framework are presented in Section 5. Finally, Section 6 gives some related work and Section 7 concludes this paper.

2. MOTIVATING SCENARIO

In this paper, we use an information service scenario from the travel agency area as a motivating example. In the future, the Internet will provide a lot of information services in this area, e.g., Web services for searching flights, hotels, attractions, and so on. These Web services can help, for example, travel agencies, to plan and carry out journeys.[1]

A provider of such services, e.g., the provider of the AttractionsSearch Web service (depicted in Figure 1), allows the use of context information with its services. The provider extends its AttractionsSearch Web service with a component that uses a consumer's location to include driving directions into the Web service's reply (the necessary data is retrieved from an appropriate Web service, e.g., the Web service RoutePlanner). Therefore, the implementation of the AttractionsSearch Web service has to be changed (as depicted in Figure 2).

Consumers are accepting this new feature. So, the provider of another Web service, e.g., HotelsSearch, also wants to enable context processing for its service. Thus, the provider must adjust the implementation of its Web service to utilize location context information. Although the Web services AttractionsSearch and HotelsSearch share the same functionality, both implement their own version of it. Also, both Web service implementations had to be changed. Furthermore, if one of these providers wants to extend its Web service again, e.g., to use context information about the consumers' clients, it must adjust the Web service's implementation a second time.

Therefore, the functionality to process and use context information should not be (deeply) integrated into the Web services themselves. Instead, the different functional duties should be isolated and they should be implemented in separate components. These components should be provided externally (see Figure 3). Their

[1]Travel portals like MapQuest.com are a first step in this direction.

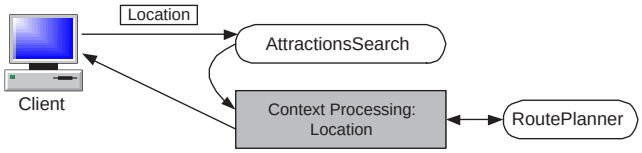

Figure 3: Example Scenario: External Context Processing

usage must be transparent for Web services and they must be used automatically if a Web service's request contains context information. Additionally, they must provide a generic solution, i.e., the same component must be usable for a variety of Web services, e.g., the AttractionsSearch Web service as well as the HotelsSearch Web service.

Our context framework presents a solution for the problems outlined above as it has precisely the desired properties: It is transparent for Web services, context processing components are automatically deployed, and these components can be used generically (of course, they must be implemented properly). In Section 5, we describe how the above scenario can be implemented with our context framework.

3. WEB SERVICES FUNDAMENTALS

There exists a variety of XML-based standards concerning Web services. We will briefly survey the most important ones that are needed to understand this work.

The SOAP Standard

SOAP [31] is an XML-based communication protocol for distributed applications. SOAP is designed to exchange messages containing structured and typed data and can be used on top of several different transfer protocols like HTTP, SMTP (Simple Mail Transfer Protocol), and FTP (File Transfer Protocol). The use of SOAP over HTTP is the de-facto standard in the current landscape of Web services.

SOAP itself does not define any application semantics and, therefore, can be used in a broad range of applications. It can be used to simply deliver a single message or for complex tasks like request/response message exchange or RPC (Remote Procedure Call). The XML document in Figure 5 shows the basic structure of a SOAP message, consisting of three parts: an envelope, an optional header, and a mandatory body.

The `Envelope` element is the root element of a message and contains the other two elements `Header` and `Body`. The `Header` element of a message offers a generic mechanism to extend the SOAP protocol in a decentralized manner. This is used for extensions like Web Service Security [17]. We defined a SOAP header extension to transmit Web service context within SOAP messages, see Section 4. The `Body` element of the message contains the payload of the message.

The UDDI Standard

UDDI (*Universal, Description, Discovery and Integration*) is designed to "provide a platform-independent way of describing services, discovering businesses, and integrating business services using the Internet" [40]. Four main data structures can be identified, which constitute the basic schema: businessEntity, businessService, bindingTemplate, and tModel. While the first three data structures form a hierarchy, the tModels can be seen as an independent structure providing technical fingerprints of services, concepts, and ideas (see Figure 4).

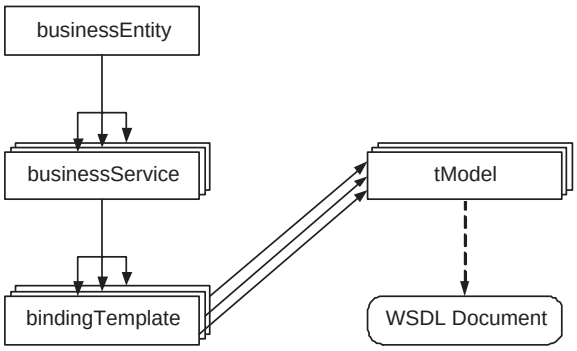

Figure 4: UDDI Data Structures

The businessEntity data structure contains data about an entire company or party that offers a family of services. A businessEntity registers several services. The businessService structure contains information about a particular service. It contains also one or more bindingTemplates specifying binding information for the service. The most important component of the bindingTemplate structure is the access point of a service, i.e., the actual URL, phone number, etc., by which the service can be invoked. A bindingTemplate may have several references to tModels.

As a technical fingerprint, tModels describe various concepts and classifications. A tModel may contain a link to a WSDL document that specifies the signature of a service in detail. Besides these service-classification-oriented tModels, also concept-oriented tModels like geographical locations or industry codes are possible.

The WSDL Standard

WSDL (*Web Service Description Language*) [6] is an XML-based language to describe the technical specifications of a Web service, including the operations offered by the Web service, the syntax of input and output documents, the communication protocol to use for communication with the service, and some further information. The exact structure of a WSDL document is out of the scope of this paper.

4. CONTEXT FOR WEB SERVICES

In the literature, there are a number of different definitions and uses of the term context [32, 8, 36, 13, 33, 14, 34]. In our work, context encompasses all information about the client of a Web service that may be utilized by the Web service to adjust execution and output to provide the client with a customized and personalized behavior.

