HIV Testing Site (HTS) Locator: Applying Current Computing Trends to Voice Applications

Mathe Maema[1] and Alfredo Terzoli[2]
Department of Computer Science
Rhodes University
Grahamstown, South Africa
E-Mail: [1]mathe@rucus.ru.ac.za [2]a.terzoli@ru.ac.za

Lorenzo Dalvit
Department of Education
Rhodes University
Grahamstown, South Africa
E-Mail: ldalvit@gmail.com

***Abstract*—The potential of voice interactions with machines, independent on the visual modality or the possibility to signify intentions through the movement of parts of the body, means that voice applications of different kinds will continue to play an important role in the future. This paper provides a simple design, in the context of telephony, based on current trends in technologies that shows how future voice applications will be built. The design is aligned to the VoiceXML specification and demonstrates a move towards use of rules and ontologies for structuring and building a rich back-end.**

***Index Terms*—Ontologies, Rules, VoiceXML.**

I. Introduction

THE use of self-service is recognised by many sectors of the economy as beneficial. Over the years, the use of IVR self-service, in particular, has raised a delicate question about where the balance of interest lies between users and service providers. On the one hand, due to benefits that may translate to reduction in operational costs, service providers have promoted and encouraged self-service. On the other hand, users have expressed frustration and dissatisfaction with this service [5].

Despite this, anecdotal evidence suggests that IVR self-service and the general use of voice applications will continue to increase. The simple reason for this is that voice communication remains good for many situations where visual communication may not be appropriate. For example, voice, as opposed to visual communication, may be better for delivering services to the semi-literate users. Also, it is certainly better for users whose eyes and bodies are engaged in other tasks such as the case when driving a car. As a case-study, this paper will discuss the implementation of a voice application that is aligned particularly to the former context of use.

The case-study application, named HIV Testing Site (HTS) Locator, is an IVR self-service system designed to help individuals locate information about service providers that offer HIV testing. The system was intended specifically for use by all members of Rhodes University community, as part of reinforcing the 'know-your-status drive'.

Besides representing a context that favours voice modality, HTS Locator was designed to provide a moderately futuristic view of next-generation IVR applications. Based on observations of current trends in computing, in addition to using VoiceXML specification, they will be influenced heavily by developments in the field of Artificial Intelligence. At a minimum, they will have explicit representation of relevant knowledge as their back-end, with inference engines for reasoning with that knowledge.

This knowledge may pertain, for example, to the dialog, the task at hand or the domain, so it is important to note that different reasoning mechanisms may be required for each type of knowledge and so different specialised engines may be used. To illustrate this idea, for the implementation of the case-study application two different types of inference engines are used: a rule engine and a reasoner. Basically, the rule engine is used for processing rules needed for generating dialog; and the reasoner is used for processing domain related knowledge for answering user queries.

The rest of the paper is structured as follows. Section II will provide context by discussing related work. Section will provide the architectural design of the entire system. Section IV will then briefly discuss the front-end and its implementation. The back-end will be discussed in Section V. This will be followed by the conclusions in Section VI.

It should be noted that in this paper, front-end refers only to what is rendered while back-end is used loosely to describe all that literally happens in the back, beyond the interface with the user. Therefore, what would otherwise be referred to as the middle tier when using architectural design patterns such as n-tier architecture, is qualified as part of the back-end. This is done to simplifying the presentation of the paper and is by no means a reflection of lack of separation of concerns in the implementation.

II. Related Work

Much of the work relevant to this paper can be gleaned by broadly looking into trends in computer science and what is fuelling them. The shift to the Internet Protocol (IP) driven networks and the ever growing access to computing processing resources (Moore's Law) are among the two drivers for the progression of voice applications. Arguably, these two factors hold true for all other advances within the broad industry of computing. Nonetheless, both are useful in pointing out that the move is toward rich back-end driven voice applications, developed no differently than web applications. In fact, through use of VoiceXML the back-end for voice and web applications will be shared.

The authors would like to acknowledge the financial support of Telkom SA, Comverse SA, Stortech, Tellabs, Amatole Telecom Services, Bright Ideas 39, and THRIP through the Telkom Centre of Excellence in the Department of Computer Science at Rhodes University.

