RUCRG IMS Client: Design and Implementation of Presence and XCAP

Walter T. Muswera , Alfredo Terzoli
Department of Computer Science
Rhodes University, P.O. Box 94, Grahamstown 6140
Tel: +27 46 6038642, Fax: +27 46 6361915
email: g09m3278@campus.ru.ac.za; A.Terzoli@ru.ac.za

***Abstract*—Presence is rapidly becoming the cornerstone for building context aware applications. It was originally developed for communicating the willingness and/or ability of a user to communicate with other users on the network in Instant Messaging (IM) systems. Presence has since been adopted for online collaboration, within enterprises as well as by service providers through the adoption of the IP Multimedia Subsystem (IMS). It has thus become important to manage application configuration data effectively and efficiently to allow for accuracy and availability of such information.**

This paper presents the integration of XML Configuration and Access Protocol (XCAP) support in the Rhodes University Convergence Research Group (RUCRG) IMS client to help in the management of application configuration data.

***Index Terms*—Presence, IMS, XCAP, SIP.**

I. Introduction

PRESENCE is defined by Perea [1] as the willingness and ability of a user to communicate with other users on the network. For instance, Chiedza's presence information might tell us that she is not connected or that she is connected but she is in a meeting and cannot accept communications. Furthermore, if her client supports extended presence, additional information may be provided about her mood, location and communication capabilities (depending on the device through which she is currently connected).

Acharya et al. [2] argue that presence is rapidly evolving to become the de-facto method of representing and querying the context of an individual, both physical (e.g., a user's location) and online (e.g., status of avatars). Furthermore, presence can be used to represent dynamic attributes of not just individuals, but devices (e.g., battery level of a cell phone) and abstract entities (e.g., number of attendees in a conference call).

Apart from the traditional usage of presence illustrated above (in the Chiedza example), a number of emerging applications and recent research efforts can benefit from presence technologies. For example, presence as an element of IP Multimedia Subsystem (IMS) technology can be used to implement a more intuitive network that satisfies a growing demand for context aware applications. According to Lei and Coulton [3], SIP based presence will be a key enabler for achieving the envisaged rich multimedia experience within the IMS. Furthermore, presence information as a service enabler can be incorporated into any number of services, such as IPTV, presence enabled address book, etc [4].

As presence technologies become more widespread, the need to manage data that they use more accurately and efficiently is becoming urgent. In this paper, we use XML Configuration and Access Protocol (XCAP) as the underlying protocol for managing application configuration data namely buddy lists. We also discuss how the RUCRG IMS client was extended to add the ability to upload/fetch/update any type of XCAP document.

II. Background and Related Work

The Rhodes University Convergence Research Group (RUCRG) IMS client is an open-source IMS client developed by students in the Department of Computer Science, at Rhodes University. The project was started with the goal of addressing the unavailability of a single, free IMS client with all the required functionality needed to test applications developed by researchers in the RUCRG [5]. Since its inception the client has grown, both in terms of its stability and its feature set. The client supports a number of functions including Session Initiation Protocol (SIP) and IMS registration and session establishment, voice and video calling, instant messaging (IM) and presence. The client uses a peer-to-peer based presence framework that lacks the ability to provide persistence for user data when the user moves from one device to another. However, Rosenberg in RFC 4825 [6] argues that user information needs to reside within the network to allow it to be accessible from any device, anywhere in the world. He further argues that this allows for the user data to be managed through a number of access points, including PC applications. In light of these arguments it became apparent that the RUCRG IMS client's model of storing and retrieving user data needed to be improved to use a network based user data storage architecture.

Although we decided to extend the RUCRG IMS client, there were other candidate clients that were considered. An overview is given below of the feature sets of three freely available IMS clients that were assessed as well as one standalone presence application.

A. SipPres

SipPres is a presence-capable application built around the flexibility of IMS. It provides three kinds of presence services: presence delivery, watcher identification and privacy control [3]. Although these are services we require, SipPres is developed in ANSI C, making it difficult to integrate within a Java based applications and reducing portability.

