ChattaBox: A Case Study in Using UML and SDL for Engineering Concurrent Communicating Software Systems

P S Kritzinger, M Chetty, J Landman, M Marconi and O Ryndina
Data Network Architecture Laboratory
Department of Computer Science
Private Bag Rondebosch
7701
South Africa
Tel: +27 21 650 3127
Fax: + 27 21 689 9465
Email: dna@cs.uct.ac.za

***Abstract*— This paper describes a case study that was conducted to investigate software engineering of concurrent communicating systems (CCSs). Best practice software engineering methodologies were used to specify and design a Voice over IP (VoIP) system, which was then implemented. The methodologies utilised were the Unified Modelling Language (UML) and the Specification and Description Language (SDL), and the project specifically explored their combined use.**

The VoIP system implemented, called ChattaBox, allowed users to communicate via voice, as well as several other features. The system requirements and static design for the system were carried out using UML diagrams. Dynamic design was done using UML initially, followed by a conversion to SDL using a tool provided by Telelogic. The resulting SDL design was verified using the tool.

The final system was tested for correctness, performance and usability. It met all of the requirements set out at the initial phase of the engineering process, whilst remaining stable and protocol compliant. After evaluating the engineering process itself, it was concluded that the software engineering paradigm is vital to the field of CCS engineering. Furthermore, UML was useful for providing fast high level design capabilities, but was unable to provide adequate verification of the design. The converted SDL diagrams made up for this, although the biggest drawback of the proposed software engineering process was the inefficient and error-prone conversion, which needed much manual correction and intervention.

***Index Terms*— VoIP, UML, SDL, Software Engineering**

I. Introduction

An effective software engineering process is of great importance when developing a software artefact in today's world of complex technology. However, the field of software engineering does not yet offer an approach to software development that is suitable to all types of systems. This paper focuses on engineering *concurrent communicating systems* (CCSs). Applications using mobile communication and Internet protocols are prime examples of these systems.

A case study investigating the engineering process for concurrent communicating software systems was conducted. The main objective of the case study was to analyse the possible software engineering routes when creating such systems, using existing best practice methodologies and Computer Aided Software Engineering (CASE) tools. A *Voice over the Internet Protocol* (VoIP) system called ChattaBox was specified, designed and implemented for this purpose.

Initially, the basic requirement for the software to be created was that it supported VoIP communication [1]. However, to make it sufficiently complex for the case study, ChattaBox was evolved into a full-grown industry-strength distributed application. Services such as voice mail and dynamic address book were added. Furthermore, it was decided that load balancing, security and data management on the server side of the ChattaBox system should be supported.

During the case study, the possible ways of combining the existing software engineering methodologies to arrive at an effective approach to building complex software systems were explored. The existing design methodologies that were utilised were the *Unified Modelling Language* (UML) [2] and the *Specification and Description Language* (SDL) [3]. Furthermore, usefulness of the object oriented design paradigm in this field was investigated. Finally, the value of CASE tools for software development was examined.

A "forward engineering" approach was followed [4]. In accordance with this approach, specification and initial design were done in UML. Thereafter, the UML design diagrams were converted to an SDL specification, which was then verified and validated. The development process is illustrated in Figure 1.

In order to put the forward engineering process into practice, a means of converting the UML design to an equivalent SDL

This project was undetaken at the Data Network Architecture (DNA) Laboratory, University of Cape Town. The DNA Lab has a formal research cooperation agreement with INT (near Paris) funded by the French Embassy in South Africa. The research of the Group is further funded by the National Research Foundation (NRF) and the national Technical and Human Resources for Industry Programme (THRIP) with Siemens Communications (South Africa) and Telkom as industry partners.

