Automating Human Performance Measurement using XSLT in Simulation-Based Exercises

Glen Cornell
General Dynamics Land Systems
Sterling Heights, MI
cornell@gdls.com

Wiley Boland, PhD
General Dynamics C4 Systems
Orlando, FL
wiley.boland@gdc4s.com

Geoffrey Frank, PhD
RTI International
Raleigh, NC
gaf@rti.org

ABSTRACT

The real-time collection and filtering of the enormous amounts of data resulting from training simulations involving hundreds of entities is a challenge for training system architects. Traditional approaches have relied on human observer/trainers (O/T) to tag key events and prepare the After Action Reviews (AAR), which identified what happened (particularly events contributing to metrics for mission success), why metrics were not met (precursor events that contributed to the metric events), how the critical sequences of events arose (identifying decision points), and provide timely learning points. For collective training simulations, the O/Ts are often overwhelmed in terms of tracking individual behaviors and skills. Automated approaches for capturing human performance data are a preferred method that can provide feedback to each member of a team or unit.

This paper describes the application of a novel use of XML Stylesheet Language Transformations to process simulation event stream using an Event-Condition-Action (ECA) rule engine. The human performance data capture used the following process:

1. **Data Collection** from multiple sources, including IEEE 1278 Distributed Interactive Simulation (DIS) data and other network data.
2. **Protocol Filtering**, which is done to reduce the processing workload in the later stages of the pipeline.
3. **XML Conversion**, where the filtered data streams are converted to a neutral XML format that allows processing using ECA rules.
4. **Event Detection**, which is done using XSLT to extract key events.
5. **Event Processing**, which is done to relate selected event sequences to human performance standards.

The Embedded Training Group at General Dynamics Land Systems has applied this approach to measure learner performance in distributed interactive simulations, including driver training and gunnery training. Ongoing work is related to using competency standards to specify the ECA rules and generate the appropriate XSLT.

ABOUT THE AUTHORS

Glen Cornell is a Software Engineer at General Dynamics Land Systems Division in Sterling Heights, Michigan. Mr. Cornell is presently managing the training content development team in support of the Manned Ground Vehicles in the Future Combat Systems program. Mr. Cornell holds a Bachelor of Science from the University of Central Florida.

Dr. Geoffrey Frank is the Principal Scientist of the Technology Assisted Learning Center at RTI International. He has a PhD from the University of North Carolina at Chapel Hill. He is a member of the IEEE Learning Technology Standards Committee. Dr. Frank has led the training analysis of web-delivered simulations for the Army Signal Center.

Dr. Wiley N. Boland, Jr. is the Training and Supportability Manager within General Dynamics' C4S Group in Orlando, Florida. He has more than thirty years experience in instructional systems design and learning products. He holds an educational doctorate from Virginia Tech and is a retired Marine aviator.

Automating Human Performance Measurement using XSLT in Simulation-Based Exercises

Glen Cornell
General Dynamics Land Systems
Sterling Heights, MI
cornell@gdls.com

Wiley Boland
General Dynamics C4 Systems
Orlando, FL
wiley.boland@gdc4s.com

Geoffrey Frank
RTI International
Raleigh, NC
gaf@rti.org

INTRODUCTION

A novel use of XML Stylesheet Language Transformations (XSLT) has been suggested as the basis of an event-condition-action (ECA) rule engine (Boglaev, 2003). The Embedded Training Group at General Dynamics Land Systems has applied this approach to measure learner performance in distributed interactive simulations. This paper describes the technique, the performance analysis of the approach taken, and the feasibility of the technique towards large-scale, collective exercises as well as commander-, crew-, operator-, and maintenance-based training.

Performance Measurement for Virtual Training Exercises

Today's collective virtual training exercises consist of much more than engaging a virtual enemy on a virtual battlefield. These exercises model Asymmetric Warfare, Improvised Explosive Devices (IED), and Digital Networks. Virtual simulations for these exercises require real-time collection and filtering of the enormous amounts of data which result from modeling hundreds of entities. Data collection and management shall continue to grow as a challenge for training system architects.

