16th CCRTS

"Collective C2 in Multinational Civil-Military Operations"

Title of Paper:
Supporting NATO C2-Simulation Experimentation with Scripted Web Services

Suggested Topics:
Topic 8: Architectures, Technologies, and Tools
Topic 10: C2, Management, and Governance in Civil-Military Operations
Topic 6: Experimentation, Metrics, and Analysis

Authors:
J. Mark Pullen, Douglas Corner, and Lisa Nicklas

Point of Contact:
J. Mark Pullen
C4I Center
George Mason University
4400 University Dr.
Fairfax, VA 22030
mpullen@c4i.gmu.edu

Abstract

The NATO Modeling and Simulation Group Technical Activity 48 (MSG-048) operated from 2006 to 2009, investigating the potential of a Command and Control (C2) Battle Management Language (BML) for Multinational and NATO C2-simulation interoperation. To achieve this, MSG-048 used an interface specification developed under US Army support called Integrated BML, enhanced to meet coalition needs. Demonstrations in 2007 and 2008 culminated in a weeklong period of experimentation in 2009. In all, six national C2 systems and five national simulations successfully interoperated, showing a high likelihood that the approach used can form the basis of a wide range of coalition collaboration.

BML Web services used by MSG-048 were developed by our group under an innovative approach called Scripted BML, in which BML is mapped to JC3IEDM and stored in a database. The range of needed functions is supported by a scripting engine that considerably simplifies requirements for development of the BML Web service. This, in turn, allows rapid response to XML schema changes in the experimental environment while at the same time reducing possible coding errors to the minimal set represented in the scripting language. The Scripted BML server implements push, pull, and publish/subscribe capabilities and has been provided with multithreading capability for better performance and an improved Condensed Scripting Language to reduce effort required for scripting. This paper provides a description of the functions and design of the Scripted BML Server along with examples of its use by MSG-048.

1. Introduction

This paper describes a developmental supporting technology for interoperation of command and control (C2) systems with simulation systems. The general technology area on which we report is Battle Management Language (BML), which aims to provide an unambiguous information exchange for such interoperation [1-3]. The supporting technology is the Scripted BML Web service (SBMLServer), which implements BML in a network-centric service paradigm.

2. Development of the SBML Concept

Early work in BML was contemporary with development of service-oriented architecture (SOA) that led to a vision of WS as an enabler for BML [4]. The Joint BML project [3] was the first to enable interoperation of multiple C2 and simulation systems in a shared environment. Based on Schade and Hieb's work in grammar for BML [5,6], that project developed an XML schema for BML and implemented the schema in a WS implemented in Java [7], and shown in Figure 1. BML orders and reports are represented in a JC3IEDM [8] database.

