# A Framework for Context-Aware Adaptable Web Services

Markus Keidl and Alfons Kemper

Universität Passau, D-94030 Passau, Germany
<*lastname*>@db.fmi.uni-passau.de

## 1 Introduction

The trend towards pervasive computing involves an increasing number of ubiquitous, connected devices. As a consequence, the heterogeneity of client capabilities and the number of methods for accessing information services on the Internet also increases. Nevertheless, consumers expect information services to be accessible from all of these devices in a similar fashion. They also expect that information services are aware of their current environment. Generally, this kind of information is called *context*. More precisely, in our work context constitutes information about consumers and their environment that may be used by Web services to provide consumers a customized and personalized behaviour.

In this paper, we present a context framework that facilitates the development and deployment of context-aware adaptable Web services. We implemented the framework within our ServiceGlobe system [1, 2], an open and distributed Web service platform.

## 2 The Context Framework

In our framework, context information is transmitted (in XML data format) as a SOAP header block within the SOAP messages that Web services receive and send. A context header block contains several *context blocks*. Each context block is associated with one dedicated *context type*, which defines the type of context information a context block is allowed to contain. At most one context block for a specific context type is allowed.

The life-cycle of a Web service's context begins at the client: First, the client application gathers all relevant context information and inserts it into the SOAP header. Then, the request is sent to the Web service. After the request was received, 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 API provided by the framework. 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 current context is automatically inserted into its response and sent back to the invoker. If the invoker is a client application, it can integrate portions of the returned context into the consumer's persistent local context (for use in future requests).

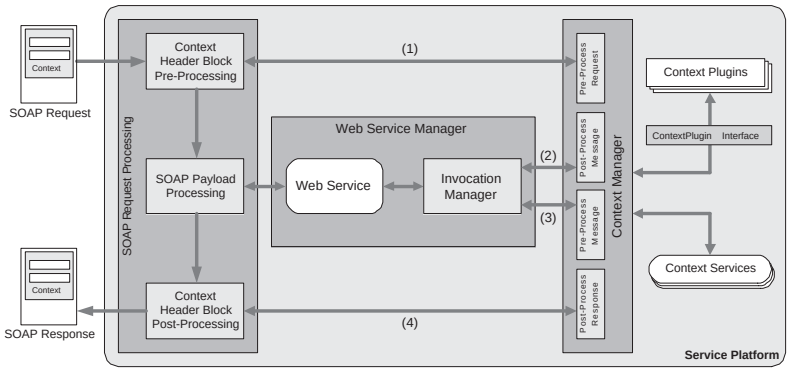


**Fig. 1.** Components for Context Processing

### 2.1 Processing Context

There are four components that process context information (see Figure 1): Web services, context plugins, context services, and clients (not shown). Invoked *Web services* themselves can always process any context information. They have full control over how the context information influences their execution and their replies, and they can also modify the context. *Context plugins* are implemented in Java and must be installed locally at a host. Every plugin is associated with one dedicated context type. *Context services* are Web services that implement the `ContextService` interface, which is defined using the WSDL standard. Just as context plugins, every context service is associated with one context type. In contrast to context plugins, context services need not be installed locally, but can be available anywhere on the Internet. The *client application* also processes context information, e.g., it converts price information in a response into the currency at the consumer's current location.

There are four occasions at which context is processed automatically, as shown in Figure 1: 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, outgoing requests are post-processed before they are actually sent (2). All incoming 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.

### 2.2 Processing Instructions

In our framework, a context block could potentially be processed by several components. Furthermore, it can be processed by all hosts on which the invoked Web service invokes other services. Context processing instructions are used to specify rules of precedence and the components and hosts that should actually be used for context processing. They encompass context service instructions and processing guidelines. With context service instructions, context services that should be used for context processing and their execution order are specified. With processing guidelines, the 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.

Context processing instructions can be inserted into the context itself as a self-contained context block. Furthermore, a Web service's UDDI metadata may be annotated with them. Providers or developers of Web services can use this option to specify context services that should be used for processing certain context blocks. Additionally, context services may be published in a UDDI registry, just as ordinary Web services. Our framework then uses the available UDDI metadata to automatically determine available context services for context processing.

### 2.3 Context Types

Our context framework provides several integrated context types. One of the most important context types for information services is *Location*. It contains information about the consumer's current location, e.g., the consumer's GPS coordinates, country, local time and time zone. It may also include semantic location information, e.g., that the consumer is currently at work. The *Consumer* context type contains information about the consumer invoking the information service, e.g., name, email address, preferences, and so on. *Client* context information is data about a consumer's client, e.g., its hardware (processor type or display resolution) as well as software (Web browser type and version).

## 3 Description of the Demo

For our demonstration, we use a well-known information service from the Internet as basis, the Amazon Web service. We developed the information service *MyBook* that enhances the Amazon service's query capabilities with context awareness. The MyBook service uses, for example, Client context information to adjust its response to the client's capabilities. If the display of the client device is small (e.g., on PDAs or cell phones), unnecessary data, e.g., customer reviews, is removed from the response. We also present a context service that uses Location context information to convert price information in the service's response into the currency at the consumer's current location.

We implemented several clients for different devices, e.g., Java-based clients for PDAs and cell phones. With them, the usefulness and the advantages of context information are demonstrated, based on the MyBook service. We also implemented a Web-based client that allows the investigation of the influence of various types of context information on information services in more detail.

## References


1. 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.
2. M. Keidl, S. Seltzsam, K. Stocker, and A. Kemper. ServiceGlobe: Distributing E-Services across the Internet (Demonstration). In *Proc. of the Conf. on Very Large Data Bases (VLDB)*, pages 1047–1050, 2002.