Proceedings of the 2004 Winter Simulation Conference
R .G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, eds.

AUTOMATED DATABASE AND SCHEMA-BASED DATA INTERCHANGE FOR MODELING AND SIMULATION

Gregory A. Harrison
David S. Maynard
Eytan Pollak

Lockheed Martin Simulation, Training & Support
12506 Lake Underhill Road
Orlando, FL 32768, U.S.A.

ABSTRACT

Creating a simulation of a large enterprise system by manually coding all the details into a simulator tool is not just time consuming, but yields a system that is difficult to maintain. By separating the model-configuration data from the models, a higher level of automation can be achieved, and enhance the usefulness of the simulation. The underlying data can be manipulated by the subject matter experts, and then transformed into the appropriate structure for simulator use. This paper describes a method that automatically configures a simulation using external data that interfaces to generic processing flow. The models and the simulation were co-designed along with the interchange data representation to enable generic models to be configured under software sequenced by a workflow system. This allowed model re-use, and automatic configuration changes, in support of optimization. We also describe the application of this technique to the simulation of an enterprise, student-training system.

1 INTRODUCTION

In the simulation of large systems, there can be great quantities of details involved in the configuration of all the subsystems in the model. Programming all the details by hand could be very tedious and error-prone. In some cases, the details would be very necessary to the validity of the model, and to the usefulness of the results of the simulation. To help overcome this difficulty we developed a method to configure a simulation automatically, using data stored in a relational database.

We concentrated on the configuration of systems of models in a discrete event simulation. Discrete event simulation is described in Banks (2001). The discrete event simulator that we used was Arena, from Rockwell Software. Arena supplies a visual interface to create the models, and a VBA interface to allow passing data in and out of the simulation. To accommodate automatic configuration using outside data, generic models were designed, having slots for external data. These configurable data slots defined many things within the model, such as resources that were required, the amount of time different operations within the model would take, and other required configurations, such as the model's hierarchical position in the overall architecture.

In the ordinary case, once the generic models were entered through the visual interface, they would be unchangeable. But the techniques described in this paper allowed us to use the capabilities of the visual-user-interface-programmed discrete simulator, as well as to control the settings of all the individual models within, using configuration data from a source outside the program.

The main technologies involved were Visual Basic, Applications Edition, or VBA and the Extensible Markup Language, XML, especially the XML schema capabilities. These allowed the transfer of large amounts of articulated data right into the Arena application. In the use of other simulators, it is still recommended to use XML, but the VBA section may change depending on the capabilities of the simulator.

A workflow system sequenced all the various software applications, and served to pass the date from one application to the next. This workflow proceeded in a completely automatic manner. It used the Java programming language to interface with the database to create the XML data that it passed to the VBA of the discrete event simulator.

This paper describes the methodology that enabled the creation of a complex, external-data-driven simulation. Discussion of the various components, and the reasons for their use is included. It also describes an application of this methodology in the simulation of an enterprise-level student-training system.

2 SEPARATING THE MODEL FROM THE DATA

There is a certain regularity in models of a similar type. Even if there are some differences between models in a type class, the differences may be small. Thus, you may be able to group your models into distinct types, enabling you to identify pertinent characteristics about each type of model, and create individual data items to control the setting of each of the pertinent characteristics in the simulator model.

Data-driven modeling allows automation to be applied so as to specify the operation of each model instance in the simulation. For instance, if you have twelve different classes of models that need to be used in a certain simulation, you can represent the different model types as each having a certain set of data that applies to that type. You can create a database that holds these sets of data, with different data sets for each model type. Then, this data would provide information as to how to represent the internal structure of each model in the simulation.

In a discrete event simulator, two of the most important variables are the temporal position in the chain of events in each model, or the event 'flow' position, and the amount of time to take in that flow position. Flow positions can be modeled as either occurring in parallel (as in simultaneous), or in a sequential flow. This modeling is specified by the configuration of the model in the simulator. It is necessary that the discrete event simulation model flows and time duration information requirements are specified early in the design process, so as to allow simultaneous development in the design team.

Now that the abstraction items of flow and time-delay are available, the specification of the state can be tabulated as part of the data, and encoded into the database, or extracted from an existing database. The model can state that there is a delay at time index three, but the amount of the delay is part of the data realm.

The existence of flow states can be turned off by not specifying them in the data, and the discrete event simulator model must know to skip this step in the resulting flow. Similarly, the amount of delay can be controlled by the data also. This is the basis behind the ability to specify the operation of the simulation by a detached data model.

