Migrating to a Real-Time Distributed Parallel Simulator Architecture

An Update [*]

Bernardt Duvenhage
The Computer Science Department of the University of Pretoria
Pretoria, South Africa
bduvenhage@csir.co.za

Categories and Subject Descriptors
J.7 [**Computers in Other Systems**]: *Command and Control*; J.7 [**Computers in Other Systems**]: *Military*

Keywords
discrete time, DTSS, discrete event, DEVS, distributed parallel simulation

ABSTRACT

A legacy non-distributed logical time simulator was previously migrated to a distributed architecture to parallelise execution. The existing Discrete Time System Specification (DTSS) modelling formalism was retained to simplify the reuse of existing models. This decision, however means that the high simulation frame rate of 100Hz used in the legacy system has to be retained in the distributed one—a known difficulty for existing distribution technologies due to interprocess communication latency.

The specialised discrete time distributed peer-to-peer message passing architecture that resulted to support the parallelised simulator requirements is analysed and the questions surrounding its performance and flexibility answered. The architecture is shown to be a suitable and cost effective distributed simulator architecture for supporting a four to five times parallelised implementation of a 100 Hz logical time DTSS modelling formalism.

From the analysis results it is however clear that the discrete time architecture poses a significant technical challenge in supporting large scale distributed parallel simulations. This is mainly due to sequential communication components within the discrete time architecture and system specification that cannot be parallelised. A hybrid DTSS/Discrete Event System Specification (DEVS) modelling formalism and simulator is proposed to lower the communication and synchronisation overhead between models and improve on the scalability of the discrete time simulator while still economically reusing the existing models.

The proposed hybrid architecture is discussed. Ideas on implementing and then analysing the new architecture to complete the author's masters dissertation are then touched upon.

[*]Progress of Masters dissertation following on original article[1] by the same name.

1. INTRODUCTION

The South-African National Defence Force's (SANDF's) need for decision support and concurrent tactical doctrine development within a Ground Based Air Defence System (GBADS) acquisition program offered an ideal opportunity to establish an indigenous and credible modelling and simulation capability within the South-African defence acquisition environment [2][3]. The broad requirement of the capability is to simulate a GBADS battery of existing and still to be acquired (possibly still under development) equipment and their related human operators at a system of systems level within a realistic Synthetic Environment (SE). A GBADS deployment, shown in Figure 1, usually consists of a layered air defence. The outer layer typically consists of eight very short range missile systems, each having a virtual operator and an accompanying buddy with a wide angle pair of binoculars. The second layer has four gun systems, each consisting of two guns, a tracking radar, a designation radar, a fire control system and at least three operators to operate the guns. The inner layer of defence usually has two short range missile systems, each consisting of a ground based launcher, a designation sensor, a fire control system and a couple of virtual operators. The deployment would defend some asset (vulnerable point) against an airborne threat scenario. A typical threat scenario would consist of one to many incoming attack aircraft which are the 'targets' to be engaged by the air defence system.

During the concept and definition phases of the acquisition life cycle [4] the capability was successfully provided by a non-distributed simulator and its architectural predecessors [5] developed by the CSIR. A selection of the models were derived from high fidelity engineering models, some by OEMs, and developed within a 100Hz logical Discrete Time System Specification (DTSS) [6] that simplified the time and causality management. The non-distributed simulator evolved within this 100Hz logical time DTSS modelling formalism

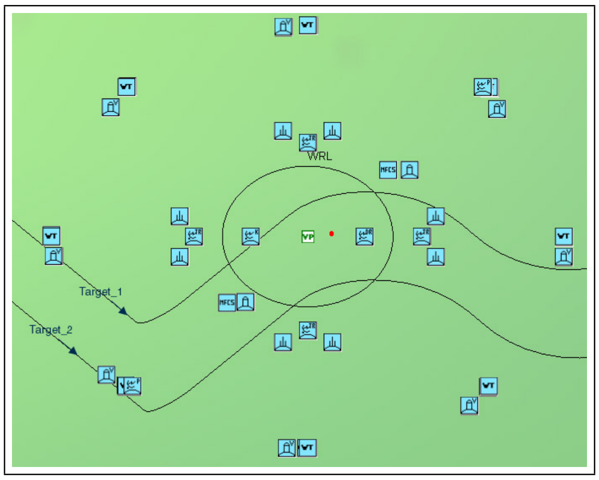

Figure 1: A Typical GBADS Deployment

and was implemented to run As Fast As Possible (AFAP).

