Implementing a Low Cost Distributed Architecture for Real-Time Behavioural Modelling and Simulation

Willem H. le Roux
Council for Scientific and Industrial Research[1]
Meiring Naude Road
Pretoria, 0001
+27 12 841 4867
whleroux@csir.co.za

Keywords:
Distributed Simulation, Simulation Architectures, Modelling & Simulation

ABSTRACT: *As part of Modelling and Simulation-based Acquisition Decision Support to a Ground-Based Air Defence acquisition programme, dedicated simulations have been used to evaluate and develop tactical doctrine, define measures of performance and effectiveness and to answer techno-military questions. The simulations are loosely matched to the different phases of the procurement programme for efficient and effective support. This paper presents the lessons learned and issues identified during the development of the series of simulations, focusing on real-time, distributed execution and behavioural modelling and how budget and time constraints affected these.*

1. Introduction

"If you think good architecture is expensive, try bad architecture" – Brian Foote and Joseph Yoder.

In order to provide acquisition decision support to the South African Armaments Corporation (ARMSCOR) for the procurement of Ground-based Air Defence System (GBADS) equipment, constructive, aggregated simulations are used. Different versions of the simulation system were developed to match the GBADS acquisition phases for efficient and effective support.

Figure 1 maps all developed versions of the simulation system, referred to as the Virtual GBADS Demonstrator (VGD), to the first GBADS acquisition phase. The GBADS procurement programme follows a phased approach of which acquisition of the SABLE[2] Air Defence System forms the first phase [1].

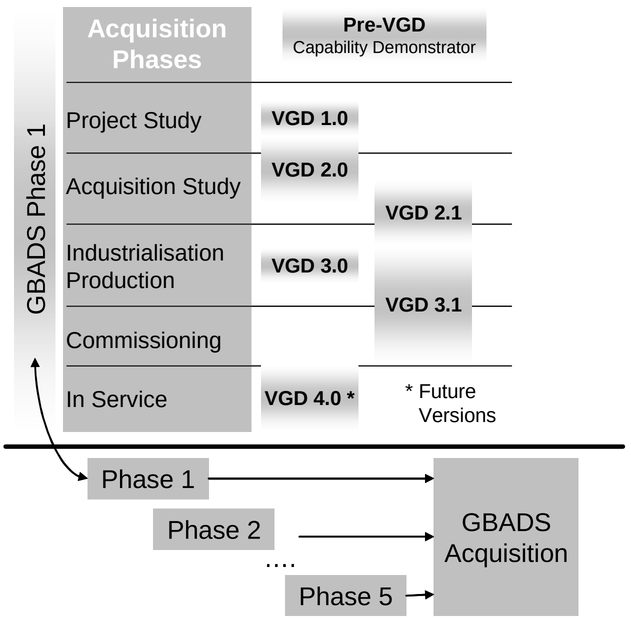

Figure 1: Simulation Versions, Acquisition Phases and GBADS Phases

The increasing capability and acceptance of Modelling and Simulation (M&S) as an effective tool to support the acquisition and utilisation of complex systems have in recent times made it an important area in which Research Institutes world-wide are developing and applying their technology bases. The need for decision support on the GBADS acquisition programme of the South African Air Defence Artillery (SAADA) offered an opportunity to

[1] The Council for Scientific and Industrial Research (CSIR) has been constituted by an Act of the South African Parliament in 1945. It is one of the leading scientific and technology research, development and implementation organisations in Africa. The organisation undertakes and applies directed research and innovation in science and technology to improve the quality of life of the country's people.
[2] The SABLE Air Defence System is an offering of the the Denel Aerospace Group.

apply M&S and in so doing to establish an indigenous M&S Acquisition Decision Support (MSADS) capability. The "window of opportunity" presented by the GBADS programme was used as a vehicle to establish a credible MSADS capability to act as a pilot Simulation-based Acquisition (SBA) project within the South African Defence Acquisition environment and which in time can be expanded to more applications [2].

This paper presents the lessons learned during the implementation of distributed simulations in an environment with resource and budget constraints, and identifies issues to be considered when developing low cost simulation architecture frameworks. After a brief discussion of the pre-VGD simulation that led to the conceptualisation of the first VGD, each version of VGD is discussed with its associated Modelling and Simulation issues.

2. Pre-VGD

An M&S prototype had to be developed with limited resources and budget within tight time scales. The prototype was developed in a Linux environment using Objective C as it provided built-in mechanisms for simplistic Two-dimensional (2D) visualisation and an object orientated abstraction layer with added features such as managing objects in a hierarchy.

The most important lesson learned with this prototype was to use the correct terminology. The same terminology as used by the end-user or client, in this case the SAADA, should be used. If and when simulations have to be discussed in any more detail than the input and output of it, the internal structures used should, as far as possible, match that of the real system being modelled. However, it can become a very time consuming and tedious task to keep up with the latest user terminology as in some cases it might evolve as the end-user or client becomes more familiar with the simulation.

The changing requirements of a simulation can take several paths – The end-user can insist on higher fidelity models or models from different vendors to conduct wider what-if type analyses with. Hardware- or human in the loop simulations can become priority if the end-user realises that training support with the system is a possibility.

Another aspect to take into account when developing simulations is at what system level is a simulation pitched? Figure 2, adapted from [3], shows a system level hierarchy applied to the M&S and military domains. For high fidelity, engineering-type models one-on-one or one-on-many simulations are preferred otherwise execution times become too long. However, if a higher order system's performance has to be evaluated, many-on-many type interactions are typically required to be able to quantify and qualify behaviour in more complex scenarios.

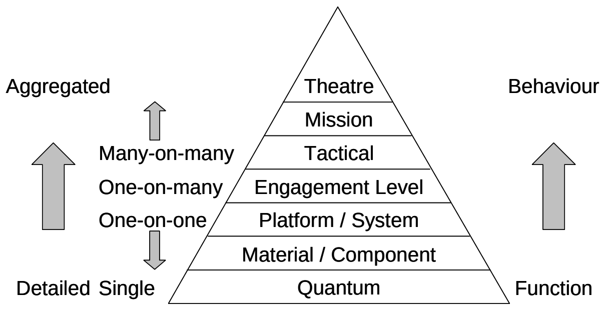

Figure 2: The Systems Hierarchy Applied to M&S

In the case of VGD, behavioural models are used to limit processing requirements and to allow the use of many-on-many interactions. This in turn enables the evaluation of system performance and not only that of individual sub-systems (e.g. specific radars).

3. VGD Version 1.0

The prototype GBADS simulation and VGD Version 1.0 (VGD1) was mainly used to explore simulation-based decision support possibilities and were implemented using a Rapid Application Development (RAD) philosophy as development resources were limited and impact had to be demonstrated quickly. VGD1 was implemented using a proprietary object-orientated visual programming environment, G2[3], which has expert system capabilities.