Context is different from the parameters of a Web service: First of all, the same context information is interesting for a number of Web services whereas parameters are only used by exactly the Web service they belong to. As a consequence, context can often be evaluated automatically, e.g., by the service platform. This simplifies the development of Web services as the evaluation of such context does not need to be integrated into the Web services themselves. A further difference is that context information is optional whereas parameters are mandatory. Context information does not need to be passed to a Web service and if it is, the Web service does not necessarily need to understand and process it.

4.1 Context Infrastructure

In our framework, context is transmitted as a SOAP header block within the SOAP messages that Web services receive and send (see Figures 5 and 6 for an example). Legacy Web service platforms that do not support context information may ignore this specific header block (in conformity with the SOAP standard).[2] As context information is optional, Web services executed on legacy platforms are nevertheless able to process such requests, but they lose the benefit of the context information.

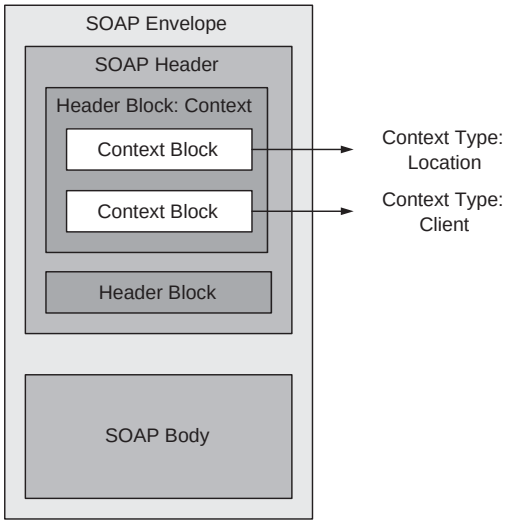

Figure 5: Context within a SOAP Message

Analogous to a SOAP header, context consists of several *context blocks*. Each context block is associated to one *context type*, which exactly defines the type of context information the context block is allowed to contain. At most one context block is allowed for a specific context type, i.e., no two context blocks can be associated to the same context type. The context in Figure 5 contains two context blocks: one associated to the context type Location (with information about a consumer's current location) and another one of context type Client (with information about the client's hard- and software). More information about the context types supported by our context framework is given in Section 5.

Every context type has a unique *context type identifier*. This identifier is equal to the qualified name of the XML elements that represent corresponding context blocks, i.e., identifiers must be valid qualified names. The qualified name of an XML element is composed of its namespace and element name. For example, in Figure 6 there is a context block element `Client` with namespace `http://sg.fmi.uni-passau.de/context`. Consequently, the associated context type is `http://sg.fmi.uni-passau.de/context:Client`. We omit the namespace part in the following and reference context blocks (and context types) only by the corresponding element names.

Context types are basically used to distinguish context blocks. For example, if a Web service wants to access information of its context, it specifies the type of context information it wants to retrieve, i.e., a context type identifier. The context infrastructure determines the corresponding context block using this identifier and returns it. For the infrastructure, the knowledge of a context type identifier is sufficient for allowing access to the context type and for guaranteeing that a context contains at most one context block of any context type.

[2] If the attribute `mustUnderstand` is set in a context header block, Web service platforms must process the context or fail processing the message [31].

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header>
    <Context xmlns="http://sg.fmi.uni-passau.de/context">
      <Location>
        <address useType="Office">
          <addressLine keyName="Street" keyValue="60">Innstrasse 33</addressLine>
          <addressLine keyName="City" keyValue="40">D-94032 Passau</addressLine>
        </address>
      </Location>
      <Client>
        <DeviceDefaults>http://example.com/context/device/PDA</DeviceDefaults>
        <Hardware>
          <ScreenSize>320x200</ScreenSize>
          <IsColorCapable>Yes</IsColorCapable>
        </Hardware>
      </Client>
    </Context>
  </env:Header>
  <env:Body>
    <!-- serialized object data -->
  </env:Body>
</env:Envelope>
```

Figure 6: A SOAP Message with a Context Header Block

Though the infrastructure does not require the validation of a context block's content against the schema of its context type, it does provide the possibility for it, especially to free Web services themselves from this task. For this purpose, a context type has to be published in a UDDI repository as a tModel.[3] In the tModel, the identifier of the context type must be specified. Also, if content validation should be possible, an XML schema document must be referenced that defines the schema to which corresponding context blocks have to conform to. If the validation of a context block fails, the context block is marked as incorrect and not used further on.

Figure 7 gives an example of a tModel which defines the context type Location. The context type's identifier `http://sg.fmi.uni-passau.de/context:Location` is specified as a keyedReference in the tModel's identifierBag. In the categoryBag, it is specified that the tModel is derived from the tModel `ContextType`, which serves as base tModel. The overviewDoc entry contains a URL that links to the XSL schema document which defines the context type's schema.

4.2 The Life-Cycle of Context Information

The life-cycle of a Web service's context, illustrated in Figure 8, starts at a client's site: First, the client gathers all relevant context information and inserts it into the SOAP request as a context header block. Then, the request is sent to the host executing the Web service.

After the request was received by a Web service platform, the context is extracted by the context framework and provided to the invoked Web service as its *current context*. During its execution, the Web service can access and modify this current context using the Context API provided by the framework. For example, in Figure 8, the first context block is modified (illustrated by the color change to gray) and a new context block (the third, black rectangle) is inserted.

When the Web service invokes another service during its execution, its current context is automatically inserted into the outgoing request. The response to such a request may also contain context information. In this case, the Web service can extract the interesting parts of the context data from the response and insert them into its current context.

After the Web service's termination, its (possibly modified) current context is automatically inserted into its response and sent back to the invoker. If the invoker is a client, as in Figure 8, it may integrate portions of the returned context into its local context (for use in future requests). Furthermore, the returned context may be utilized by the client to adjust the Web service's response.

In the entire context life-cycle, potential privacy and security issues have to be considered. For example, clients should be able to specify what modifications a Web service is allowed to perform on the context and also what parts of the context the Web service is allowed to insert into requests to other Web services. Furthermore, the policies must state if and how a client is allowed to modify a its local context. Partly, these issues are considered in Section 4.4. Elaborate privacy and security policies are out of the scope of this paper.

4.3 Context Processing

In our context framework, we distinguish two types of context processing: explicit processing by Web services or clients and automatic processing by the context framework.

Explicit Context Processing

Explicit processing means that Web services or clients directly access the context contained in a SOAP message using the framework's Context API and, consequently, that the context processing functionality is part of their code. Thus, there is a tight coupling between such Web services and clients and the context types they are able to process. That is, such Web services and clients can only utilize context types that were known and integrated at their development time. A further disadvantage is the additional coding effort, as the same or at least similar context processing functionality is basically contained in many Web services and clients. Also, a strict separation of concerns is missing. An advantage of this type of context processing is that Web services and clients have full control over how the context information influences their control flow and their replies. Additionally, they can access the context information to modify it.

An example for explicit processing is the client we implemented. It processes the returned context information to finally adjust the response to its device capabilities, e.g., by using stylesheet information inserted into the returned context.

Automatic Context Processing

Automatic context processing means that SOAP messages are pre- and/or post-processed (from a Web service's point of view) based

[3]These tModels are also used by other parts of the context framework, see Sections 4.4 for further details.

