The Evolution of a C2 Protocol Gateway

Arno Duvenhage
Luther Terblanche
Council for Scientific and Industrial Research
Meiring Naude Rd
Pretoria
South Africa
aduvenhage@csir.co.za, lterblanche@csir.co.za

Keywords:
C2, Gateway, Bridge, Middleware, Interoperability, Collaboration, Tactical Awareness

ABSTRACT: *Live, Virtual and Constructive (LVC) simulations can share tactical awareness data using command and control (C2) protocol gateways as bridges: Multiple training simulations can be linked to create large scale collaborative environments; simulations can be connected to real-time data sources such as local warning radars, air traffic control centres, flight simulators, etc.; simulators can collaborate with other simulators and systems; simulation states and events can be displayed using 3D viewers and other displays. This paper presents the ongoing development of a C2 gateway concept that has evolved over a period of more than three years. The functional and non-functional requirements that were discovered, software architecture and design, time management and synchronization, the lessons learned, as well as possible future developments are discussed.*

1 Introduction

Different simulations have unique requirements. The data and spatial models, temporal references, and time stepping algorithms may differ. A gateway or simulation bridge has to map between two or more information models with different formats in order to transfer data from one system or application to another. It also has to provide time management and synchronization between different systems.

Sharing *Tactical awareness* (TA) information includes sharing [1]:

- The position and movement of own forces,
- the tracking of the position and movement of contacts detected by own sensors, and
- the classification of both friendly and hostile contacts being tracked by own sensors.

This paper presents the ongoing development of a C2 gateway concept that has evolved over a period of more than three years, called the *XMLGateway*. The *XMLGateway* represents all data internally using the Extensible Markup Language (XML) and can connect to different systems using the appropriate link standard or protocol. Various international, open and proprietary link standards are currently supported.

The *XMLGateway* has a well defined object-oriented architecture that allows easy addition of new link standard components. Logging and playback of the XML data and the raw link data are inherently supported by the gateway architecture. The *XMLGateway* application and protocol components are written in C++.

The gateway Graphical User Interface (GUI) was created using Microsoft Foundation Classes (MFC), but the link components are reusable and in general don't depend on a Microsoft specific environment. Currently the GUI is limited to only two simultaneous instances of link components and therefore the gateway operates as a C2 protocol bridge.

2 Background

Over the past three years a group within the Council for Scientific and Industrial Research (CSIR), South Africa, called Mathematical and Computational Modelling (MCM), has been increasing the interoperability of their simulation base. This makes collaboration with a wider range of simulators and external systems possible.

MCM frequently attend field exercises in support of the South African National Defence Force (SANDF). The type of support has ranged from running simulated batteries, and performing live aircraft engagement with the simulation base, to performing a key C2 system integration role during antiterrorism exercises. This requires the use of a gateway to import and export TA data to and from the simulation base, as well as to translate TA data from one C2 protocol to another, enabling two or more external systems to collaborate with each other or with the simulation base.

2.1 Common Communications Link Structure

Most link standards can be represented using the 7 layer Open Systems Interconnection (OSI) model [2]. The rest of this article will refer to a communications link standard as either a protocol or the link. Figure 1 shows how this relates to the OSI model. The link represents the bottom four layers of the OSI model and includes the TCP/IP and UDP protocols, as well as file access.

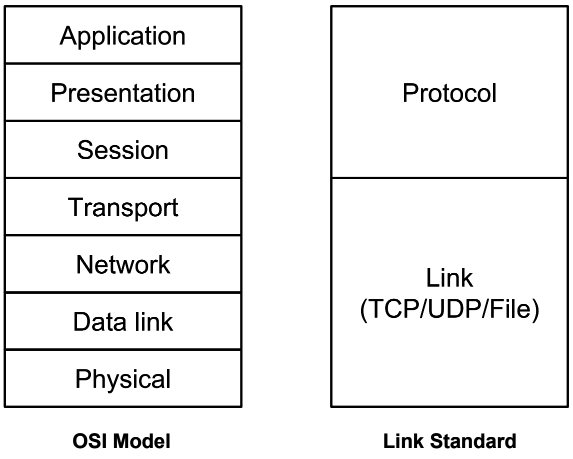

Figure 1: A graphical representation of the relationship between the 7 layer OSI and communications link standard layering

All the protocols discussed in the next section run on TCP/IP or UDP links. A file access link is used for logging and playback. The file access link has the same interface as the TCP/IP or UDP links allowing inherent support for logging and playback of the raw protocol data.

