Presence-Based Runtime Composition of IMS Services Deployed in a SIP Servlet Platform

Juan Miguel Espinosa Carlín

Communication and Distributed Systems, RWTH Aachen University,
52074 Aachen, Germany,
e-mail: espinosa@i4.informatik.rwth-aachen.de

Abstract The IP Multimedia Subsystem (IMS) is aimed to enable the delivery of rich multimedia services in converged networks. Due to the current needs of the mobile telecommunications market to deliver tailored experiences to its users, IMS operators must be able to manage the interoperability and cooperation between the deployed services, in order to provide a high level of customization to their subscribers. With this goal in mind, this paper presents an extension to the Default Application Router introduced in the Java SIP Servlet API v1.1. The proposed Presence AR allows runtime changes in the precedence relationships and application subscriptions that build the composition chains, based on the Presence information of the users involved in the session.

1 Introduction

The convergence-driven need to seamlessly deliver services, has forced operators to adopt new approaches, like the Intelligent Networks, Web Services API's, and, most recently, the IP Multimedia Subsystem (IMS) [7]. In the context of the IMS architecture, Java SIP Servlets are a suitable option to implement a SIP Application Server (AS). In March 2003, the Java Community Process (JCP) released the Java SIP Servlet API (SSAPI), under the Java Specification Request (JSR) 116 [13], for building and deploying SIP applications. In such environment, a SIP Servlet container acts as an AS, and hosts one or more applications that are invoked according

Please use the following format when citing this chapter:

Espinosa Carlín, J. M., 2008 in IFIP International Federation for Information Processing, Volume 286, Towards Sustainable Society on Ubiquitous Networks, eds. Oya, M., Uda, R., Yasunobu, C., (Boston: Springer), pp. 113–124.

to specific mapping rules. Although the JSR 116 states that application composition is desirable, no standard compositions mechanisms are defined, and its definition is left entirely to the container implementation.

With the goal of promoting software modularity and reuse, the Java Community Process specified the SIP Servlet API v1.1 under the JSR 289 [15]. In the application composition framework of this specification, the core entity is the Application Router (AR), to which the container communicates to know the sequence in which the applications have to be invoked. Although the AR is essential for the proper operation of the container, the specification only provides the definition of a Default AR (DAR), which has no processing logic besides the declaration of the order in which applications will be invoked; the implementation of more powerful AR's that make use of complex rules and diverse data repositories, is left to the container implementations. With the goal of providing a richer component, this paper presents an extension that allows the AR to perform runtime composition, based on the Presence information associated to the users involved in a session.

The rest of this paper is structured as follows. Section 2 gives an overview of the available approaches for enabling IMS service composition. Then, Sect. 3 describes the proposed AR in detail. Section 4 presents the Proof-of-Concept prototype developed for implementing the Presence AR. Finally, the conclusions and an outline of future work are given in Sect. 5.

2 Available Approaches for IMS Service Composition

Although there are many techniques for enabling service composition in the IMS, most of them are based on at leat one of the approaches shown in Fig. 1.