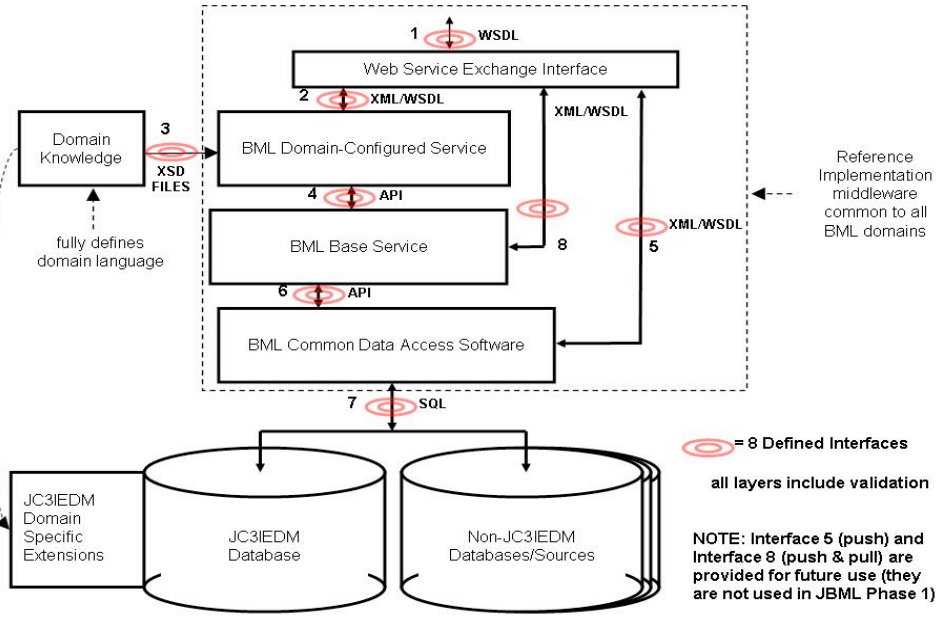

Figure 1. JBML Web Service Architecture

Development of the JBML WS led to important understanding regarding the BML WS:

- The server is a critical central component for BML.
- As a developmental capability, BML will be growing and changing for the next several years.
- With the application and schema in a state of flux, the server requires frequent changes; these tend to be a source of "bugs" (program errors).
- Functions implemented in the WS are simple: push and pull of BML documents in XML, implemented as logical operations on a relational database.

3. Scripted BML use in MSG-048

NATO Modeling and Simulation Group (MSG) technical activity 048 (MSG-048) was chartered to evaluate the potential of BML for coalition C2-simulation interoperability. Coalition operations have a need for interoperability that is even greater than that of national military Service and Joint operations. Because coalitions must function under greater complexity due to significant differences among doctrine and human language barriers, the agility to train and rehearse rapidly before the actual operation is highly important [9]. MSG-048 adopted a SOA approach [10]; its first major demonstration employed the WS developed for JBML. The authors developed SBMLServer to provide flexible support to subsequent demonstration and experimentation by MSG-048. The scope of the final experimentation conducted by MSG-048 [11,12] is evident in Figure 2. The Appendix to this paper provides an abridged example of an MSG-048 BML Order.

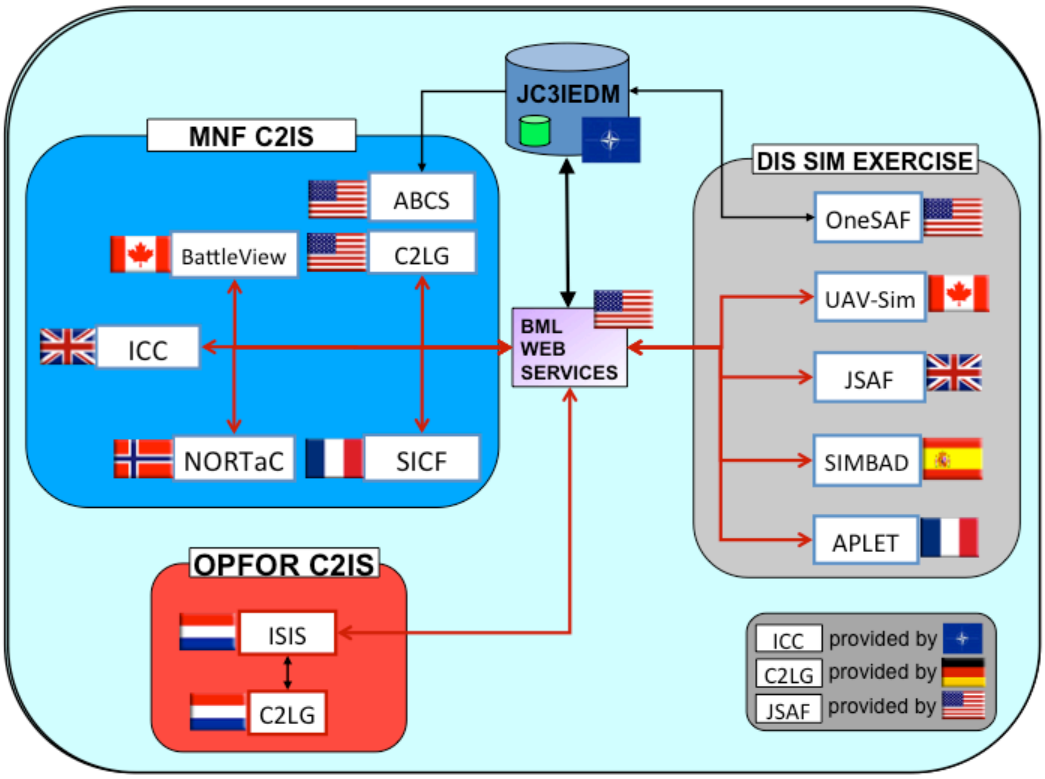

Figure 2. MSG-048 Experimentation Architecture

The scripted approach employed in SBMLServer is widely used in software systems. For BML, it has these characteristics:

- While the details of BML electronic documents continue to grow and evolve, the basic functions of the server remain as described at the end of section 2 above.
- The script is capable only of the limited functionality needed to express mappings to and from BML and the relational data model used (for MSG-048, JC3IEDM).
- Skills needed to create the script are narrower than those needed to create a general-purpose WS since scripts are written in the simpler special purpose scripting language.
- Development of the scripting engine can be a focus separate from the data mappings, resulting in improved performance and robustness.
- Ability to change the service rapidly, by modifying the script, reduces cost and facilitates prototyping.

We implemented SBMLServer with these characteristics and have continued to mature and refine it [13-18]. The remainder of this paper describes the design of SBMLServer in detail, including explanation of how its functionality facilitates coalition BML experimentation.

4. Architecture of SBMLServer

Figure 3 shows the architecture of SBMLServer. The BML Input may be a *push* containing data (e.g. an Order) or may be a *pull* request for data. If successful, a push returns a response indicating success; a pull returns the requested data, formatted in BML per the script. If unsuccessful, either push or pull will return an error message. The SBMLServer operation is driven by elements of the BML that are individually processed by the script. These elements are XML aggregates, known as *BusinessObjects* (BO). (Alternately, they could be described by their grammatical role; they are *constituents* of the BML grammar [5].)

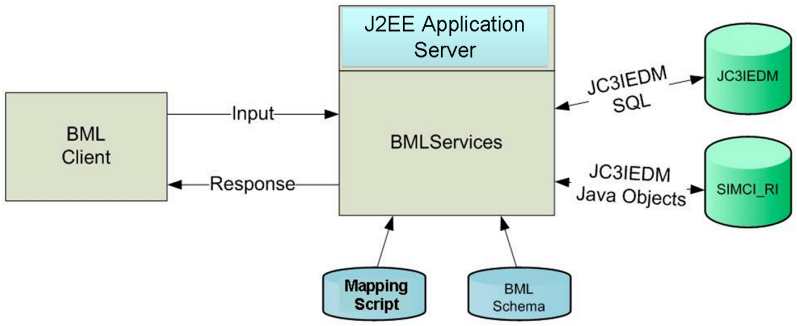

Figure 3. SBMLServer Operating Configuration

The SBML service runs under the JBoss J2EE Web service environment [19]. Methods available provide for push and pull of a collection of Orders, Reports, and supporting services (such as NewUnitType and NewUnit, for database initialization). SBMLServer is capable of persisting the supporting information, using either a SQL-based relational database or Java Objects exchanged with the Reference Implementation (RI) JC3IEDM persistence service [20]. This dual capability enabled MSG-048 to combine US Army systems based on the RI with other NATO national systems that used the SQL database.

Two files control the BML/JC3IEDM conversion. The *BML schema* is an XML schema document (XSD) that specifies the structure and contents of the input document, while the *mapping script* contains scripting to process each BO. The BO is treated as an XML subtree rooted at a particular XML tag in the BML input. The BO script contains all the variable definitions and processing instructions needed for that subtree; it may be thought of as a subroutine, with parameters passed in and return variables passed back. The first phase of BML operation identifies the tags and the BO names with which they are associated. A BML transaction input may cause the invocation of multiple BOs. The root of the BML input document is also the name of the root BO; all other BOs are invoked by calls in the script. The script itself is coded in XML, allowing SBMLServer to use the open source Java Document Object Model (DOM) parser [21].