From a software architecture perspective, each link has a corresponding *link component* that is responsible for opening and closing the connection, as well as reading and writing data and managing the connection. The protocol has a *link decoder component* that translates the raw protocol data to and from XML. The link component and link decoder component, together with some time management and helper components, form the *protocol component*.

2.2 Interoperability Developments

The interoperability developments timeline is shown in Figure 2. These developments were all eventually incorporated into the *XMLGateway* and will be discussed in this section. The various link decoder components were either implemented directly form the protocol specifications or reverse engineered from code excerpts from the relevant system if a specification was not available.

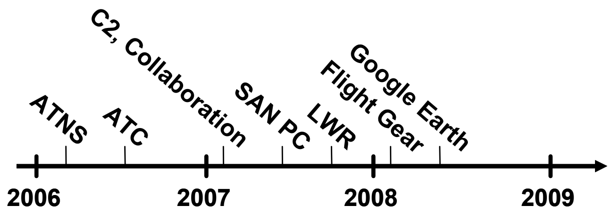

Figure 2: Interoperability development timeline

Air Traffic and Navigations Services

The first system that was integrated with is an Air Picture Display System (APDS), developed by Saab Systems South Africa, which gets its data from the South African Air Traffic and Navigation Services (ATNS) as well as some medium range radar systems. The display system exports track information using the concepts, but not the specific format, of the **A**ll Purpose **ST**ructured **E**urocontrol su**R**veillance **I**nformation e**X**change (ASTERIX) protocol. ASTERIX is a complex binary application/presentation protocol responsible for data definition and assembly. It was developed to support surveillance data transmission and exchanges between Air Traffic Control (ATC) systems [3]. The link decoder implementation was done from the APDS link specification [4]. The track information is decoded and used to give the simulation base access to a live air picture.

Reutech Radar Systems ATC

The second system that was integrated with is another air picture display system that gets its data from a specific air traffic control (ATC) centre via a dialup link. The display system exports track information using a proprietary binary protocol from Reutech Radar Systems (RRS). The protocol transfers track information using a simple structure and the link decoder implementation had to be reverse engineered from a code excerpt from the display system. The track information is used to give the simulation base access to a live air picture.

LinkZA

LinkZA is a tactical data link standard that facilitates interoperability between various South African military systems and simulators in support of C2 [1]. The standard is currently used by the CSIR for interoperability of the simulation base with air picture display systems, air defence and fire control systems and C2 simulators. This makes various levels of collaboration with C2 systems and simulators possible. LinkZA facilitates the sharing of tactical awareness data in a C2 environment, as well as the sharing of Air Raid Warning States (ARWS), Air Defence Readiness States (ADRS), Text Messages (Free Text), images and other control messages [1]. The link decoder implementation has an ASCII interface and uses components from Saab Systems South Africa to do the LinkZA ASCII to binary conversions.

SA Navy Patrol Corvettes

The Institute of Maritime Technology (IMT) of South Africa developed a surface plot display system that is currently deployed as part of the onboard data capture suit on SA Navy Patrol Corvettes (SAN PC) [5]. The display system provides access to primary and secondary search radar tracks, tracking radar tracks and Automatic Identification System (AIS) tracks through a proprietary binary protocol. It is a complex protocol with several message structures defined and the link decoder component was implemented from the specification [5]. The track information is used to give the simulation base access to surface plot (AIS) data. AIS tracks help identify and locate sea vessels.