According to Schoeller [17], the adoption of VoiceXML as a standard, influenced by the movement to the IP network, is driving "the evolution toward a more fully web-application-oriented stack". Although this was said a few years ago, this remains true and with the soon to be released VoiceXML 3.0 [13], the full alignment may not be that far. As an improvement to previous versions, VoiceXML 3.0 encourages robust separation of concerns. For example, it offers clear separation of flow from presentation in that the presentation layer is not tangled with <*goto*> logic. This is important because separation of concerns enables an application to be built or extended quickly and cost-effectively, since the building process is necessarily modular.

In as far as the processing resources are concerned, an argument has been put forth that also has implications for improving user satisfaction with the use of voice applications and IVR applications in particular. This argument pertains to development of intelligent dialogs. According to authors like Flycht-Eriksson *et al*. [8], practical implementation of intelligent dialogs requires a back-end with multiple processing engines for reasoning with specific types of knowledge that need to be represented. In essence, they are suggesting that a dedicated processing engine be available for reasoning with either domain knowledge, task knowledge, generation knowledge or any other knowledge represented within a dialog based system. As it may be deduced, whilst this setup may increase the computational power of the entire system, it comes at a cost. In the past, the cost might have been prohibitive, but today this not necessarily the case (precisely because of Moore's law, which roughly translates to a growing trend of more computational power being availed at a lesser cost).

Artificial Intelligence (AI), a field of study dedicated to machine intelligence, has identified knowledge representation as one of the key ingredients in understanding and generating intelligent behaviour. As a consequence, there has been a lot of ongoing research in this area. The research has provided many forms of representation. However, ontologies in comparison to other forms of representation have received more popularity within AI and other disciplines of computing [6]. As a result, current trends indicate a move towards a broad adoption of ontologies in any system that enables knowledge and/or information exchange.

III. System Overview

Figure 1 provides a picture of the architecture used for the HTS Locator system. This architecture is based on the architectural model proposed for use with VoiceXML applications, as described in [7]. To provide clarity on how the various components of the architecture interact with each other, a use case for the system is discussed below.

When a call is received, the telephony platform handles the processing details of the call such as interactions with the telephone network (PSTN). The call is then forwarded to the VoiceXML browser. As shown in the figure, I6NET VoiceXML browser called VXI* [2] is used. It was selected primarily because it provided easy integration with iLanga (an Asterisk based platform developed at Rhodes University [16]). An initial experiment integrating VXI with iLanga is reported in [11][12].

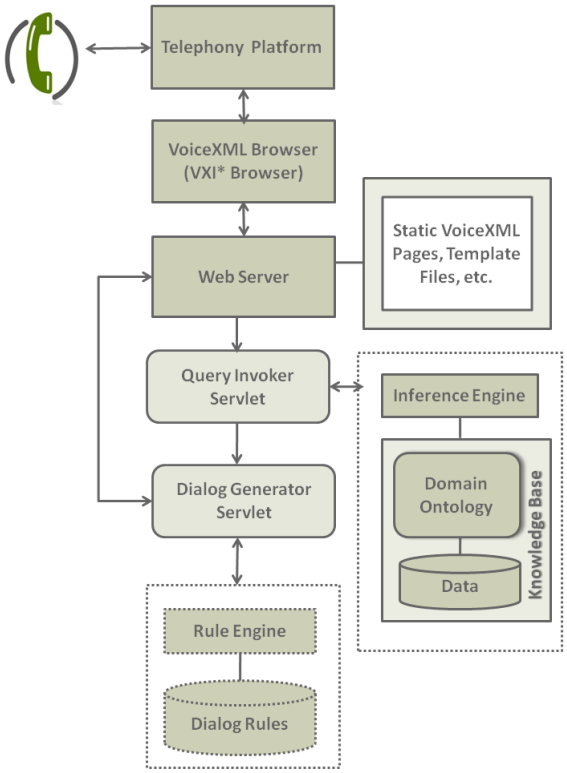

Figure 1. Overall System Architecture

After the call is forwarded, the browser maps the number called to the URI of the initial document to fetch. A request is then sent to the web server for the initial document. In line with the request, the web server sends the document to the browser. The browser then interprets the document and renders it appropriately using services offered by the telephony platform.

The result of the interpretation is a welcome message that prompts the user for relevant criteria for locating a suitable HIV testing facility. Assuming the caller does not hang up, he/she will provide the criteria as numeric input (DTMF). This input is used to generate a query to the knowledge base via the inference engine following a process that is identical to an HTML form being submitted to a servlet. The only difference is that VoiceXML is used and not HTML.