An essential element of any collective training exercise is an After Action Review (AAR). Traditional approaches for preparing an AAR relies on human observers tagging key events and preparing the AAR presentations, which identify what happened (particularly events contributing to metrics for mission success), why metrics were not met (precursor events that contributed to the metric events), and how the critical sequences of events arose (identifying decision points). For collective training simulations, the quantity of individual behaviors and skills often overwhelm observers. Consequently, individualized learning diminishes in such situations. Automated approaches for capturing human performance data are a preferred method that can provide feedback to each member of a team or unit to either supplant or augment an instructor in-the-loop.

This paper proposes a generalized technique for automating human performance assessment, which is a fundamental need in the area of "simulation-based" virtual training. This paper evaluates the technique with a description of two scenarios which are meant to verify, exercise, and refine the technique presented here.

THE PERFORMANCE MEASUREMENT PIPELINE

This paper describes the adaptation of an automated human performance measurement technique. The technique applies to many simulation-based exercises including weapons, combined arms, battle staff and maintenance training. A software architecture resembling the Unix text processing pipeline was chosen to accommodate the large amounts of data transmitted over several heterogeneous protocols. The pipeline architecture, as depicted in Figure 1, collects and filters data, converts it to XML, detects events, and then processes the events in real-time.

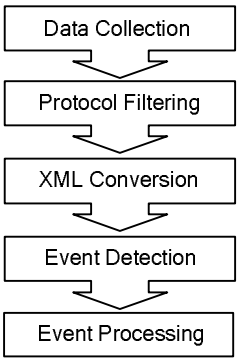

Figure 1. The Performance Measurement Pipeline

Stage 1: Data Collection

In the first stage of the performance measurement pipeline, the data collection software determines which information is collected, which device accesses the information, and how the information is obtained.

Human performance measures reveal themselves in the instructional system design process and then are mapped to data sources in the learning environment. Once mapped, appropriate data collectors are then employed to extract the information in the exercise. For example, in a conduct of fire exercise, some of the performance measures are as follows:

- The crew communicates according to doctrine
- Proper battle command messages are sent
- The crew properly identifies, acquires, destroys and senses the threat.

In this example, information sources are identified as the intercom system, the vehicle data bus (MIL-STD-1553B), the tactical battle command network (FBCB2), and the simulation network (DIS/HLA). Because these information sources are transmitted over separate physical transports, multiple data collectors are employed.

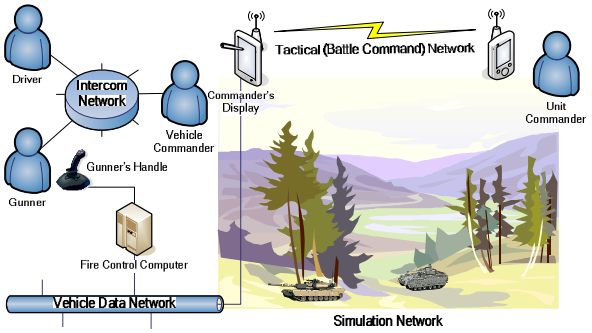

Figure 2. Information Sources for Data Collection in Weapons Training

Stage 2: Protocol Filtering

Once the raw data is obtained from the proper network device, filters are placed in the pipeline to eliminate all but the pertinent information from the incoming data stream. Experience has shown that protocol filtering is a necessary step, especially for large scale distributed simulations. Filters have proven necessary to reduce the processing workload in the later stages of the pipeline.

Stage 3: XML Conversion

In order for data to be used by the XSLT processor, it must first be represented in XML. The sole purpose of this stage of the performance measurement pipeline is to normalize the data for event detection using the technique described by Boglaev (2003). Two methods of XML conversion were experimented with. The first method is to represent all network data with a common XML schema. Another approach is through the use of XML schemas which characterize specific network protocols in XML form. As will be described later in this paper, each method affects the development of training support packages.