Local Warning Radar

The signal processor board of a Local Warning Radar (LWR), built by Reutech Radar Systems (RRS), exports track and plot information. A connection can be made directly to the signal processor board using a proprietary binary protocol from RRS to read track and plot information [6]. The protocol transfers the track information using a simple structure and the link decoder implementation was done from the specification [6]. Track information is also exported from the control systems onboard the radar using the LinkZA standard.

Flight Gear

There is a requirement to be able to inject live aircraft into the simulation base. For this purpose an open source flight simulation package, *Flight Gear,* was configured to report the current aircraft state using a proprietary binary protocol [7]. The protocol transfers track information using the simple Flight Dynamics Model (FDM) structure and the link decoder implementation had to be reverse engineered from the *FlightGear* source code. The aircraft state can then be injected, via the *XMLGateway*, into the simulation base or any other system connected to the gateway, as a live track.

Google Earth

Google Earth (GE) has an extensive terrain map database and can be used to visualise tactical awareness information. The *XMLGateway* can interface with the Google Earth Viewer application using the Pitch *Google Earth Adaptor* [8]. The Google Earth Adapter is a High Level Architecture (HLA) [9], [10] compliant Keyhole Markup Language (KML) [11] web-service. KML allows features such as place marks, images, polygons, 3D models and textual descriptions to be sent directly to GE to be visualised. GE can then request a current view from the Google Earth Adaptor federate using KML. The gateway acts as another federate and the protocol component was implemented from example code provided by Pitch that uses the Pitch Portable Run Time Infrastructure (pRTI1516) middleware [12], [13]. The Real-time Platform Reference Federate Object Model (SISO RPR FOM v2.0D17) is used as the data standard for sharing object and interaction data with the RTI. The gateway protocol component builds up a situational awareness view based on the information received from the gateway and makes that available to the Google Earth Adaptor through the RTI.

Another way the *XMLGateway* interfaces with the Google Earth Viewer application is directly through the KML protocol. The protocol component consists of a XML server component and a KML web-service that connects to the XML server using a TCP/IP link. The XML server builds up a situational awareness view based on the information received from the gateway. GE can then requests KML updates and the required graphical content from the web-service which in turn request the current view from the XML server.

2.3 Native Code, Gateway Or Middleware

There are three ways in which a new communications link standard can be added to an application. One way is to integrate the new link standard into the application's code base (using native code) [14]. This involves creating a class or component and re-compiling the application with the new component. This approach requires access to the source code and development tools of the application which isn't always feasible.

Another way is to build the application on a reusable middleware layer that supports adding new communications links [14]. This can also be called an internal gateway. This allows the application to support new link standards without having to modify the applications source code or relying on additional applications to do the conversions. This approach will not work for applications that don't use the middleware.

The third way is to use a separate application, called a gateway, which converts the application's existing communications link standard to the new link standard [14]. The *XMLGateway* uses this approach since a gateway can run independently, making it reusable and applicable to a wide range of existing systems. A gateway can be run on a different machine than the simulation if CPU utilization becomes too high. Multiple gateways communicating with the same or different link standards can also run simultaneously.

2.4 Services That A Gateway Can Provide

According to the literature a gateway can provide useful services besides converting information from one protocol to another. It can also have added functionality to help integrate systems and simulations that would otherwise be incompatible with each other.

Different simulations use different time stepping techniques. A gateway can, for example, translate model state data from a discrete time based system to a discrete event based system by performing state quantization and/or quantized state integration. Different systems also run at different rates and data smoothing and sampling can be implemented in the gateway. This could include implementing algorithms like Dead Reckoning in the gateway [14].

Coordinate conversion is required if the connected systems or applications use different coordinate frameworks [14]. A good example of this would be the conversion of model state updates from a simulation environment that models the world using a *flat earth* coordinate system to a simulation environment using a *spherical earth* coordinate system.

A gateway can filter data to limit the amount or type of information that is sent to a specific system or application [14]. Tactical Awareness data can be filtered based on location, speed, altitude, track classification, etc. The gateway can also conserve bandwidth, if required, by limiting the rate at which data is sent through.