Both the prototype simulation and VGD1 were non-distributed stand-alone applications and were typically used in faster than real-time mode to generate reports for statistical analysis. Low fidelity, behavioural models were used. The architecture and models were also tightly integrated. Aspects that were identified as important philosophical or design choices are:

1. Should rapid prototyping be used?
2. What is the impact of model fidelity?
3. Confirm a common understanding of behavioural modelling.
4. Whether discrete event, time-based or a combination of discrete event and time-based simulations should be used.

[3] G2, a product and registered trademark of Gensym, is a real-time business rules engine for mission critical applications.

3.1 Should Rapid Prototyping be Used?

In a situation where it is not clear what can be achieved through M&S support, rapid prototyping offers some value. However, the client should be made aware of the fact that rapid prototypes have limitations in terms of flexibility, modularity and extendibility. It is often necessary to discard a prototype during subsequent phases.

Boundaries between models tend to be vague as most models will be of low fidelity. This drives more simplistic interfaces between models and the simulation architecture itself.

During rapid prototyping some aspects may not be modelled such as ignoring line-of-sight (LOS) calculations due to terrain obscuration and when a flat earth model is assumed.

It is advisable to use a RAD tool for prototyping as lower level computer languages requires infrastructure (also referred to as frameworks) to be laid down before actual development can commence. A disadvantage of most RAD tools is that they generally use interpreted or intermediate interpreted languages limiting the processing power available for intensive mathematical or logistical implementations due to run-time code interpretation. On the other hand integration via gateways or bridges or execution of binary routines are well supported allowing a simulation developer to model parts of a simulation in a RAD tool and other processing intensive parts in a more appropriate environment.

It is precisely this approach that led to a distributed simulation requirement for VGD. A RAD tool was used to prototype some models, but the more stable models that did not require further development were implemented in a lower level language. This required models to be executed over different machines as the RAD tool required a dedicated machine.

3.2 Impact of Model Fidelity

In general the higher the fidelity of a model, the higher the processing requirements and inter-model information load. Higher fidelity models tend to run at faster update rates which require more processing power as calculations have to be made in shorter time slots, if real-time execution is required. To reduce the processing burden, distributed simulations are used to share the processing load of models amongst processing nodes. This in turn requires distributed Inter-Process Communication (IPC) frameworks to allow for the efficient development and integration of models capable of taking part in such distributed simulations. Coupled to these requirements, model and simulation interfaces need to be standardised for interoperability and extendibility. All these issues have to be addressed in a unified architecturally sound framework to minimise rework, debugging and maintenance effort. Fidelity can be addressed at different levels of which some are:

1. Comprehensiveness (completeness) – Are the sun and moon positions accurate for the time of day and time of year in the flight simulator? Are wind gusts taken into account for the projectile trajectory?
2. Externally observable behaviour – Does the missile model have the same phases (power on, arming, lock-on, launch and detonation) as the real system? Will it be inactive of the power is switched off? Does it have a power button?
3. Internal implementation – Is the control law of the missile model based on the same mathematical principles and operation as the real missile? Are the internal electronics that manipulate the control surfaces of the missile modelled?
4. External environment interaction – Will the missile model fly a different path if there are wind gusts present?
5. Visualisation – How realistic is the representation of a simulation scenario? Is a participant immersed such that it will not affect his/her responses?

This leads to the concept of behavioural modelling where a piece of equipment is modelled for inclusion in a simulation.

3.3 Common Understanding of Behavioural Modelling

Behavioural modelling can span over one or more of the classes of fidelity listed above, but primarily resorts under externally observable behaviour. A behavioural model can also be complete (see fidelity levels in Subsection 3.2) or provide visual output that is very realistic, but in general, behavioural models do not have high fidelity internal implementations. A model of equipment should have the same aggregated behaviour and interfaces as the real system. However, this can be filtered with the application of the simulation in mind. An example illustrates this best: a search radar is modelled and the simulation span only covers actual battles (when targets are engaged) as opposed to including the times before and after a battle (preparation and maintenance of the radar, respectively). Assumptions can then be made, such as that the modelled radar can be considered to be in optimal condition to perform its given tasks during a simulated battle (unless break-downs have to be modelled). The model interfaces required to configure the radar during simulated battle can then be omitted, as it will not be used. If it is required to model the times before and after battle, hence radar preparation and maintenance as well, the radar model should support these activities, both with its internal states and its external interfaces.

To illustrate how behavioural models can be used effectively, a radar model is again used. Target signatures or radar cross sections (RCS) are generally highly dependent on a target's orientation relative to the radar detecting it. Figure 3 shows the probability of detection for a specific target as a function of the target distance and height for a single RCS value. In a simulation where the flight profiles of targets are not predictable, the radar model should allow for arbitrary target positions and orientations. This coupled with effects such as multipath reflections, complex aircraft geometry (leads to complex RCS signature variations) and tracking filter performance, radar detection and tracking performance quickly lead to complex and processing intensive calculations.

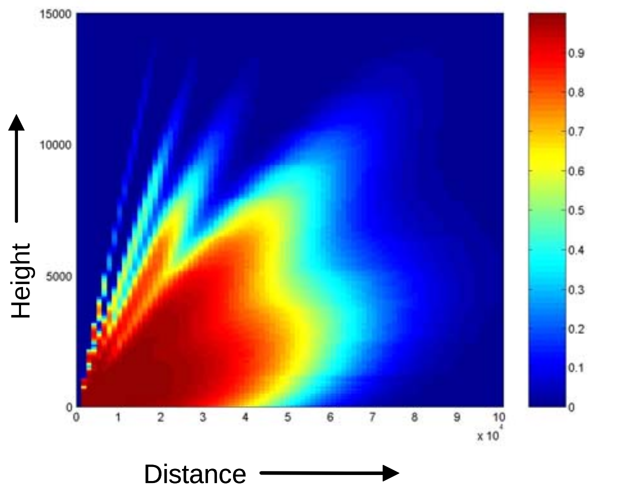

Figure 3: 2D Radar Probability of Detection Coverage Diagram – High Fidelity.

However, when a radar model is included in an aggregated simulation such as VGD, using a high fidelity coverage diagram-based models (as shown in Figure 3) is not practical as processing requirements cannot be met for real-time execution. Instead all factors influencing radar performance are combined into a behaviourally sound model that can be verified and validated against field tests or subject matter expert (SME) opinion. A possible solution for the radar case might be to use target position with Gaussian measurement errors and a simple distance and elevation angle threshold (see Figure 4).