```
<tModel>
  <name>Location Context Type</name>
  <overviewDoc>
    <overviewURL useType="xmlSchema">http://sg.fmi.uni-passau.de/context/context-location.xsd</overviewURL>
  </overviewDoc>
  <identifierBag>
    <keyedReference keyName="ContextTypeID"
                    keyValue="http://sg.fmi.uni-passau.de/context:Location"
                    tModelKey="uddi:serviceglobe:identifier:contexttype"/>
  </identifierBag>
  <categoryBag>
    <keyedReference keyName="derivedFrom:ContextType"
                    keyValue="uddi:serviceglobe:categorization:contexttypes"
                    tModelKey="uddi:uddi.org:categorization:derivedFrom"/>
    <keyedReference keyName="uddi-org:types" keyValue="categorization"
                    tModelKey="uddi:uddi.org:categorization:types"/>
  </categoryBag>
</tModel>
```

Figure 7: The tModel for the Location Context Type

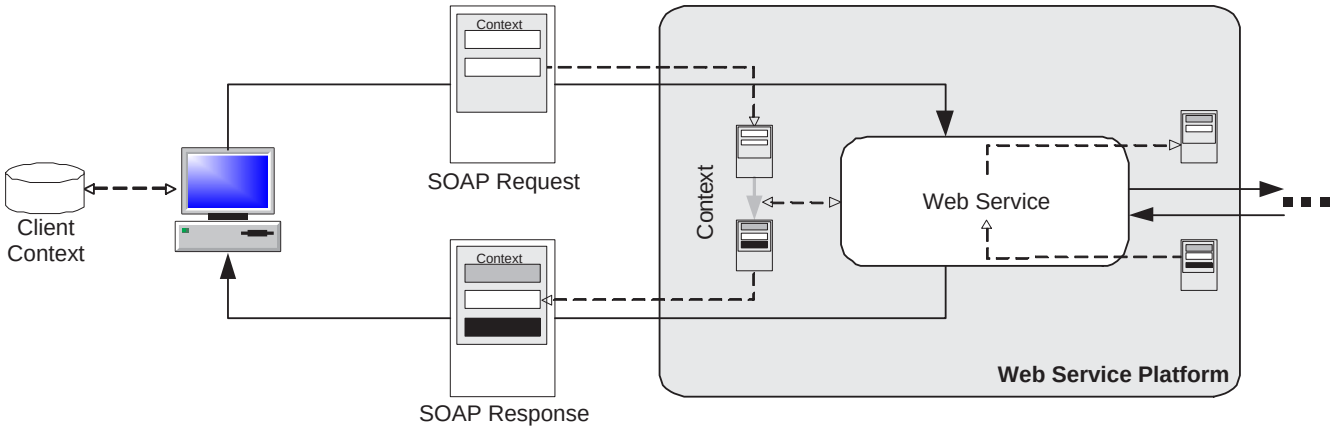

Figure 8: Context Life-Cycle

on the context information they contain. Automatic context processing is done by the context framework, i.e., Web services are not involved in it. As a consequence, the context processing task is moved from the Web services to the service platform and the coding effort for Web services is reduced. A disadvantage is, of course, that only a Web service's requests and responses can be modified, its internal process flow cannot be adjusted by this means.

There are four different points in time at which context is processed automatically (see Figure 9): First, the incoming SOAP request of an invoked Web service is pre-processed (1), based on the context in the request. Furthermore, whenever the Web service invokes other services (using the invocation manager), outgoing messages, i.e., requests to other services, are post-processed before they are actually sent (2) and incoming messages, i.e., responses to outgoing requests, are pre-processed before they are returned to the Web service (3). Finally, the outgoing response of the invoked Web service is post-processed (4), based on the service's current context (which might be a modified version of the received one). To sum up, this means that all messages to and from an invoked Web service can be modified based on context information.[4]

Of course, modification of messages also implies that the messages' content can be modified, e.g., the content of a Web service's reply. For example, in our demonstration at the EDBT'04 conference [18], we used automatic context processing to convert the price information within a Web service's reply content into the currency of the consumer's location.

In the following, we refer to the procedures when a Web service's message is pre- or post-processed as *context operations* and we call them *PreprocessRequest* (1), *PostprocessMessageRequest* (2), *PreprocessMessageResponse* (3), and *PostprocessResponse* (4), respectively.

In every context operation, automatic context processing is done by processing the context blocks of the SOAP message's context in arbitrary order. Consequently, during the processing of a context block, no assumptions can be made on the processing state of any other context block, i.e., if some other context block has already been processed or not.

After selecting an arbitrary, not yet processed context block, the context framework determines its context type. For every context type, the context manager (see Figure 9) manages a list of components capable of processing context information of the associated type. The actual processing of the selected context block is delegated to these components, which are described in the following section. There are several ways to configure which of these components should actually be used for processing and in what order. Details are given in Section 4.4.

Components for Automatic Context Processing

The context framework delegates automatic context processing to two types of components, as shown in Figure 9: context plugins and

[4] In our work, we assume a request-response message exchange pattern as, e.g., used for remote procedure calls (RPC). If a different exchange pattern is used, e.g., if a Web service does not return a response, the corresponding processing steps are omitted.

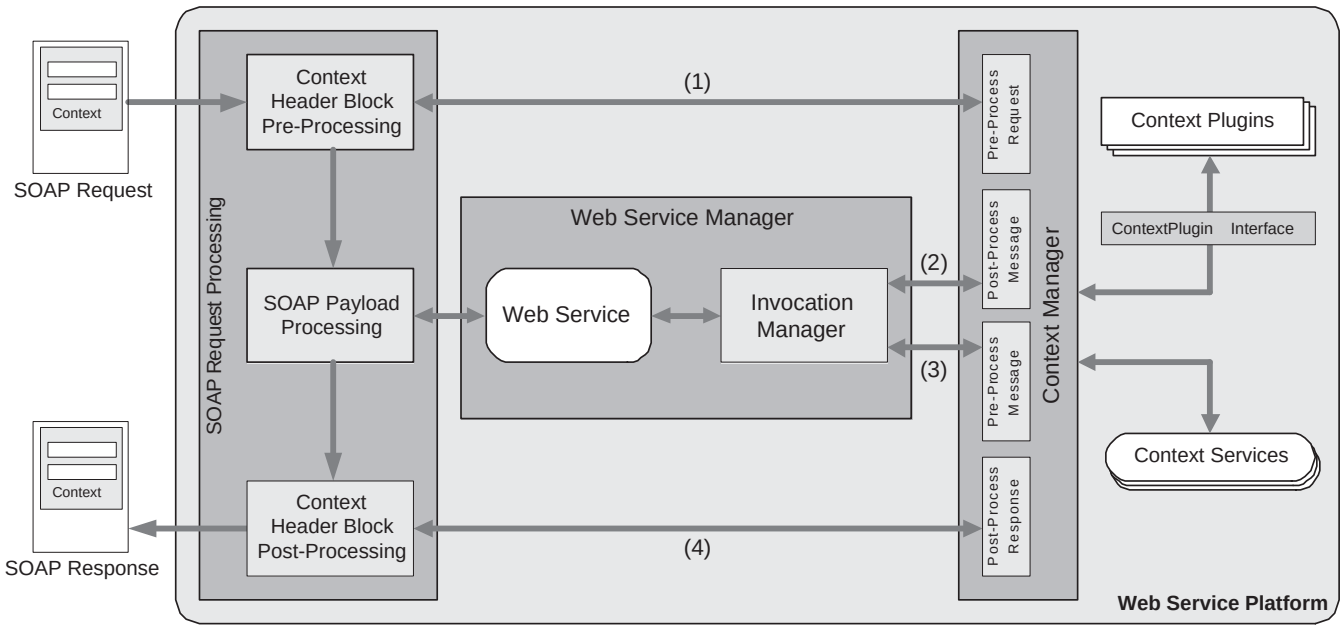

Figure 9: Components for Context Processing