B. IMS Clients

Table I shows a comparative assessment of three IMS clients which are currently available to the RUCRG, two of which possess both presence and XCAP features needed for testing some of the developed applications. They support only a subset of functions that are required [7, 8] and this poses challenges. Furthermore, most SIP/IMS clients can only be used on specific platforms [9, 10, 11]. Researchers are then forced to switch between clients or adjust their systems. A detailed discussion of the IMS clients considered is given below. It is worth noting that no mobile client applications were considered.

1) UCT IMS Client: The UCT IMS client is a free open source implementation of a 3GPP IMS client. Like SipPres, it is developed in ANSI C [10]; thus it suffers from the same limitations. It supports a variety of IMS applications such as IM, presence, VoD/IPTV, and the XCAP protocol. It was designed to be used on the Linux platform, and this makes it unavailable for use on other major platforms like Windows and Mac OS. Furthermore, the UCT IMS client is no longer under active development; the last release was published in February 2009. It was tested under a version of Ubuntu that is no longer supported (Ubuntu 8.10, which is not a Long Term Support release) and mailing list discussions have almost ceased.

2) Mercuro IMS Client: Mercuro IMS client is closed source and proprietary; thus it cannot be extended. It comes in various versions one of which is free (Mercuro Bronze) and supports a limited set of functions [12]. Similar to the UCT IMS client, Mercuro is not cross platform; it is built only for the Windows environment. The Mercuro IMS client project has been stopped and the development team has been dissolved [9]. This presents a big challenge because the development and standardization of presence and context aware services is an ongoing process; client development needs to keep up to date with changes in the IMS to remain compatible with IMS standards.

3) IMS Communicator: IMS Communicator is an IMS softphone based on the SIP Communicator Java project [13]. It is implemented on top of the JAIN-SIP stack [14] and the Java Media Framework (JMF) API [15]. Like the RUCRG IMS client, IMS Communicator does not store user data in a central repository. The presence list is stored on the client meaning that subscriptions are created and managed for each presentity in the list by the client. Furthermore, IMS Communicator classes are overloaded with responsibilities making them difficult to debug, test and extend. For example, SIP messages are received and processed by the same class. Registration with the Opensource IMS Core fails (There are existing bugs that have not been fixed in a long time).

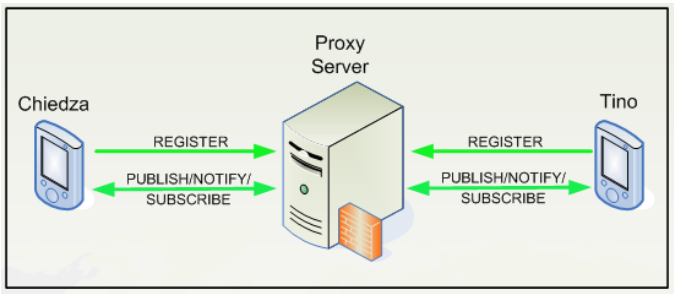
III. RUCRG IMS Client: Presence Model and Limitations

As indicated in Section II, the RUCRG IMS client does not store user data in a central repository. As such, we needed to extend the client to support a network based storage architecture. Before any changes could be made to the client there was need to perform an extensive assessment of the current presence architecture of the RUCRG IMS client, particularly because it already supported some form of SIP based presence and we wanted to preserve what was potentially useful. The assessment included:

- tracing messages sent by the client using network analysis tools to check
  - their sequencing and
  - that they were well formed
- reverse engineering to find out the relationship among the various classes

Presence Model: Having gone through the aforementioned process, a structure of how the client handled its presence was drawn up. Figure 1 gives an overview of the structure that resulted from the experiments that were carried out. To briefly explain, the RUCRG IMS client, represented by Chiedza in Figure 1, utilises SIP subscription mechanisms to subscribe to another user's presence status. The Proxy server is responsible of administering the users' accounts. The client can subscribe to the presence events of another user (Tino in Figure 1) and will be notified whenever the status changes or the user goes offline, through the Proxy. However, as earlier pointed out, the client fetches the presence record and the permission roster from a self administered database at each logon. Furthermore, all subscriptions require user intervention. For example, in the case of subscription failing to reach its target after a specified number of attempts, an alert will be provided on the screen before any further action.