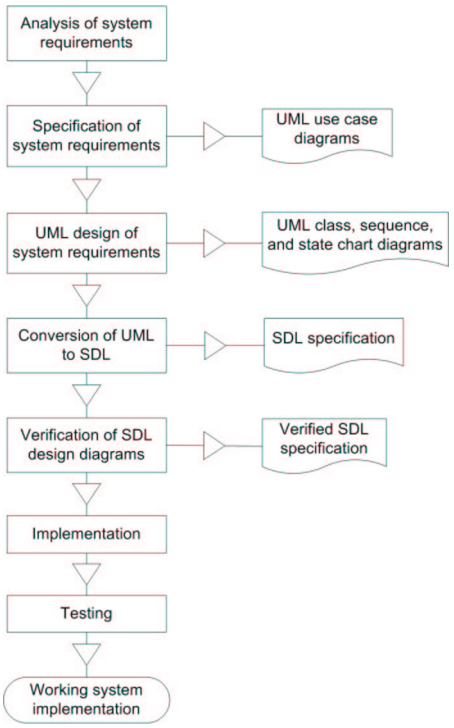

Fig. 1. Forward engineering with UML and SDL

specification was required. UML Suite 4.5 as well as SDL and TTCN Suite 4.3 offered by Telelogic were employed for this purpose [5] .

II. Background

Before the actual development of the system, background research was undertaken on the best practice methodologies UML, SDL and the protocols involved in Voice over IP.

A. Unified Modelling Language

According to Booch et al [2], UML is a graphical modelling technique used for software engineering. Its use has become very widespread during the recent years due to the increased interest in the object oriented design methodology. UML has been standardised by the Object Management Group (OMG) since November 17, 1997. It can be used to model many different types of systems although it was primarily designed to aid software development. Its principal purpose is to facilitate communication between programmers, software architects and clients. In order to do this, it provides facilities for modelling each phase of the software development life cycle from requirements analysis to implementation.

An advantage of using UML is that it has many different diagrams on offer, which allow one to have many different views on a particular system. Another positive aspect of using UML is that there is currently extensive CASE tool support for UML.

On the other hand, as mentioned previously, the major disadvantage of UML is that it has no formal semantics. This makes it difficult to simulate UML models in order to test them for correctness and is a major concern when engineering CCSs.

B. Specification and Description Language

As stated in Belina et al [3], SDL is a formal language used to specify and describe the functional behaviour of software systems. It is standardised by the ITU (International Telecommunication Industry). SDL has both a graphical format (SDL/GR) and a textual format (SDL/PR).

The major strength of SDL lies in its ability to simulate or meta execute models. This allows one to verify and validate systems specified with this language. When specifying concurrent communicating systems, in particular, this allows one to detect errors before implementation. This is cost effective and efficient. Moreover, SDL diagrams form a clear hierarchy of detail from high level system diagrams to low level process diagrams.

One disadvantage of using SDL is that, at present, there is very little CASE tool support. A further disadvantage is that SDL has fewer diagrams and therefore not as many views on the system being specified. Lastly, SDL does not provide for requirements specification.

It is for this reason that SDL was chosen only for the latter part of the design of the ChattaBox system that was implemented. A brief discussion of Voice over IP is given next.

C. Voice over Internet Protocol

VoIP comprises a protocol stack designed for sending voice data over packet switched networks. Voice signals are digitally encoded using codecs or coder-decoders. This data is then packetised and transported over an IP network (LAN, Internet, etc.) between terminals.

The VoIP system that was designed and built for the purpose of this case study involved implementations of *Real Time Protocol* (RTP), *Real Time Control Protocol* (RTCP), *Session Description Protocol* (SDP) and *Session Initiation Protocol* (SIP). These are described in more detail in [6] and [1]. At the top level of a VoIP system, SIP is the standard protocol for initiating an interactive user session that involves multimedia elements such as voice [7]. SIP is used in conjunction with SDP to convey information about media streams to participants of a multimedia session [8].

The actual voice packets are transported using RTP , a protocol that specifies a way for programs to manage the real-time transmission of multimedia data over either unicast or multicast network services [9]. Finally, RTP is complemented by RTCP which makes it possible to monitor data delivery. Monitoring allows the receiver to detect if there is any packet loss and to compensate for any delay jitter. The protocols involved in VoIP were deemed sufficiently complex to fully test the use of UML and SDL for CCS engineering.