Real-time simulation execution became a prioritised requirement during the development phase of the acquisition life cycle due to the realised impact of *realistic* human-simulation interaction when doing tactical doctrine development. Human interaction would happen through an Operator In the Loop (OIL) console with the possibility to record the operator's actions to be re-used in statistical simulation runs when and as required. To support the real-time requirement it was decided to parallelise the simulator across multiple Commercial Off the Shelf (COTS) PC nodes connected with Gigabit Ethernet. For economical reusability of all the existing models it was also decided to retain the 100Hz logical time and DTSS modelling formalism.

Several case studies of value for embedding a discrete time modelling approach within existing simulator distribution technologies, presented by Duvenhage and Kourie [1], showed that these reach a frame rate ceiling of 20 to 30Hz. A specialised 100Hz logical discrete time distributed simulator was developed to bridge the gap from 30Hz to 100Hz and this simulator is currently in use to provide the required simulation capability. To achieve real-time execution the logical time simulator's execution is throttled to not exceed real-time. Guaranteeing that at least real-time execution can be reached does place severe real-time constraints on the inter-node communication latency though, as each simulation frame is a mere 10ms.

The distributed discrete time simulator design is briefly discussed in the next section followed by some performance analysis results. The results are discussed and shown to indicate a technical difficulty in the future scalability of the discrete time simulator. A hybrid DTSS/Discrete Event System Specification (DEVS) modelling approach and simulator is then proposed to improve on the scalability of the discrete time simulator. Ideas on implementing and then analysing the new architecture to complete the author's Masters dissertation is finally presented.

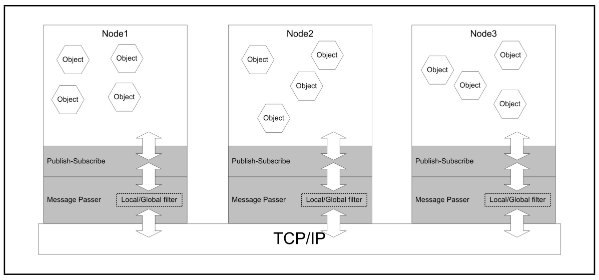

Figure 2: Layered Peer-to-Peer Simulator Architecture

2. THE DISTRIBUTED DISCRETE TIME SIMULATOR

The discussion on the new publish-subscribe simulator architecture is structured around the layered architecture of the simulator (shown in Figure 2), which includes a publish subscribe simulation layer, a message passing implementation of the simulation model and at the bottom layer a low latency TCP messaging protocol for Gigabit Ethernet.

The attraction of the layered architecture was the separation of concerns, in terms of design, between the simulation model and the distributed execution thereof. An additional advantage is of course the ability to change the implementation of the bottom layers without affecting the top layer simulation application.

2.1 Publish-Subscribe Simulation Model

The top layer simulation model encompasses a couple of aspects, which include the simulation time management, the system specification modelling formalism, the object communication framework and the synthetic environment services.

As mentioned, the pre-existing models have been implemented within a conservative logical time management scheme and a DTSS modelling formalism. It was decided to keep these aspects unchanged to simplify the reuse of the existing models. The object communication framework that is under investigation for the simulation model is a specialised publish-subscribe framework to be discussed next. Discussions on the synthetic environment services will then follow.

2.1.1 The Object Communication Framework

The publish-subscribe paradigm is well known to anyone that has ever needed to organise to get information, for example a magazine, on his or her topic of interest on a regular basis. Each magazine within your topic (category) of interest has a title and a regular interval at which the categorical information is made available (published). You, the subscriber, may request that the information be delivered to your doorstep in the form of, say, a weekly or a monthly magazine issue.

The publish-subscribe simulation framework is a direct analogy to the magazine example. An instance of a simulation model (an object) may express its desire to receive information within a certain category of interest, e.g. aircraft positions, by adding the category (and title name, if known) to its *Subscription Wish List*. An object may also express its willingness to share information within a certain category, such as its own position, by adding a title (name and category) to its *Owned Title List*.

At simulation run-time each object will go through regular increment, publish and gather cycles. Within the DTSS modelling formalism an object is incremented every n'th discrete time simulation frame where n is the object's *trigger frame*. Each wish list subscription, and thus each subscriber in a title's subscriber list, is also associated with a trigger frame. During a simulation frame, each subscriber of each owned title will be visited and an issue sent to the subscriber if it is the subscription's trigger frame.