Figure 1: RUCRG IMS Client Presence Architecture



IV. Design and Implementation

In light of the arguments presented, the RUCRG IMS client's model of storing and retrieving user data needed to be improved to use a network based data storage architecture. This section will outline how RUCRG IMS client was modified to store and retrieve user data on the network using XCAP. XCAP is a configuration access protocol used to store application/user configuration data, in XML format on a server. XCAP allows a client to read, write, create, delete and modify the same configuration data. XCAP is a set of conventions for mapping XML documents and document components onto HTTP URLs. It specifies rules for how the modification of one resource affects another, data validation constraints, and authorization policies associated with access to those resources [6]. Because of this structure, normal HTTP primitives can be used to manipulate

Table I: Feature Summary of UCT IMS client, Mercuro and IMS Communicator [10]

	UCT IMS client	Mercuro (Bronze)	IMS Communicator
Registration	AKAv1/2-MD5; MD5	AKAv1/2-MD5; MD5	AKAv1-MD5; MD5
IMS Signalling	PRACK support Pre-condition support	PRACK support Pre-condition support	PRACK support Pre-condition support
Media Support	Audio / Video	Audio	Audio / Video
Presence Support	Presence support Watcher authentication	Presence support Watcher authentication	Presence support
Instant Messaging	Pager mode / Session-based	Pager mode / Session-based	No support
XCAP Support	XCAP support	XCAP support	No XCAP support
Platform	Linux	Windows	Windows / Linux
License	GPLv3	Free, Closed Source	LGLP

the data. XCAP is meant to support the configuration needs for a multiplicity of applications, rather than just a single one.

A. Choice of Technology

In line with the RUCRG philosophy of producing applications that are free and open-source, it was important to use appropriate software libraries. The implementation of the XCAP client was achieved by the use of free open-source Java libraries. There were two main reasons for using Java libraries. Firstly, the libraries integrated well since the client was originally developed in Java. Secondly, their use allowed us to retain the the platform independence feature of the client.

The Mobicents XCAP client API was required to provide a means to send XCAP requests to an XCAP Server such as the Mobicents XDM Server. The Mobicents XCAP client API depends on Java HTTP client API to provide the core HTTP functionality and Java HTTP client core API to provide low level HTTP transport components for building services with a minimal footprint.

B. Implementation: Enhancements to the RUCRG IMS Client

In order to add XCAP support, a full analysis (similar to the one done in Section III) of RUCRG IMS client classes was carried out and the classes which needed to be modified, removed or replaced were identified. The process involved:

- Studying the structure of the RUCRG IMS client and identifying the classes which needed to be modified to add XCAP support
- Adding helper classes for populating XCAP specific parameters
- Removing and/or replacing some existing classes with optimised ones that allow the support of XCAP
- Adding XML support to allow the populating of IMS/SIP attributes and to allow persistence