III. Related Work

There is much interest in the combined use of UML and SDL. Holz [10], feels that the software engineering process should incorporate the use of both languages. He claims that UML is better suited to analysis and the early design of a system, whereas SDL is more useful for detailed design and as a high level implementation language. Furthermore, he

suggests extending SDL to include UML concepts. In a similar vein, Bjorkander [11] and Dimitrov et al [4] advocate the combination of UML's expressive power with SDL's coherence and semantics.

Moreover, many researchers have investigated mapping one language to the other. For instance, Selic et al [12] have specified how to map SDL to UML through certain extensions. Similarly, ITU have released recommendation Z.109 detailing how to map UML to SDL.

Lastly, the organisations standardising both languages are extending the languages to include concepts from both. ITU's latest version of SDL, SDL 2000, attempts to align SDL with UML. Likewise, OMG is currently working on UML 2.0 which will include concepts that will allow for the validation of UML specifications.

Clearly, both languages have their respective supporters and there is much scope for investigating the combined use of the languages.

IV. The Software Engineering Process

The ChattaBox system developed for the purpose of the case study was engineered using the process outlined below.

A. Requirements Analysis and Specification

The first phase of any software engineering process is analysis and specification of requirements. Initially misinterpreted system requirements may translate into a flawed product.

The ChattaBox requirements were specified using UML use case diagrams, which specifically allow for engineering of system requirements. These diagrams have three main components: an *actor* (portrayed as a stick figure), a *system* with which the user interacts (portrayed as a box/block) and *use cases* (shown as bubbles inside the system block). Actors can represent human users or systems and their components.

In the case of the ChattaBox system, two main requirement groups were identified: *end user* requirements and *internal system* requirements.

End user requirements, as stated above, refer to the functionality provided by ChattaBox to a human user. Establishing these requirements involved identifying all the possible desirable functions of the software. The main function that the ChattaBox software had to provide was to allow users to establish voice calls. Additionally, a number of other feature requirements were identified. The diagram in Figure 2 contains a use case diagram that formed a part of the end user requirements specification for ChattaBox.

From the use case diagram it is easy to see what is required of the ChattaBox software from the end user's perspective. ChattaBox was required to keep an account for each user, and provide a logging in facility. Accounts would allow users to keep personalised settings and an address book. Additionally, the software had to allow the user the basic call functionality, such as to place and answer a call. A status service was also to be provided to reflect users' availability to accept calls.

Besides the basic functionality mentioned above, advanced requirements were established for ChattaBox, which included voice mail services among others.

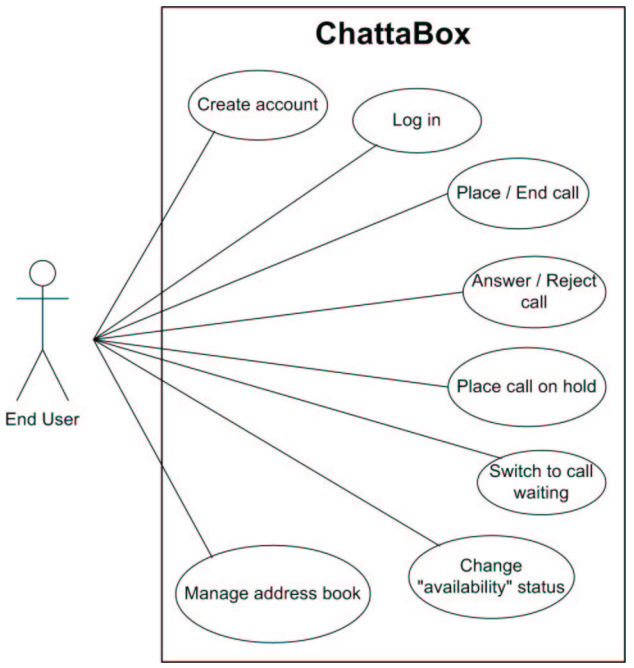

Fig. 2. End user requirements for ChattaBox

The internal system requirements were used to describe the needs of various components of the ChattaBox's distributed architecture. Each component would have to provide a service to other components. These requirements were also mapped out using use case diagrams.