Time stamping of data can be done if an accurate time stamp is required but not available from the data source [14]. Model state updates normally require accurate timestamps to ensure that track correlation, prediction, etc. is done accurately enough.

A gateway can perform entity storage to allow for partial updates from data sources [14]. A system might for example send classification and hostility information less frequently than position updates, but the gateway might be required to output complete information on each update. In order to do this the gateway might have to maintain a list of all possible objects and update the objects as the relevant information is received. The more complete information from the list may then be used when sending out updates.

A gateway can guard against common problems in distributed systems such as connections terminating abruptly [14]. If one connection goes down illegally the gateway will close the other connection(s) gracefully, following the relevant link procedures.

A gateway can provide connectivity to cheaper, commercially available systems, outside the standard military environment, like 3D game engines [15]. This can provide rapid and low-cost solutions for visualization if the game engine has an interface that allows data to be imported from the gateway and/or the view to be controlled.

3 Lessons Learned

This section will discuss the advantages of specific features that were required and subsequently added to the *XMLGateway* during the ongoing development, as well as during various field exercises conducted by the SANDF. The gateway functionality has evolved over time and compares well to expected gateway requirements from the literature as discussed in the previous section. There are some unique requirements though.

Different link standards might use different data models for representing TA information. The Classification codes, hostility codes, symbol codes, coordinate frameworks, etc. might have to be translated when information is converted from one protocol to another. A real problem with this is that classification codes, for example, from the source protocol does not necessarily have a one-to-one mapping with classification codes from the destination protocol. The gateway's internal XML based data model is also not complete enough to represent information from all the different protocols perfectly.

The gateway stores every object that flows through the gateway in a specific *incoming* or *outgoing* list that displays the XML structure of the objects for debugging purposes. This was required to trace the data flow among different systems. It can also help find protocol errors when integrating with new systems or when using new protocols. The information can additionally be used to verify whether a system is sending information and at what rate.

The gateway can generate a preconfigured set of test messages in order to test data flow among the systems or stimulate the external systems for debugging purposes. This was required to test the systems and links when it is unclear whether or not a system is sending data when data flow among the systems has to be tested.

The raw protocol data is logged by the link component while the protocol component is connected. This makes all the data received, from any system, available off-line. This was required for playback of the protocol data or debugging and testing of the relevant protocol component.

The gateway can filter data, but it should translate and send out objects when they are received. This allows the TA events and states to flow among the systems with the correct timing. Any change in timing or update rates of TA information can introduce unexpected behaviour in the connected systems, which make debugging of potential problems difficult. Some protocol components can report partial updates and all the objects have to be stored in an object list that gets updated as information is received. The updates are then sent through using the more complete object information from the list. A simple spatial filter was also required to reduce the amount of objects received when only a specific area of interest in activity applies.

The gateway can receive data faster than real-time or slower than real-time, depending on the type of link and the data source: The gateway normally reads logged raw protocol data much faster than real-time; a simulation could also start running slower than real-time if it requires too much resources. Figure 5 shows the *dual timer* concept that is used to synchronise links that run at different speeds. The gateway queues the objects received from a protocol component until the other side is ready to receive the information. The time management and synchronisation is explained in more detail in the next section. The queuing is also required when reading logged raw protocol data from a file access link since synchronisation can only be done once an object is decoded.

Normally the gateway protocol components continuously read data from a link until all pending data has been read. This allows the protocol component to read all the data being received without falling behind. However the gateway has to *throttle* the reading to prevent the gateway from becoming unresponsive if too much data is available. This is done by only allowing a preset amount of objects to be read consecutively.

Logged raw protocol data is read using a file access link with all the data available once the file is opened. The file access link component also has to *throttle* the reading to prevent the protocol component from reading and decoding the whole file before relinquishing to the gateway. This is required not only to prevent the gateway from becoming unresponsive, but also to ensure that the protocol component time managers (Figure 3) estimate the time correctly (this will be explained in more detail in the next section).