An object has an issue pigeon hole for each of its wish list subscriptions. When an issue is received (gather phase) it is placed in the appropriate pigeon hole. A pigeon hole may have subscription history turned off or on. If history is off then a newer version of an issue replaces all old issues that may remain in the pigeon hole. If history is turned on then issues will be added to the pigeon hole in chronological order. The object may then read issues and manually delete them as required during increment cycles. Turning history on for a specific wish list subscription is typically required when a subscriber doesn't want to miss any important updates (events) for that subscription. Having history off allows the subscriber to always have access to the current issue without the overhead of always caching and processing a subscription's recent history.

2.1.2 *The Synthetic Environment Services*

The two types of simulation services supported are, firstly, low level services that are built into the simulation model and, secondly, high level services that run on top of the simulation model as simulation objects. The only low level service currently implemented is that of delayed issues. An issue may be given a future delivery time by either the publisher, or the subscriber upon delivery. Such an issue would be delivered to the subscriber immediately, but once there it resides in a delayed issue list until the time of delivery arrives at which point the issue is put into the appropriate pigeon hole of the subscriber. Delayed issues are handy if transmission delays of messages within the SE are to be modelled. In the current simulator the issue delays of tactical communication subscriptions are, when required, calculated by a radio and cable network model.

High level synthetic environment services subscribe to the objects' state titles and then apply environmental tools such as Line Of Sight (LOS) and terrain engines to give each object individual feedback on its height, which objects it can see, etc. To accomplish the personalised feedback a service advertises what is called a differentiated title. Each time a subscription is made to a differentiated title the simulator automatically creates a personalised title and subscription for the subscriber. The service may then use the created titles to publish to individual objects.

A service need not always publish data back to the simulation, though. Logging, for example, is a high level service that accumulates object states and other information. The logging service may then apply user configured data analysers to the accumulated data and log the results to disk.

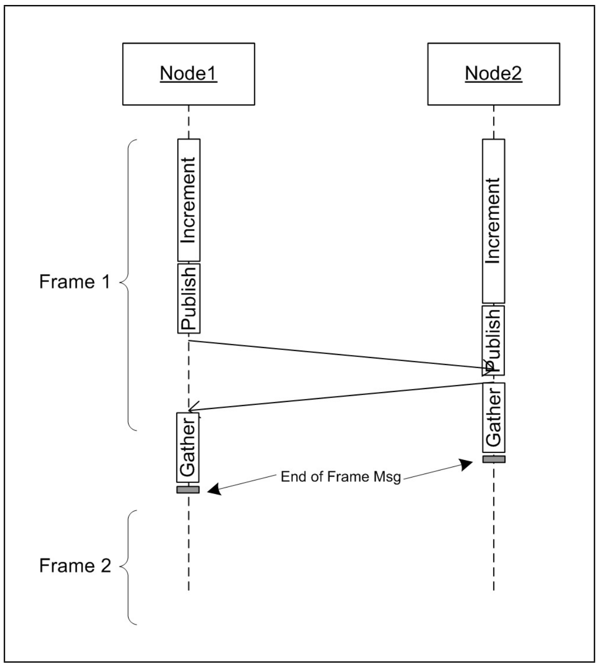

Figure 3: Peer-to-Peer Message Passing and Simulation Synchronisation

2.2 Peer-to-Peer Message Passing and Node Synchronisation

The publish-subscribe communication framework and the simulator synchronisation is implemented with a peer-to-peer message passing architecture. A peer-to-peer architecture is specifically preferred above a client-server architecture to avoid the double latency that exists when communicating via a server to a third machine. The messaging implementation of the publish-subscribe communication framework is presented, followed by the implementation of the simulation synchronisation.

2.2.1 *Messaging Implementation of Publish-Subscribe*

The publish-subscribe framework naturally translates to a messaging architecture containing only three message types. A title may be advertised as a title message containing all the title and publisher details. A wish list subscription may similarly be a message containing the details of the wish list subscription and the subscriber. The third message type is an issue message that contains the subscriber's node-number delivery address, the targeted wish list subscription pigeon hole and the actual issue payload. The messaging implementation has a local/global filter (see Figure 2) that will loop a node's self addressed messages back to be cached for the next simulation frame without passing anything down to the TCP layer.

