A RUN TIME ENVIRONMENT EXTENSION FOR PERSONALIZED WEB SERVICES

Julia Gross, Joachim Zeiss, Sandford Bessler
FTW (Telecommunications Research Centre Vienna), Donau-City-Str. 1, Vienna 1220, Austria

Keywords: Web services, web services platform, OSA, Parlay X, profiling, UML design

Abstract: This paper considers the web service environment for Telco applications, as defined recently by the Parlay-X Working Group within the Parlay organisation. These services include messaging, location, payment, call control, presence, group management, etc. and open the way to rapid service creation and integration with internet applications. However, some vital issues for the telecommunications industry such as the secure user access to a service and its personalization have not been addressed yet sufficiently. The Web Services Run Time Environment Extension (WSRTE) presented in this paper is a light, vendor-independent framework that facilitates the creation of personalized web services by combining security and profiling functions.

1 INTRODUCTION

Nowadays web services start to find a use in all conceivable branches of industry for whose operation distributed computing is vital. It is especially true for the telecommunications field where web services technology is currently getting adopted and telecom web services are being created. However, the web services technology is not yet completely mature and some aspects, vital for telecom services creation, have not been addressed yet sufficiently. Specifically, we consider the secure user access to a service and its personalization, two crucial requirements. To our knowledge, the interaction of security and profiling functions has not been so far treated in the literature. Often, access data and user profiling data are kept separated in different storages/databases. This is due to structural diversity of the security mechanisms and proprietary nature of organizing user data. We show that the user credentials obtained from the authentication process are used for user profile lookup, leading to a flexible model that relieves a system from ambiguity and necessity of user credentials mapping.

The proposed web services runtime environment extension (WSRTE) should rely on the security standards supported by the web services platform (for example WS-Security (OASIS Standard 200401, 2004)), and thereby service providers and users are not forced to using a certain security mechanism, since many web services platforms provide multitude of diverse security mechanisms and allow plugging proprietary ones (see Manes 2003, O'Neill 2003, IBM Corp., Microsoft Corp., 2002, PC Magazine, 2002 for security mechanisms overview).

Telecom providers have to offer personalized services in which the user related data is stored by the service and precisely customizes it. It is hard to overestimate the value of personalized services in the telecom branch. However, there is yet another type of data necessary for the service operation. Web services, just as any other kind of distributed systems, rely on various configuration and startup data for communication with the software infrastructure they are a part of (Alonso, G., et al., 2004). Diverse system and services addresses, names, properties, etc. are needed for that purpose. All this information should be managed centrally in a system profile (Newcomer, E., 2004).

The idea of a profiling database is not new and is further developed in 3GPP under Generic User Profile (GUP) (3GPP TS 23.240, 2003). In fact the majority of providers already possesses user and service databases and expects to use them when deploying new (web) services. A goal in our design is therefore to keep the web services platform independent of the profile database. The result of our work is generic enough to port it from one web services platform to another without having to change a single line of the web service code.

Gross J., Zeiss J. and Bessler S. (2004).
A RUN TIME ENVIRONMENT EXTENSION FOR PERSONALIZED WEB SERVICES.
In *Proceedings of the First International Conference on E-Business and Telecommunication Networks*, pages 251-256
DOI: 10.5220/0001383902510256

Other than in the IT world, the most important Services in the Telecom have been standardized in form of Web Services APIs by the Parlay Group under the name Parlay-X. These are open interfaces between network operators and service providers specified in UML or WSDL (Parlay X Working Group, 2003) and used for call control, messaging, content based charging, location, presence, etc. An open issue is currently the so called WS Framework, an entity that mediates between the web service provider and a web service. The solution, we present in the next sections, implements those parts of the WS Framework dealing with authentication, access (authorization) and service personalization.

The rest of the paper is organized as follows: section 2 presents the architecture and design of a generic web service personalization solution. After a telecom specific example in section 3, we conclude and give further research directions.

2 SYSTEM ARCHITECTURE AND DESIGN

2.1 High level overview

As mentioned above, the WSRTE is a light framework with extensible architecture that can be plugged into a web services platform. WSRTE mainly enables web services profiling taking advantage of a security framework provided by a web services platform. Thus, WSRTE can be seen as an abstraction layer between a web service and a given web services platform implementation that enables the retrieval of profile data necessary for the web service operation.