5. Publish/subscribe functions

Client C2 and simulation systems have the requirement to receive information in Orders and Reports that the server receives from the other clients. In the first implementation of SBMLServer, clients had to poll the server to get any information supplied by other clients. A publish/subscribe capability was added to the SBML server in order to overcome the inefficiencies of the polling interface, by using capabilities of JBoss 4.2.2 [19]. The messaging service provided by JBoss: JBoss Messaging or JBossMQ is an implementation of the Java Message Service (JMS) 1.1 [22]. JBossMQ provides both point-to-point messaging between two entities (using JMS Queues) and a subscription-based distribution mechanism (using JMS Topics) for publishing messages to multiple subscribers. JMS provides reliable delivery of messages for all subscribers to a particular topic.

SBMLServer 2.3 provides a set of preconfigured JBossMQ Topics, which are used for the distribution of incoming orders and periodic reports to any interested subscribers. As BML messages are received, they are processed by the appropriate script and written to the database. Successful completion of the transaction indicates that there were no errors in incoming data and that the message can be forwarded to subscribers. Within the server, there is an XPath [23,24] statement associated with each Topic, which is matched against the input data to determine if a particular message should be written to that Topic. If application of the XPath statement to the message produces a non-null result, the message is written to that Topic. A particular BML message may match more than one XPath statement and therefore could be transmitted to more than one Topic. A client that was subscribed to multiple Topics might therefore receive the same message more than once. The SBML publish/subscribe architecture is depicted in Figure 4.

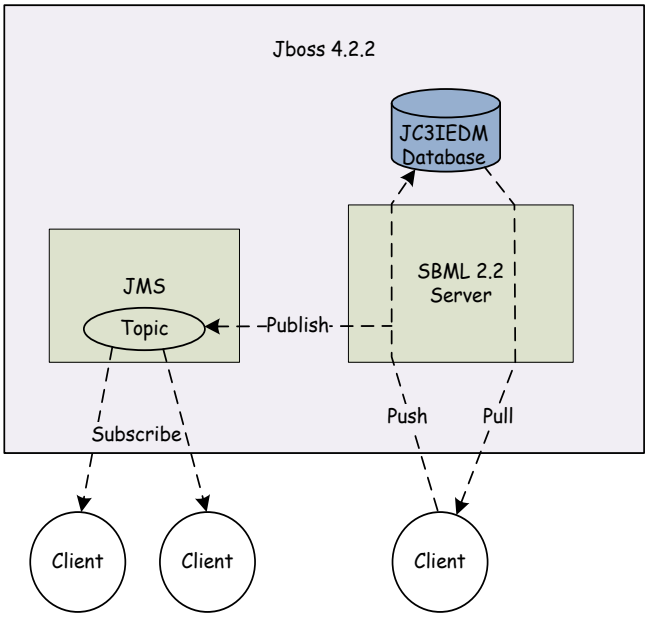

Figure 4. Publish/Subscribe Architecture Used By SBMLServer

JMS is built for the Java environment; thus, interfacing with JMS presents an additional requirement for clients written in C++ and other languages. We have provided an interface for C++ users, built under the Java Native Interface (JNI) framework. This interface works well; however, it separates the actual client code from a direct interface with the messaging service and thus adds another layer of complexity to C++ clients.

Because creating clients that set up subscriptions in languages other than Java is not straightforward, we have developed a RESTful version of SBMLServer using the RESTEasy implementation of JAX-RS for JBoss [25]. With this RESTful version of SBMLServer and the HornetQ implementation of JMS [26], clients create subscriptions could be written in any language that has access to an HTTP client library.

6. Dynamic publish/subscribe topics

Analysis by the MSG-048 technical subgroup indicated that a more flexible publish/subscribe mechanism is desirable. As used by MSG-048, the SBMLServer implementation of publish/subscribe predefined static topics to which subscribers could subscribe dynamically. During initialization, the SBMLServer reads in a configuration file *topicDefinitions.xml* that contains statically defined topics. The SBMLServer then uses each XPath query string in this configuration file to determine whether to publish a BML transaction to a particular Topic. Clients can subscribe to the static Topics they want, thereby establishing the BML transactions that they would receive. This approach greatly improved efficiency over a polling interface in that:

- Each message is posted to each topic at most one time.
- Poll requests to the server are not required.
- Database queries in response to polling were eliminated.
- Clients receive BML transactions immediately rather than waiting for the next poll cycle.