The gateway runs in a real-world environment and the link and protocol components are required to be very robust. Connections dropped without the appropriate teardown procedures being followed should not *break* the gateway. Links could be broken illegally when the physical media is disconnected or the remote systems crash or exit abnormally. The gateway has to be able to recover from connection problems.

4 Software Architecture And Design

This section will discuss the architecture of the *XMLGateway*. The gateway benefits from both an object-oriented architecture and a layered architecture as described in [16]:

In the style based on data abstraction and object-oriented organization, data representations and their associated primitive operations are encapsulated in an abstract data type or object.

A layered system is organized hierarchically, each layer providing service to the layer above it and serving as a client to the layer below. In some layered systems inner layers are hidden from all except the adjacent outer layer.

The advantages of data abstraction and object-oriented organization are [16]:

- It is possible to change the implementation of a component without affecting the interface to it.
- Problems can be decomposed into collections of interacting agents.

Figure 3 shows the architecture of the *XMLGateway*. The object-oriented architecture allows the protocol component and the gateway to be decomposed into functional parts. The part implementations can easily be changed or updated without affecting the rest of the architecture. This allows for easy addition of new protocol components and link types.

The advantages of layered systems are [16]:

- Support design based on increasing levels of abstraction.
- Changes to the function of one layer affect at most the two adjacent layers.
- Supports reuse. Different implementations of the same layer can be used interchangeably.

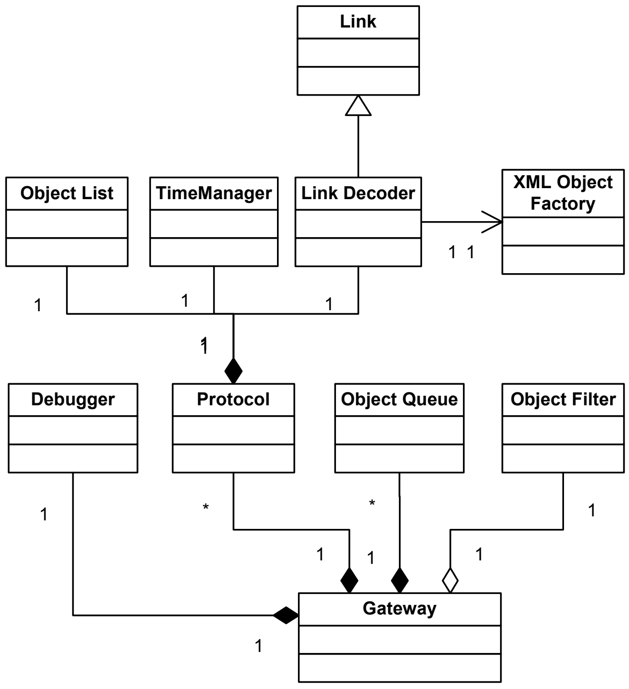

Figure 3: Object Oriented XMLGateway architecture

Figure 4 shows the layered protocol component architecture of the gateway. The layered architecture allows the different parts, like the link decoder component for example, to be changed with a decoder component that supports a different protocol. The *Link* layer (refer to Figure 1) manages the transport level connections. The *XML Object Factory* converts the protocol data to and from XML objects. The *Object Filter* discards any objects received that the gateway doesn't have an interest in. The *Object Queue* temporarily stores the received XML objects for the gateway to be able to synchronise the sending of the incoming updates based on object timestamps.

Figure 5 shows how objects are received from one protocol component, filtered, queued and then sent out to the protocol component on the other side. The layered architecture of the protocol component allows the gateway to process real-time data from external sources in exactly the same way as data being read from logs: All the protocol components present the same interface to the gateway and the interface supports synchronisation through the object queue.