The birds view shows a 3-tier architecture tailored to the needs of web services which require profiling and security features. The client can be any application capable of talking SOAP

In Telecom environments we see a number of platforms for services using, for example, CORBA technology (ETSI ES 202 915-3, 2003). If we want to build new Web Services on top of these legacy systems, we come out necessarily to a similar architecture to that depicted in Figure 1.

The client sends a SOAP request to a given service URL. In our case, a web services platform is listening to requests for this URL. The request is intercepted by the WSRTE that extracts and abstracts relevant context information for the web service. After interception, the request reaches the web service implementation. At this point the implementation accesses the WSRTE to obtain the context data related to the requesting user, the service itself or the system/software infrastructure. The runtime

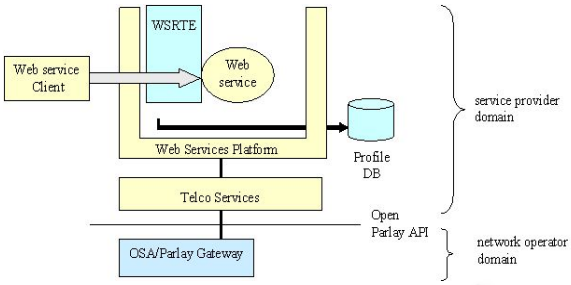

Figure 1: Architecture Overview.

environment interacts with a database and returns this information to the requesting service via abstracted profiles. For that purpose, the WSRTE objects mediate between the web services platform and database implementation (SQL database, for example).

The novelty of the presented architecture is in the fact that WSRTE is vendor independent, i.e. can be ported to any web services platform that allows plugging modules for additional SOAP message processing (most of the current products do) and supports Java. In our case we plugged WSRTE into the WASP platform from Systinet (Systinet Corp., 2004).

WSRTE is web service transparent in sense that once the web service code has been written there will be no further changes to it when porting the web service to another web services platform implementation.

Another feature of WSRTE architecture is that it is profiling database independent, i.e. any profiling database structure can be used with WSRTE, including flat files, etc.

All of these issues will be explained in greater detail later in this chapter.

2.2 Design of Runtime environment extension

To enable the WSRTE flexibility and vendor independence, the entities have been grouped in three categories (see Fig. 2):

1. Classes and interfaces provided by web services platform (e.g. WASP) being used by WSRTE and enabling plug-in functionality (packages org.systinet.wasp.webservice and org.idoox on Fig. 2
2. Classes implemented by WSRTE to fulfil web services platform specific tasks (package at.ftw.wasp on Fig.2)
3. Classes and interfaces implemented by WSRTE that are platform independent. A web service