Stage 4: Event Detection

An event, in this context, is defined as a discrete, measurable occurrence. Events effect both exercise behavior and performance assessment. In an Event-Condition-Action (ECA) rule engine, an event triggers a Boolean condition to be evaluated. When the condition is satisfied, the resulting action is taken. This design idiom is represented in the following form (Bailey 2002):

```
when event
if   condition
then action
```

This ECA technique described by Boglaev makes use of the XML stylesheet language transformation to represent business logic rules in e-commerce applications (Boglaev 2005). Because of the widespread applicability of the ECA idiom, the same technique is applied here to automate human performance measurement. This technique takes advantage of the logic processing capabilities of XSLT to transform one XML format to another. In this case, the source data is supplied by the previous three stages in the pipeline. The source data format may vary from one exercise to another and is defined by the XML schemas supplied to the XSLT processor. Conversely, the invariant output format defines a common interface to the last stage of the pipeline and is discussed in the next section. Simply put, however, the XLST processor dynamically constructs remote method invocations to the event processing stage of the automated performance measurement pipeline. This remote method invocation may come in the form of the Object Management Group's (OMG) Interface Definition Language (IDL), Web Services Design Language (WSDL), XML-RPC, or other similar technologies.

Stage 5: Event Processing

Event processing concludes the performance measurement pipeline. Events detected in previous stages are used here for two purposes. First, events are used to help form an evaluation of the learner's performance. For example, the time that a fault was detected (one

event) to the time that the learner repaired or replaced the faulty component (another event) may be used in the assessment of a maintenance training exercise. The second use of the event processing stage is to shape the behavior of the exercise. Exercise behavior is predicated upon events that occur dynamically. The resulting actions are to provide evaluative feedback, corrective feedback, changes in subsequent human performance measures to apply, and changes in input conditions presented to the user.

APPLICATIONS OF THE MEASUREMENT PIPELINE

Two simulation-based training vignettes were developed to verify, exercise and refine the performance measurement pipeline. The purpose of these vignettes was not to instruct, but rather to serve as a test case of the automated performance measurement pipeline. Each scenario began with a storyboard developed by subject matter experts and instructional system designers. The instructional system design process would normally require a training task analysis to be performed prior to identification of the task to be trained and the proper method of delivery of the instruction to the learner. In this case, however, the training task analysis data was obtained from previous work captured in the Automated System Approach to Training (ASAT) database.

The initial task was to provide storyboards for Training Support Packages (TSPs). The storyboards are based upon task analysis data from military occupational specialties (MOS), Soldier Training Publications (STP), Army Training Evaluation Program (ARTEP) Mission Training Plans (MTP), and Army training doctrine based upon the TRADOC Regulation 350-70 and associated Pamphlets.

Storyboard development began with a textual description of the scenario and tasks involved in each TSP. Common interactive multimedia instruction (IMI) storyboard templates, Army Warrior TSP (WTEP), and the Close Combat Tactical Trainer (CCTT) TSPs were adapted into a storyboard for the simulation-based training exercises.

Each storyboard was not only used by the instructional systems design team to create the training content, but also by systems engineers to create requirements for human performance measures in the simulation. Systems engineers reference the storyboards when developing models of the system in the Unified Modeling Language (UML) (OMG 2003). The system modeling process results in a clear understanding of the specific human performance measurements and how the measurements will be implemented in software.

Scenario 1: Driver Trainer Exercise

The objective of the first experiment was to demonstrate the performance measurement pipeline.

Description

The first experiment using the performance measurement pipeline described in this paper typifies self-paced, initial operator training. The lesson begins with a presentation of multimedia instruction describing the proper technique of driving a tracked combat vehicle when traversing a trench. The learner is then immersed in a simulation that allows him to practice the technique, thereby reinforcing the instruction. Because the exercise is intended for the novice driver, audio and visual cues lead the learner through his task and provide him with immediate, corrective feedback. Feedback and cues are the resulting actions of the performance measurement pipeline described above. The UML state charts in Figure 3 through Figure 6 depict the states in the scenario.

Protocols

Being the simplest example, the only protocol used in this exercise is the IEEE Std. 1278.1A-1998 Distributed Interactive Simulation (DIS) protocol (IEEE 1998).