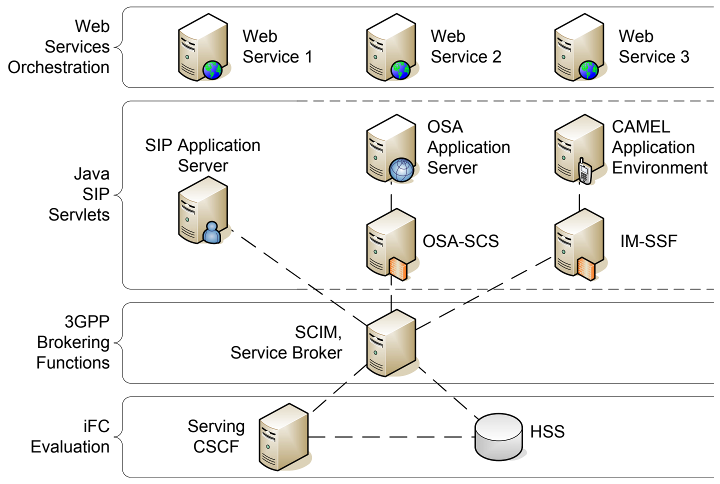

Fig. 1 Existing Approaches for IMS Service Composition

2.1 iFC Evaluation at the S-CSCF

For delivering services to the end users, the IMS makes use of the information stored in the user profiles residing in the HSS's. According to TS 23.218 [3], a service profile is composed by a Private User Identity to which the profile is applicable, and one or more service profiles associated to it. For each one of these service profiles, a Public User Identity (PUI) and zero or more initial Filter Criteria (iFC) are defined. iFC contain all the necessary information that helps the Serving Call-Session Control Function (S-CSCF) to decide when a specific AS must be involved in order to provide a service to the subscriber. iFC are evaluated by the S-CSCF for all those SIP requests that either create a dialog, or are stand-alone requests (e.g. `SUBSCRIBE`, `INVITE`, `OPTIONS`).

For the case in which there is more that one AS involved in the signaling path, the S-CSCF will forward the matched request to each one of them, according to the priority field of the iFC. Although this chaining mechanism is both efficient and reasonably simple, it only allows to sequence the interactions at the AS level. In order to coordinate the orchestration of individual services, a composing approach achieving refined granularity is needed [4].

2.2 3GPP Brokering Functions

The 3GPP has defined two entities aimed to deal with the feature interaction management in an IMS network. The first concept is the Service Capability Interaction Manager (SCIM), a function aimed to manage service capability coordination. The second one is the OSA/Parlay Service Broker (SB), aimed to broker OSA/Parlay applications, application hosted on SIP AS's, and services offered by legacy networks (e.g. CAMEL). Both components were supposed to be standalone entities placed between the S-CSCF and the AS, communicating with them via a SIP interface. However, the SCIM was never further defined, and for the case of the SB, the progress is being slowly documented in 3GPP TR 23.810 as part 3GPP Release 8.

Although these entities are theoretically able to manage the interaction of applications deployed in different domains (e.g. OSA/Parlay and CAMEL), a scenario in which such approach would be needed is not very realistic [4], notably limiting the practical use of both approaches.

2.3 Web Services Orchestration

Web Services orchestration is a topic on which a lot of research has been done in the past years. Based on the Universal Description, Discovery, and Integration (UDDI) for listing, on the Web Service Description Language (WSDL) for description, and on the Web Services Business Process Execution Language (BPEL) for orchestration, Web Services are a successful technology that allowed the IT community to realize key features behind the Service Oriented Architecture (SOA) concept.

Even though many efforts have been done for enabling the use of Web Services in the telecommunications environment [5], the process-driven nature of their orchestration techniques is not suitable for composing real-time communications services, being the later better described from an event-driven perspective [2].

2.4 Java SIP Servlets

The AR concept introduced in the JSR 289 allows for a clear separation between concerns and responsibilities, with developers implementing the application logic of the services, and deployers controlling the application selection and invocation order.

Because the container must support the invocation of originating applications for the caller and terminating applications for the callee, the specification introduces the concepts of `ORIGINATING` and `TERMINATING` routing regions. When the container queries the AR for the next application to invoke, it also sends the routing region in which the invocation has to be performed; a third `NEUTRAL` routing region is defined for applications that do not serve a particular subscriber.

As stated in the JSR 289, the Default AR (DAR) should be available in every compliant implementation, and should invoke applications driven by a properties file, in which the name of the property is the invoked SIP method, and the value of the property is the `SipRouterInfo` object converted into a string. The information contained by this string consists of the following [15]:

- The name of the application to be invoked, as identified by the application deployment descriptors.
- The subscriber's identity returned by the DAR, that can be any header present in the SIP request.
- The routing region in which the application has to be invoked (i.e. `ORIGINATING`, `TERMINATING`, or `NEUTRAL`).
- A SIP URI that indicates the route as returned by the AR.
- A route modifier, to indicate the direction to be followed in the invocation chain (i.e. `ROUTE`, `ROUTE_BACK`, or `NO_ROUTE`).
- A sting representing the AR's internal state. Because this information will be exclusively used by the AR, its contents are up to the individual DAR implementations.