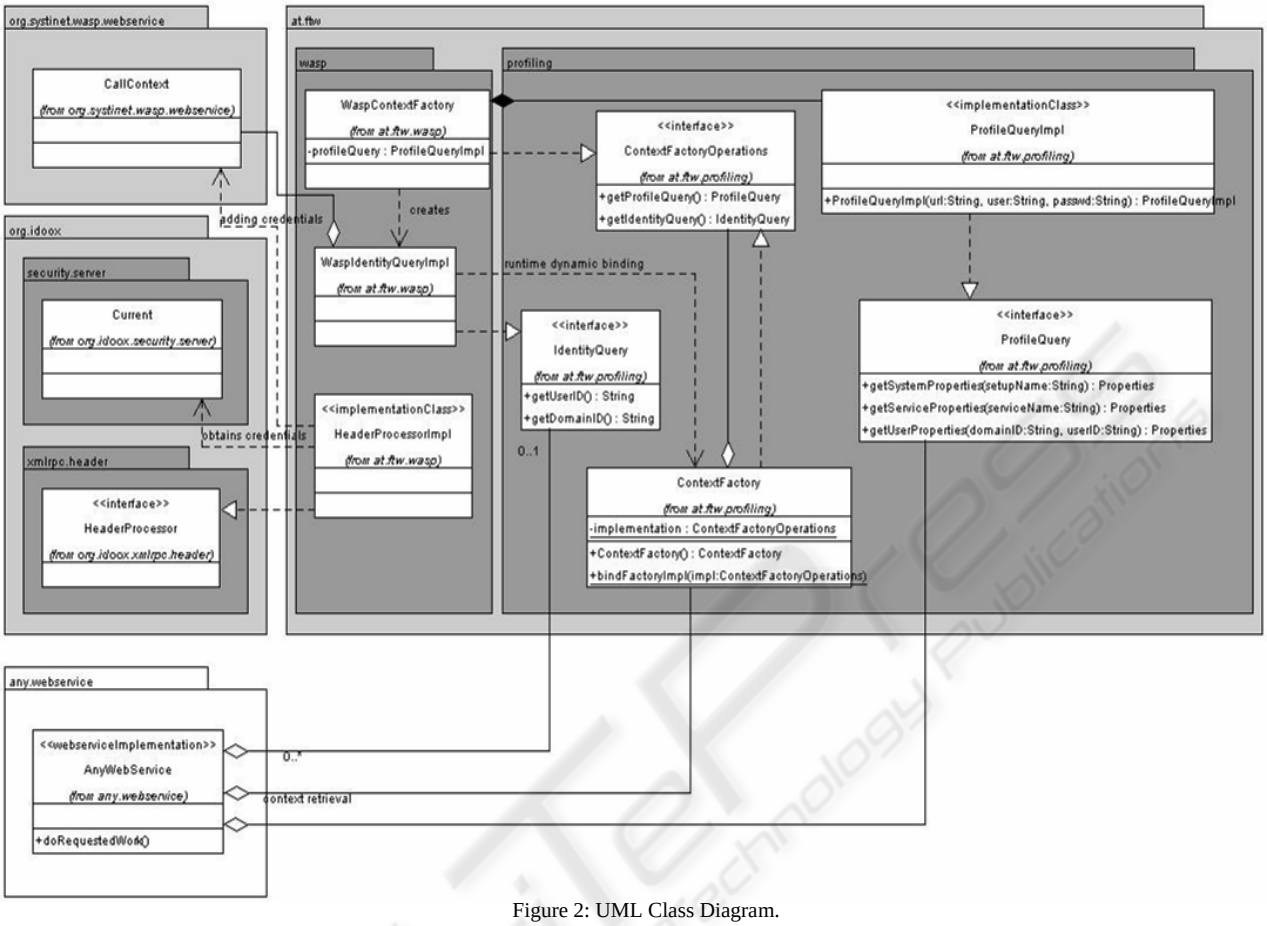

Figure 2: UML Class Diagram.

implementation is interacting with these classes and interfaces only to guarantee product independent realization of a web service (package at.ftw.profiling on Fig. 2)

WSRTE is plugged in WASP by means of a `HeaderProcessorImpl` class implementing WASP `HeaderProcessor` interface that serves message interception purposes. The WASP server offers the opportunity to define header processors for each deployed service. These header processors are being called on every request to the related service ahead of calling the service implementation but after security framework did its job.

As initial access point to the WSRTE, a web service implementation creates an object of `ContextFactory` type. This class is simply wrapping the actual platform dependent implementation of a `ContextFactory`. In addition to the `ContextFactoryOperations` implementation, it provides the method bindFactoryImpl which is called by the `WaspContextFactory` at creation time. This allows different factory implementations to be loaded at program runtime. The web service implementation will not notice if it is using a WASP related, AXIS related or any other specific implementation of a `ContextFactory`. Due to this mechanism the web services platform specifics remain hidden to a web service implementation.

The `WaspContextFactory` is created at WASP start up and has the task to provide actual implementations of identity and profile query objects to the web service implementation. As the web service sees these objects via WSRTE `IdentityQuery` and `ProfileQuery` interfaces, it is unaware of the actual profiling database structure and doesn't have to perform any security framework specific actions to obtain the user's credentials. These operations are performed by `ProfileQueryImpl` and `WaspIdentityQueryImpl` respectively.