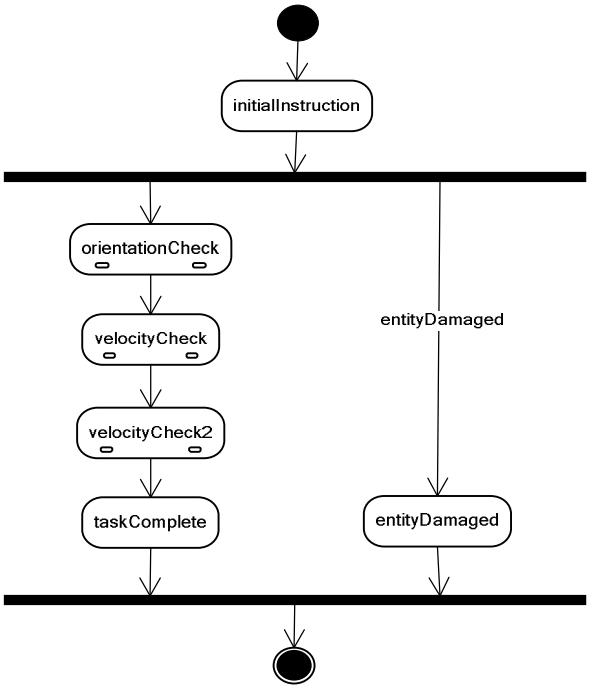

Figure 3. State Chart of Driver Training Scenario

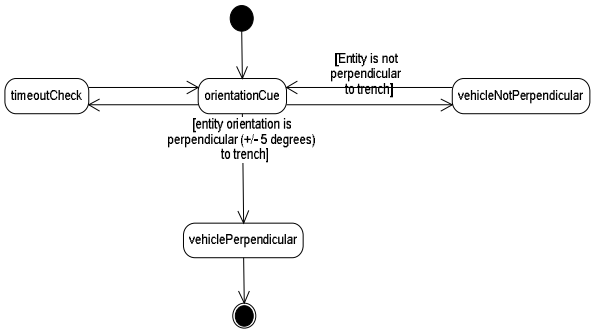

Figure 4. orientationCheck Composite State

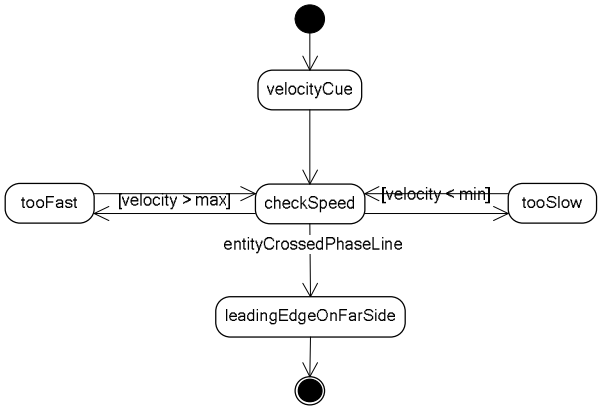

Figure 5. velocityCheck Composite State

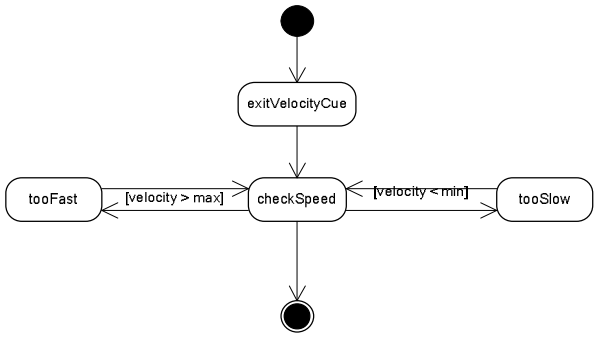

Figure 6. velocityCheck2 Composite State

Human Performance Measures

Modeling the scenario in UML identified the following Human Performance Measures (HPM).

Many of the HPMs described below do not directly measure the performance of the learner. Instead, some of these HPMs detect the effects of the learner's intent or actions, which may be difficult, if not impossible, to measure. For example, a performance measure for a hypothetical exercise may be that the user maintains control of the vehicle throughout the exercise. Although it may be difficult to directly measure the learner's ability to maintain control, the measure may be composed of a collection of smaller, more discrete measurements such as the vehicle was not damaged, the vehicle did not exceed a certain velocity threshold, and the vehicle stayed within the exercise boundary.

Entity Crossed Phase Line

This HPM is used in determining when an entity crosses a phase line. The entity identifier, the phase line and the normal are parameters to this event detector. The entity identifier, the simulation time, and the phase line identifier are passed as parameters when the event is detected. A phase line is defined here as a bounded rectangle with two sides tangential to the ground plane. The rectangle's normal vector describes the direction that an entity must pass through in order for the event to be detected. The normal vector is optional. If the normal vector is absent, then the event is detected whenever an entity completely passes through the bounded plane in either direction. The application, site and entity id is passed in as an optional parameter. If the entity id parameter is present, the event detector will report when only the given entity crosses the phase line. Conversely, the event detector will report all entities that cross the phase line if the entity id parameter is not present.