B. Design

The design phase began with an overview of the architecture of the ChattaBox system, which was represented using UML component diagrams. These diagrams are a good place to show the high level communication and distribution of components and objects.

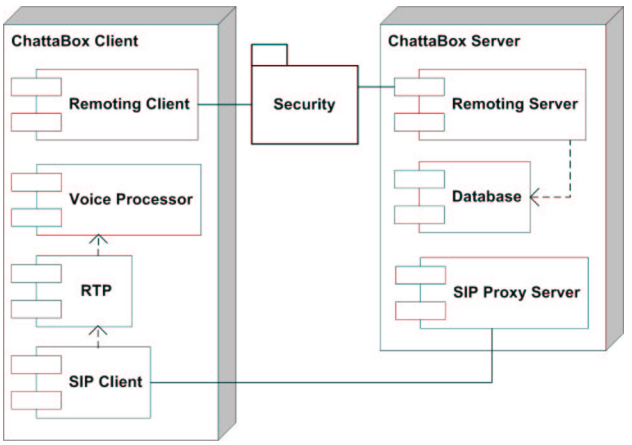

Fig. 3. Overview of the ChattaBox system

Figure 3 presents the architectural overview of the ChattaBox system. The system is roughly composed of two entities, the *client* and the *server*.

The ChattaBox server consists of three major components:

1) Remoting Server - This component relies on Microsoft's .NET Remoting infrastructure to make abstracted network calls. It enables facilities such as user status changes and voice mail. Note that this component relies on the *Security* package for authentication purposes.
2) Database - The database component manages persistence of data.
3) SIP Proxy Server - The SIP Proxy server component is responsible for forwarding SIP messages, which are used during session establishment.

The ChattaBox client is composed of four major components:

1) Remoting Client - The Remoting client communicates with the Remoting server, using the Security package for authentication of users.
2) SIP Client - The SIP client sends and receives SIP messages to allow for session establishment.
3) RTP - The RTP component processes real-time voice data, transporting it between two ChattaBox clients.
4) Voice Processor - The Voice processor component captures audio data from the sound input stream to be sent via RTP.

The Security package includes software that allows for client and server authentication using asymmetric keys and a certificate scheme based on X.509 [13].

1) Static Design : The next phase concerned the structural, static aspect of the design. UML class diagrams were employed during this phase. Class diagrams describe the types of objects in the system and the various kinds of static relationships that exist between them. They also show the attributes and methods of a class and constraints concerning the relationships between objects. An example of a class diagram can be seen in Figure 4.

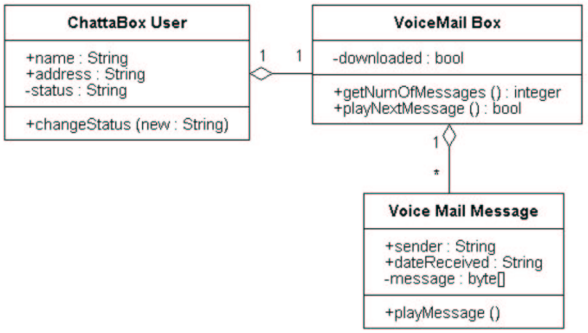

Fig. 4. ChattaBox User, VoiceMail Box and Voice Mail Message classes

2) Dynamic Design: The final phase of the design process dealt with the dynamic behavior of the system. UML interaction and statechart diagrams were produced to capture this aspect of the design. Interaction diagrams are models that describe how groups of objects collaborate in some behaviour.

A particular type of interaction diagram, a sequence diagram, was used to capture the behaviour of individual use cases. Sequence diagrams describe messages that are passed between objects, allowing the designer to gain an understanding of the overall flow of control. They are especially valuable for modelling concurrent processes. An example sequence diagram is shown in Figure 5.