An example of such configuration file is the following:

```
INVITE: ("OriginatingCallWaiting", "DAR:From",
        "ORIGINATING", "", "NO_ROUTE", "0"),
        ("CallForwarding", "DAR:To",
        "TERMINATING", "", "NO_ROUTE", "1")
```

For this example, the DAR is configured to invoke two applications on receiving an `INVITE` request, one for the originating region, and the other for the terminating region. The applications are identified by their names, and the returned subscriber identities are bonded with the contents of the `From` and `To` headers of the request.

The specification of the DAR is heavily based on the Distributed Feature Composition (DFC) algorithm [6], with the only available implementation, called the DFC-AR, being statically configured with no dependencies on external databases or other data stores [1]. DFC is realized from a pipe-and-filter perspective, in which multiple *feature boxes (FB's)* are linked by internal featureless calls, thus forming a sequences of FB's. In the context of the DFC-AR, these sequences are called *application chains*. A typical arrangement of feature boxes is shown in Fig. 2.

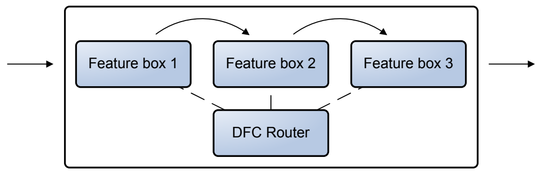

Fig. 2 Architecture of the Distributed Feature Composition

Table 1 shows a comparison between the described composition techniques in terms of the granularity offered by the approach, its provider interoperability, and the paradigm used for doing composition.

Table 1 Comparative Analysis of IMS Composition Approaches

Technique	Granularity	Interoperability	Paradigm
iFC Evaluation	Coarse	High (3GPP Standard)	Event-driven
3GPP Brokering Functions	Refined	Low (Propietary)	Event-driven
Web Services Orchestration	Refined	High (OASIS Standard)	Data-driven
JSR 289 Application Router	Refined	High (Java-based)	Event-driven

Because the SIP Servlet API builds on the well known API for deploying HTTP servlets, the programming model is already known to a considerably large community of developers; at the time of writing, there are at least five commercial and one open source implementations of the standard.

3 A Presence-Driven IMS Application Router

3.1 Functional Architecture

The proposed architecture for the Presence AR is shown in Fig. 3. A brief description of each of the entities is given next. For the aim of clarity, let's take the example of Alice establishing a call with Bob. Both of them have a contract to use the following services: *Speed Dial*, *Call Waiting*, *Call Forwarding*, *Voice Mail*, and *SMS*. Additionally, their IMS provider enables the *Enhanced Call* service, which composes the mentioned services depending on Alice's and Bob's Presence status.

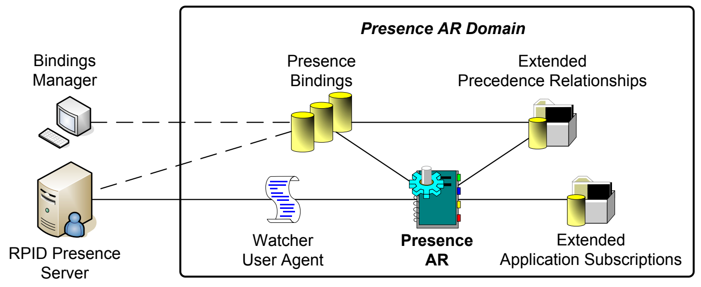

Fig. 3 The Presence Application Router

3.1.1 Extended Precedence Relationships

The application chains built by the DFC-AR are based on Precedence Relationships statically stored in an XML file. These relationships are enclosed within `<ordering>` elements, and establish a partial order among applications in each one of the routing regions. The higher an application name appears in the list, the higher priority that the application has in the region in which is listed. Due to the fact that not all the deployed applications keep a precedence relation with the others, it is absolutely valid that some applications are not listed on the relationships. Because the priority assigned to the applications is based on their proximity to the subscribers, applications with higher precedence will appear closer to the endpoints when invoked.