A versatile simulation model needs to be provided by the simulation tool. It must be possible to specify the various states and delays inside the model using data that comes from a source external to the simulation tool. Many different model operations should be able to be specified through the data, enough so that the set of models can represent the totality of all possible configurations expected in the final system.

One of the benefits of this representation is that having the knowledge of how the underlying model operates, which should closely match how the actual physical process operates, then the Subject Matter Experts (SME) can have an expedited method to specify the underlying system performance properties, while the model would remain unchanged. The SME can specify the control data using ordinary desktop tools, such as a spreadsheet or database, and not have to be experts in the simulation tool. The families of generic models are automatically configured using the SME data. This exemplifies code reuse, if the underlying models can be made flexible enough.

2.1 The Link Between Database and Simulator

To accomplish the inter-process communication between the creation of the data and the simulation use by the models in the simulator, some method must be used to transfer the data. With the prevalent, and expanding, use of XML, it was apparent that the best technology for the current and future data transfer was XML. XML allowed a detailed, hierarchical transfer of data, while still being readable by human operators.

One benefit of XML was that development could continue in parallel, after the XML schema was decided upon. The modern XML parsers and generation tools all rely upon a data format specification mechanism, called a schema to ensure that the information transmitted corresponds to the information that was sent. The sender has to ensure that their data is valid with respect to the schema, and the recipient of the data can use library routines to ensure that the data format is valid with respect to the published schema. This is a great improvement over prior, stove-piped, implementations, where the utmost care would be required to ensure that the binary or ASCII data met the expectations of the sender and the user.

2.2 Database Choice

In the application that we will describe later, the source data was already encoded in the Microsoft® SQL Server relational database system. Due to this constraint, we developed a system to translate the source data into the data formats that were immediately useful by our discrete event simulator. While other database products may perform equally admirably, we stayed with the Microsoft® SQL Server because it allowed the suitable creation of the XML data-transfer format, and was robust enough for our application. We used the stored-procedure system to create intermediate tables that were more directly converted into XML text. This system is described in Šunderić (2003).

The XML text was obtained in two ways, depending on the stage of system operation we were in, whether engineering or on-line operation. As the maximum record size for Microsoft® SQL Server was 8192 characters, it required some newer techniques to extract 1000 pages of XML text from the system. During the engineering stage, we were able to obtain the full XML information through a Microsoft® Internet Information Services link, that allowed us to manually combine the information using Altova XMLspy, an XML editor. But during the automated, on-line operation, our best application of the available technology involved using Java Database Connectivity software, JDBC, to get the bulky information, and supply it to the next level in the workflow. This allowed a fully-automated method to get to the database information, and supply it to the discrete simulator.

As a separate issue, the generation of suitable XML from the Microsoft® SQL Server proved to be one of the hardest issues to overcome. We had come up with an XML schema before trying to make the database accommodate the schema, and even so, due to the complexity of information to be transmitted, it would not have been easy to use the 'standard' XML format that the database produces. Instead, we had to go to the SQL Server 'Explicit XML' format, to create XML text that matched the pre-existing schema for data transfer. This is a rather difficult, but accomplishable, process, and it enabled us to match the schema that the discrete simulator parser was looking for.

2.3 XML Schema

The XML schema proved to be an enabling technology for the creation of the application. Since XML lets you provide hierarchical data separation, we took advantage of this capability to supply detail for the models. An example of the schema is shown in Figure 1. The hierarchical nature of the data is seen by inspection.

There are multiple 'Configuration Class' entities that each describe different portions of the underlying system to be simulated. Each entity may contain differently formatted hierarchies of data. This is quite a different mechanism than would be found in a relational database. Relational databases organize the data in tables, which supplies a 'breadth' view of the information. Instead, XML, as well as object-oriented programming, organizes the data in a hierarchical, 'depth'-type format. This was the primary hurdle to overcome in the creation of an XML document to represent the information in the relational database. It required making a huge table, called a Universal Table, full of a high number of null locations, in order to contain the 'depth'-type information in a 'breadth'-type format.

Delving deeper into the schema, it is seen that in Configuration Class 3, there are models available. There may be thousands of models at this level. Each module contains the information that the discrete event simulator needs in order to configure itself to represent the individual module correctly.