It is not advisable to have a mix of fidelities within the same model, although using models with varying fidelity levels in a simulation might still be feasible. Higher fidelity models might require more information than available from lower fidelity models interacted with.

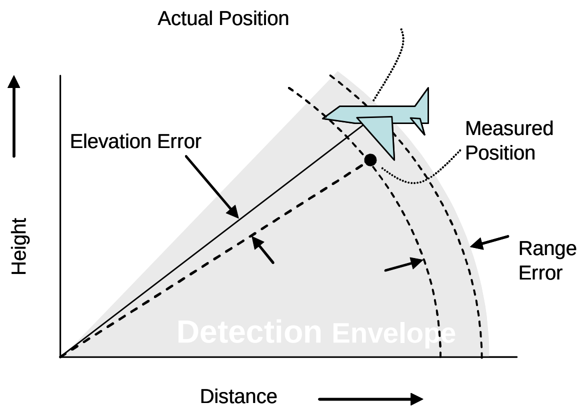

Figure 4: 2D Radar Probability of Detection Coverage Diagram – Low Fidelity.

If lower fidelity modules cannot match the required information fidelity that they are dependent on, assumptions have to be made that neutralises the advantage of using the higher fidelity. Continuing with the radar model example: if the model uses electromagnetic wave propagation formulas that are dependent on the ambient temperature, but the environmental model cannot provide a temperature to the model, the temperature value has to be fixed at an acceptable value. This in effect neutralises the parameter's value to the simulation. It may be acceptable in low fidelity simulations to use high fidelity models, but with parameter constraints and at the cost of processing power. Care should be taken in such cases that false impressions are not created that all of the functionalities of a high fidelity model is used or influencing the outcomes generated with a low fidelity simulation.

3.4 Discrete Event *versus* Time-Based Simulation

A technique to increase the execution speed of simulations is to use a discrete event timing scheme [4]. In principle it requires the calculation of the time events occur and the identification of inter-event dependencies. The event times are then sorted first to last. The simulation time is advanced from event time to event time until all events have occurred and been handled. This method assumes that dependencies can be identified prior to the time calculation and that it is possible to calculate the time an events occurs using a mathematical formula. It also assumes that events occur at a single time instant and not over a period. If this assumption cannot be made, the start and end times of the event can be used as two separate sub-events which can be calculated. It is very difficult or sometimes not possible to define all event dependencies or to accurately calculate event occurrence times. A solution to this type of problem is to predict event occurrences just prior to the time of actual occurrence. The simulation is then switched to a time-based scheme for a short while at the predicted time until the event occurs or a new event time can be predicted. Discrete event time schemes are typically used with statistical models to simulate the occurrence of events [4]. The VGD series of simulations are all time-based as it precludes operator-in-the-loop (OIL) simulation. Human operator reaction cannot be accurately predicted.

4. VGD Version 2.0

VGD Version 2.0 (VGD2.0) was mainly used in support of GBADS Phase 1 towards the end of project study and during the acquisition study phases [5]:

1. To conduct hit probability analyses for GBADS system performance quantification.
2. Comparison and evaluation of GBADS system architectures.
3. Timeline and sensitivity analyses to determine critical performance areas.

The two main requirements identified for VGD2.0 were to include Human Behaviour Modelling (HBM) through G2® in VGD and to use higher fidelity C++ models, supplied by external model vendors. HBM became a requirement as statistical analyses of simulated GBADS performance required between 30 and 100 iterations per configuration, which is not practical with OIL simulations. Higher fidelity models were required as the models used in VGD1.0 were of very low fidelity with questionable timeline characteristics.

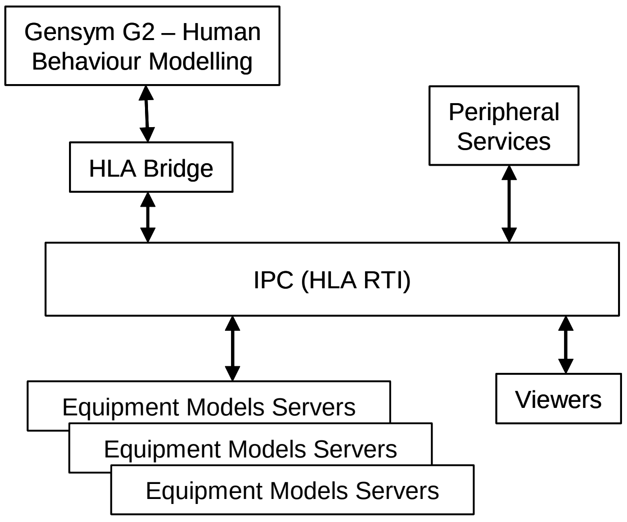

Figure 5: VGD2.0 Architecture.

Figure 5 shows the VGD2.0 architecture using the High Level Architecture (HLA[4]) as IPC framework. Note that G2® is connected to the IPC via an HLA bridge (also referred to as an HLA gateway). Peripheral services include support functions such as line-of-sight and terrain collision detection. Viewers can be immersive, three-dimensional (3D) scenario viewers or 2D plan-view scenario viewers. Models have all been grouped by type (class) and bundled in model servers (federates) such that a single federate maintains all instances of a model class (objects). This approach was preferred over using a model instance per federate or all model instances bundled into one federate. Too many federates slows a simulation down and using one federate only restricts ownership possibilities. Typically each block connected to the IPC executes on a dedicated computer, as well as the IPC server. The computers are connected via a standard 100 Megabit per second (Mbps) Ethernet Local Area Network (LAN). Computer platforms are Intel-based (Pentium III and IV) standard desktops with Microsoft Windows-based Operating Systems. This is the case for all simulations described in this paper. VGD2.0 was developed in approximately two person-years. Important lessons learned while developing VGD2.0 are highlighted in the following Subsections.

Aspects that were identified during the design and implementation of VGD2.0, and discussed in the following Subsections, are:

1. The use of HLA as simulation infrastructure.
2. Selecting the correct software language level for model implementation.
3. Human behaviour modelling.
4. Using models supplied by external model vendors.
5. Batch-mode execution.

4.1 The High Level Architecture

As G2® was used for HBM and is a processor intensive application the C++ models had to be executed on a separate machine. This led to a secondary requirement for distributed simulation. As a distributed IPC was required and external parties had to be involved in VGD2.0's implementation, HLA were selected. HLA is a standard that governs the entire simulation development effort. It is in essence a simulation engineering specification attempting to create interoperable simulations that can be combined into larger simulations [6].

[4] The High Level Architecture was developed by the Defense Modeling and Simulation Office in the Department of Defense of the United States of America.