2.2.2 *Peer-to-Peer Node Synchronisation*

The peer-to-peer synchronisation scheme is shown in Figure 3. Each simulation frame has three consecutive execution phases. Within the first phase, which is the *increment phase*, all the objects are put through their increment-publish cycles. The published issues are not messaged directly, but are grouped per destination node and cached until the second, so called publish, phase. The cached issue groups may now be sent to their respective destination nodes. The publish phase must be followed by a time-stamped end-of-frame message to each peer node to signify that all the issues for the current simulation frame have been sent. The end-of-frame messages perform a similar function as Chandy-Misra null messages [7] for dead-lock avoidance and time management in DEVS implementations. A simulator node will wait in the gather phase until it has received and processed an end-of-frame message from each of the other simulator nodes after which it starts with the increment phase of the next simulation frame.

2.3 TCP Message Passing Implementation

The TCP messaging implementation consists of two components. The first of which is an address translation from destination node number to destination IP and port before any message can be sent via TCP. This translation is pre-configured and fixed for each distribution configuration.

The second component is a two-tiered approach to lowering TCP message latency. The first tier is to ensure that as much as possible of the TCP send and receive overhead happens in parallel to the node execution. This is accomplished by increasing TCP's send and receive buffers to an adequate size such that the buffers have enough space for two simulation frames worth of data. This ensures that all TCP sends are non-blocking. It also facilitates CPU time, from a second CPU or hyper-thread or that's not used by the simulation, to be used to transport as much data as possible from the nodes' send buffers across TCP to their receive buffers for quick retrieval when needed.

The second tier takes control of the TCP message send times. TCP's Nagle algorithm tries to optimise bandwidth usage by conglomerating sent messages in the send buffer until it is large enough to fill a TCP packet or until a certain time-out is reached. The unfortunate side effect of the Nagle algorithm is that control over message latencies is lost. To give control over the message latency back to the simulator the Nagle algorithm is disabled.

2.4 Analysis and Results

The proposed architecture's real-time performance is analysed over distributions of one to six simulator nodes on the target infrastructure. With each node configuration the number of objects per node will be limited to achieve a real-time frame-rate. Finally a simple predictive model for the distributed performance behaviour is derived from the analysis data and used to do a first order estimate of the simulator's scalability to seven and more nodes. Accurate evaluation over more nodes should however be part of the future work section if such accuracy is required.

For the experimental setup the simulator nodes are similar Pentium 4 3.2GHz machines with 2GB of dual-channel RAM each and WindowsXP SP2. The network infrastructure is, as mentioned, Gigabit Ethernet with a D-Link DGS-3324SR managed switch. Each node has an Intel D945PAW mother

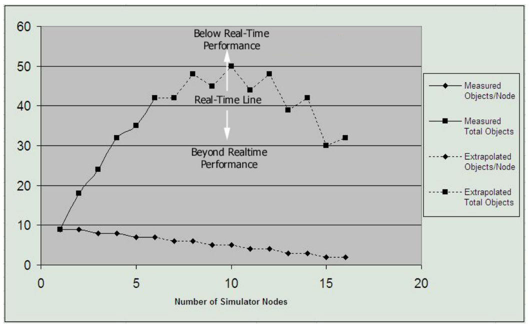

Figure 4: Total Object Performance of 100Hz Peer-to-Peer Simulator

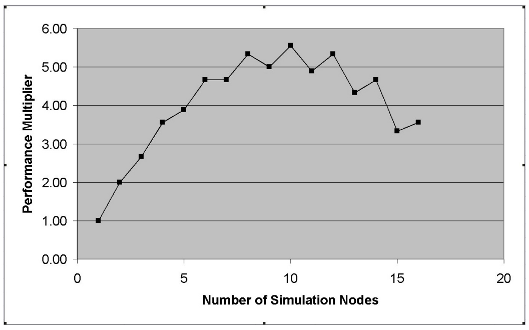

Figure 5: Real-Time Performance Speed-Up of 100Hz Peer-to-Peer Simulator

board with an on-board Intel Pro/1000 PM Gigabit Ethernet network card. The simulator nodes will be populated with instances of a "test" model. The test model has a fixed processing requirement of 1ms per 10ms simulation frame and an owned title with a fixed issue size of 512 bytes. Furthermore each instance of the test model subscribes to every other instance, creating the worst case communication scenario of a fully connected communication graph.

The performance result that is recorded is the maximum number of objects per node (see Figure 4) such that the simulation can still reach real-time. If the total number of objects are increased above the "Total Objects" graph, the performance will drop below real-time. Conversely, if the total number of objects are decreased below the "Total Objects" graph, the performance will grow beyond real-time. Both the total number of objects and the performance speed-up graphs are derived from the measured objects-per-node graph (Figure 4 and Figure 5).