context services. *Context plugins* are basically Java objects implementing a special Java interface. They must be installed locally at a host and they are loaded by the service platform during startup. *Context services* are Web services that implement the `ContextService` interface. This interface, defined using the WSDL standard, describes the four context operations a context service should implement. A UDDI tModel `ContextService` is provided that links to the WSDL document with the interface description. Context services should refer to this tModel when published in UDDI. In their bindingTemplate entries, context services may also specify which context operations they actually support.

Every component, i.e., every context plugin and every context service, is associated with one context type and it is used to process context information of this type only. When invoked, a component requires two parameters: The first one is a context block of the component's associated context type. The second one depends on the context operation that is invoked on the plugin: It is the request to a Web service, the service's response, an outgoing request of the Web service (to invoke another Web service) or an incoming response of such an invoked service.

Context services are very similar to context plugins as they basically implement the same interface. Both enable automatic processing of context information and are essential for the easy extensibility of context. On the other hand, context services are Web services. They need not be installed locally, as context plugins must be, but can be available anywhere on the Internet. If implemented as mobile Web services, as it is possible in the ServiceGlobe system, context services may be loaded dynamically and on demand and they may be executed on the local host.

Besides the reduction of the coding effort, context plugins and context services have the advantage that they constitute a generic solution. A context plugin or service for a specific context type can be used for a variety of Web services without any need for specific adjustments depending on the Web service they are used for. Even more important, Web services can now utilize context types they do not support themselves. This is also beneficial for legacy Web services which cannot be modified.

4.4 Context Processing Instructions

In Section 4.3, several components for context processing were introduced. A context block in a SOAP request is possibly processed by context plugins, context services, and/or the invoked Web service itself. Therefore, rules of precedence are required and also information about which components should actually be used for processing. Additionally, the same context block is probably not only processed at a Web service's local host, but also at hosts on which the Web service invokes other services, as context is inserted into outgoing messages.

Precautions have to be taken to prevent these problems and ambiguities. In our framework, *context processing instructions* are used for this purpose.[5] If no context processing instructions are specified, defaults are used: Context plugins are invoked as configured locally at the service platform (default here is alphabetical order according to the class name). Context services are not used by default. Invoked Web services themselves can always process their context information.

Context processing instructions are specified within a `ContextProcessingInstructions` element, as depicted in the example in Figure 10 (for the moment, we ignore the enclosing UDDI elements). For every context type, they can contain at most one `ContextType` subelement. Within this element, instructions for the corresponding context type are specified. Currently, component instructions and processing guidelines can be specified. But we are still investigating these issues and we are going to consider them in more detail in future work.

Component Instructions

With component instructions, context plugins and context services that should be used for processing context information of the enclosing context type and their execution order are specified. Con-

[5] Apart from their name, context processing instructions and XML processing instructions are very different. Context processing instructions are ordinary context information, just like location or client context information.