4.1.1 HLA-based Software Development

As the South African industry had not yet adopted HLA or had any experience with it at the time, a framework or Software Development Kit (SDK) was developed to assist model development. The SDK provided an Application Programmer's Interface (API) in C++ that shielded the model developer completely from the HLA APIs. The federate development process was not shared between all the parties but entirely handled by CSIR. The HLA SDK allowed seamless use of models in the on-line simulation as well as in test environments at the supplier's site. Figure 6 shows the relation of the HLA SDK to the HLA Run-Time Infrastructure (RTI) API. Although the HLA SDK provides a more traditional C++ interface to the model developer, the developer can still access the HLA RTI API directly.

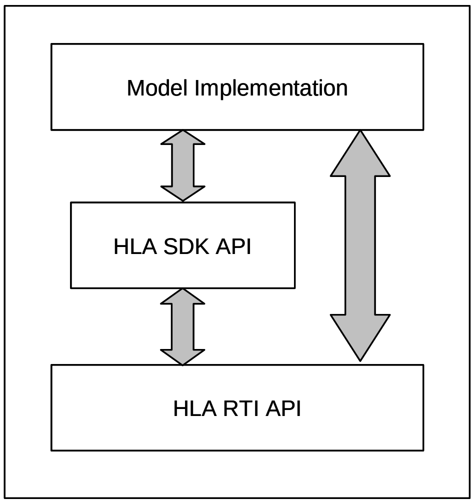

Figure 6: Relation Between the HLA RTI and HLA SDK APIs.

Some advantages of using a single team for simulation development are:

1. A single team at CSIR is responsible for the overall architecture and has control over it – This reduces the logistical effort compared to multiple teams being responsible for developing a framework.
2. The philosophy of the simulation architecture can be kept uniform and focused. It is difficult to contain a larger group and to keep design ideas and options focused especially if most are only affected by a subset of the architecture. Design choices are made according to the subset only, which might not be useful or practical in the bigger scheme of things.

Some disadvantages are:

1. When model vendors are not experienced with many-on-many simulations, assumptions made during the development of models can lead to integration errors, such as assuming that a supplied model will always be placed at (0, 0, 0). Such errors are not always visible in complex scenarios and can lead to totally incorrect results. Revising models to remove such errors might also not be trivial, as underlying assumptions can complicate matters.
2. All possible interface requirements have to be predicted by the team responsible for implementing the simulation architecture. This becomes even more complex if future models (or types) to be integrated are not known at the start. This specifically occurs in cases where concept support is rendered and system knowledge is low or non-existent. In the case of concept modelling it is better to use a larger and more diverse design team as it will then identify more interesting options.

4.1.2 Simulation Granularity

The HLA RTI provides the IPC for a simulation and supports the execution of a simulation on one or more processing nodes without reconfiguration. The granularity of a simulation is left to the developer, as well as the design and composition of federates, objects, attributes and interactions. A set of rules is enforced by the RTI (directly and indirectly) and services are provided and have to be used or adhered to if a simulation is to remain HLA compliant. The HLA SDK supports the most used services and those not supported can still be accessed via the HLA RTI API itself.

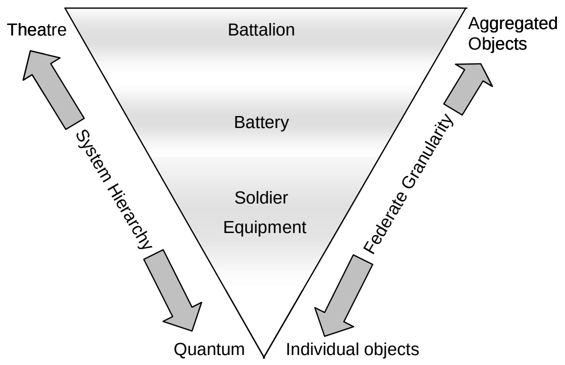

Figure 7: Object Granularity *versus* Systems Hierarchy.

Figure 7 relates object granularity against the systems hierarchy of Figure 2. Higher systems levels generally require more aggregated objects to be used in a federate, such as battalions instead of single soldiers. On the other hand, lower systems levels require more detailed or individual objects. Granularity is also dictated by processing power (more detailed objects at higher systems levels) and data availability (individual soldier data limited at battalion level).

VGD was implemented in such a way that each type of model (e.g. a 2D search radar or missile) is maintained by a dedicated federate. The federate can maintain multiple instances of the same type of model at the same time. Integration with other simulations, visualisation packages or external models are done by using a dedicated federate. As sensor and effector models typically require accurate position and time information, a conservative timing scheme was adopted – This means that whenever a federate wants to increment its time, all other federates must do the same. Federates that do not influence the flow of a simulation cannot hold other federates back, but cannot advance further than the others would allow. The conservative time scheme imposes high overheads as all federates have to be time synchronised.

4.1.3 HLA Compatibility and Compliance

As HLA was used more, it was suspected that in some cases organisations claiming HLA compliance do not necessarily use it internally in their simulations, but rather implement HLA gateways or bridges to simulations. Using HLA in this way rather results in HLA compatibility than compliance. This approach allows for the use of more efficient IPC schemes internally, but can cause slow down if a simulation is governed (in terms of time) by a gateway. To illustrate: a simulation runs in optimistic time management mode, close to real-time. If a gateway is connected that requires conservative time management, the simulation will be slowed down, as additional overheads will be incurred for synchronisation. It also seems that many HLA-based implementations do not use the RTI services or do not adhere to the HLA rules. This reduces the level of interoperability between simulations.

4.1.4 Interoperability

Another aspect that was discovered was that an HLA-compliant simulation implementation is governed by the Simulation Object Model (SOM). It implies if the SOMs of two HLA-based simulations do not match, the simulations are not interoperable. The SOM defines the hierarchy of objects, interaction and their defining attributes and parameters. SOM compatibility between two simulations can be at different levels:

1. The two SOMs can match at conceptual as well as syntax (naming) levels – The two simulations (federations) can be joined by either using a dedicated federate on each side that connects via a third party (can also be a federate in a third federation). Another option is to move all federates from both simulations to a new common federation.
2. The SOMs can match at conceptual level but not at the syntax level. This will require either renaming items in the SOM of one of the federations and then to integrate as discussed in (1) or translate on-line by means of the mechanism used to integrate.
3. The SOMs do not match at conceptual or syntax levels – This will require more complicated translation mechanisms and might not even be possible in some cases.
4. The SOMs do not match at conceptual level but at syntax level – This is a dangerous situation as it can lead to grave errors. Complicated translations are required, but a practical problem will be to keep the names apart in the integration mechanisms.