Orientation Threshold

This HPM detects when the vehicle is oriented at a specified number of degrees from true North. The entity identifier, the orientation and a tolerance are passed as parameters to this event detector. The entity identifier, the simulation time, and the vehicle orientation are passed as parameters when the event is detected.

Entity Exceeds a Given Velocity Threshold

This HPM detects when an entity's speed is greater than or equal to a specified velocity. The entity identifier and the velocity threshold are passed as parameters to the event detector. The entity identifier and the simulation time are passed as parameters when the event is detected.

Entity Falls below a Given Velocity Threshold

This HPM detects when an entity's speed is less than or equal to a specified velocity. The entity identifier and the velocity threshold are passed as parameters to the event detector. The entity identifier and the simulation time are passed as parameters when the event is detected.

Timeout Expires

This HPM detects when the specified simulation timeout expires.

Entity Damaged

This HPM detects when the given entity is damaged. In this scenario, vehicle damage and mobility kills are determined by the virtual vehicle's real-time dynamics calculations.

Observations

As a result of developing the simulation system and executing the scenario, several observations are worthy of reporting. Observations were made in each stage of the performance measurement pipeline and are discussed below.

Data Collection, Protocol Filtering, and XML Conversion

In this experiment, a custom application was written which combines the data collection, protocol filtering and XML conversion stages of the performance measurement pipeline. When the experiment began, the performance measurement pipeline was originally conceived as one step. However, it became evident that this application was performing three functions. As a result, the pipeline was expanded to the five stages presented in this paper.

Event Detection

An objective of the ECA technique presented here is to use an unmodified XSLT processor to detect events. Therefore, care was taken to create transformations entirely in XSLT 2.0. As was described earlier, the destination format of the XSLT transformation is a remote method invocation. In this experiment, the XML-RPC specification was chosen for its simplicity. An example that transforms the damaged bits on an entity state DIS PDU into a properly formatted XML-RPC call is graphically represented using Altova's MapForce XSLT visual data mapping tool in Figure 7.