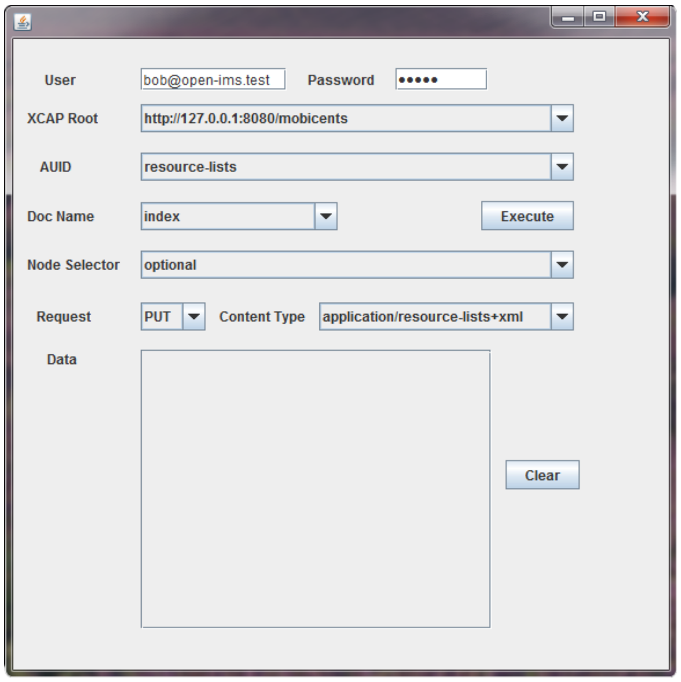
C. Implementation: XCAP support

The RUCRG IMS client was extended to allow authentication with an XCAP server and to permit extracting, parsing and displaying of XCAP documents. A full discussion of how extensions were made follows on.

1) Authentication and Authorisation: Since SIP and XCAP are different protocols, they use different authentication and authorization (AA) mechanisms. Fortunately, the XCAP requests sent to the XCAP server embed the user's authentication details, thus making it fairly simple to capture and embed the user's authentication parameters. This meant that an additional interface had to be created on the client program that was used to collect the user's authentication parameters, which could be used as long as the user was sending requests to the same XCAP server within the same session. Figure 2 shows the interface.

It is interesting to note that within an IMS supported XCAP setting, an additional server program is designed to operate in parallel with other servers and aims to unify the verification management by creating an identical username and password for IMS and XCAP at the moment of registering new IMS account [3].

Figure 2: RUCRG XCAP Client Interface


User: bob@open-ims.test
Password: •••••
XCAP Root: http://127.0.0.1:8080/mobicents
AUID: resource-lists
Doc Name: index
Node Selector: optional
Request: PUT
Content Type: application/resource-lists+xml
Data

2) Client Operations:

- Adding and Modifying
  Adding and modifying work in similar ways; they both use the HTTP PUT request and the body of the request always contains content. However, their behaviour depends on the URI that the client constructs. In the case of adding or modifying a document, the client constructs a document URI that references the location where the document is to be placed. This URI contains the XCAP

root and a document selector. The MIME content type is set to the type defined by the application usage. For example, it would be "application/resource-lists+xml" for a Resource List Server (RLS) services document. The XCAP server checks if the resource exists. If the URI resolves to an existing document, the new content replaces the content selected by the URI. If not, the operation adds the new content such that the URI resolves to the content in the body. Adding or modifying elements and attributes works in a similar manner to adding or modifying documents. To create/replace an element (within an existing document) or an attribute (in an existing element of a document), the client constructs a URI whose document selector points to the document to be modified. A node selector is also added to the URI to help identify a single element or attribute. The MIME content types are set to "application/xcap-el+xml" and "application/xcap-att+xml" respectively. An illustration of how a PUT operation is performed on a document is shown below:

```
PUT    http://127.0.0.1:8080/mobicents/services/resource-
    lists/users/Chiedza/friends.xml HTTP/1.1
Content-Type:application/resource-lists+xml

<?xml version="1.0" encoding="UTF-8"?>
<resource-lists xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance">
<list name="friends" uri="sip:friends@open-ims.test" sub-
    scribable="true">
</list>
</resource-lists>
```

- Retrieving
  Fetching/retrieving is accomplished by performing an HTTP GET request. In order to retrieve a document the client sets the request URI to the document URI. In the case of retrieving an element of a document, the client constructs a URI whose document selector points to the document containing the element to be fetched. The node selector identifies the element to be fetched. To fetch an attribute in a document, the client constructs a URI whose document selector points to the document containing the attribute to be fetched. The node selector contains an expression identifying the attribute whose value is to be fetched. Retrieving is always followed by a 200 OK response from the server if the request was successful. In the case of retrieving an attribute, the 200 OK response will contain an "application/xcap-att+xml" document with the specified attribute. An illustration of how a GET operation is performed on an attribute is shown below:

```
GET    http://127.0.0.1:8080/mobicents/services/resource-
    lists/users/Chiedza/friends.xml?
resource-lists/list/list/entry[@name="Ruvarashe"]/@uri
    HTTP/1.1
```

```
The server responds:

HTTP/1.1 200 OK
Content-Type:application/xcap-att+xml
Content-Length: ...

sip:Ruvarashe@open-ims.test
```