To briefly articulate this process, when the dialog is completed for collecting user input, the form is submitted and the *Query Invoker* servlet is invoked to process the user's input. In this case, this involves querying the knowledge base and forwarding the results to the *Dialog Generator* servlet. This servlet is responsible for deciding what to send to the user as a response. This decision is governed by rules for generating dialogs, which will be discussed in Section V.

Overall, as it may be deduced from the use of servlets, java was used as the glue for making the various components of the system work together. Java was chosen precisely because of trends that indicate that we are steadily moving into a generation of Java based tools especially within telephony. This can be evidenced by the growing adoption of platforms like Mobicents, a popular open source VoIP platform that is based on efforts by the Java community [19].

IV. FRONT-END IMPLEMENTATION

The most important thing in implementing the front-end of voice applications is the design of conversations (dialogs) that the user will have with the computer. These conversations effectively qualify as the user interface for the application.

There are a number of strategies based on voice user interface design principles that can be followed to ensure that these conversations do not lead to user frustration. The broad objective of these strategies is to ensure that the short-term memory of users is not overloaded nor is their time wasted. With this in mind, the conversations for different scenarios were written out in a drama script format to simulate communication by the user and the system.

On the basis of the produced scripts, pages based on VoiceXML 2 were created. An example page is shown below. Essentially, this page includes two dialogs represented within *form* tags. The first dialog greets the user and the *goto* tag moves it to the options dialog. Within the options dialog, the user is asked to make choices that would aid in locating a suitable provider. These choices are submitted to a servlet on the web server for further processing.

```
<?xml version="1.0"?>
<vxml version="2.1" xmlns="http://www.w3.org/2001/vxml"
    >
  <!--Error handling logic-->
  <!--Variable declaration-->

<!-- Start initial dialog -->
  <form id="greeting">
    <block>
      <audio src="">
        <!-- Welcome message -->
      </audio>
      <goto next="#options"/>
    </block>
  </form>

 <!--Present user with options -->
  <form id = "options">
    <!--Determine cost preferance -->
    <field name ="costParam" type="boolean?y=1;n=2;">
      <prompt>Are you interested in a free or low cost
          testing site? Press 1 for yes and press 2
          for no.</prompt>
      <filled>
        <if cond="costParam=='true'">
          <assign name="cost" expr="'lowCost'"/>
        <elseif cond="costParam=='false'"/>
          <assign name="cost" expr="'moderateCost'"/>
        </if>
      </filled>
    </field>

  <!--Determine distance preferance -->

  <!--Submit user selection -->
  <block>
     <submit next="http://localhost/voiceLocApp/query"
         method="post" namelist="distance cost"/>
  </block>
</form>
</vxml>
```

In this section, details of the generation of the pages was not discussed. To an extend, this is because the task itself is not complicated. It is however worth mentioning that these pages were either generated statically or dynamically by the back-end.

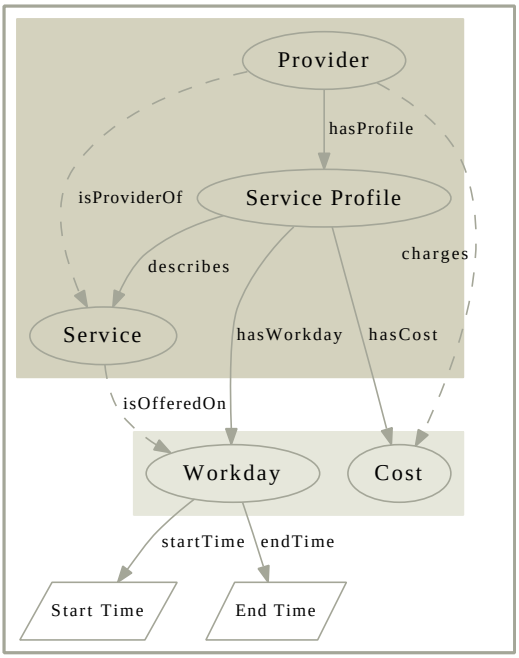

Figure 2. High-level Conceptual Model

V. BACK-END IMPLEMENTATION

As it has been alluded to before, the back-end of HTS Locator is driven by two inference engines. Although the engines work together, each performs its own designated tasks. This section will discuss the overall approach followed for their integration into the application.