For the Presence AR, the idea is to reuse these relationships to give the AR alternative *Paths* inside an extended application chain: the taken Path will depend on the Presence information of the user being called. Because the published information represents only a weak form of contract, it does not ensure that it will always be possible to successfully contact the presentity, so a *Default Path* inside the extended chain has to be always defined. For the case of our example, a possible extended application chain built by the Extended Precedence Relationships is shown in Fig. 4.

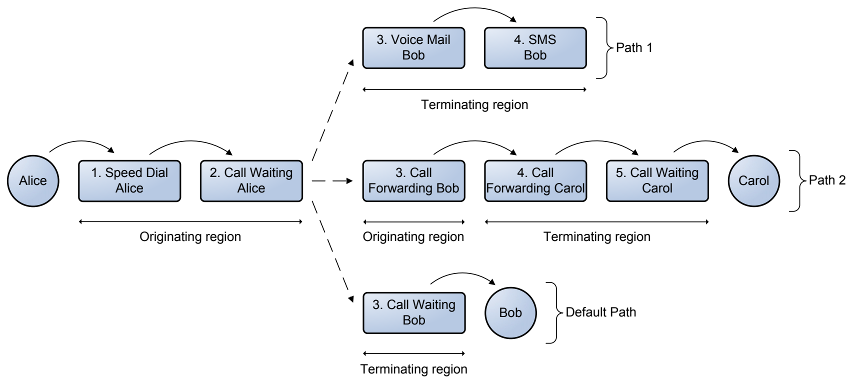

Fig. 4 Extended Application Chain for the Enhanced Call Service

The Extended Precedence Relationships configured on the Presence AR for Path 2 in both routing regions are structured as follows:

```
<originating-region>
  <ordering>
    <app-name>SD</app-name>
    <app-name>CW</app-name>
    <path id=2>
      <app-name>CF</app-name>
    </path>
  </ordering>
</originating-region>

<terminating-region>
  <ordering>
    <path id=2>
      <app-name>FW</app-name>
      <app-name>CW</app-name>
    </path>
  </ordering>
</terminating-region>
```

3.1.2 Extended Application Subscriptions

In the DFC-AR, the bindings between the received SIP requests and the invoked applications are statically configured by the Application Subscriptions. These Java regular expressions are called Address Patterns, and are enclosed by `<mapping>` XML elements that are evaluated against the `From` header, including its display-name portion for the case of the originating address, and against the `Request-URI`, for the case of the terminating address of the processed request.

The Presence AR uses an extended version of these subscriptions, and defines the bindings for each one of the Paths configured by the Extended Precedence Relationships. For the case of our example, the Extended Application Subscriptions for the originating routing region of the Default Path are the following:

```
<originating-region-mapping>
  <mapping>
    <path id=default>
      <address-pattern>.*sip:*</address-pattern>
        <app-name>SD</app-name>
        <app-name>CW</app-name>
    </path>
  </mapping>
</originating-region-mapping>
```

The additional subscriptions are configured as follows. For the originating region of Path 1, the Address Patterns contain the same two applications shown above (i.e. Speed Dial and Call Waiting), while for the originating region of Path 2, they contain one extra application (i.e. Call Forwarding). For the terminating routing region, the Address Patterns contain two applications for the first Path (i.e. Voice Mail and SMS), two for the second Path (i.e. Call Forwarding and Call Waiting), and one for the Default Path (i.e. Call Waiting).