- Deleting
  Deleting is achieved by invoking an HTTP DELETE request. To delete a document, the client constructs a URI that references the document to be deleted. In a similar way to creating or replacing a document, the URI is a document URI. In the case of deleting an element from a document, the client constructs a URI whose document selector points to the document containing the element to delete. The node selector identifies the element to be deleted. To delete an attribute from the document, the client constructs a URI whose document selector points to the document containing the attribute to be deleted. The node selector evaluates to an attribute in the document to be deleted. An illustration of how a DELETE operation is performed on an element is shown below:

```
DELETE http://127.0.0.1:8080/mobicents/services/resource-
lists/users/Chiedza/friends.xml?
resource-lists/list/list/entry[@name="Tasara"] HTTP/1.1
```

It is important to note that these operations are invoked through the use of a user interface as shown in Figure 2. The user supplies parameters through this interface and these are then used to construct the relevant XCAP requests that trigger the behaviour explained above. For example, deleting a user will call the XCAP DELETE operation in the background to delete the element that corresponds to that particular user in the buddy list.

D. Presence Models

There are several ways of accessing presence information for a list but only two will be discussed. One way is to subscribe to a resource which represents that list. In this case, the Resource List Server (RLS) requires access to this list in order to process a SIP SUBSCRIBE request for it as shown in Figure 3.

Another way to obtain presence for the users on the list, is for a watcher to subscribe to each user individually. In this case, it is convenient to have a server store the list, and when the client boots, it fetches the list from the server. This would allow a user to access their resource lists from different clients, as shown in Figure 4. This is the model that was implemented in the RUCRG IMS client.

E. Testing

Testing was carried to determine whether the client was still capable of performing SIP based presence functions using the new XCAP architecture. This scenario was based on end-to-end systems testing, that is, higher level functionality, rather than on specific protocol layers or requirements. Figure 4 shows

Figure 3: Presence Architecture using RLS with XCAP

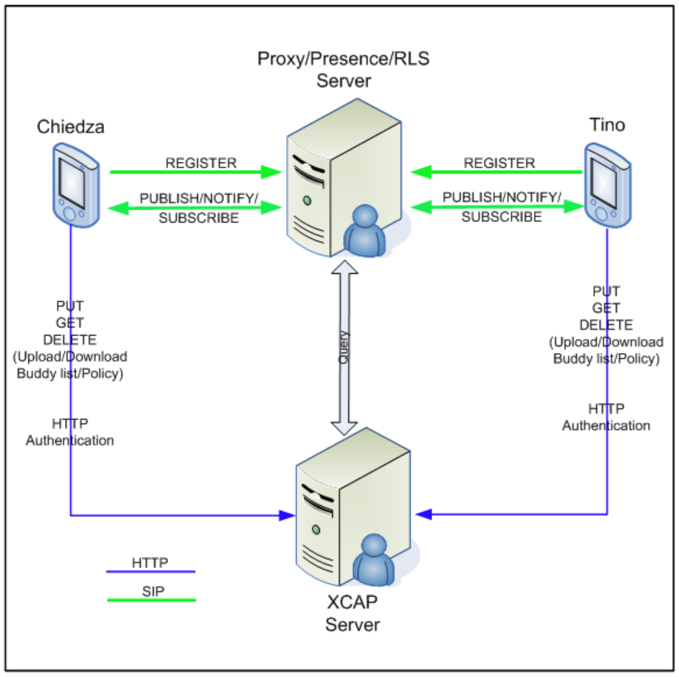
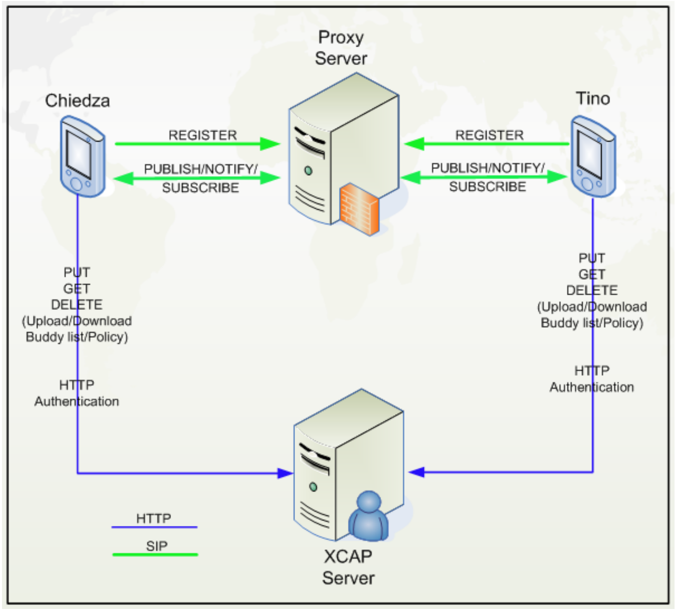