A. Interfacing with the ontology knowledge base

Over the years, the process for maintaining data has evolved from using a flat file structure to use of databases. Currently, we are at a turning point moving towards wide use of ontologies.

Although numerous definitions exist for what an ontology is, the most widely used definition describes an ontology as “an explicit specification of a conceptualisation” [9]. The process of building an ontology is a non-trivial one [14]. It is iterative and demands careful analysis, feedback and redesign through each stage of the development life cycle.

This process and its ‘grounding’ will not be discussed in detail within this paper. However, for purposes of clarifying what is meant by specification of conceptualisation, Figure 2 captures the high-level conceptual model yielded through the construction process. As the figure shows, there are concepts such as *Provider*, *Service Profile*, *Service* and *Cost*. These concepts are related to each other via binary relations such as *hasProfile, describes* and *isProviderOf*. Also shown is that binary relations can exist between concepts and attribute values. In the figure, this idea is represented by binary relation *startTime,* which links concept *Workday* to a start time value, represented in the figure using a parallelogram.

Further, the figure illustrates that there are at least two types of binary relations. There are simple relations, which have been represented using solid arcs. These explicitly state that two concepts are related to each other. The second variety are composite relations and these have been represented

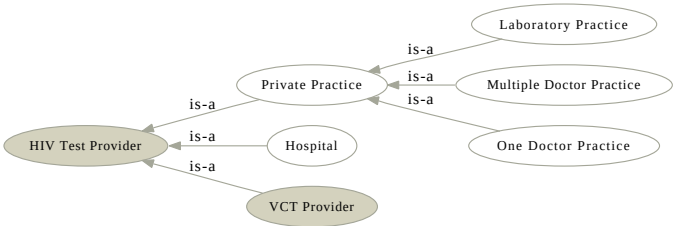

Figure 3. Inferred Model for HIV Test Providers

using dashed arcs. These relations indicate how deductions can be made in order to conclude that a relation exists between connected concepts. For example, *isProviderOf* is a composition of *hasProfile* and *describes* (i.e. *hasProfile* o *describes*). This means that if some provider X has profile Y that describes some service Z, then it can be deduced that X is provider of Y.

From the definition, it can be gleaned that conceptual modelling is the main activity within the ontology construction process. Indeed, this is true, due to the availability of tools for automating the conversion of the conceptual model to one that is executable by the computer. Protégé-4 [1], a free and open source ontology editor was used for this purpose. We chose Protégé primarily because it is regarded as a leading ontological engineering tool that enables easy creation of ontologies that can be represented in various formats.

Following this, the ontology was instantiated i.e. instances (individuals) and their attributes were added in order to produce a working knowledge base (since without instances the concept of knowledge base does not exist). The produced ontology was then rendered as an XML document.

With the knowledge base created, the next vital step was building a small application that would act as an interface to it and a reasoner so that user queries may be processed. The purpose of the reasoner is to perform tasks such as checking for consistency and answering queries. Pellet [18] reasoner was chosen for use based on two reasons. First, at the time of construction, it showed a high degree of conformity with Web Ontology Language version 2 (OWL 2). This was important in that, inadvertently, a commitment was made to produce OWL 2 ontologies by selecting to use Protégé-4 and its features. Second, as a reasoner, it is reported to perform reasonably well for many use cases. (In some cases other reasoners may out-perform it [18]).

OWL API [3], [10], a high level Application Programming Interface (API) for working with OWL 2 ontologies was chosen for the construction of HTS Locator. Using this API, the application was built to carry out the tasks of loading the ontology, setting up the reasoner and processing queries.

As a strategy for bringing efficiency, a decision was made to allow only queries that pertain to core objectives of the IVR application, which in this case, is to locate HIV test providers. This translated in grounding the operation of the interface to the IVR, but this was not deemed to be a negative thing; given that with voice, it is not possible to present information in parallel.

To put into context how this grounding effect was achieved, in loading the ontology, only the inferred model relating to HIV test providers is loaded into the working memory of the reasoner. This model can be visualised as a taxonomy tree, as shown in Figure 3. From the perspective of the interface, all the answers are captured within this tree. Therefore, the act of answering any query, amounts to traversing the tree using algorithms provided by the reasoner. It should be noted that based on some specified criteria, the reasoner can reconfigure the tree as part of refining an answer.