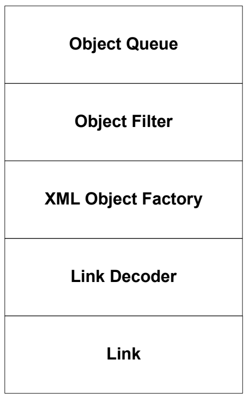

Figure 4: Layered Protocol Component Architecture

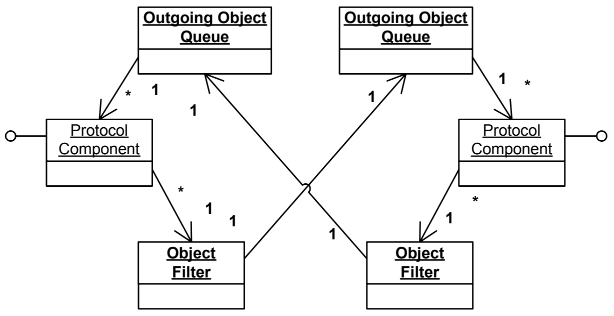

Figure 5: Data flow within XMLGateway

Each protocol component has a time manager component as shown in Figure 3. The time manager is responsible for estimating the current time of the external system based on the timestamps contained in the incoming information. It uses the actual time difference between consecutive objects or groups of objects with timestamps to calculate the speed the time manager should run at. This is also why *throttling* data being read from a log is important.

The time manager remembers the difference between the first timestamp received and Greenwich Mean Time (GMT) when the relevant object was received. The time manager can then calculate the current estimated GMT time of the external system by using that time difference. The estimated GMT time might actually progress slower or faster than real-time depending on the speed the external system is running at. The time manager has an option of running at GMT if the protocol component doesn't receive timestamped data from the external system.

Incoming objects are tagged with the estimated GMT time from the relevant time manger as they are received and stored in the protocol component's object queue. Any timestamps are also adjusted with the same time difference to agree with the estimated GMT time. The gateway takes objects from the object queue and sends it over to the protocol component on the other side (see Figure 5) only if the estimated GMT of the other side is larger than, or equal to the estimated GMT time of the object.

The protocol components use their time managers to convert any timestamps of objects sent out again, back from estimated GMT time to the estimated time of the relevant external system. This synchronisation is called the *dual timer* approach since two time managers are required for the gateway to properly synchronise the incoming and outgoing data.

5 Conclusion

The *XMLGateway* is now a proven concept within the South African C2 environment with various successful field trials under its belt. The architecture and design is constantly being refined to accommodate new requirements and protocols as the application of the concept grows.

The *XMLGateway* runs independently and can facilitate interoperability and collaboration of a large array of existing systems and simulators within the South African C2 environment. The gateway's novel time management approach can accommodate data sources that run faster than real time, as well as data destinations that run slower than the data sources.

6 Future Work

This section will discuss possible future work on the *XMLGateway* based on additional requirements and the lessons learned. New requirements will also be introduced while preparing for or attending future SANDF field exercises.

The protocol component interfaces should be abstracted to a level where the entire protocol component, including the link component and protocol settings management, can be incorporated into a Dynamically Linked Library (DLL). This will make the protocol components reusable and also make the development of new protocol components for the *XMLGateway* easier in the sense that the development environment and framework will be much more controlled and mature. The gateway will also be able to use multiple protocol components easily by just loading the relevant DLLs.

A generic link layer that can handle all the link types transparently should be created. The gateway protocol components use a layered architecture which will ensure that all the protocol components also handle the link types transparently. Each protocol component will then be able to connect to a system using any of the supported connection types without extensive configuration or any development being required. Currently only a specific connection type is supported by each protocol component.

Advanced debugging and message generation features will enable the *XMLGateway* to find and debug, link and protocol problems more effectively. A good message generation system can also enable the *XMLGateway* to be set up to emulate defective or missing systems (filling functionality gaps).

Research on the usability of alternative C2 data models is currently underway to make the command and control infrastructure more effective and interoperable. The correct data model can also contribute to the *XMLGateway* being more compliant with international trends and standards.