Apart from SOM compatibility, HLA rules and RTI services compatibility can affect the successful integration of two simulations. If a simulation uses an optimistic timing scheme – And in most cases an HLA rule is broken as synchronisation is typically not achieved through the RTI but through an external mechanism – and another a conservative scheme, it will be more complex to synchronise the two simulations. A simulation might also rely on Data Distribution Management (DDM) to be able to be real-time compatible, and another not.

It is suggested to simulation developers that when various parties are involved in development of a simulation, it is definitely worthwhile to follow the complete HLA process. On the other hand, if only one party is responsible for implementing an HLA compatible (note not compliant as HLA should then be used internally as well) simulation, other processes may be followed and only a gateway provided that is HLA compliant and compatible.

4.2 Software Language Levels

Each type of simulation development environment has its pros and cons – a RAD environment supports quick investigations but might lack scope. A lower level language environment has huge scope but requires infrastructure before actual simulation development can begin. It is for this reason that a visual-based object orientated programming environment has been selected for implementing models of tasks performed by humans. However, this environment is not suitable for processing intensive tasks, such as radar models that require mathematical formulas to be evaluated repeatedly. The C++ language has been selected for this purpose as it is object orientated but with higher execution efficiency. Using Gensym G2® (RAD tool) from VGD1, human behaviour was to be included in VGD2.0 with higher fidelity equipment C++ models.

4.3 Human Behaviour Modelling

In addition to higher fidelity models, an important part of systems level modelling of military systems is the element of human performance. HBM is specifically important where many-on-many simulations are used, as the timeline effect of a human operator cannot be easily isolated. Several approximations and models of human behaviour can be defined, but the crux is that the model should be supported by the architecture.

The HBM effort with VGD2.0 resulted in concurrent doctrine activities, as it became possible to experiment with doctrinal concepts using equipment that was not yet available to the end-user. Furthermore it empowered the end-user to identify areas requiring more exploration and analysis.

4.4 External Model Vendors

The decision was made to use models supplied by equipment vendors to ensure that the models were impartially validated and maintained. This proved to be viable for customer furnished equipment. It is necessary with vendor-supplied models to ensure correct use in a simulation environment, as the operational performance of the models can easily be affected. This will then lead to incorrect results and conclusions.

4.5 Batch-mode Execution

Batch-mode execution was found to be complex to implement with the combination of HLA, G2® and STAGE[5]. The federation design did not cater for batch-mode execution from the start, and controlling simulation runs via the G2® and STAGE gateways are not trivial. Although G2® had a commercially available gateway, STAGE did not have one. With limited resources it was more cost effective to manually repeat simulation runs than to implement batch-mode execution support and an extended gateway for STAGE.

5. VGD Version 2.1

It became apparent towards the end of the acquisition study support that more realistic target flight profiles are required for accurate timeline analyses. This was also anticipated to be a requirement for future acquisition phases. In order to support human pilots flying against simulated batteries, the simulation should at least be soft real-time compliant, hence the requirement to upgrade VGD2.0.

[5] STAGE is a product of Engenuity Technologies, Inc.

VGD Version 2.1 (VGD2.1) used a similar architecture as VGD2.0. The SOM has been optimised for execution performance (speed) to be able to achieve soft real-time, distributed simulation. An HLA gateway to a commercial flight simulator (Virtual Prototypes FLSim) that is flown by a human operator is supported in addition. Two aspects are therefore highlighted in this section, OIL and soft real-time simulation.

5.1 Operator-in-the-loop Simulation

In cases where HBM is not practical or it is too complex to achieve an acceptable degree of realism, human OIL simulation is an alternative. Advantages of OIL simulations include:

1. More realistic operator behaviour – Note it is still not perfect as the situational awareness fidelity in which the human operator finds itself may not be adequate for realistic decisions. Stress conditions are typically not the same as in real life situations because operators are aware the fact that it is only a simulation. Only a subset of information may also be available to the operator to base its decisions on. Stress conditions can be artificially raised by using a faster paced simulation or by introducing competitive elements between operators.
2. OIL simulations can be converted to training simulations, although the factor of situational fidelity comes into play again.
3. Human behaviour, typically action or response times, can be recorded on-line and analysed for input to higher fidelity models.
4. By running a simulation in OIL mode, internal software errors or interface errors can be uncovered if inexplicable simulation behaviour is observed. It is then used as a software debugging tool.

Some disadvantages are:

1. Batch-mode simulation executions are limited if large samples are required to calculate accurate statistics. The element of predictability and simulation “quirks” are easily learned within a few iterations by a human subject and then abused to gain an unfair advantage during a simulation execution.
2. For simulations that lack graphical interfaces, dedicated OIL interfaces have to be developed – This might be complex as visualisation of concepts is not always straightforward.
3. Human boredom can play a role, specifically if there are long times when no operator interaction is required – As example: The operator has to wait until the targets enter the launch envelope of a missile system.

4. Simulations have to be at least soft real-time compatible. If the simulation executes too fast, the operator can be subjected to unrealistic stress when making decisions. In some cases this approach can actually be used to analyse human performance or to make scenarios more complex. On the other hand, if the simulation runs too slow, the operator will have an unrealistic advantage to make decisions in time.
5. If too many human operators are required to execute a simulation it can become expensive and impractical to coordinate, schedule and "choreograph" the run.
6. When using humans, simulation execution outcomes may vary significantly enough such that it is difficult to draw proper conclusions from a few runs.

To get more accurate results with OIL simulations operators have to be briefed with the relevant contextual information, specifically when an operator has control over elements in the simulation. This has not been done with VGD-based experiments and is an aspect that will be expanded in future. More sophisticated briefing, debriefing and after-action reviewing tools are required which also drives data logging requirements.

5.2 Real-time and Distributed Simulations

As mentioned in the previous Subsection, OIL simulations require soft real-time execution. Human operators are tolerant (or oblivious) towards intermittent time lapses (where the simulation fails to maintain real-time execution), except if noticeable events are omitted. Hardware-in-the-loop simulations on the other hand generally require hard real-time execution to maintain synchronisation.

In a real-time simulation, limited processing power, memory and bandwidth all dictate available time slices for entity models to be processed in. The more models included in the simulation execution, the shorter the processing time slice becomes (given the number of processing nodes is constant). A popular solution to this problem is to add more processing nodes, but then the IPC efficiency coupled with the bandwidth will dictate the practical maximum number of nodes. Data transfer between models also has to be optimised to gain even more efficiency.

Distributed simulations are generally more difficult to implement and debug as the execution of a simulation occurs over more than one node. IPC schemes exist that allows seamless changes between executing an entire simulation on one or more nodes. If an IPC scheme supports the execution on one node only, albeit not in real-time, development and debugging become easier.