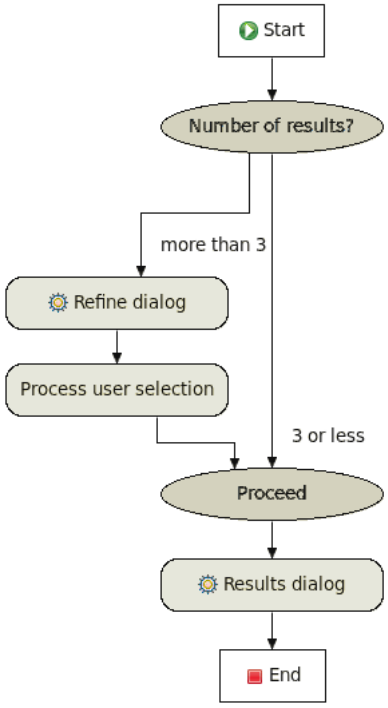

Figure 4. Process flow for generating VXML dialogs

B. Dialog Processing with rules

Given that it can never be predicted how many results will be generated by a query, and that with voice only one result can be given at a time, a question of presentation arose. This question was: how can results be presented to users without taxing their short-term memory?

The answer was to either present the results in batches to the user or find some criteria that could be used to shorten the list. With regard to not overloading the users' short-term memory, the latter was preferred. Based on this decision, a dialog flow (see Figure 4) was generated to capture how the implementation would be achieved. As shown, if the number of results produced is less than or equal the thresh-hold value, which is three in this case, the results are presented to the user i.e. VoiceXML page for the results is generated and delivered to the user. Otherwise a VoiceXML page is generated to ask the user for criteria to be used to refine the results. After the user has provided an answer, the results are refined and presented to the user.

For all the pages generated, StringTemplate [15] was used. StringTemplate is a rendering engine that distinguishes itself by its ability to enforce separation of generation logic from the output format.

Processing the results with provided criteria could have been achieved by matching criteria with *if-then* statements.

However, current trends are moving away from imperative to declarative programming, where rules are used. The former focuses on how to perform a task while the latter focuses on what to do. As cited in [4], the primary benefit of this shift of *if-then* to *when-then*, is that in separating knowledge from implementation, maintenance costs are reduced in a sense that a change in the rules does not lead to a change in the source code. There are many other benefits that include easy representation of knowledge (business logic) in manner that is readable and easy to update by non-technical users.

Given the benefits, the next question was which rule engine to use? A number of alternatives were considered including using Pellet used above for implementing the ontology processing engine. However, after due consideration, we decided to use a different engine to show how multiple technology offerings can be used together to enrich service development. We selected Drools (JBoss Rules) [4] based on evidence of community support and the fact that is free.

In line with the process flow in Figure 4, the necessary implementation using Drools was done such that the rule engine is invoked after the user has submitted their criteria for refinement. Then rules stored in repository are fired and matched against the criteria. Following this, the refined results are presented to the user.

```
rule "Match criteria 1 - Alphabetic sort "
  salience 3
  when
    $c:Criteria(userSelection==Criteria.CRITERIA_1)
    $q:QueryResultMap($map:resultMap)
  then
    storeResultsMap = new TreeMap($map);
    $q.setResultMap(storeResultsMap);
    $q.setRefined(true);
end
```

The rules authored for use in Drools are represented in form shown above. As shown, the example rule matches criteria 1 which is alphabetic sort. This rule is matched when user selection is *CRITERIA_1* and the query result map (*QueryResultMap*) exists. When this conditions are satisfied the rule fires and the query result map is put into a tree map, which performs the sorting and the results are stored in the *storeResultsMap* variable. Within the engine, the instance of the query result map (*$q*) is updated with stored results map and the refine variable is set to true.

VI. Conclusions

Although voice applications may have a limitation of presenting information serially, they offer a powerful alternative for contexts where visual modality may not be appropriate. For this reason, there is no doubt that the development of these applications will continue to rise. The question, however, is how the development will evolve?

In this paper, through a simple voice application in which users query a knowledge base for information, we demonstrated how the development of voice applications might look in the future.

References

[1] Protégé. Online: http://protege.stanford.edu/ [Last accessed: 8 April 2008].