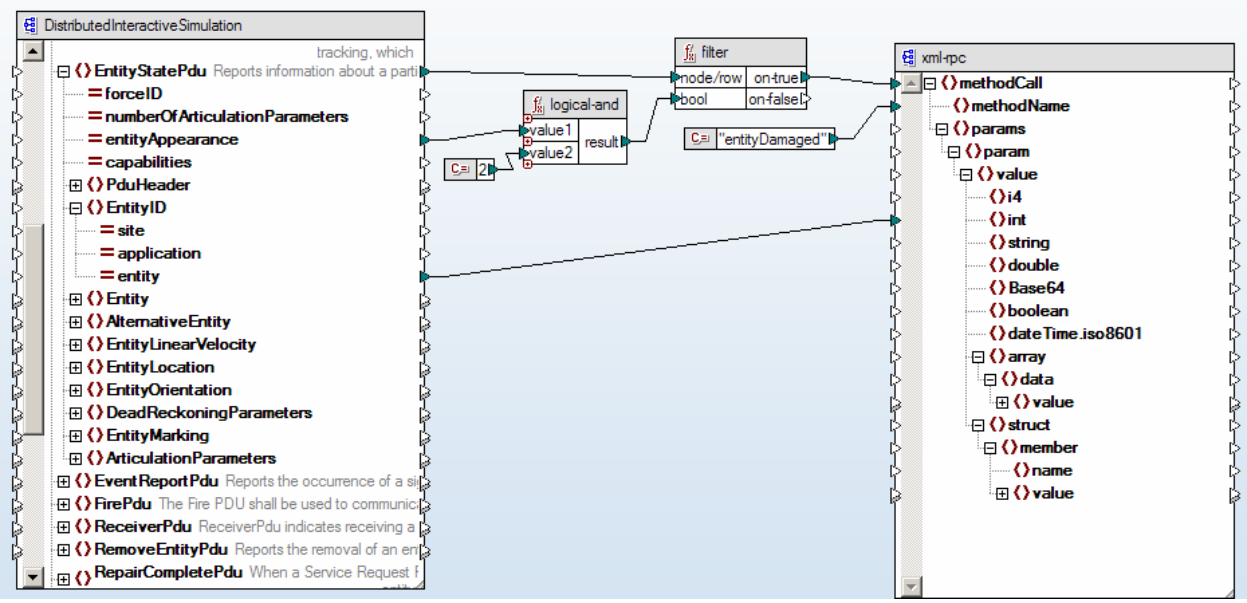

Figure 7. entityDamaged XSLT Mapping

As was stated earlier, the only protocol used in the event detection stage of this exercise was DIS. The custom collector, filter and conversion tool only evaluated the Entity State Protocol Data Unit (PDU). The XML schema describing the DIS protocol is part of the DIS-XML project developed at the Naval Postgraduate School as part of a larger project called the Extensible Modeling and Simulation Framework (XMSF 2004).

Event Processing

The state chart depicted in Figure 3 was implemented in the ECMA-262 (JavaScript) scripting language with the SpiderMonkey interpreter. The JavaScript code for this exercise consists of a number of event handlers mapped to XML-RPC calls. Each XML-RPC call is dynamically constructed by the event detection engine (stages 1-4). Upon the receipt of an XML-RPC call, the event processor dispatches the JavaScript event handler. The event handlers generally provide

instruction and evaluative feedback to the learner, as well as form a final assessment.

The storyboard for this scenario called for immediate, evaluative feedback to the learner. In each of the composite states identified in Figure 3, the event processor provided multimedia instruction to the learner, configured the simulation for the learner to practice, and provided multimedia feedback to the learner when either the learner performed the task correctly, performed the task incorrectly, or a timeout expired. The assessment was determined when the vehicle traversed the ditch properly. Accurate vehicle dynamics prevented the learner to traverse the ditch in any other way than was prescribed. Any failure would damage the vehicle.

Lessons Learned from the Driver Training Example

Several key learning points were noted from the results of the first experiment. First, the experiment demonstrated to a satisfactory level that XSLT may be used as an ECA rule language. Second, user-defined functions embedded in the XSLT processor reduced the complexity of the transformations dramatically. Finally, the loosely-structured event handling JavaScript application code requires a more formal implementation which would be able to enforce the structure of the state machine in Figure 3.

Scenario 2: Gunnery Trainer Exercise

The primary objective of the second experiment was to refine the performance measurement pipeline. Secondly, the technique was extended to detect events on protocols other than IEEE 1278.1-1995 (DIS).

Description

The scenario used in this experiment typifies a gunnery training exercise. This exercise is intended to be for a tank crew who has received initial, individual operator training. The scenario begins with a warning order sent by the unit commander to briefly describe the mission. As the crew prepares for the mission, the vehicle commander receives an operational order (OPORD) describing the mission is greater detail. The crew makes their final preparations and waits for the threat to enter the area of interest. As the threat exposes itself, the crew performs a proper conduct of fire to identify, acquire, destroy and sense the target according to Army doctrine. The scenario is depicted in the UML sequence diagram in Figure 8.

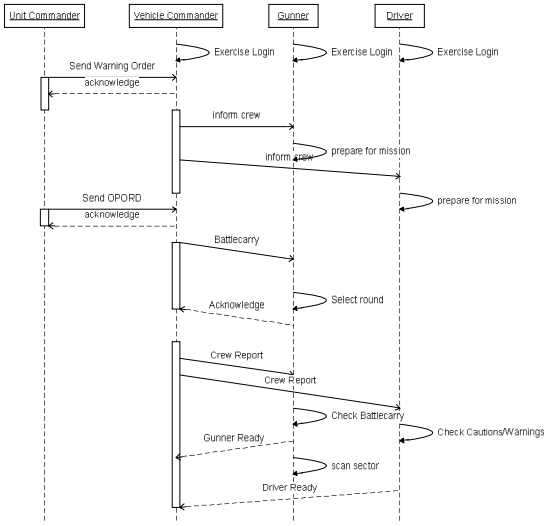

Figure 8. Sequence Diagram of Gunnery Scenario

Human Performance Measures

As in the previous experiment, the training support package scenario description was modeled in UML. From the UML model, the performance measures identified in Table 1 were determined to be required for this exercise and categorized by protocol.