Quantifying the measured communication overhead it seems that each time a simulator node is added, the number of model instances per node must be decreased by an average of 0.5 to maintain real-time which is a 0.5% overhead of the 10ms simulation frame. Assuming for the purpose of first order performance predictions, that the results do indeed indicate a linear distribution overhead of 5% for each simulator node added, such a linear overhead would most probably be in the receive loop of each simulation frame. Amdahl's Law [www.wikipedia.org] states that the speed-up attainable by parallel execution is limited by the sequential components of the system which in this case is proposed to be the single NIC, and thus single communication channel, per simulator node.

A linear performance might seem counter intuitive to what is expected of an n node and fully connected peer-to-peer structure where the total number of connections grows by n^2. The linear nature does however make sense if one remembers that the processing is done by n nodes resulting in a processing time of $\frac{n^2}{c.n}$ which is proportional to n and therefore linear. In other words, each node must strictly receive data from each of the other nodes in turn, limiting the potential parallelisation.

The first order objects-per-node performance for seven and more nodes is estimated by linearly extrapolating the measured objects-per-node curve (Figure 4) under the previous assumption. The spikiness of the performance graphs is due to the granularity of the objects which, in general, leaves a fragmented processing slot (idle time) on each node. The linear extrapolation provides an estimate for the scalability of the simulator, but as the number of nodes increases to beyond 10 the total number of objects eventually start to decrease which implies that the communication bandwidth will also decrease again. Around this point it is expected that the linear nature of the objects-per-node curve might change which requires, as mentioned earlier, analysis over more nodes to draw accurate scalability conclusions beyond 10 nodes.

From the analysis of the results it seems that the new 100Hz logical time DTSS publish-subscribe peer-to-peer simulator architecture achieves a measured speed increase, due to execution parallelisation, of above 4.5 when distributed over six simulator nodes, but not higher than approximately 6 even when distributed over ten and more nodes. This simulator is currently in use and working as expected, but the DTSS modelling formalism does seem to pose a technical difficulty in implementing still larger scale parallelisation of high resolution logical time simulations due to the sequential components of the architecture that cannot be parallelised.

3. A HYBRID DEVS/DTSS MODELLING APPROACH

In hind sight it seems like a good idea to rather develop a hybrid DTSS-DEVS modelling formalism, that has a DEVS layer enveloping the DTSS layer, to further migrate this specific simulation capability towards supporting large scale parallelisation. The two layer approach allows the existing DTSS models to be grouped and aggregated into systems level models for example, which may then be better suited to a DEVS modelling formalism. The DEVS layer then communicates only what is required and its parallelisation is not constrained by the underlying DTSS layer's logical time resolution which would require strict high resolution time synchronisation between nodes.

It is known that a DTSS may be embedded within a DEVS [6].

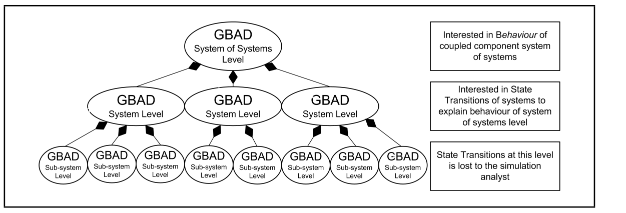
GBAD
System of Systems Level
GBAD
System Level
GBAD
System Level
GBAD
System Level
GBAD
Sub-system Level
Interested in *Behaviour* of coupled component system of systems
Interested in State Transitions of systems to explain behaviour of system of systems level
State Transitions at this level is lost to the simulation analyst

Figure 6: Double Structure Level of Discrete Time Simulator

This alone, however, does not improve the scalability of the simulator. The proposed aspects to improve the scalability is *Aggregating DTSS Models into DEVS Models* and *Using Dead-Reckoning Techniques* on the remaining high resolution data links. It is worth noting again here that reusing the current discrete time models, their publish-subcribe communication framework and the peer-to-peer messaging on TCP architecture is still of importance for economical reasons, minimising duplication of effort.

3.1 Aggregating DTSS Models into DEVS Models

The typical layout of a GBADS battery was described in the introduction. Most of the current GBADS models are at the level of GBAD sub-systems of systems within the battery. These models are typically modelled at a state transition system specification level or higher. At the GBAD system level the sub-system models are brought together to create system level models at a coupled system specification level as shown in Figure 6. The GBAD system models are then coupled again to create a GBAD system of systems level model also at the coupled component system specification level.