An example of XML code that would be generated from this schema is shown below. In actual XML code, there would be much data below the Configuration Class headings, and likely many models to be addressed.

```
<Top Level System >
  <Configuration Class 1/>
  <Configuration Class 2/>
  <Configuration Class 3>
    <Model>
      <Model Identifier>14</Model Identifier>
        <Model Name>Your Model</Model Name>
        <Model Parameters>
           <Model Parameter 1/>
           <Model Parameter 2>
             <Setting 1>6</Setting 1>
             <Setting 2>5</Setting 2>
             <Duration>20</Duration>
             <Details>anon</Details>
           </Model Parameter 2>
         </Model Parameters>
       </Model>
     </Configuration Class 3>
   <Configuration Class 4/>
 </Top Level System>
```

2.4 Content of Database Tables

The data and the organization of the underlying database, in our application, was previously established by subject matter experts. It was established for another purpose, but contained the details we needed. Therefore, we had to supply some method to interpret the information, and make it amenable to the discrete event simulator. In a simulation system that was created all at one time, it would be helpful to organize the data so that it would be easier to transform it into a form that would be readily acceptable by the simulator. As it was, the table organization, and the data content, was such that we had to create a secondary set of data tables to use in the creation of the XML that would be directly read by the simulator.

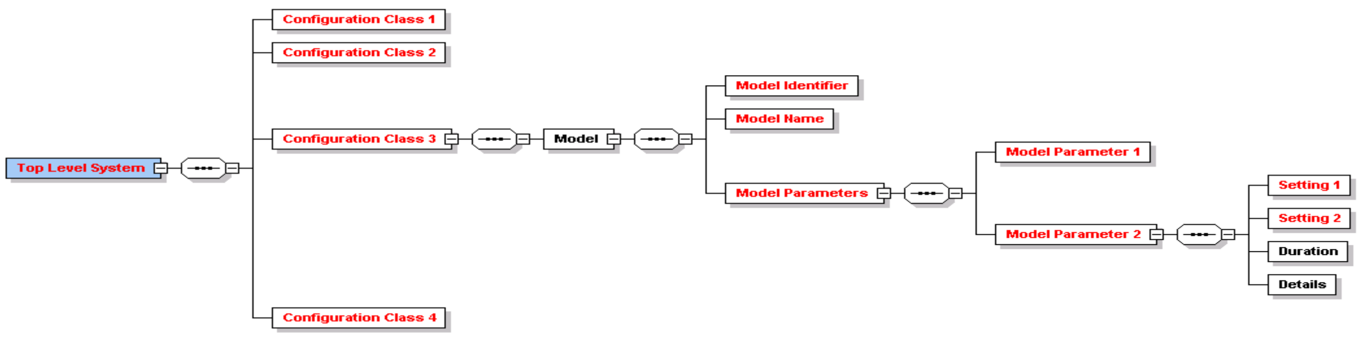

Figure 1: An Example of an XML Schema for Data-Driven Model Information

This was accomplished through the use of stored procedures, that could be activated by the workflow engine we used. This resulted in tables that were more-easily parsed by the XML-generation utilities that were part of the database. It was important to allow one record for each of the underlying XML syntax elements, and to supply suitable sorting mechanisms to ensure that the information was output in the correct locations in the resulting file to support the XML Explicit mechanism.

3 TECHNOLOGIES USED

Modern software technology was used to support the passing of data fluidly throughout the entire application, starting from the requirements, through to the databases, and finally to the discrete event simulation. These technologies were chosen for their generic properties and highly extensible usages, so as to allow for the additions to, or subtractions from, the overall application system with minimal extra development.

3.1 JDBC

Since the workflow engine used the Java programming language, having a way for Java to interface with the database proved to be very helpful. The database supported a Java Database Connection, JDBC, system, which allowed Java code to link to the database, perform queries, and get results. The results obtained through this method avoided the 8192 character SQL Server limit on record size, and we were able to get 1000-page XML files.

3.2 Stored Procedures

Stored procedures are a product of the Microsoft® SQL Server environment. They allow batches of Transact-SQL code to be run, controlling the process of making the XML from the source data. The workflow engine was able to cause these stored procedures to run, using the JDBC connection.

3.3 Web Server Integration

This interface allowed the XML data to be obtainable from a PC workstation. Again, the 8192 character limit on record size, caused the returned XML to have just 8192 characters, when using the ordinary database query tools. This problem was overcome with the JDBC interface, but for quick checks, and manual creation of the interface file, the database would output the xml results through a web server. Then the file would appear on your web browser, and you would just copy it out.

3.4 Why Use XML?

XML provides a great improvement over binary or textual information, in that it indicates a hierarchical relationship between all the data elements. Parsers can be written using parser standards such as DOM or SAX in all modern languages such as C++ and Java. Many desktop tools and applications are now being written with Import / Export options to write out, and accept, XML. Therefore, much support was available, with more on the way. It seemed like a good technology to invest in.