```
<tModelInstanceInfo tModelKey="uddi:serviceglobe:context:processing-instructions">
  <instanceDetails>
    <instanceParms><![CDATA[
        <?xml version="1.0" encoding="utf-8" ?>
        <pi:ContextProcessingInstructions xmlns:pi="urn:serviceglobe:context">
          <pi:ContextType ID="http://sg.fmi.uni-passau.de/context:Location">
            <pi:ContextService>
              <pi:AccessPoint useType="http">http://example.com/services/CurrencyConverter</pi:AccessPoint>
              <pi:ContextOperations>post</pi:ContextOperations>
            </pi:ContextService>
            <pi:ProcessingGuideline>
              <pi:ServiceHost>Next</pi:ServiceHost>
              <pi:ComponentTypes>ContextPlugin+ContextService</pi:ComponentTypes>
            </pi:ProcessingGuideline>
          </pi:ContextType>
          <pi:ContextType ID="http://sg.fmi.uni-passau.de/context:Client">
            <pi:ContextPlugin>serviceglobe.context.plugins.StylesheetFinder</pi:ContextPlugin>
          </pi:ContextType>
        </pi:ContextProcessingInstructions>]]>
    </instanceParms>
  </instanceDetails>
</tModelInstanceInfo>
```

Figure 10: Context Processing Instructions in the tModelInstanceInfo entry of a bindingTemplate

text plugin instructions must be defined using `ContextPlugin` elements, context service instructions using `ContextService` elements. If several context plugins and services are specified, they are executed in the same order as they are specified.

In the example in Figure 10, a context service with access point `http://example.com/services/CurrencyConverter` is used for processing Location context. The context framework supports several types of access points: SOAP-HTTP URLs (as in the example), ServiceGlobe URLs, or bindingTemplate UUIDs referencing context services published in a UDDI repository. The instructions in the example also state that only the context operation *PostprocessResponse* (keyword `post`) should be invoked. Other keywords are `pre` for *PreprocessRequest*, `postmessage` for *PostprocessMessageRequest*, and `premessage` for *PreprocessMessageResponse*. Furthermore, the context plugin StylesheetFinder (specified by its class name) is used for processing Client context (for a description of this plugin, see Section 5).

Processing Guidelines

With processing guidelines, the types of components that should be used to process a certain context block are specified as well as the hosts at which the context block should be processed. A processing guideline is defined using the `ProcessingGuideline` element (which must be a child element of the `ContextType` element for which the guideline is specified). Figure 10 shows an example. The child element `ServiceHost` specifies the host at which corresponding context blocks should be processed, and the child element `ComponentTypes` specifies the actual component types that should be used for processing.

Possible values of the `ServiceHost` element are `next` and `all`. (The meaning of this element is similar to the `role` attribute that can be used in SOAP header blocks [31].) If `next` is specified, only the next host should receive and process a context block. For that reason, the context block is not included into outgoing requests of the invoked Web service. When using `all`, the corresponding context block is inserted into outgoing requests and all hosts that receive it may also process it.

Possible values that can be used within the `ComponentTypes` element are `ContextPlugin` and `ContextService`. The meaning of them is obvious. Both values can be combined using the + operator. In this case, both components are used sequentially for processing. In the example of Figure 10, context plugins are applied first. Then, context services are invoked.

Without any processing guidelines, defaults are used: `next` for `ServiceHost` and `ContextPlugin` for `ComponentTypes`.

Web services themselves can always access and process any context block passed to them, even if a context block was already processed by a context plugin or context service. Obviously, it would be possible to remove a processed context block from the context to prevent Web services from processing it a second time. But then, the context block would only be used to modify the Web services' messages (due to the constraints of context plugins and context services).

Providing Context Processing Instructions

There are several possibilities to make context processing instructions available to the context framework. The first two possibilities are especially useful if it should be enforced that only particular context plugins and context services are used to process context blocks of certain context types.

First of all, the *context itself* can contain context processing instructions. For this purpose, the instructions, e.g., the `ContextProcessingInstructions` element of the example in Figure 10, are inserted into the context as a self-contained context block. The context block is then processed by a special context plugin provided by the context framework.

Second, a *Web service's UDDI metadata* may be annotated with context processing instructions. Therefore, the bindingTemplate entry of the Web service must contain a tModelInstanceInfo entry that specifies the instructions. An example is shown in Figure 10. The context processing instructions are contained within the `instanceParms` element (as a string, according to the UDDI standard[6]). The format of the context processing instructions is the same as if used within the context of a SOAP request.

Providers or developers of Web services can use this second option to specify context services that should be used for processing certain context blocks. Even operators of hosts executing Web services may utilize UDDI this way to force the use of certain context plugins or context services with Web services executed at their hosts.

The third possibility is different to the preceding ones: Instead of relying on explicitly specified context processing instructions,

[6]According to [40], the content of an `instanceParms` element must be of type string. The suggested format is a namespace-qualified XML document.