3.1.3 Presence Bindings, Binding Manager, and Watcher User Agent

The Presence Bindings are managed through the Bindings Manager, and are defined to indicate the Presence AR which of the defined Paths has to be taken in the extended application chain, depending on whether or not specific parts of the conveyed Presence information meet certain criteria. These bindings are linked to both, the IMS Public User Identities (PUI's) stored in the HSS, and the Paths defined in the Extended Precedence Relationships. In order to get the Presence status of the users, the Presence AR implements a Watcher User Agent that fetches the data from the Presence Server. Assuming that the Rich Presence Information Data Format (RPID) [10] is used for publication, Bob could define the following Presence Bindings for the Enhanced Call service:

```
if ((RPID.person.place-type == residence) &&
  (RPID.person.activities == tv)) then path 1

if ((RPID.person.place-type == office) &&
  (RPID.person.activities == meeting)) then path 2
```

In case that the Presence data is provided by a large number of sources, the Presence AR can make use of already available mechanisms for integrating this information in a consistent and unified way [11].

3.1.4 Presence Application Router

The Presence AR is the key element of the proposed extension. At runtime, it fetches information from the Extended Precedence Relationships, from the Extended Application Subscriptions, from the Presence Bindings, and from the Watcher User Agent. It first verifies if one of the defined Paths is compatible with the received Presence information; if this is the case, the application chain enabled by the matched Path is used, and the first application in the chain is sent to the container. For example, if Bob's Presence indicates that he is at home watching TV, he doesn't want to be disturbed, so Alice will be rerouted to Bob's Voice Mail and Bob will receive and SMS indicating that Alice just tried to contact him (i.e. Path 1 in Fig. 4). In a second scenario, if Bob is at his office on a meeting, all his incoming calls are forwarded to his secretary, Carol (i.e. Path 2 in Fig. 4). Finally, if none of the defined paths is compatible with the current Presence status, or if the Presence service is currently unavailable, the service sets up a call with Bob, as indicated by the Default Path.

3.2 Deployment in an IMS Network

Figure 5 shows the placement of the Presence AR in a basic IMS deployment. On reception of an initial SIP request filtered through the iFC, the S-CSCF invokes the involved AS (i.e. the Servlet container). Next, the container queries the Presence AR to find out the order in which the services have to be invoked. Based on the chosen application chain, the Presence AR answers the container with the list of applications to be executed, together with the corresponding Routing Region (i.e. `ORIGINATING`, `TERMINATING`, or `NEUTRAL`) and the proper Routing Directive (i.e. `ROUTE`, `ROUTE_BACK`, or `NO_ROUTE`).

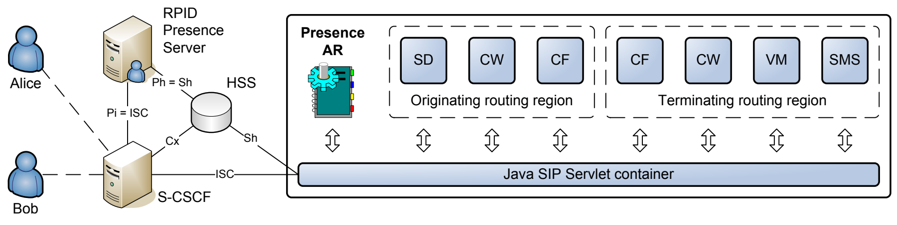

Fig. 5 The Presence AR in an IMS Environment

4 Proof-of-Concept Prototype

An overview of the prototype's architecture is shown in Fig. 6.