However, the approach described above has a major drawback in that Topics were statically defined. A client could not use a Topic that was not predefined in *topicDefinitions.xml*. Under version 2.3 of the SBML server, adding a new topic to the server requires the following manual steps:

- Updating the *topicDefinitions.xml* file with the new topic name and XPath formated search criteria
- Creating a new message bean definition in the *SBMLTopics-services.xml* configuration file with the same topic name as defined in topicDefinitions.xml

After completing both steps, the SBML server and its supporting JBoss must be restarted for the new Topic to be available. Restarting the SBML server is required to pick up the new Topic since the topic definition file is read only during server initialization, while restarting JBoss is needed to create a message bean for the newly defined Topic.

To implement dynamic topics, version 2.4 of SBMLServer makes use of JMS message selectors with a single static topic. This allows the server to provide client controlled

filtering of BML. JMS message selectors provide a way for the clients to be more selective about the messages that they receive for a given topic. Figure 5 provides a visual of how message selectors are used for publish/subscribe with SBMLServer. In the example there is one static topic defined, named *SBMLTopic*.

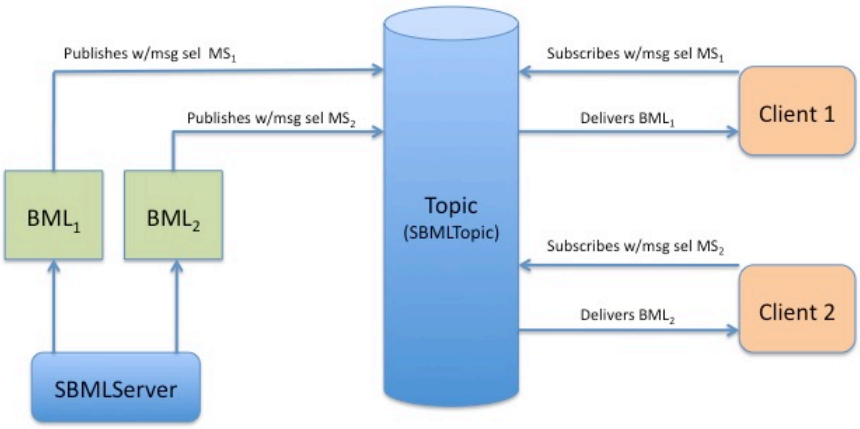

Figure 5. Message Selectors in SBML

The topic configuration file, *topicDefinitions.xml*, has been replaced with another configuration file that allows the initialization of message selectors. The new file is called *msgSelectors.xml* and is also read by SBMLServer at initialization. An example of file *msgSelectors.xml* is shown in figure 6. Each message selector has a name and an associated search string that must be a valid XPath query.

To support dynamic topics, two new web services are available to clients. They are:

- getMsgSelectors()
- addMsgSelector(String search)

The *getMsgSelectors* method provides clients a list of currently defined message selectors. SBMLServer returns to the client a list of message selector names and associated XPath queries. The second new web service, *addMsgSelector*, allows clients to dynamically define a new message selector on the server. For this web service, the client supplies one parameter: a search string, which is a valid XPath formatted query. The server generates and returns a message selector name that is to be associated with the supplied query.

```
<?xml version="1.0" encoding="UTF-8"?>
<Message >
  <Selector>
    <name>allGSR</name>
    <search>//TypeOfReport[. = 'GeneralStatusReport']
     </search>
  </Selector>
  <Selector>
    <name>allOrder</name>
    <search>//OrderPush</search>
  </Selector>
  <Selector>
    <name>allSIMCI</name>
    <search>/*[contains(name(),'REP')]</search>
  </Selector>
</Message>
```