6. VGD Version 3.0

VGD Version 3.0 (VGD3.0) was mostly used during the industrialisation and production acquisition phases for statistical analyses of GBADS performance evaluations. The decision was made to discontinue the use of proprietary software products, due to expensive licensing fees, product lock-in and easier batch-mode execution support. Peripheral services in VGD2.0 and VGD2.1 (See Figure 5) were implemented with STAGE and included LOS, terrain, threat and scenario management services. In addition, operator modelling of multiple target handling had to be supported, as VGD2.0/2.1 had limited capabilities in this regard.

VGD3.0 uses the same entity models as VGD2.1, except for the HBM part that is replaced with a populated C++ framework. A target-centric approach was adopted to define operator tasks and parameters. All equipment models were re-used by means of intermediate interfaces (wrappers). In addition to this, batch mode execution of the simulation was a more prominent requirement to support statistical analysis of results. As neither VGD2.0 nor VGD2.1 was implemented to support batch runs, and developer resources were low, a less complex and more maintainable system was required. Two options were investigated for an IPC framework. These are highlighted in the next two Subsections.

6.1 Common Object Request Broker Architecture

A quick investigation was conducted in using the Common Object Request Broker Architecture (CORBA®[6]) for the purpose of an IPC framework. The HLA RTI used in VGD uses CORBA® as an underlying framework. The investigation was done, as at the time none of the local model vendors used HLA or CORBA® and it was decided to move to a more simplistic IPC (less distributed in a sense) as resources were limited. However, CORBA® will still be considered in future as it is a language independent, distributed architecture with an established user community.

6.2 Transfer Control Protocol

Between the Transfer Control Protocol (TCP), HLA and CORBA®, TCP is the most widely used IPC framework. Most programmers also have experience with it. As the real-time requirement for VGD became less of a priority – the focus was more on batch-mode execution for statistical analyses – it was not pursued anymore. The decision was made to lump all models into a single executable, and to only distribute the visualisation aspects

[6] CORBA is a registered trademark of the Object Management Group.

of the simulation and the operator modelling modules. All models were reused from the distributed simulation and reintegrated using the HLA SDK API wrappers as is. The models were stripped of the HLA SDK eventually as it caused execution overheads and nothing was gained. More focus was also placed on developing an environment in which operator tasks, processes and decisions could be modelled more efficiently and flexible.

At the point where VGD was almost completed and ready for experiments to be defined, implemented, conducted and analysed, the requirement for OIL simulations were raised again. This was due to the fact that human behaviour modelling became very complex as certain tasks that were originally thought to be very well-defined and rule-driven, were in fact not as well defined. In addition to this it was realised that a big driver of overall system performance, given the set of rules, actions and procedures (doctrine, weapon drills and standard operating procedures) was still human performance. Specific aspects are target acquisition (with and without assistance), visual identification and weapon drills. Although these aspects could eventually be modelled, the end-user felt strongly that more flexibility is required to experiment with tactical doctrine by means of OIL simulations. Soft real-time execution is a requirement for OIL simulation. However, at this point it was decided that a simpler and more cost efficient approach should be adopted for a new IPC framework. Although VGD3.0 was used for experiments (batch-mode executions), it was never completed fully.

7. VGD Version 3.1

The lessons learned with the development and application of the preceding versions of VGD culminated in the design and implementation of VGD Version 3.1 (VGD3.1). It is the current operational version, and is applied in GBABS acquisition phases I and II. For phase I it is used up to the commissioning phase support and for phase II up to the production phase. VGD3.1 has been extended to demonstrate concepts in other domains, such as Joint Air Defence (navy) and persistent, real-time maritime surveillance.

VGD3.1 was required to support OIL simulations thus soft real-time execution support was necessary. As the number of models added to VGD increased considerably, and the scope of support of VGD expanded to include Operational Test and Evaluation (OT&E) and subsequent phases of the acquisition programme, the architecture of VGD3.0 had to be updated.

7.1 A Simplified, Distributed IPC Architecture

The use of a central global memory (implemented in hardware) connected to processing nodes via high bandwidth connections such as FireWire800™ (Apple Computer Incorporated – IEEE 1394b), Universal Serial Bus Version 2.0 (USB2.0) or Gigabit (Gb) Local Area Networks (LAN), was investigated. To evaluate the concept a software-based central global memory was developed based on TCP for communications over 100 Mbps or Gigabit per second (Gbps) LANs. A standard client-server configuration was used. It was soon apparent that the client-server architecture had severe drawbacks: in essence if a message is sent from one client to another it has two legs to pass over – the first from the transmitting client to the manager and then from the manager to the receiving client. The number of connections at the manager also becomes a bottleneck as it will have as many connections as there are clients. A peer-to-peer solution was proposed, in which all nodes are considered peers and are connected to all other nodes (fully connected). It results in the same number of connections at each node, but communications between peers are now direct (a single leg only). A further improvement is that parallel communications can occur over independent connections but is still limited by the underlying hardware architecture (network switch backplane fabric). Figure 8 shows a client-manager versus peer-to-peer architecture.

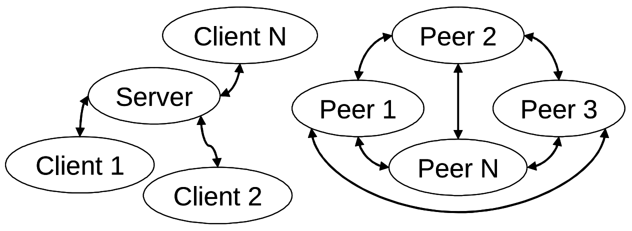

Figure 8: Client-Manager vs. Peer-to-Peer Architectures.

To control the flow of information between peers, a mechanism similar to the publication and subscription object management service of the HLA RTI is used. Models within peers have to indicate which information is available to other models by publishing a title. Other models can subscribe to published titles and will receive issues (data updates) at the rate specified in the subscription. If a model subscribes faster than what another can offer issues at, the last valid issue will be provided until a new update is available (which will then be provided).

Time management is an integral part of the framework and is a conservative scheme allowing models to operate at an integer multiple of the internal framework clock. The internal clock is configured at a 100Hz update rate but can be changed which will then affect the execution performance. Models can be configured to run slower than the internal clock, but not faster.

An additional advantage of the framework is that start-up and shut-down sequences are well defined, as well as the fact that a single executable is used for all peers (processing nodes). A peer therefore only loads the models that are required to be executed by it, allowing future extensions such as automatic load balancing. Debugging is easier, as the entire simulation can be run as a single executable with all models configured to be executed from a single peer – This is controlled with the configuration of a single switch in the model start-up configuration.