Table 1. Scenario 2 Human Performance Measures Organized by Protocol Type

DIS	Battle Command Message	Intercom
Entity Becomes Active	Warning Order Sent	Phrase is Spoken
Entity Becomes Inactive	OPORD Sent	
Entity Is Destroyed	Situation Report Sent	
Munitions Fired		
Munitions Detonated		
Target Designated		

Observations

The objectives of this experiment were to refine the technique and to extend the performance measurement pipeline to other protocols.

As was previously mentioned, the data collection, protocol filtering and XML conversion stages were originally envisioned as one discrete step. In this experiment, however, the steps were identified as unique and an attempt to separate them was made

wherever practical. As a result, an oversimplified implementation of the refined performance measurement pipeline, which is the basis for the ECA technique as it exists in this paper, may be described by the Unix-like command in Figure 9. Wireshark performs the first four stages of the pipeline. Wireshark uses the Packet Capture (pcap) Library for execution of the data collection stage. The "-f" (filter) option to wireshark serves as the filter stage in the pipeline. The XML conversion stage in the pipeline is characterized by the "-T pdml" option. This option displays network data in the Packet Data Markup Language (PDML 2006). PDML provides a universal XML schema for most common protocols on the network device. The XSL stylesheet implements the event detection stage. The output of the XML stylesheet is a properly formatted XML-RPC method invocation to be sent by the Unix application curl to transmit data over an HTTP POST request to the event processor.

```
tshark -f "filter" -T pdml | \
        xsltproc stylesheet | \
        curl --header \
        "Content-Type: text/xml" \
        --data @- http://www.foo.com/RPC2/
```

Figure 9. Performance Measurement Pipeline Example

Data Collection

In this experiment, a more general approach to data collection was taken. This approach employs standard network protocol analyzer tools without the need to create custom "protocol dissectors", as was performed in the previous exercise.

This wireshark network packet analyzer uses pcap for data collection. Pcap was a practical solution because it already is able to passively view all network traffic on a variety of network devices. The wireshark application was configured to capture and analyze DIS traffic in this experiment.

Protocol Filtering

The pcap packet capture library also includes a robust packet filtering capability that reduces the amount of data to be analyzed. The packet filtering language is well documented and flexible for general purpose use. The pcap filter was constructed to allow only DIS Entity State PDUs to subsequent stages in the pipeline.

XML Conversion

Wireshark has an additional capability to represent network traffic in the in the Packet Data Markup Language (PDML). PDML is an XML markup language describing network packets in a generalized fashion. Despite the general applicability of PDML, several protocols were not implemented due to their complexity. The team chose to implement only one (DIS) of the three protocols identified in Table 1.

XML conversion of the battle command network traffic was not implemented in this experiment. Rather, an instructor in the loop observed the network traffic using battle command network monitoring tools. When the instructor determined that the learner sent a properly formatted battle command message, the instructor manually sent the notification of the event to the event processor.

Voice recognition is another area which was not addressed in this experiment. Again, an instructor in the loop determined when doctrinally correct fire commands were issued. As fire commands were performed, the instructor manually sent the results of the commands as event notifications to the event processor.

Event Detection

Network packets described in XML are processed in this experiment with xsltproc, the next stage of the pipeline. This processor transforms XML data into any text stream as described by the XML style sheet language transformation. The destination XML schema for the XSLT processor is an XML-RPC remote procedure call to the event processor.

Event Processing

The event processor architecture improved in several ways. First, the event processor matured from an event dispatching loop to an event processing model similar to that as defined in the W3C DOM event model (W3C 2003). This approach greatly simplified the exercise JavaScript code. For example, each exercise executed by the first generation event processor was written in the following manner:

```
switch (getNextEvent()) {
case "Event1":
      …;
      break;
case "Event2":
      …;
      break;
}
```

The second generation event processor hides the event dispatch loop. The resulting exercise JavaScript resembles the following code fragment:

```
onEvent1() {
        …;
}

onEvent2() {
        …;
}
```

Additionally, the second generation event processor included a finite state machine (FSM) which not only simplified the JavaScript code even more, but also enforced structure on the exercise script. Originally, the state chart depicted in Figure 3 was implemented as a finite state machine (FSM) in JavaScript. The result of this implementation is a collection of commonly re-used state transition variables and methods intermixed with exercise-specific code. In the second generation event processor, the FSM code was implemented in the event processor. In this architecture, the remote method invocations from the even detection stage trigger state transitions in the FSM. As the system changes state, the event processor invokes the appropriate state transition event handlers in JavaScript.