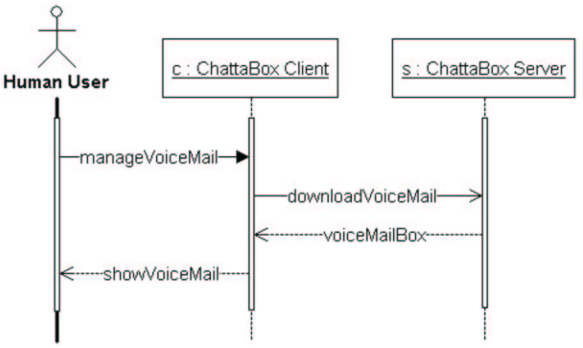

Fig. 5. Sequence diagram for voice mail download

Statechart diagrams describe all the possible states that a particular object can be in and how that state can change as a result of events reaching the object. Statechart diagrams were found to be useful for describing object behaviour across several use cases. An example of a statechart diagram is given in Figure 6. They are not particularly useful for describing behaviour that involves a number of collaborating objects. The combination of state and sequence diagrams however, allows a complete overview of the dynamic aspects of the design.

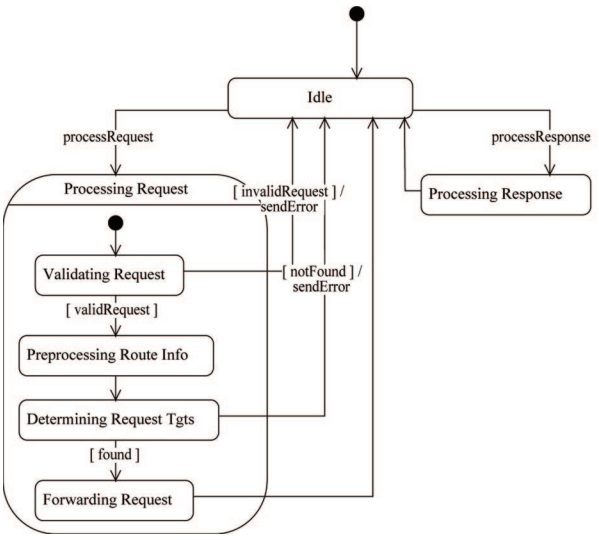

Fig. 6. Statechart for processing of SIP messages

The primary aim of the case study was to prove the correctness of the SIP protocol implementation, so this aspect of the system was concentrated upon during the design phase.

C. Converting UML to SDL

The resultant complete design of the SIP component of ChattaBox, consisted of two sets of class diagrams and statecharts, divided according to the client/server model. Each set

of diagrams was converted separately to form a separate SDL system, which interacted with its environment.

The conversion was not a straightforward process, as a particular syntax had to be followed in UML for this step. The initial class diagrams and statecharts, therefore, had to be revised a number of times before they were suitable to initiate a conversion. For instance, to facilitate for correct translation of UML classes to SDL constructs, use of *stereotypes* was required.

The client and server parts of the system design were converted separately. It was discovered upon conversion, that certain parts of the original UML design were not converted at all. Therefore the SDL diagrams had to be augmented to retain their meaning and at the same time to be syntactically and semantically correct in SDL. As a result, a detailed knowledge of SDL was required and the process of refining the SDL diagrams was extremely time consuming.

D. Verification of the Design

After the SDL description of the SIP portion of ChattaBox was deemed to be complete and equivalent of the UML design, verification was conducted. This was done using the simulator and validator tool provided by Telelogic SDL and TTCN Suite 4.3.

The simulator allowed the team to send signals of choice to the system and to monitor the resultant behaviour. The reaction and initiated actions of the two systems were observed for each of the signals sent from the environment. This showed where design was lacking and could be improved. In addition, the validator tool was used to do an exhaustive state exploration of each set of SDL diagrams. The results showed that the systems was free from deadlock.