```
<businessService>
  <name>CurrencyConverterContextService</name>
  <categoryBag>
    <keyedReference keyName="ContextService" keyValue="true"
                    tModelKey="uddi:serviceglobe:interfaces:contextservice"/>
    <keyedReference keyName="ContextType"
                    keyValue="http://sg.fmi.uni-passau.de/context:Location"
                    tModelKey="uddi:serviceglobe:categorization:contexttypes"/>
  </categoryBag>
</businessService>
```

Figure 11: UDDI Metadata of a Context Service

the context framework uses available UDDI metadata to automatically determine available context services for context processing. Context plugin instructions and processing guidelines cannot be determined that way.

Just as ordinary Web services, context services may be published in UDDI. Every context service that should be found by the context framework when searching for appropriate context services must be associated to two tModels: the tModel `ContextService`, which marks a Web service as context service, and the tModel `Context-Type`. The latter's association contains a parameter that specifies the context type the context service is able to process. For an example, see Figure 11. When processing a context block, the framework queries UDDI for all services associated to both tModels and having the correct parameters. As all of these services provide semantically equivalent functionality, one of them is chosen randomly. For the future, we are going to consider the utilization of ServiceGlobe's dynamic service selection [21, 22] in this process.

4.5 Implementation Status

A prototype implementation of our context framework within the ServiceGlobe system is completed and its current state is as described in this work. There exist also parts that are still subject to change, e.g., context processing instructions, as we are still investigating these issues. A number of context types, context services, context plugins, and example Web services have been implemented successfully (some are presented in Section 5). But we are still investigating further possible context types and usage scenarios.

We also finished the implementation of clients for different types of client devices, e.g., Java-based clients for PDAs and cell phones. They are used to demonstrate the usefulness and the advantages of context information based on example Web services of our motivating scenario. Furthermore, we implemented a Web-based client. With this client, the influence of various types of context information can be investigated in more detail. A demonstration of our context framework and these clients is presented at the EDBT'04 conference [18].

5. CONTEXT TYPES

In this section, several context types are explained that are provided by our context framework. Our framework is not limited to this set of context types. As it is extendable at any time, new context types can be added by inserting corresponding context information into the context and providing appropriate context services and/or context plugins for processing it. Neither the context framework nor Web services must be adjusted for this.

An important context type is *Location*. It contains information about the consumer's current location, e.g., the consumer's current address, GPS coordinates, country, or local time and time-zone. An example was shown in Figure 6. Location context may also include semantic location information, e.g., that a consumer is currently at work. We implemented, e.g., a context service CurrencyConverter that converts price information in a Web service's reply into the currency of the consumer's location.

Client context information comprises data about a client's device. It includes information about hardware, e.g., processor type or display resolution, as well as software, e.g, operating system or Web browser type and version. An example of such a context block was also shown in Figure 6. Two schemas are supported for this context type: a rather simple one as used in Figure 6 and an RDF-based one as defined in the CC/PP standard [27] of the W3C.

The main purpose of this context type is to allow Web services to adjust their output to the client device's properties. For example, Web services from the information systems area, which often query data from a database management system (DBMS), can use this context information to optimize their database queries and to query only data that can actually be displayed at the client. The Amazon Web service is an example for such a Web service. In its replies, it includes several lengthy customer reviews. If viewing a reply of this Web service on a PDA or cell phone, the inclusion of these reviews is rather pointless as they require too much space. Optimally, the corresponding data should not be retrieved from the DBMS in the first place. But if this is not possible, e.g., because modifications of the Web service are impossible, context services can be used to adjust the reply of the Web service.

We also implemented a context plugin StylesheetFinder which uses the Client context to provide the client with a stylesheet that can be used to format the Web service reply. A Web service must specify XSL stylesheets that should be used for the various client types in its UDDI metadata. Figure 12 shows an example for such metadata. Based on this metadata and the Client context information, the plugin inserts a new context block of type *ReplyProperties*. This context block, see Figure 13 for an example, is processed by clients. They use the specified stylesheet to transform the reply's XML data into HTML.

The *Consumer* context type contains information about the consumer invoking the Web service, e.g., name, email address, and so on. Although it is very important, it can actually be used only by the Web services themselves in a sensible way.

The *Connection Preferences* context type allows to specify properties of the connections to Web services. It was added by implementing a context service ConnnectionPreferencesService. Based on the content of the corresponding context block, the context service compresses and decompresses Web service messages using gzip [7] or the XML compressor XMill [29]. The context service could, e.g., be extended to support encryption, too.

With these context types, the motivating scenario of Section 2 can be implemented as depicted in Figure 14. Although the AttractionsSearch Web service was not modified, the reply that the client receives contains personalized, context-dependent information. Client and Location context information are processed automatically by two context services and one context plugin. After AttractionsSearch generated its reply, the context service DrivingDirections uses the RoutePlanner Web service to insert driving direc-