The nature of the system of systems simulation experiments requires objectives and outcome measures at the GBAD system level. In such an experimental frame the output variables within the GBAD system level coupled component models (the system structural knowledge) is hidden from the simulation analyst and argued to therefore be superfluous. The aim of *aggregation of the DTSS models* is the act of explicitly hiding the *double layer* of intermediate GBAD system structural information within a DEVS model. The new GBAD system level DEVS model is then a state transition system specification envelope which shields the model interconnect infrastructure from the communication overhead of the internal structure.

3.2 Using Dead-Reckoning Techniques

Some models, such as the incoming aircraft, are already at a GBAD system level. Nevertheless, these models still communicate at a high data rate to some of the other GBAD systems. The current tracking radar models, for example, require high time resolution target position input for their tracking filters to operate properly. This is due to the tracking radar, a GBAD sub-system, internally also being modelled at a coupled component level. This requires the correct external stimulation for all the components to operate together within the experimental frame for which they were originally developed.

The aim of the dead-reckoning technique is to trade accuracy for communication bandwidth, but in a clever way. An aircraft will, along with its position, make known to the radar how to best predict its path of motion up to x seconds into the future. The radar may then calculate for itself the aircraft's position as frequently as required. The aircraft will however keep track of were the radar thinks the aircraft is as the aircraft knows what prediction algorithm the radar is using. As soon as the aircraft's actual and predicted positions are outside a predefined error boundary of each other, the aircraft refreshes its current position and prediction method to the radar.

4. CONCLUSION

The analysis results of the discrete time simulator indicated that sequential hardware communication components of the infrastructure limit its scalability. A hybrid discrete time and discrete event modelling approach is proposed that will increase the scalability of the simulator by making more efficient use of the communication infrastructure while reusing the existing discrete time GBAD sub-system and system level models.

The proposed modelling aspects that will implement the hybrid modelling approach is *Aggregating DTSS Models into DEVS Models* and *Using Dead-Reckoning.* However, the open issues to investigate further are:

- The effect of this modelling approach on the simulation fidelity eg. the performance of the tracking filters once dead-reckoning is included, and
- how to compare the scalability of the discrete event simulator to that of the discrete time simulator.

Main advantage of this work is of course greater scalability of the real-time simulation capability, but additional advantages of the DEVS based simulator is:

- Improved scalability is the result of making more efficient use of the distribution infrastructure which implies that the simulator may now be distributed over longer distance lower bandwidth connections, and
- easier migration to The High Level Architecture (HLA), an IEEE standard for large scale distributed simulation interoperability, which is based on DEVS and popular within the military simulation domain.

5. REFERENCES

[1] Bernardt Duvenhage and Derrick G. Kourie. Migrating to a real-time distributed parallel simulator architecture. In *Proceedings of the 2007 Summer Computer Simulation Conference*, 2007.

[2] Johannes Lodewikus Pretorius. Feasibility considerations for a tailored simulation based acquisition (SBA) approach. Master's thesis, University of Pretoria, 2003.

[3] Jacques Baird and Cobus Nel. The evolution of M&S as part of smart acquisition using the SANDF GBADS programme as an example. In *Proceedings of the 12th European Air Defence Symposium*, volume 3694, pages 173–182, 2005.

[4] Shahen Naidoo and Cobus Nel. Modelling and simulation of a ground based air defence system and associated tactical doctrine as part of acquisition support. In *Proceedings of the 2006 Fall Simulation Interoperability Workshop*, 2006.

[5] Willem H. le Roux. Implementing a low cost distributed architecture for real-time behavioural modelling and simulation. In *Proceedings of the 2006 European Simulation Interoperability Workshop*, 2006.

[6] Bernard P. Zeigler, Tag Gon Kim, and Herbert Praehofer. *Theory of Modelling and Simulation, second edition.* Academic Press, 2000.

[7] K. Chandy and Jayadev Misra. Distrubuted simulation: A case study in design and verification of distributed programs. In *IEEE Transactions on Software Engineering*, volume SE-5, 2003.

Biography

Bernardt Duvenhage obtained his B.Sc (Honours) degree in Computer Science from the University of Pretoria in 2005 and is currently pursuing a Masters Degree. While part of the Mathematical and Computational Modelling Research Group of the Council for Scientific and Industrial Research (CSIR) in South Africa, he played a key role in developing the group's distributed simulator architecture; the simulation's terrain and LOS services; and the 3D visualisation and analysis tool of the synthetic environment. He is currently employed in the Optronic Sensor Systems Competency Area of a division within the CSIR. He intends further research in virtual environment simulation and visualisation.