The *XMLGateway* could also benefit from an integrated track manager and display system. This will add to the debugging and *gap filling* services that can be provided. Reporting the time difference between received object timestamps and the current time will also help identify systems that aren't synchronized with the current time.

7 Acknowledgements

The authors would like to thank the members of the Mathematical and Computational Modelling (MCM) research group within the CSIR, South Africa for their ongoing support and effort in the development of the *XMLGateway* concept. MCM is part the Defence Peace Safety and Security (DPSS) operating unit of the CSIR and the authors would like to additionally acknowledge the support of the whole operating unit as well as several industry partners, including Saab Systems South Africa.

8 References

[1] "Combat Net Interoperability Standard (DODI/CMI/00009/2001), Section B" Classified Restricted, SANDF, Feb. 2006.

[2] B.A. Forouzan, TCP/IP Protocol Suite, Chapter 2, McGraw-Hill, 2003.

[3] “Eurocontrol Standard Document for Surveillance Data Exchange, Part 1, ASTERIX” SUR.ET1.ST05.2000-STD-01-01, Nov. 2001.
[4] “Air Picture and Display System, Interface Specification – Track Message Output” Classified Restricted, Saab, APIR-012, June 2006.
[5] “IMT Mobile Surface Plot Data Distribution Format (IMT0111/3)” Classified Restricted, Institute of Maritime Technology, May 2007.
[6] “ESR220 Processor Unit Data Interface, Signal Interface Control Document” Classified Restricted, Reutech Radar Systems, May 2006.
[7] FlightGear 1.0.0 Source Code (\src\Network\net_fdm.hxx), http://www.flightgear.org/Downloads/source.shtm
[8] Pitch GE Adapter, Pitch, http://www.pitch.se/products/ge-adapter/pitch-ge-adapter.html
[9] "IEEE standard for modeling and simulation (M&S) high level architecture (HLA) - framework and rules" *IEEE Std 1516-2000*, pp.i-22, Sep. 2000.
[10] F. Huhl, R. Weatherly and J. Dahman, Creating Computer Simulation Systems, An Introduction to the High Level Architecture, Prentice Hall, 1999.
[11] Google KML Documentation, http://code.google.com/apis/kml/documentation/
[12] Pitch pRTI-1516 documentation, Pitch, http://www.pitch.se/products/pitch-prti/pitch-prti-overview.html
[13] SISO-STD-004.1-2004 - Dynamic Link Compatible HLA API Standard for the HLA Interface Specification (IEEE 1516.1 Version).
[14] D.J. Paterson, E.S. Hougland, J.J. Sanmiguel: “A Gateway/Middleware HLA implementation and the extra Services that can be provided to the Simulation” Fall Simulation Interoperability Workshop, Sep. 2000.
[15] P. Prasithsangaree, J.M. Manojlovich, J. Chen, M. Lewis: "UTSAF: a simulation bridge between OneSAF and the Unreal game engine" IEEE International Conference on Systems, Man and Cybernetics, vol.2, pp. 1333-1338, 5-8 Oct. 2003.
[16] M. Shaw and D. Garlan, Software Architecture, Perspectives on an Emerging Discipline, Prentice Hall, 2000.

ARNO DUVENHAGE is a Researcher for the Council for Scientific and Industrial Research (CSIR), South Africa. He joined the CSIR’s Mathematical and Computational Modelling Research Group in January 2005 as a Software Engineer. Arno’s current work involves modelling and simulation for acquisition decision support, focusing on air defence, specializing in distributed and networked systems and system interoperability. Arno holds a BEng Degree in Computer Engineering from the University of Pretoria, South Africa. He is currently specializing in software engineering.

LUTHER TERBLANCE is a researcher at the Council for Scientific and Industrial Research (CSIR), South Africa. He joined the CSIR’s Mathematical and Computational Modelling Research Group in August 2007 as a Computer Scientist. Luther holds an MSc Eng degree in Applied Mathematics from the University of Stellenbosch, South Africa. His current focus is on modelling and simulation of defence systems.