[2] VXI* VoiceXML Browser Manual. Online: http://www.i6net.com/support/documents/ [Last accessed: 9 Feb 2008].

[3] *The OWL API: A Java API for Working with OWL 2 Ontologies*, 2009.

[4] Paul Browne. *JBoss Drools Business Rules*. Packt Publishing, 2009.

[5] Dwane H. Dean. What's wrong with IVR self-service. *Managing Service Quality*, 18(6):594–609, 2008.

[6] Gašević Dragan, Djurić Dragan, and Devedžić Vladan. *Model Driven Architecture and Ontology Development*. Springer, 2006.

[7] Jim Ferrans, Brad Porter, Steph Tryphonas, Bruce Lucas, Peter Danielsen, Daniel C. Burnett, Andrew Hunt, Scott McGlashan, Ken Rehor, and Jerry Carter. Voice Extensible Markup Language (VoiceXML) Version 2.0. W3C recommendation, W3C, March 2004. http://www.w3.org/TR/2004/REC-voicexml20-20040316/.

[8] Annika Flycht-Eriksson and Arne Jönsson. Dialogue and domain knowledge management in dialogue systems. In *Proceedings of the 1st SIGdial workshop on Discourse and dialogue*, pages 121–130, Morristown, NJ, USA, 2000. Association for Computational Linguistics.

[9] T. R. Gruber. Towards principles for the design of ontologies used for knowledge sharing. In N. Guarino and R. Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Deventer, The Netherlands, 1993. Kluwer Academic Publishers.

[10] Matthew Horridge and Sean Bechhofer. Igniting the owl 1.1 touch paper: The owl api. In *In Proc. OWL-ED 2007, volume 258 of CEUR*, 2007.

[11] Adam King. Constructing a low-cost, open-source, VoiceXML Gateway. Master's thesis, Rhodes University, 2007.

[12] Adam King, Alfredo Terzoli, and Peter Clayton. Creating a low cost VoiceXML Gateway to replace IVR systems for rapid deployment of voice applications. In *Southern African Telecommunications, Networks and Applications Conference (SATNAC)*, pages 131–136, September 2006.

[13] Scott McGlashan, Daniel C. Burnett, Rahul Akolkar, RJ Auburn, Paolo Baggia, Jim Barnett, Michael Bodell, Jerry Carter, Matt Oshry, Kenneth Rehor, Milan Young, and Rafah Hosn. Voice Extensible Markup Language (VoiceXML) 3.0. Online: http://www.w3.org/TR/2010/WD-voicexml30-20100304/ [Last accessed: 2 May 2010].

[14] N.F. Noy and D.L. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical report, Stanford University, 2001.

[15] Terence John Parr. A functional language for generating structured text. Online: http://www.cs.usfca.edu/~parrt/papers/ST.pdf [Last accessed: 5 March 2010], 2006.

[16] J. Penton and A. Terzoli. iLanga: A next generation VOIP-based,TDM-enabled PBX. In *Southern African Telecommunications Networks and Applications Conference (SATNAC)*, 2004.

[17] Art Schoeller. Voice Self-Service Aligns with Web Architectures to Reduce TCO, Increase Flexibility and Ensure Content Consistency. Technical report, YANKEE Group, 2006. Online: http://www.cisco.com/en/US/prod/collateral/voicesw/custcosw/ps5694/ps1006/prod_white_paper0900aecd8051711b.pdf[Last accessed: 7 May 2010].

[18] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical owl-dl reasoner. *Journal of Web Semantics*, 5(2):51–53, 2007.

[19] Mosiuoa Tsietsi, Alfredo Terzoli, and George Wells. Mobicents as a Service Creation and Deployment Environment for the Open IMS Core. In *Southern African Telecommunications, Networks and Applications Conference (SATNAC)*, 2009.

Mathe Maema completed her BSc(Hons) at Rhodes University. She is currently reading towards an MSc degree at the same institution. Her research interests are in the area of telephony and knowledge management.

Alfredo Terzoli is Professor of Computer Science at Rhodes University, where he heads the Telkom Centre 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.

Lorenzo Dalvit is a lecturer in Education, where he is responsible for Information Communication Technology (ICT), both at an academic and practical level. He also works in close collaboration with the Telkom Centre of Excellence in Distributed Multimedia; with the South Africa-Norway Tertiary Education Development Programme and with the NGO Translate.org.za.