Lessons Learned from the Gunnery Training Example

The objectives set forth at the beginning of the second experiment were to refine the technique and to use alternative protocols with the same ECA technique as defined in this paper.

CONCLUSION

The proposed technique for automated assessment is a viable solution for "simulation-based" training. However, further research is required to refine this technique into a more practical and manageable solution. Several key challenges for this approach are complexity, ubiquity, and emerging best practices.

Event detection XSL Transformations are non-trivial. Developing an XSLT file requires a solid understanding of the input protocols, their relationships to the assessment desired, and the XSLT programming language. To liberate the training support package author from this specific domain knowledge, computer-aided software tools will be needed.

Another challenge that this approach faces is that the XSL transformations for human performance measurement may not be used in all XSLT processors. Automated human performance measures of distributed interactive simulations require extensive use of domain-specific algorithms, as was shown in the examples above. Implementation of such algorithms in XSLT 2.0, albeit possible, exceeds the capabilities and real-time performance of a stock XSLT processor. To meet performance requirements and to simplify the transformations, the algorithms are best implemented in a custom-built XSLT processor as user-defined functions. The XSL transformations, although they conform to the XSLT 2.0 specification, invoke routines only available in a special-purpose XSLT processor. This constraint hinders the ubiquity of the technique.

When considering this technique for applicability beyond the examples presented here, one must also be aware of emerging standards and best practices in the areas of agent-based behavioral modeling, the rule markup language (RuleML 2007), and the human performance markup language (Stacey, W., et. al. 2005).

REFERENCES

Boglaev, Yuri. (2003), *A Design Pattern for a Rule Engine*, JavaPro on-line magazine. Retreived July 03, 2005 from http://www.ftponline.com/javapro/2003_08/online/xml_yboglaev_08_01_03/

Boglaev, Yuri. (2005), *Interchange of ECA Rules with Xpath expressions*, Procedings of the W3C Workshop on Rule Languages for Interoperability. Washington, D.C. 27-28 April 2005.

XMSF (2004). Extensible Modeling and Simulation Framework. Retrieved June 16, 2007 from https://www.movesinstitute.org/xmsf/xmsf.html

IEEE (1995). IEEE Std 1278.1-1995 IEEE Standard for Distributed Interactive Simulation. Retrieved February 22, 2005 from http://standards.ieee.org/reading/ieee/std_public/description/compsim/1278.1-1995_desc.html

Wireshark (2007). The Wireshark Network Protocol Analyzer Project. Retreived June 18, 2007 from http://www.wireshark.org/

OMG (2003). Unified Modeling Language (UML). Retrieved June 16, 2007 from http://www.omg.org

W3C (2003). XML Extensible Markup Language. Retrieved June 16, 2007 from http://www.w3.org/XML/

Stacy, W., Freeman J., Lackey, S., and Merket, D. (2004).*Enhancing Simulation-Based Training with Performance Measurement Objects*. Proceedings of

the Interservice/Industry Training, Simulation & Education Conference, Orlando, December, 2004.

Stacy, W., Merket, D., Freeman, J., Wiese, E., Jackson, C. (2005) *A Language for Rapidly Creating Performance Measures in Simulators*. Interservice/ Industry Training, Simulation and Education Conference, Orlando, December, 2005.

ECMA-262 (1999). Standard ECMA-262: ECMAScript Language Specification. Retrieved January 20, 2005 from http://www.ecma-international.org/publications/ standards/Ecma-262.htm

J. Bailey, A. Poulovassilis, and P.T. Wood (2002). *An Event Condition-Action Language for XML*. In Proceedings of WWW'2002, Hawaii, 2002.

RuleML (2007). The Rule Markup Language Specification. Retreived June 2, 2007 from http://www.ruleml.org/spec/

PDML (2006). Packet Data Markup Language. Retrieved August 2006 from http://analyzer.mirror.ethereal.com/30alpha/docs/dissectors/PDMLSpec.htm

XML-RPC (2003). The XML-RPC Specification. Retrieved June 2007 from http://www.xmlrpc.com/spec