3.5 Generic Properties of XML

XML describes its content in terms of what the data is that is being described. For example, an <AuthName> parent tag can indicate that the data following it is the author's name, and an <AuthAffil> as a child tag can indicate the affiliation of same author. This type of definition allows an XML file to be processed purely as data by an application as well as being organized and viewed via most Internet Browsers. XML is called 'extensible' since, unlike HTML, the markup symbols are unlimited, customizable, and self defining.

4 INTERNAL SYSTEM OPERATION

We used a workflow engine as a collection of related tasks that are organized into, and associated with, a collaborated set of resources or assets. It controlled the execution of the business process we designed by triggering, through application automation, all tools and processes in the overall application. This methodology not only triggers one application after another in order, but also includes using the output of one sequential application as the input to another. Figure 2 illustrates the overall system.

The workflow process can start with a Java GUI that allows for certain custom-built choices to be selected. These choices can typically be selections of whether or not to have the internal simulation output be optimized, to allow for what-if scenarios to be run, or to view runtime simulation animation.

The first application in the workflow process could be a requirements analysis tool to enter certain custom data into the SQL Server databases. Once the data was populated within the databases the workflow would trigger stored procedures, Explicit XML and conversion technologies to create the complete XML file. The workflow engine would then write the XML file to disk to be used as an input to the discrete event simulator. During and after the simulation runs, the discrete event simulator writes student logs, Microsoft® Excel plots and graphs, and web deployable reports, as outputs. After the simulation runs to completion these outputs can be presented visually by having the workflow engine automatically bring up these various applications. Also, if the selection was made in the beginning of the workflow run to allow for what-if scenarios, the workflow would then present the user with more Java GUIs. This allows the user to make changes to the database entries or the XML directly, an then perform a new full simulation run.