Once the SDL design was correctly verified, the implementation of ChattaBox began. For the implementation, both the UML and SDL designs were used as a reference model to ensure that it reflected the outcomes of the design process accurately. Implementation details are beyond the scope of this paper.

E. Testing

In order to establish whether the concurrent communicating and distributed system implemented was correct, testing was performed. Two main measures of correctness were established. The first tested whether all the components of the system were performing their functions appropriately. The second tested whether the end user was satisfied with the completed ChattaBox product. Both are described below.

1) Component Testing: Individual components were tested separately during implementation. Testing involved a combination of black-box and glass-box testing. The following results were derived from the tests:

- SIP Correctness: SIP calls were able to be connected across multiple domains and multiple proxy servers successfully. The SIP calling sequence adhered to the SIP RFC standard [7]. SIP calls could correctly handle errors, as well as graceful terminations (call teardowns).
- RTP Correctness: RTP connections were correctly handled, and sessions were cleanly closed. Voice data was audible, therefore successfully transmitted.
- Distributed Architecture Integrity: The distributed architecture balanced loads effectively. Servers would not run unless valid certificates were found, and profiles and voice mail could be successfully managed.

2) User Testing: The system was tested on a group of ten individuals ranging in age from 18 to 49. Tests were carried out in groups of two, with the chosen two, communicating with each other using ChattaBox. The users were required to perform tasks using the system. They were then asked to judge the system via a questionnaire, which had been created beforehand. Having analysed the data sampled from the user tests, the following conclusions were made regarding the ChattaBox system:

- The quality of sound was very good, although a noticeable lag was present.
- The system was responsive to actions such as initiating calls, profile changes, and voice mail management.
- The user interface was effective and fairly easy to use.
- The system was not unpredictable and behaved in the manner in which a user intended it to behave.

Overall, ChattaBox succeeded in meeting its requirements. It proved to be a stable, correct system based on the results of the component and user testing.

V. Conclusions

The following conclusions were reached at the end of the case study.

A. Software Engineering with UML and SDL

The chosen path using UML for the requirements analysis and the initial design and SDL for the verification and testing of the design provided insight into software engineering as a whole. The experience gained during the case study confirmed that a structured software development process is of great importance when engineering a complex system.

Furthermore, the value of using UML and SDL for engineering a CCS was assessed. UML was very beneficial in that it allowed the group to view the system being developed at a very high level and quite quickly. The conversion process however, needs improvements if this path is to be feasible and efficient.

The worth of using a software engineering path that proves the correctness of a system before implementation has been noted. Developing a large communicating and distributed system proved to be very difficult especially when integrating the individual components into the final system.

After performing exhaustive testing sessions needed to ensure that the system was indeed running smoothly, it is evident that any way to cut down on this part of the SDLC would be useful. Furthermore, the knowledge that the design logic of a system is correct saves a great deal of time during the implementation phase. With correct design, the errors during implementation are less likely to be fundamental logical errors which often take a long time to solve.

B. Suitability of the Object Oriented Design Paradigm

The design using UML was entirely object oriented. It was discovered that many concepts employed in the final implementation were not suited to being modelled using an object oriented paradigm. These included the concepts of events, remote objects and threads. Implementation of most distributed systems requires the use of such concepts. It is therefore suggested that extensions to UML that allow for better modelling of these concepts are needed.

Moreover, without an in depth knowledge of the technologies one is using, a complete detailed design is not always possible. For instance, many of the constructs used in the final implementation were not known about at the design stage. Additionally, modelling without this knowledge meant that the design had to remain at a fairly high level.

C. Usefulness of CASE Tools

The place of CASE tools in the software engineering process is now well understood by the authors. Diagrams are essential for the communication of ideas between team members and can be used to clarify concepts that are not well understood. The use of the Telelogic and Microsoft CASE tools greatly enhanced the Software Development Life Cycle (SDLC).

To conclude, it has been a long journey from the initial stages of the ChattaBox case study until the actual realisation of the system. Yet, much insight into the software engineering of a complex system and team work has been gained.