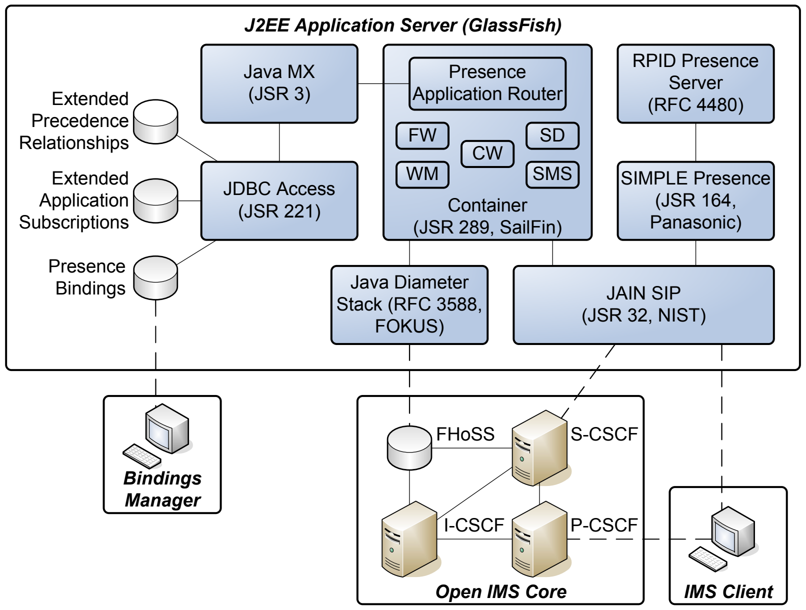
J2EE Application Server (GlassFish)
Java MX (JSR 3)
Presence Application Router
RPID Presence Server (RFC 4480)
Extended Precedence Relationships
FW
CW
SD
WM
SMS
Container (JSR 289, SailFin)
Extended Application Subscriptions
JDBC Access (JSR 221)
SIMPLE Presence (JSR 164, Panasonic)
Presence Bindings
Java Diameter Stack (RFC 3588, FOKUS)
JAIN SIP (JSR 32, NIST)
FHoSS
S-CSCF
I-CSCF
P-CSCF
Bindings Manager
Open IMS Core
IMS Client

Fig. 6 The Prototype's Architecture

4.1 SIP Servlet Container

The abstraction level provided by the SSAPI is located between the transactions users and the SIP transaction layer. As such, it enables applications developers to access and control protocol details like the contents and the headers of the SIP messages, while issues as the formatting, retransmission, and correlation of the requests are fully managed by the container.

The used implementation is the one provided by the SailFin Project [16] as an extension to the GlassFish Application Server [14]. SailFin fully implements JSR 116, and current work is focused towards achieving full JSR 289 compatibility, adding high availability and clustering features.

For communicating with the IMS network, the prototype uses the JAIN SIP implementation provided by NIST as the SIP stack for SailFin, and for accessing the subscribers' information stored in the HSS, SailFin uses the Java Diameter stack developed by the FOKUS Fraunhofer Institute.

4.2 Application Router and Presence Server

Regarding the AR, four approaches were evaluated. The first one was SailFin's Alphabetical AR, which routes every initial SIP request to the deployed applications in alphabetical order. The second one was SailFin's DAR implementation, as defined in the Appendix C of the JSR 289. As already discussed, the DAR routes every initial request to the deployed applications based on the contents of the properties file. The third one was the DFC-AR, which is a DAR implementation based on the approach presented in [12]. These three AR's don't allow changing their behavior without modifying the source code and without doing a redeployment of the application, so they are not suited for the solution proposed in this paper.

A fourth option, taken as base for the Presence AR, is an AR that makes use of the Java Management Extensions (JMX) technology (JMX-AR), and is also included in the SailFin distribution. The JMX-AR offers a runtime configuration interface that allows to dynamically modify the application chains.

4.3 Data Repositories, Core IMS Network, and IMS Client

The Extended Application Subscriptions, the Extended Precedence Relationships, and the Presence Bindings, were deployed as Derby databases embeded through the GlassFish JDBC driver. Finally, the Bindings Manager was developed in Java Swing as a stand alone application, with the needed functionality to manipulate the Presence Bindings database.

For testing the Presence AR, a fully 3GPP compliant IMS environment based on the FOKUS Open IMS Core implementation [8] was deployed. For the IMS client, a prototype was developed in Java Swing, based on the UCT IMS Client Project [17]. The client was extended with the needed funtionality to manipulate the Presence information via XCAP [9], and an XCAP server developed at our research group was used.

5 Conclusions and Future Work

This paper presented an AR proposal based on the DAR specified by the JSR 289, and on the DFC-AR implementation presented in [12].