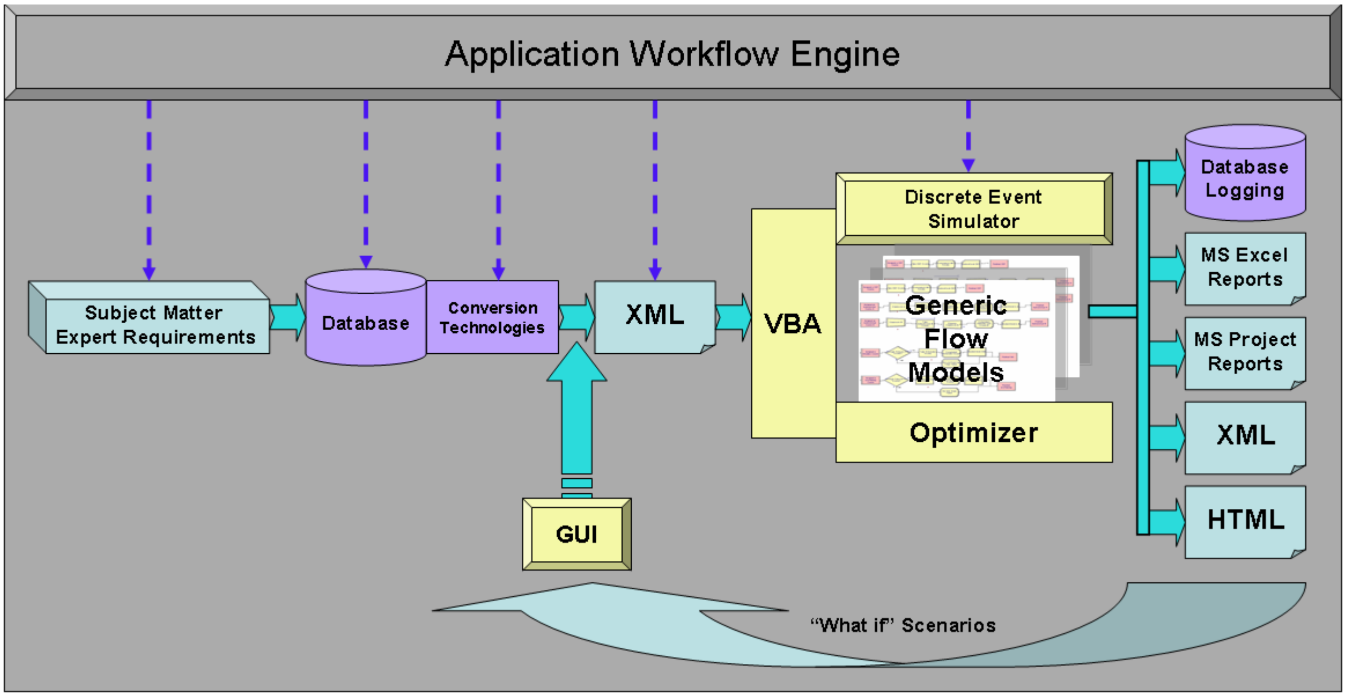

Figure 2: Overall System Application Superset

4.1 Workflow Engine

The WfMOpen workflow engine was used within the application superset, which is based on WfMC; an GNU Licensed Open Source standard. WfMOpen also implements XPDL, or XML Process Definition Language; which is inclusive in the same standard. XPDL provides a framework for implementing business process management and workflow engines.

4.2 XML Generation

Two methods were used to create the XML files. One method involved hand creation of the XML files directly from the schema, using XMLspy tools. The XML created in this manner would have the proper form, but need to have the data put in, to be useful. This method was helpful early on in the development, and provided a way to quickly get XML text to experiment with.

The online, automatic creation of the XML file used a technology inside Microsoft SQL Server called XML Explicit to create an intermediate table, and output the table as XML text. This method required cryptic, and tricky coding of it's instructions, but was the most powerful and flexible method.

4.3 XML to VBA

Visual Basic, Applications Edition, or VBA, is a Microsoft® derivative of the Visual Basic programming language that allows its host application to interoperate with the Microsoft® Windows Platform Environment. VBA was used to create a generic, customizable XML parser via the MSXML parsing framework. Using Visual Basic coding standards, object classes and data structures were created, in order to map, and hold within memory, all of the pertinent data from the XML file. Typically, parent tags within the XML are mapped to Visual Basic class names, and the children tags are mapped to class member variables. These data structures were not only used for reference, but also used to populate all applicable variables and modeling object constructs within the Discrete Event Simulation. The organization of these data structures is shown in Figure 3, which is a Unified Markup Language class diagram that implements the schema of Figure 1.

4.4 Discrete Event Simulator

Discrete Event Simulation concerns the modeling of a system as it evolves over time by representing any changes within the system as separate events. Bank, et al (2001) provides more information about discrete event simulation. Arena® was used due to its versatile VBA standard interface and graphical nature for building small to large scale models.

The user interface is a simple drag and drop mechanism for creating and populating all 'modules' or 'blocks' within the model. The model consists of one or multiple process flows much like that of a flow-chart. As the entity, or actor, moves through the flow it will enter model object constructs such as 1-to-many decisions, processes where resources can be seized and utilized, delays consisting of

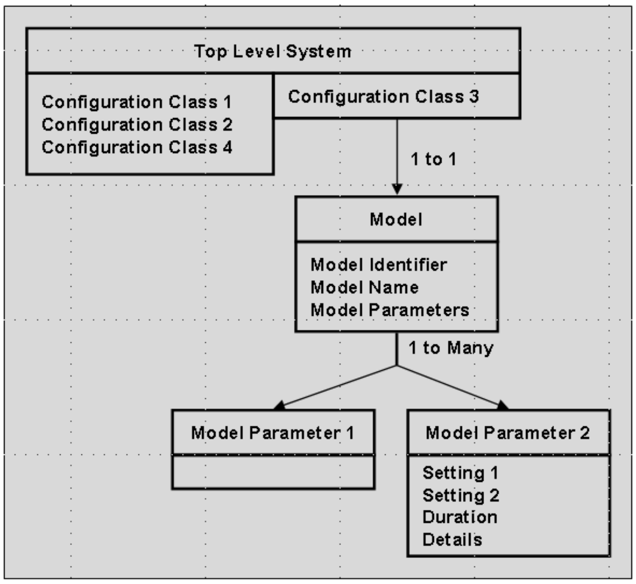

Figure 3: UML Class Diagram of Model Schema

both static digits and random statistical distributions, and output modules where numerical data can be captured and written out to disk in many different mediums.

Because the XML data being is populated into VBA classes and subclasses, all the data is within the discrete event simulation's memory. During runtime, but before the simulation begins, the generic process model can be modified using the aforementioned data. Resources can be given specific names, costing data, and capacities. Internal variables, expressions and process delays can be populated with static numerical data, or random statistical distributions that are specific to the given dataset. Within our example, an entire specific curriculum with a plethora of course, lesson and student data was populated all within the generic discrete event simulation model turning it into a runtime specific model.