VI. Future Work

Several suggestions for future work and alterations to the software engineering path followed in this case study are provided below.

The process of converting UML design to SDL should be made more efficient. Efficiency would be gained if no augmentation to the translated SDL diagrams was required after conversion.

Further investigation should be undertaken into developing a tool that allows reverse engineering between UML and SDL. With such a tool, developers would be ensured that one representation of the system is manipulated at all times. Consequently, design could be tested for correctness in SDL, while still being documented with the more user friendly UML.

Another alternative would be to undertake a similar case study using UML 2.0 or SDL 2000. Such a case study would help to establish whether the extensions to these methodologies allow for the use of only one technique throughout the entire software development life cycle, i.e. from requirements and design through verification and finally to implementation, testing and documentation. Should either of these languages offer a complete path for engineering CCSs correctly, they would be a valuable tool for future software engineers.

References

[1] H. Schulzrinne and J. Rosenberg, "The IETF internet telephony architecture and protocols," *IEEE Network*, vol. 13, pp. 18 – 23, May/June 1999.

[2] I. J. Grady Booch, James Rumbaugh, *The Unified Modelling Language User Guide*. Addison-Wesley, 1999.

[3] D. H. Ferenc Belina, "The CCITT-Specification and Description Language SDL," *Computer Networks and ISDN Systems*, vol. 16, pp. 311–341, 1989.

[4] R. D. Evgeni Dimitrov, Andreas Schmietendorf, "UML-Based Performance Engineering Possibilities and Techniques," *IEEE Software*, vol. 19, pp. 74–83, January/February 2002.

[5] Telelogic, "Telelogic Web Site," http://www.telelogic.com.

[6] H. Schulzrinne and J. Rosenberg, "The Session Initiation Protocol (SIP): Internet-centric signaling," *IEEE Communications Magazine*, vol. 38, pp. 134 – 141, October 2000.

[7] M. H. H. S. E. S. J. Rosenberg, "RFC2543: Session Initiation Protocol," http://www.faqs.org/rfcs/rfc2543.html.

[8] M. Handley and V. Jacobson, "RFC2327: Session Description Protocol," http://www.faqs.org/rfcs/rfc2327.html.

[9] S. C. H. S. R. F. V. Jacobson, "RFC1889: Real-time Transport Protocol," http://www.faqs.org/rfcs/rfc1889.html.

[10] E. Holz, "Application of UML in the SDL Design Process," http://citeseer.nj.nec.com/268645.html, June 1998.

[11] M. Bjorkander, "Graphical Programming Using UML and SDL," http://www.telelogic.com/download/paper/graphicalprogramming.

[12] B. S. J. Rumbaugh, "Mapping SDL to UML," http://www.rational.com/products/rosert/prodinfo/reading/sdl2umlv13.pdf.

[13] W. Stallings, *Network Security Essentials: Applications and Standards*. Prentice, 2000.

P S Kritzinger P Kritzinger obtained his PhD from Waterloo University, Canada, in 1972 where he became Assistant Professor for 2 years. Thereafter, he taught at the University of London before he returned to take a senior lecturer position at Stellenbosch University. He joined University of Cape Town in 1985 as a full professor. Pieter is the founder of the Data Network Architectures (DNA) Group.

M Chetty M Chetty obtained her BSc and BSc(Hons)from UCT. She is currently doing her MSc with the Collaborative Visual Computing Laboratory at UCT. Her project is investigating how Voice over IP may be used for development purposes in rural South Africa.

J Landman J Landman obtained his BSc and BSc(Hons) from the UCT. He is currently doing his MSc with the DNA laboratory at UCT. His project focuses on Markov models for fading channels on UMTS air interfaces.

M Marconi M Marconi obtained his BSc and BSc(Hons)from the UCT. He is currently working for a software company in the United States.

O Ryndina O Ryndina obtained her Business Science degree from the UCT. She is currently pursuing an MSc with the DNA laboratory at UCT. Her project will investigate methodologies for Requirements Engineering.