With such a design we achieve WSRTE portability and ability to use any profiling database:

- to port WSRTE to another web services platform, only the classes from the second category in the list above have to be replaced by classes meeting the new platform requirements. Since the separation between actual implementation and presentation to a web service is kept clean,

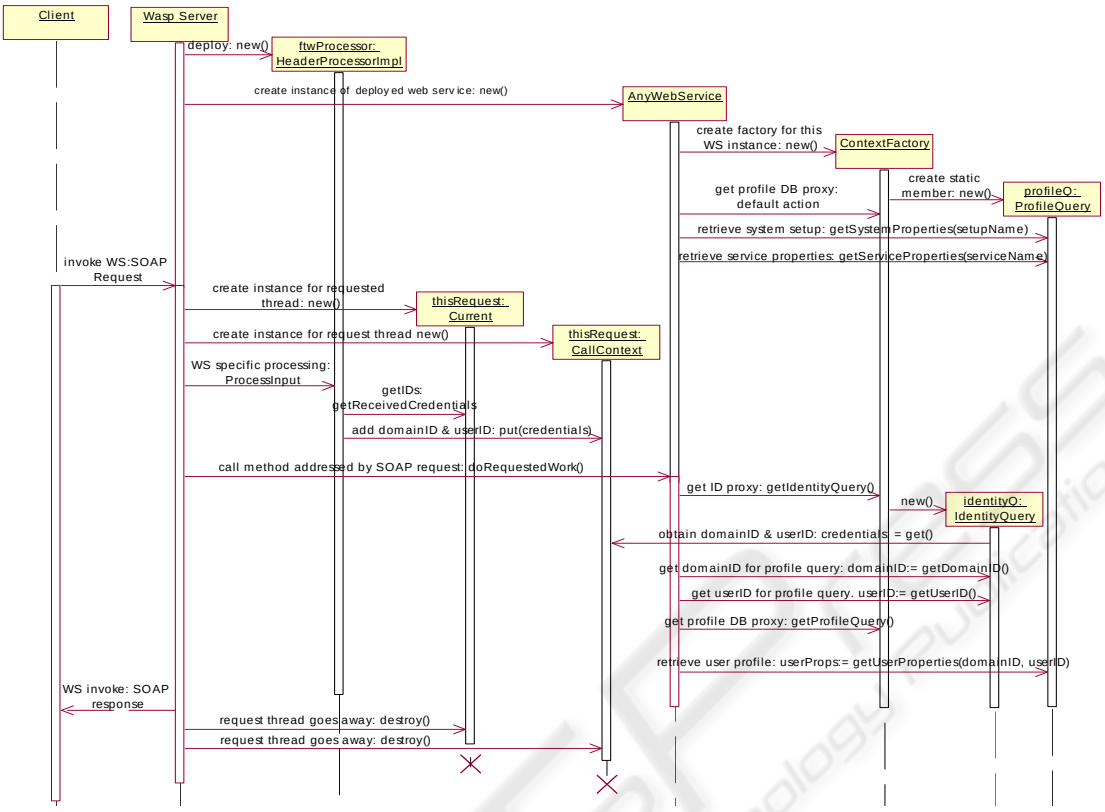

Figure 3: Sequence Diagram.

the web service implementation will not be affected by such a change.

- to use another database structure, only a new `ProfileQueryImpl` class implementing `ProfileQuery` interface needs to be written. This action will not influence the web service code.

2.3 Processing a SOAP request

The sequence diagram under Fig. 3 shows the typical interaction of a web client with a web service. It gives details on request processing and interactions between the platform (in our case WASP), the WSRTE and the web service implementation.

1. **Deployment:** On deploying the service onto the WASP platform, a new `HeaderProcessorImpl` object is created and associated to the web service. If the web service is required to be preloaded, WASP creates an instance of its implementation class. The web services constructor will typically create a `ContextFactory` and obtain a `ProfileQuery` object to retrieve system and service profiles.
2. **Processing incoming request:** The WASP server performs security tasks. If not preloaded, the web service implementation is at this time instantiated as described above. The WASP server creates a `Current` and a `CallContext` object for this single request. After that, the `HeaderProcessor` is being called. It gets the user credentials from the Current object and stores them in the `CallContext` object as userID and domain ID.
3. **Web Service invocation:** After invoking the header processor, the WASP server invokes the requested method on the web service implementation. Inside this method call, the web service obtains an `IdentityQuery` object using the `ContextFactory`. The `ContextFactory` implementation will in turn create an `IdentityQuery` object for each call of getIdentityQuery. This is required, as the `CallContext` object accessed by the `IdentityQuery` only exists while processing this single request. Therefore, on creation of the `IdentityQuery` object, it asks the `CallContext` object for user ID and Domain ID. Those two IDs

have been set on `IdentityQuery` by the `HeaderProcessor` implementation and are delivered when the web service calls getUserID or getDomainID on it. The web service obtains the user profile by calling getUserProperties on the `ProfileQuery` object. The web service implementation may now proceed with the processing of the request.