Proxy/Presence/RLS Server
Chiedza
Tino
REGISTER
PUBLISH/NOTIFY/ SUBSCRIBE
REGISTER
PUBLISH/NOTIFY/ SUBSCRIBE
Query
PUT
GET
DELETE
(Upload/Download Buddy list/Policy)
HTTP Authentication
PUT
GET
DELETE
(Upload/Download Buddy list/Policy)
HTTP Authentication
HTTP
SIP
XCAP Server

Figure 4: RUCRG IMS Client Presence Architecture with XCAP


Proxy Server
Chiedza
Tino
REGISTER
PUBLISH/NOTIFY/ SUBSCRIBE
REGISTER
PUBLISH/NOTIFY/ SUBSCRIBE
PUT
GET
DELETE
(Upload/Download Buddy list/Policy)
HTTP Authentication
PUT
GET
DELETE
(Upload/Download Buddy list/Policy)
HTTP Authentication
HTTP
SIP
XCAP Server

the test arrangement. The client automatically retrieves the buddy list from the XCAP server after successfully logging in with either the IMS or a SIP network. It then uses the list to send subscriptions to all the buddies listed. Similarly, the client uploads the latest buddy list onto the XCAP server when the user requests to be logged out and automatically cancels all subscriptions.

The next test scenario entailed evaluation of other XCAP functions that were added to the client, but were not tested by the scenario explained above. Test cases were defined in which the client had to perform the following operations:

- Create an XCAP Document (resource list) on the server
- Add elements to the created XCAP Document (add users)
- Replace the XCAP Document with one that had different elements (users) but the same Document name and structure
- Fetch the newly created Document
- Delete and element (user)

These tests conducted to verify the ability of the client to perform the functions described in Section IV were successful. This means that the client can now upload presence information, and all other clients become watchers and subscribe to this presence information - and vice versa. The purpose of this test is to verify whether the clients support presence functionality, and can subscribe to another clients presentity.

V. Future Work

Notwithstanding the important modifications made to the client to support the core XCAP framework, some work still needs to be done to enhance the client. When subscribing to a presence list using the RUCRG IMS client, subscriptions are created and managed for each presentity in the list by the client. This does not scale well for large lists, particularly in cases where a user is connecting via a wireless network. Thus there is need to integrate the SIP Event Notification mechanism for subscribing to a homogeneous list of resources as described in RFC 4662 [16] instead of sending individual SUBSCRIBE requests for each resource. The watcher (subscriber) can then be able to subscribe to an entire list and receive notifications when the state of any of the resources in the list changes.

VI. Conclusions

In this paper, we provided context and discussed the work done to provide support for storing user data on the network; particularly relating to presence. This was achieved by extending the RUCRG IMS client to make use of XCAP in the storage and retrieval of user data. There is still some work that needs to be done as discussed in Section V, but in terms of providing persistence for user data, the implementation presented in this paper fullfills the initial requirements.