```
<tModelInstanceInfo tModelKey="uddi:serviceglobe:contexttype:client:stylesheets">
  <instanceDetails>
    <instanceParms><![CDATA[
        <?xml version="1.0" encoding="utf-8" ?>
        <UDDIinstanceParmsContainer xmlns="urn:uddi-org:policy_v3_instanceParms">
          <Stylesheet deviceType="http://example.com/context/device/PDA">
            http://example.org/service-stylesheet-pda.xsl
          </Stylesheet>
          <Stylesheet deviceType="http://example.com/context/device/Desktop">
            http://example.org/service-stylesheet-desktop.xsl
          </Stylesheet>
        </UDDIinstanceParmsContainer>]]>
    </instanceParms>
  </instanceDetails>
</tModelInstanceInfo>
```

Figure 12: UDDI Metadata: Stylesheets for a Web Service's Reply

```
<ReplyProperties xmlns="urn:serviceglobe:context">
  <Stylesheet>http://example.org/service-stylesheet-desktop.xsl</Stylesheet>
</ReplyProperties>
```

Figure 13: A Context Block of Context Type ReplyProperties

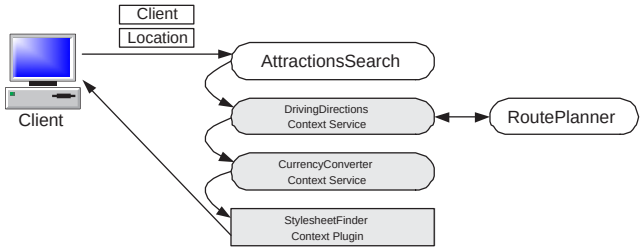

Figure 14: Example Scenario: Context Processing with the Context Framework

tions from the client's location into the reply. Second, the context service CurrencyConverter transforms all price information in the AttractionsSearch's reply into the currency of the consumer's location. Third, the context plugin StylesheetFinder chooses an XSL stylesheet which fits best to the current client device and inserts this information into the reply. After the client received the reply, it uses this stylesheet information to display the XML reply on the specified device in the appropriate way.

6. RELATED WORK

There are several technologies which are related to our context processing architecture, i.e., the automatic context processing of context by successively invoking context plugins and context services. The Chain of Responsibility design pattern [11], for example, describes how to decouple the receiver of a request from the sender by chaining the receiving objects and passing the request along the chain until an object handles it. On the other hand, in our framework several receivers, i.e., context plugins and services, can process the same request. Aspect-oriented programming (AOP) [24] allows the modification of applications with so-called aspects. Aspects are modular units of functionality which are used across the application's code. They are woven into an application's code at so-called pointcuts, thereby allowing to transparently extend, e.g., objects with new functionality. This is similar to the way Web services are extended with new context processing functionality using context plugins and services. In Java, AOP is supported, for example, by the AspectJ [23] toolkit or in the J2EE application server JBoss [16].

In CORBA, technologies like interceptors or smart proxies can be used to insert new functionality into existing applications. They are, for example, supported in IANA's Orbix [15]. Interceptors and smart proxies have also been integrated in Java RMI [35]. The Java Servlet specification [39] describes filters that could be used to intercept and modify messages. In the Axis framework [2], chains of handlers can be created. Requests and responses are passed along these chains and they may be modified by the chains' handlers.

Although our context framework shares similarities with the above approaches, there are also differences. In our framework, the context information contained within the request determines which context plugins and context services are actually invoked. Context plugins and services are not chained sequentially and not all of them are invoked every time. With context services, our context framework is extensible at runtime, just by adding appropriate context information into requests. Furthermore, context plugins and services selection may depend on the consumer's preferences, which can be integrated into the the client's context as context processing instructions. Thus, we facilitate fine-grained, dynamic control over the context processing.

In the mobile computing area, context has been investigated for several years. Best known are the location-based services. The PLIM framework [33], for example, provides an infrastructure for the distribution and retrieval of location information of (Bluetooth-connected) mobile devices using a publish/subscribe mechanism. [14] presents a context model for pervasive systems based on the CC/PP standard and points out some limitations of this standard. [12] describes a system that builds a dynamic model of the environment where the locations of the environment's objects are updated using location sensors. Also, an event-based monitoring system is provided that allows applications to detect location changes and to query the relationship of objects regarding their location. In Jini [41], a Java lookup service for services, extensions have been proposed to support search attributes that provide context information about services [28].

In the Web service area, there are several research projects that deal with context. The CB-SeC framework [32] is an agent-based architecture that provides service selection based on a rating system for Web services, where these ratings are calculated using so-called context of interest functions and context information about consumers and services. Aura [38] is an architectural framework that models user tasks as coalitions of abstract services. Aura migrates such tasks from one environment to another one, if the user's location changes. Also, tasks can be adjusted if the envi-

ronment changes, e.g., if a provider for a currently used service disappears. In [9], the concept of dynamic bookmarks is presented. Dynamic bookmarks are descriptions of services that are bound to actual services based on a user's location. Clients use a dynamic bookmark service to update their dynamic bookmarks if their location changes. [8] describes a distributed infrastructure to support context-aware applications based on context widgets. Context widgets gather context information from low-level sensors. They may also aggregate and interpret it. Applications use these widgets by subscribing to them. In [30], an architecture is presented which allows to develop applications where the application logic is decoupled of the UI based on an event-graph. With it, UIs for different client types can be developed independent of the implementation of the application itself.

Our focus in this work is on automatic and transparent processing of context and on easy extensibility of context types, not on how the context is stored locally at clients. A distributed storage system for context information is, e.g., presented in [34]. In this system, context data is pro-actively replicated and migrated on mobile devices, dependent on the clients' behavior. In [10], a context service is presented which is basically a storage for context information. Context sources deliver their information to this service. Applications access the context service to query for context information. [13] discusses requirements for a representation format for context information and examines different, existing formats. As result of the discussion, they present a novel, RDF-based representation format. In [36], data about the environment is collected using collections of low-level sensors. This data is analyzed and the context—a set of two-dimensional vectors—is updated accordingly. Using a script language, actions can be defined based on context changes. Actions can, e.g., be commands to applications.

The need for Web service personalization also poses challenges to other areas of computer science. In the information systems area, preferences are gaining noticeable attention [25, 1, 5], as they are a way to support personalization of Web services, especially for information services that often use a DBMS as backend. Preferences, also called soft constraints, are more like wishes: The result of a query should be a perfect match, but a best possible match is also acceptable. For example, [4] shows how quality of service preferences can be considered in distributed query processing. Recently, approaches to integrate preferences into Web services have also been proposed [3, 26].

7. CONCLUSION