4. **Response sending:** The result of the method call on the web service implementation is packed into a SOAP response by the web service and sent back to the client. `CallContext` and `Current` objects are destroyed. Note that callback and network event support is left to the application servers capabilities. It might be conceivable, however, to integrate this functionality into the WSRTE to support dynamic reconfiguration or work flow mechanisms.

2.4 Profile Management

This section discusses design topics related to profile management as such, in particular the representation of profile data in a relational database.

In Figure 4 we show the entity relationship diagram of our profiling database. The structure introduced here is not only useful for the WSRTE but also a generic solution for storing profiles of any kind. As the profile query mechanism design uses the proxy pattern, any existing database structure, flat files, etc. could be chosen to implement data storage.

We distinguish between four different profile types:

- **System** profiles contain information related to the server on which the application is running. The information is common to all services and users, for example the class path, execution path.
- **Service** profiles contain information relevant to a particular service, for example a JDBC URL or service IDs. This data is common to all users using this service.
- **User** profiles describe information belonging to the user and independent of the service. A phone number or an e-mail address could go here. It is information of a particular user common to all services the user is requesting.
- **Service-user** profiles are related to a particular service and a particular user. The range of telephone numbers to which the user is allowed to dial out using a 3rd party call service is an example for that.

Those four categories are represented by corresponding tables (see Figure 4). They all inherit from table *PropertyOwner* that contains the owner description and provides the unique property owner ID. Finally, table *Property* contains the actual profile attributes.

To access the profile system, a user and profile administration client has been developed.

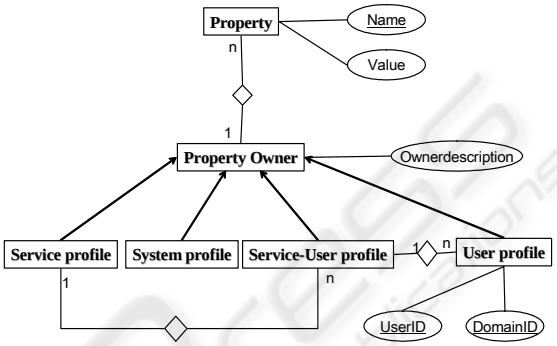

Figure 4: Profile System: Entity Relationship diagram.

3 A TELECOM 3RD PARTY CALL SERVICE EXAMPLE

In this section we show a telecom application using a Parlay X (Parlay X Working Group, 2003) service implemented with the WSRTE extensions, taking advantage of service security and personalization support. Parlay-X services specifications are sets of APIs, standardized by the Parlay Group. They have a high level abstraction and map to low level Parlay APIs such as messaging, mobility, call control or charging. Parlay X services are supposed to be secure and customisable. While fulfilling these requirements we still treat a Parlay X service as being nothing else but a web service. This means that all tasks related to security and profiling, former treated by a Parlay Framework (F/W) (ETSI ES 202 915-3, 2003), will now be offered by the web services platform together with our runtime environment. The advantage of this model is that those tasks are performed basing on a combination of existing WS standards. In Figure 5, a number of different SOAP clients are able to access the Parlay X service implementation (Systinet Corp., 2004, Apache Software Foundation, 2004). When implementing Parlay X services, extensive support for CORBA and client side implementation towards a Parlay framework and Parlay is required. The web service implementation are kept separate from the tasks that require CORBA interactions, thus the web service itself is deployed in a web services platform, whereas the CORBA related tasks are implemented as CORBA

components and reside in their own container (CORBA components, CCM). The web service implementation delegates CORBA related tasks to a proxy component which authenticates the service at the Parlay Framework and mediates between the web service and the basic CORBA service (call setup in this case). In addition, the proxy object can handle connections to different Frameworks inside or outside the web services domain.