Due to the VBA coding interface the outputs can be in the form of Microsoft® Excel spreadsheets using the Microsoft® standard ODBC object. By having the discrete event simulation write to the same area in an Excel spreadsheet between simulation runs, dynamic graphs and plots can be fully realized. The outputs can also be written out to an XML schema for use within other applications, or to be deployed over the web. HTML can be written to show the output data in a standard web format for viewing over the web. Standard comma delimited data can also be written out to be utilized within other applications such as Microsoft® Project for showing the progression and traceability of events.

5 SIMULATION OF AN ENTERPRISE TRAINING SYSTEM

We created a generic model of an enterprise training system. An enterprise training system could be viewed as a schoolhouse, having students, lessons, syllabi, and resources. We investigated helicopter training. The students would come to the school daily, and take the lessons that were specified by the syllabus for the course they were in. We simulated a detailed syllabus that was created by Instructional System Designers for an actual enterprise training system. Students would take lessons an entire specific curriculum with a plethora of course, lesson and student data was populated all within the generic discrete event simulation model turning it into a runtime specific model.

Different types of lessons were modeled, such as instructor-led, computer-based-training, and flying lessons. Each type of lesson had its own set of control variables that were set using data from the database. These variables would include what resources were required, and how long each section of the lesson takes, such as time to drive to the airfield, and how long to fly.

CONCLUSIONS

While these data-driven modeling techniques were developed primarily for a given application, they have general applicability for a large class of simulations, where many detailed elements must be configured, and potentially changed. Using modern software practices, and tools, allowed the effort to be completed without being limited by the tools. As it was, we had to reach out onto the some of the newest branches of the application capabilities, such as the automatic XML generation, and VBA simulator control. And, once the choice for use of these newer technologies was made, it set the minimum capability level for all the other tools that were used in the system. Having the workflow engine enabled all the various simulator components, including the database, to interoperate. This framework, and modeling paradigm can be used for many different applications, especially where extensive and detailed information is involved.

ACKNOWLEDGMENTS

Thanks go to the following Lockheed Martin individuals for their efforts in creating the system described in the text: Lorie Ingraham, Vivian Wong, Yushan Chen, Bruce Castle, Fred Madsen, Ed Reed, Richard Brent, Bruce Eddy and Mark Falash.

REFERENCES

Banks, J., Carson, J.S, II, Nelson, B.L, Nicol, D. M. 2001. *Discrete-Event System Simulation, Third Edition.* Upper Saddle River, NJ.: Prentice-Hall.

Šunderić, D. 2003. *SQL Server 2000 Stored Procedure & XML Programming, Second Edition.* New York, NY: McGraw-Hill/Osbourne.

AUTHOR BIOGRAPHIES

GREGORY A. HARRISON is a research and development engineer at Lockheed Martin Simulation, Training & Support in Orlando, Florida. He investigates modeling and simulation technologies, and was actively involved in the automation of discrete simulations. He received his Ph.D. in Computer and Electrical Engineering from the University of Florida. His research includes large-scale simulations, coordinate systems, digital signal processing, real-time computing, robotics, mathematical modeling, and artificial intelligence techniques such as neural networks, genetic algorithms and intelligent agents. He is a member of the Graduate Faculty of the Florida Institute of Technology. Dr. Harrison is the author of numerous papers, with two patents in artificial intelligence techniques. He is a member of IEEE. His e-mail address is `<gregory.a.harrison@lmco.com>`.

DAVID S. MAYNARD is a Software Engineer with Lockheed Martin Simulation, Training & Support within the Independent Research and Development department. He received his B.S. in Computer Science with a Minor is mathematics from Florida State University (2001) and is working on his M.S. in Computer Science from the Florida Institute of Technology with a emphasis in Artificial Intelligence using genetic algorithms. He has worked for several years in the Open-Source Commercial Video Game Modification community. His current research interests include the use of commercial video game technologies in Military / Special Forces Training Systems and the use of genetic algorithms to create a more realistic video game opponent. His e-mail address is `<david.s.maynard@lmco.com>`.

EYTAN POLLAK is a Research and Development Manager for Lockheed Martin Simulation, Training & Support in Orlando, Florida. He has published papers dealing with distributed simulation, embedded simulation, reconfigurable simulators, and common architecture for simulation systems. Dr. Pollak serves as adjunct professor of Electrical Engineering at the University of Central Florida. He received his Ph.D. from Purdue University. His e-mail address is `<eytan.pollak@lmco.com>`.