In this paper, we presented a context framework that facilitates the development and deployment of context-aware adaptable Web services. We introduced our context model and gave a detailed description of the framework's main parts: the distributed infrastructure, which transmits context between clients and Web services and manages the context processing, and the context types, which are extensible at any time.

We showed how the actual context processing is done by Web services themselves, context plugins, or context services. Context plugins and context services pre- and post-process Web service messages based on available context information. Context processing instructions, specified within the UDDI metadata of Web services or within the consumers' context, are a means to specify the host to which context is transmitted and at which hosts and by which components it is actually processed. We also introduced a basic set of context types that are supported by our context framework.

For the future, we plan to further investigate additional context types and to study context processing instructions in more detail. A further issue of interest are security policies that enable clients to specify which Web services should receive their context and what operations these services are allowed to perform on it.

8. REFERENCES

[1] R. Agrawal and E. L. Wimmers. A Framework for Expressing and Combining Preferences. In *Proc. of the Intl. Conf. on Very Large Data Bases (VLDB)*, pages 297–306, 2002.

[2] Axis Architecture Guide. `http://ws.apache.org/axis/java/architecture-guide.html`.

[3] W.-T. Balke, W. Kießling, and C. Unbehend. Performance and Quality Evaluation of a Personalized Route Planning System. In *Proc. of the Brazilian Symposium on Databases (SBBD)*, pages 328–340, 2003.

[4] R. Braumandl, A. Kemper, and D. Kossmann. Quality of Service in an Information Economy. *ACM Transactions on Internet Technology (TOIT)*, 3(4):291–333, 2003.

[5] J. Chomicki. Querying with Intrinsic Preferences. In *Proc. of the Intl. Conf. on Extending Database Technology (EDBT)*, volume 2287 of *Lecture Notes in Computer Science (LNCS)*, pages 34–51, 2002.

[6] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. `http://www.w3.org/TR/wsdl`, 2001. W3C Note.

[7] P. Deutsch. GZIP file format specification version 4.3. RFC 1952, 1996. Network Working Group.

[8] A. K. Dey, D. Salber, and G. D. Abowd. A Context-based Infrastructure for Smart Environments. In *Proc. of the Intl. Workshop on Managing Interactions in Smart Environments (MANSE)*, pages 114–128, 1999.

[9] S. Duri, A. Cole, J. Munson, and J. Christensen. An Approach to Providing a Seamless End-User Experience for Location-Aware Applications. In *Proc. of the Intl. Workshop on Mobile Commerce (WMC)*, pages 20–25, 2001.

[10] M. Ebling, G. Hunt, and H. Lei. Issues for Context Services for Pervasive Computing. In *Proc. of the Advanced Workshop on Middleware for Mobile Computing*, 2001.

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1997.

[12] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster. The Anatomy of a Context-Aware Application. *Wireless Networks*, 8(2-3):187–197, 2002.

[13] A. Held, S. Buchholz, and A. Schill. Modeling of Context Information for Pervasive Computing Applications. In *Proc. of the World Multiconference on Systemics, Cybernetics and Informatics (SCI)*, 2002.

[14] J. Indulska, R. Robinson, A. Rakotonirainy, and K. Henricksen. Experiences in Using CC/PP in Context-Aware Systems. In *Proc. of the Intl. Conf. on Mobile Data Management (MDM)*, volume 2574 of *Lecture Notes in Computer Science (LNCS)*, pages 247–261. Springer, 2003.

[15] IONA Technologies Inc. Orbix. `http://www.iona.com/products/orbix.htm`.

[16] JBoss Aspect Oriented Programming. `http://www.jboss.org/developers/projects/jboss/aop`.

[17] C. Kaler, editor. Web Service Security (WS-Security). `http://www-106.ibm.com/developerworks/webservices/library/ws-secure/`, 2002.

[18] M. Keidl and A. Kemper. A Framework for Context-Aware Adaptable Web Services (Demonstration). In *Proc. of the Intl. Conf. on Extending Database Technology (EDBT)*, 2004. Accepted for publication.

[19] M. Keidl, S. Seltzsam, and A. Kemper. Flexible and Reliable Web Service Execution. In *Proc. of the Workshop on Entwicklung von Anwendungen auf der Basis der XML Web-Service Technologie*, pages 17–30, 2002.

[20] M. Keidl, S. Seltzsam, and A. Kemper. Reliable Web Service Execution and Deployment in Dynamic Environments. In *Proc. of the Intl. Workshop on Technologies for E-Services (TES)*, volume 2819 of *Lecture Notes in Computer Science (LNCS)*, pages 104–118, 2003.

[21] M. Keidl, S. Seltzsam, C. König, and A. Kemper. Kontext-basierte Personalisierung von Web Services. In *Proc. of the GI Conf. on Database Systems for Business, Technology and Web (BTW)*, Lecture Notes in Informatics, pages 344–363, 2003.

[22] M. Keidl, S. Seltzsam, K. Stocker, and A. Kemper. ServiceGlobe: Distributing E-Services across the Internet (Demonstration). In *Proc. of the Intl. Conf. on Very Large Data Bases (VLDB)*, pages 1047–1050, 2002.

[23] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. An Overview of AspectJ. In *Proc. of the European Conf. on Object-Oriented Programming (ECOOP)*, pages 18–22, 2001.

[24] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In *Proc. of the European Conf. on Object-Oriented Programming (ECOOP)*, pages 220–242, 1997.

[25] W. Kießling. Foundations of Preferences in Database Systems. In *Proc. of the Intl. Conf. on Very Large Data Bases (VLDB)*, pages 311–322, 2002.

[26] W. Kießling and B. Hafenrichter. Optimizing Preference Queries for Personalized Web Services. In *Proc. of the IASTED Intl. Conf. on Communications, Internet and Information Technology*, pages 461–466, 2002.

[27] G. Klyne, F. Reynolds, C. Woodrow, H. Ohto, J. Hjelm, M. H. Butler, and L. Tran. Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies. `http://www.w3.org/TR/CCPP-struct-vocab/`, 2004. W3C Recommendation.

[28] C. Lee and S. Helal. Context Attributes: An Approach to Enable Context-awareness for Service Discovery. In *Proc. of the Symposium on Applications and the Internet (SAINT)*, pages 22–30, 2003.

[29] H. Liefke and D. Suciu. XMill: An Efficient Compressor for XML Data. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 153–164, 2000.

[30] G. Menkhaus. Architecture for Client-Independent Web-Based Applications. In *Proc. of the Intl. Conf. on Technology of Object-Oriented Languages and Systems (TOOLS)*, pages 32–40, 2001.

[31] N. Mitra, editor. SOAP Version 1.2 Part 0: Primer. `http://www.w3.org/TR/soap12-part0/`, 2003. W3C Recommendation.

[32] S. K. Mostéfaoui and G. K. Mostéfaoui. Towards A Contextualisation of Service Discovery and Composition for Pervasive Environments. In *Proc. of the Workshop on Web-services and Agent-based Engineering (WSABE)*, 2003.

[33] A. J. H. Peddemors, M. M. Lankhorst, and J. de Heer. Presence, Location, and Instant Messaging in a Context-Aware Application Framework. In *Proc. of the Intl. Conf. on Mobile Data Management (MDM)*, volume 2574 of *Lecture Notes in Computer Science (LNCS)*, pages 325–330, 2003.

[34] S. Riché and G. Brebner. Storing and Accessing User Context. In *Proc. of the Intl. Conf. on Mobile Data Management (MDM)*, volume 2574 of *Lecture Notes in Computer Science (LNCS)*, pages 1–12, 2003.

[35] N. Santos, P. Marques, and L. Silva. A Framework for Smart Proxies and Interceptors in RMI. In *Proc. of the Intl. Conf. on Parallel and Distributed Computing Systems (PDCS)*, 2002.

[36] A. Schmidt, K. A. Aidoo, A. Takaluoma, U. Tuomela, K. Van Laerhoven, and W. Van de Velde. Advanced Interaction in Context. In *Proc. of the Intl. Symposium on Handheld and Ubiquitous Computing (HUC)*, volume 1707 of *Lecture Notes in Computer Science (LNCS)*, pages 89–101, 1999.

[37] S. Seltzsam, S. Börzsönyi, and A. Kemper. Security for Distributed E-Service Composition. In *Proc. of the Intl. Workshop on Technologies for E-Services (TES)*, volume 2193 of *Lecture Notes in Computer Science (LNCS)*, pages 147–162, 2001.

[38] J. P. Sousa and D. Garlan. Aura: an Architectural Framework for User Mobility in Ubiquitous Computing Environments. In *Proc. of the Working IEEE/IFIP Conf. on Software Architecture (WICSA)*, pages 29–43, 2002.

[39] Sun Microsystems Inc. The Java Servlet Specification 2.4. `http://java.sun.com`.

[40] Universal Description, Discovery and Integration (UDDI) Technical White Paper. `http://www.uddi.org`, 2000.

[41] J. Waldo. The Jini Architecture for Network-centric Computing. *Communications of the ACM*, 42(7):76–82, 1999.