7.2 Architecture-integrated SOM

The SOM is also a built-in feature of the architecture and provides generic interfaces for:

1. Models – All entity models such as missiles and radars are derived from this interface.
2. Services – Provide for shared services such as terrain collision detection, ground-height information, LOS and time of day. Data logging can also be implemented as a service.
3. Consoles – Provide the means for gateways, stealth viewers and interactive visualisation consoles (OIL).
4. Titles – A base template is provided for all title definitions.
5. Spatial Reference Model (SRM) – Built-in support for a spherical earth model with a mean sea level (MSL) radius as an average of the equatorial and polar radii of the World Geodetic System 1984 (WGS1984). Constructs for North-East-Down (NED) Cartesian coordinates and orientations (heading, pitch and roll) are provided, as well as Earth Centred, Earth Fixed (ECEF) Cartesian and Meridian coordinates. Local Level, Local North (LLLN) coordinates can also be used.
6. Peers – Each peer has a standard interface to support models, services and consoles and to connect to the simulation backbone (TCP network). Each peer is in essence a single application (executable) that connects to other peers.

Each peer can be configured to support any number of models, services and consoles. However, the loading of a peer should be balanced with the rest as a single overloaded peer can limit the execution performance of the entire simulation. Load balancing is still a manual process at this point in time.

Figure 9 shows the VGD3.1 architecture as collections of models, services and consoles (not all instances are indicated). The figure does not indicate which peer executes which model, service or console, but only that they connect to the IPC via the peer's API.

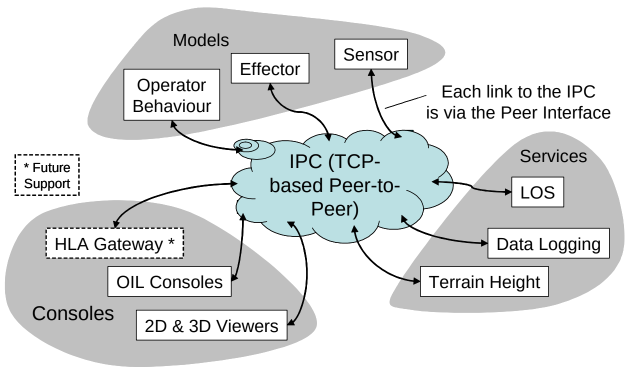

Figure 9: VGD3.1 Architecture.

7.3 Architecture Implementation

The architecture has primarily been designed and implemented by a single software developer over a 6 month period, but is used by a group of developers. It is a flexible environment and is suitable for different types of simulations, but with customised features for air defence simulations.

7.4 VGD3.1's Size

Simulation executions involve a number of models, consoles and services. A typical configuration is shown in Table 1.

Simulation Object Count	
Type	**Quantity**
Models	125
Consoles	9
Services	6
Objects (Total)	140

Table 1: Typical SOM Object Count for VGD3.1.

The simulation can be executed on any number of processing nodes, but to maintain soft real-time compliance, approximately four to seven nodes are required. The number of nodes is determined empirically.

The complete simulation code base, including models, viewers and the architecture itself is 466420 lines (386400 excluding empty lines) in 1307 files. Other statistics are shown in Table 2.

Several extensions have already been made to the architecture to allow amongst others, dynamic (late) publication of titles, third party brokering of publication-subscription between models and integrated data logging for models.

File and Line Count Statistics			
Type	File Count	Line Count	
		Total	Excl. Empty
All Code	1307	466420	386400
All C/C++	1160	391375	316888
All C#	147	75045	69512
IPC Only	25	12021	9347
SOM Only	8	1263	977
SRM Only	18	3821	2858

Table 2: Line and File Count Statistics.

8. Beyond VGD3.1

The natural growth path for VGD would be to cover more GBADS phases and then to move into the Joint Air Defence (JAD) domain where other Services and Divisions of defence are involved such as the South African Navy (SAN) or South African Air Force (SAAF). Beyond JAD lies Joint Operations, which implies that even more models will be developed and integrated with VGD. External developers for models, consoles and services may then be used. This will require a more process-driven approach (such as HLA) as a single team will not be responsible for all development anymore.

9. General Simulation Development Issues

This section addresses some general software development issues in context of simulation development.

9.1 Development Costs

An important factor to consider when developing simulations is the cost of ownership. If commercial frameworks or architectures are used as the foundation of a simulation, training, maintenance and flexibility should all be kept in mind. As most commercial packages do not provide source code, flexibility is not possible or expensive if the vendor is requested to make alterations. On the other hand, in-house development can be just as expensive, as debugging is time consuming and changes to a large code base are complex and difficult. However, having full control over the architecture is also an advantage, as internal mechanisms can be accessed to adapt, extend or modify functionality or performance. Still, compared to the cost of military acquisitions, the development costs of a simulation such as VGD3.1 is minimal.

9.2 Modularity

Modularity is implied in two ways. One is the way in which models, services and other peripheral aspects are handled in a simulation. Is it easy to add a new type of service or a new model? The second is at the level of the simulation itself. Can the SOM be extended without serious repercussions, such as adding additional attributes to the base model? Can additional mechanisms be introduced such as DDM? Another question is whether the underlying communications layer can be exchanged for another such as the User Datagram Protocol (UDP), CORBA® or the HLA RTI. Provision for such actions can be designed into an architecture but will be at a performance overhead cost.

9.3 Re-use

Models developed with a particular API always tend to have embedded traces of it in addition to the standard way of implementing the model (e.g. inheritance). It should be limited as far as possible as it often limits the level of re-use. A preferred method is to use an intermediate (integration) class and implement the model as a pure C++ (or other applicable language) class. A slight performance overhead will be paid, but maintenance and upgrading of the model will be much more isolated and efficient. Of course if upgrades require interface updates, the intermediate class has to be updated as well, but experience has shown that the intermediate class maintenance is not necessarily a complex task. This method also allows isolated testing of a model and the interface.

9.4 Incremental Development

In order to provide APIs for model development and other peripheral services for a simulation, a minimum infrastructure is required. Up to the point where the interfaces are stable, development of software that depends on the architecture is difficult and may require rework or restructuring should the interfaces change. It is thus important to first design and implement interfaces and underlying data structures, such as coordinate frameworks. Once a basic infrastructure has been established, an incremental (also known as iterative or spiral development) process can be followed to add functionality to the architecture. Typically additions are made as dependent software requirements arise. This approach reduces development costs, as a large scale design effort up front is expensive and the lead time until the first lines of code are written long. It is also difficult to preconceive all required functionalities right at the start as knowledge is captured and encoded in a software system as the developers, systems engineers and end-users of the system mature in their proficiency with the domain. Another danger of doing a grand design up front is functionality that is anticipated to be very useful ends up not being used due to complexity, obsoleteness or cost.