The Click2Call (C2C) application uses the so-called 3rd Parlay X 3rd Party Call Service and takes advantage of the proposed profiling system.

The Click2Call application could be incorporated into a real-estate purchase/rent web site. The end user browses the site, views photos and reads description of some houses and if he is interested he pushes the button to activate the C2C application and connect to an estate broker for discussing further details.

The service subscription process is expected to be done offline and is not discussed here. At service subscription, the user is assigned a service and user profiles by the WSRTE administration tool. Following a request, the service access rights are checked by the web services platform. Which security mechanism should be applied for the client-service communication is a part of the service contract.

The user logs in from the welcome page. For a call we need two addresses – of the caller and the callee. In our case, the own address is stored in the User Profile and is looked up by the service with the help of WSRTE after the authentication process. Moreover, the service can perform an authorization check for the called user by looking up the Service-User Profile for this user, and for example, deny international calls in the PSTN. The System Profile in this example contains the address of the Naming Service which leads to the address of the underlying FW Proxy component. Therefore, one could move the underlying network level service infrastructure, modify an entry in System Profile for the Parlay X 3rd Party Call Service and continue using the latter without having to restart the web services platform or re-deploy the Parlay X service.

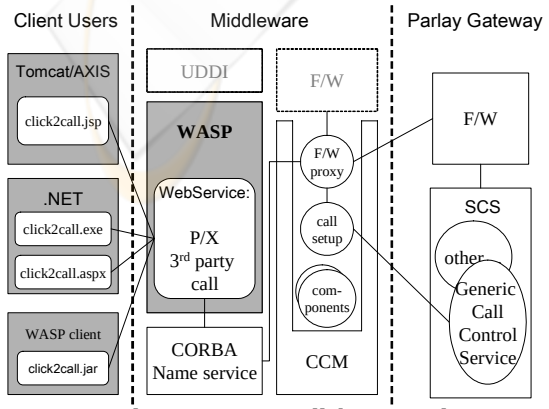

Figure 5: Telecom service collaboration diagram.

4 CONCLUSIONS AND FURTHER RESEARCH

We have shown in this work how one can extend an existing web service platform to support profiling for users and services. We discussed as well architectural considerations for embedding the web service layer in a telecom service infrastructure.

When working with the prototype, we found out that parts of the profiles (System Profiles) are used simply to find and access other services. In the future, service discovery, service inter-working (aggregation, orchestration) and reconfiguration is going to play a major role and should be done dynamically. One way to do it, is to perform the interaction between web services via connection proxies, which would be dynamically configured by a connection manager. Connection reconfiguration and service discovery would be transparent to the web service.

REFERENCES

Alonso, G., et al., 2004. *Web Services. Concepts, Architectures and Applications*. Springer-Verlag Berlin Heidelberg.

Apache Software Foundation, 2004. *Apache web services project Axis*, http://ws.apache.org/axis/index.html

ETSI ES 202 915-3, 2003. Parlay/OSA Framework API, http://www.parlay.org

IBM Corp., Microsoft Corp., 2002. *Security in a Web Services World: A proposed Architecture and Roadmap*, http://www-106.ibm.com/developerworks/webservices/library/ws-secmap/

Manes, A., T., 2003. *Web Services. A Manager's guide*. Addison-Wesley

Newcomer, E., 2004. *Context, Coordinators, and Transactions – The Importance of WS-CAF*, http://www.webservices.org/index.php/article/view/1297/

OASIS Standard 200401, 2004. *Web Services Security; SOAP Message Security 1.0*, http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0

O'Neill, M., et al., 2003. *Web Services Security*. McGraw-Hill/Osborne

Parlay X Working Group, 2003. *Parlay-X Web Services Specification*, http://www.parlay.org/specs/index.asp

PC Magazine, October 2002. *Securing Web Services*,

Systinet Corp., 2004. *WASP Server for Java*, http://www.systinet.com/doc/wasp_jserver/

3GPP TS 23.240, 2003. *Generic User Profile (GUP) requirements; Architecture.*