References

[1] R. Perea, *Internet Multimedia Communications Using SIP: A Modern Approach Including Java Practice*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008.

[2] A. Acharya, N. Banerjee, D. Chakraborty, and S. Sharma, “Pressentials: a flexible middleware for presence-enabled applications,” in *Proceedings of the 5th International Conference on Principles, Systems and Applications of IP Telecommunications*, ser. IPTcomm ’11. New York, NY, USA: ACM, 2011, pp. 15:1–15:12. [Online]. Available: http://doi.acm.org/10.1145/2124436.2124456

[3] Z. Lei and P. Coulton, “IMS Based Mobile Presence Service,” in *Future Mobile Experiences: next generation mobile interaction and contextualization, Workshop at NordiCHI*, 2008.

[4] R. Marston, "Multimedia Content Adaptation for IPTV Services in IMS," Master's thesis, University of Cape town, 2008.

[5] W. Muswera and A. Terzoli, "Development of an ims compliant, cross platform client using the jain sip applet phone," *Southern Africa Telecommunication Networks and Applications Conference (SATNAC)*, 2011.

[6] J. Rosenberg, "The Extensible Markup Language (XML) Configuration Access Protocol (XCAP)," RFC 4825 (Proposed Standard), Internet Engineering Task Force, May 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4825.txt

[7] A. Kuwadekar, C. Balakrishna, and K. Al-Begain, "GenXfone-Design and Implementation of Next-Generation Ubiquitous SIP Client," in *The Second International Conference on Next Generation Mobile Applications, Services, and Technologies*. IEEE, 2008, pp. 118–124. [Online]. Available: http://www.computer.org/portal/web/csdl/doi/10.1109/NGMAST.2008.98

[8] A. Bachmann, S. Wahle, and T. Magedanz, "An IMS Client Application Framework for Emerging NGN Testbeds," in *Proceedings of the 4th International Conference on Testbeds and research infrastructures for the development of networks & communities*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, pp. 1–6.

[9] L. Etiemble and M. Diop. (2010, August) Mercuro IMS Client. Website. [Online]. Available: http://imsclient.blogspot.com/

[10] D. Waiting, R. Good, R. Spiers, and N. Ventura, "The UCT IMS Client." IEEE, 2009. [Online]. Available: http://www.computer.org/portal/web/csdl/doi/10.1109/TRIDENTCOM.2009.4976254

[11] UCT IMS Client. Website. UCT. [Online]. Available: http://uctimsclient.berlios.de/

[12] M. Diop. (2009) Inside IMS/NGN. Website. Inexbee. [Online]. Available: http://betelco.blogspot.com/2009/02/new-version-of-mercuro-ims-client.html

[13] E. Ivov. (2007) SIP Communicator. Slides. [Online]. Available: jres.org/planning/slides/132.pdf

[14] JAIN SIP API. [Online]. Available: https://jain-sip.dev.java.net/

[15] SunMicrosystems. Java Media Framework. Website. [Online]. Available: http://java.sun.com/products/java-media/jmf/

[16] A. B. Roach, B. Campbell, and J. Rosenberg, "A Session Initiation Protocol (SIP) Event Notification Extension for Resource Lists," RFC 4662 (Proposed Standard), Internet Engineering Task Force, Aug. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4662.txt

Walter Muswera received his honours degree in 2009 from Rhodes University and is presently studying towards his Master of Science Degree in Computer Science at the same institution. His research interests include converged telecommunication networks and mobility.

Alfredo Terzoli is Professor of Computer Science at Rhodes University, where he heads the Telkom Center's of Excellence in Distributed Multimedia. He is also Research Director of the Telkom Centre of Excellence in ICT for Development at the University of Fort Hare. His main areas of academic interest are converged telecommunication networks and ICT for development.

The work reported in this paper was undertaken in the Telkom Centre of Excellence in Distributed Multimedia at Rhodes University, with financial support from Telkom SA, Tellabs, Genband, Easttel, Bright Ideas 39 Projects, and THRIP.