The Presence AR queries at runtime the Extended Precedence Relationships and the Extended Application Subscriptions associated with a composed service, and does the application routing based on bindings that relate specific portions of the published Presence information with Paths that have to be followed in an extended application chain. For the time being, only sequential application chains were considered. Additionally, a roadmap towards a Presence AR prototype implementation and its deployment in an IMS environment was also presented.

Further improvements on the Presence AR include the analysis and implementation of mechanisms that allow the composition of externally deployed applications, the structuring of hierarchical applications chains, and the cooperation between applications deployed in different environments.

References

1. Cheung, E., Purdy, K.H.: An Application Router for SIP Servlet Application Composition. IEEE International Conference on Communications, 2008. ICC '08 (2008)
2. Dinsing, T., Eriksson, G.A., Fikouras, I., Gronowski, K., Levenshteyn, R., Pettersson, P., Wiss, P.: Service composition in IMS using Java EE SIP servlet containers. Ericsson Review **3**, 92–96 (2007)
3. 3rd Generation Partnership Project: IP Multimedia (IM) session handling; IM call model; Stage 2. 3GPP TS 23.218 (2007)
4. Gourraud, C.: The IMS Latern. http://theimslantern.blogspot.com/ (2008). Last retrieved on 25.03.2008
5. Griffin, D., Pesch, D.: A Survey on Web Services in Telecommunications. IEEE Communications Magazine **45**(7), 28–35 (July 2007). DOI 10.1109/MCOM.2007.382657
6. Jackson, M., Zave, P.: Distributed Feature Composition: A Virtual Architecture for Telecommunications Services. IEEE Transactions on Software Engineering **24**(10), 831–847 (Oct 1998). DOI 10.1109/32.729683
7. Magedanz, T., Blum, N., Dutkowski, S.: Evolution of SOA Concepts in Telecommunications. Computer **40**(11), 46–50 (Nov. 2007). DOI 10.1109/MC.2007.384
8. Magedanz, T., Witaszek, D., Knuettel, K.: The IMS playground @ FOKUS-an open testbed for generation network multimedia services. First International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities, 2005. Tridentcom 2005 pp. 2–11 (23-25 Feb. 2005). DOI 10.1109/TRIDNT.2005.35
9. Rosenberg, J.: The Extensible Markup Language (XML) Configuration Access Protocol (XCAP). RFC 4825 (Proposed Standard) (2007). URL http://www.ietf.org/rfc/rfc4825.txt
10. Schulzrinne, H., Gurbani, V., Kyzivat, P., Rosenberg, J.: RPID: Rich Presence Extensions to the Presence Information Data Format (PIDF). RFC 4480 (Proposed Standard) (2006). URL http://www.ietf.org/rfc/rfc4480.txt
11. Shacham, R., Kellerer, W., Schulzrinne, H., Thakolsri, S.: Composition for Enhanced SIP Presence. 12th IEEE Symposium on Computers and Communications, 2007. ISCC 2007 pp. 203–210 (1-4 July 2007). DOI 10.1109/ISCC.2007.4381531
12. Smith, T.M., Bond, G.W.: ECharts for SIP servlets: a state-machine programming environment for VoIP applications. In: IPTComm '07: Proceedings of the 1st international conference on Principles, systems and applications of IP telecommunications, pp. 89–98. ACM, New York, NY, USA (2007). DOI http://doi.acm.org/10.1145/1326304.1326318
13. Sun Microsystems: Java Specification Request 116: SIP Servlet API. http://jcp.org/en/jsr/detail?id=116 (2003). Last retrieved on 20.04.2008
14. Sun Microsystems: GlassFish. http://glassfish.dev.java.net (2008). Last retrieved on the 20.04.2008
15. Sun Microsystems: Java Specification Request 289: SIP Servlet v1.1. http://jcp.org/en/jsr/detail?id=289 (2008). Last retrieved on 20.04.2008
16. Sun Microsystems: SailFin. http://sailfin.dev.java.net (2008). Last retrieved on the 20.04.2008
17. University of Cape Town: UCT IMS Client. http://uctimsclient.berlios.de (2007). Last retrieved on the 15.11.2007