9.5 Update Rates

During the development of the distributed TCP-based architecture, it was proposed to set the update rates of types of models to appropriate values, and to spread the update times of models throughout the available time slices for real-time execution (0.01s for 100Hz real-time update rate). Although this scheme seems to be a good solution to the problem, it will still result in a worst-case scenario that will cause a cyclical slip in real-time compliance. Every so many cycles, the updates rates of different models will result in a clock cycle that has a maximum number of models to be updated at any given time. At this time the available time slice might not be adequate to perform all processing, causing a slip in the real-time compliance. If this cycle repeats too often, the simulation will slip further and further from real-time. Other clock cycles might be under-utilised, which can be used to process models in advance but only if a model is independent of the state or output of other models which is rarely the case.

9.6 Architecture Abuse

The HLA RTI API enforces the rule that a simulation federate is not allowed to change its state during a publication update, whether it was pushed or pulled from the RTI. This principle prevents possible simulation state inconsistencies as the subscribing federate can request publications at a faster rate than the federate can supply. If it is allowed to change its state during the supply of data to the requesting federate, its state will become inconsistent with its own internal update cycles. This is one example of architecture abuse by end-user programmers and has to be prevented by using well-designed architectures. The current distributed TCP-based architecture does not enforce this policy strictly.

9.7 Software Error Prevention

Software errors, First Incidence Reports (FIR), or bugs as more commonly known, can be prevented in several ways, although not completely:

1. Use an applicable design methodology such as the Unified Modelling Language (UML).
2. Draft complete software specifications before implementation.
3. Use software development standards for actual coding (variable naming, class names, etc.).
4. Use the strictest possible specification for access to variables, classes, attributes, etc. If a class method is not allowed to change the attributes of its owning object, prevent it from specifying the correct control measures.
5. Paired coding from the Extreme Programming methodology is a very effective way of identifying problems on the go [7].
6. Subject newly developed software to as many as possible end-users and other developers.

10. Future Work

Dynamic load balancing can be easily added to the distributed TCP-based architecture (VGD3.1) because the required information is already available at all peers. As a conservative timing scheme is used, each peer will detect when other peers have finished processing the current time slice, and which peer was the first to be completed with its processing tasks. The assumption is then that the peer that completed first will be the one with available processing capacity. Models can be transferred to it one by one until another peer becomes the first to complete its processing cycle. Control mechanisms have to put in place to prevent passing models too often instead of processing them. A good measure would be to stop transferring models once real-time is achieved within a comfortable margin. The only aspect that has to be implemented is to transfer models from one peer to another and to control the process. The transfer of modules requires each model to be able to serialise itself.

It will also be an interesting exercise to evaluate the use of more HLA rules and RTI services in the distributed TCP-based architecture. This will result in a peer-to-peer instead of client-server architecture as used in the HLA RTI. The pros and cons for the two architectures can then be compared with experiments.

11. Conclusion

Some lessons learned while implementing several versions of a GBADS simulation have been presented. The different versions ranged from single applications to fully distributed simulations employing the latest IPC frameworks. The most important lesson learned is that a good architecture does not need to be expensive. Well designed mechanisms provide adequate integration scope and a light-weight API reduces complexity and limits the breeding ground for software bugs.

It should also be kept in mind that simulation systems tend to capture knowledge of subject experts (of the models) and systems engineers (of the “glue” that connects the models – not to be confused with the IPC). This implies that a simulation system that is delivered to an end-user is not the only impact made. It is the entire process of building the simulation that adds the most value. The simulation developers have to ask apparently trivial questions, and if models of real systems are used, they have to figure out how to interface the models with each other. As this process unfolds, both the developers and domain experts are educated in a structured way about the modelled system and a "body of knowledge" build up and maintained. Table 3 summarises the lessons learned with the development and application of the VGD series of simulations.

Simulation	Important Lessons Learned
Pre-VGD	Use the correct terminology (same as end-user) from the start.
	Match real system as far as possible with simulated – interfaces, structure, etc.
	Decide which systems levels are addressed with the simulation to determine fidelity level of models
VGD1.0	Use behavioural modelling for higher systems levels to reduce processing requirements and data requirements
	Use RAD when M&S goals are not clear
VGD2.0	OIL simulation requires time-stepped mode
	HLA compliance is not the same as HLA compatible (interoperability)
	HLA interoperability only achieved when SOMs match
VGD2.1	OIL simulation provides more flexibility than HBM
	OIL simulation requires soft real-time execution
	Batch-mode execution not trivial in HLA and gateway environment
VGD3.0	Proprietary software: product lock-in, and expensive to extend
VGD3.1	Lightweight IPC possible if architecture is not too generic
	Peer-to-peer architectures offer performance advantages over client-server approaches.

Table 3: Summary of Lessons Learned with VGD.

12. References

[1] H.J. Baird and J.J. Nel: "The Evolution of M&S as part of Smart Acquisition using the SANDF GBADS Programme as Example" Twelfth European Air Defence Symposium, Shrivenham, June 2005.

[2] J.L. Pretorius: "Feasibility Considerations for a Tailored Simulation Based Acquisition (SBA) Approach" M. Eng. Project Report, University of Pretoria, 2003.

[3] J.S. Roodt: "Modelling and Simulation Verification, Validation and Accreditation Process" Technical Report DEFT-MSADS-00049 Rev 2, CSIR DPSS, Pretoria, March 2002.

[4] C. Tonkinson: "A Basic Course in G2 Version 5.0. Course Notes" Knowledge Base Engineering, Sandton, 1998.

[5] R. Oosthuizen, "VGD-2 Requirement Definition," Technical Report DEFT-MSADS-00052 Rev 1, CSIR DPSS, Pretoria, July 2001.

[6] F. Kuhl, R. Weatherly and J. Dahman: Creating Computer Simulation Systems: An Introduction to the High Level Architecture, Prentice-Hall, Upper Saddle River, 1999.

[7] J. Highsmith: "Extreme Programming: Agile Project Management Advisory Service" White paper, Cutter Consortium, Arlington, 2000.

13. Acknowledgements

The author thanks his colleagues at the CSIR for taking the time to review this paper and provide valuable suggestions. The development of the Virtual GBADS Demonstrators was supported by the South African Armaments Corporation.

HERMAN LE ROUX has been with the South African Council for Scientific and Industrial Research since April 1998 and is at present a Senior Researcher in the Mathematical and Computational Research Group. He is involved in Modelling & Simulation-based Decision Support, specifically for the South African National Defence Force. Interests include data fusion, biometrics, artificial intelligence and software engineering. Le Roux completed a Masters Degree in Computer Engineering at the University of Pretoria in 1999 and is currently pursuing a PhD in Multisensor Data Fusion.