Figure 6. Sample msgSelectors.xml

7. Pushing a complete thought

When interacting with a relational database, typically new rows are written one at a time and are held in a pending state by the DBMS. When processing of a transaction is complete, the entire transaction is committed and written in its final form to the database. The RI Java Object persistence service [20] supports a different approach to transaction control, in which all tables updated are combined into a single "push" to the RI interface. This approach is in consonance with MIP documentation [27], which discusses the concept of a "Complete Military Thought" in that a single JC3IEDM transaction should consist of all the elements that permit it to stand alone as a logical "thought." A capability to support this approach has been added to SBMLServer in the form of the `ri_start` script command, which signals the beginning of a complete thought. The `ri_start` also identifies the parent object and the primary key of the parent object. As additional elements are written, they are collected in memory and linked together as Java references and are also identified by temporary object identifiers (OID).

At the end of the complete thought, the script issues the `ri_end` command, causing the set of linked objects to be passed to the RI interface, which translates the linked objects into a JC3IEDM XML document. This document then is passed to subscribers of any object included in the complete thought and is also used to update the RI database.

The entire transaction (for example, a BML Order) could be pushed as one "thought" or it might be broken up such that each task and each control feature is pushed separately. The complete thoughts also may be nested, enabling the server to push each complete thought and then to link it to its parents using database keys returned by the initial push, rather using than Java references. The script changes needed to effect such a transaction are quite simple.

8. BML Namespaces

One shortcoming of the SBMLServer used to support MSG-048 was that it ignored namespaces in the BML input and did not return output BML that used XML namespaces. This made validation of the BML difficult and required that there be no conflicting names in the various namespaces used. Members of the MSG-048 Technical Group pointed out that this placed a constraint on experimentation; therefore, the SBMLServer has been modified to support XML namespaces. The modified version expects that (1) any input BML is specified with the correct namespaces and (2) any generated output BML also has namespaces specified. This enables all input and output BML/XML to be validated against their XML schemas. This improvement required changes to the SBML web service as well as the scripting. Within the server, an option was added to the input properties file to enable validation of all input BML. Because the currently implemented BML uses a variety of independent schemas, the server was modified to contain a mapping of BML root nodes to corresponding schemas. This was needed to interpret BML elements within the scripting correctly.

Changes to existing scripts to support this capability were one-time and straightforward. Achieving the improvement requires that the scripter specify what namespaces will be expected in BML input and output. This is done in a separate namespace mapping file, by defining what namespace prefixes will be used within the script's references to elements that are part of the BML namespaces. Using a mapping file is necessary because the SBML script contains tags with namespace prefixes in field contents; therefore, the normal XML namespace prefix mappings won't work. The matching prefix must be used in any references to BML elements within the script's body.

9. Multithreading SBML for performance

A major concern during MSG-048 experimentation was that performance of the BML server might prove inadequate. This was dealt with pragmatically, by constraining each reporting element to one report each minute. This number was set by military subject matter experts who confirmed that no actual unit would manually generate reports more frequently. Nevertheless the issue of performance remains a concern in that larger military forces may have enough units to exceed the server's capacity, which was about one report per second. Ultimately the problem may be addressed by creating higher-powered servers; in operational use, they will not need to implement a scripting capability, which adds overhead.

It is easy to foresee that near-term experimental use of BML might need higher server performance. Therefore, performance of the SBMLServer has been improved by providing for multithreaded operation. In previous versions, client requests of the web service were serialized at the server input. In the current version, multiple requests are processed simultaneously to the greatest extent possible in order to improve performance.

The following modifications were necessary to allow for multithreaded SBML web services:

- To allow for concurrency, several resources that were global had to be made local:
  - Each instance of SBMLServices (the top level object of SBMLServer) now has its own connection to the MySQL database.
  - Each instance of SBMLServices has its own copy of the publish/subscribe topics.
  - Formerly static data conversion methods are no longer static but are instead instantiated within each instance of the server.
  - Since DOM is not thread safe, each instance of SBMLServices services must parse the input scripts.
- Semaphores are created to insure serial access to the remaining global resources:
  - Since there can only be one connection to the JC3IEDM RI, that connection now must be shared among all instances; this requires a semaphore to control access.
  - Initialization of SBMLServer is also now protected by a semaphore.
- Setting and using object identifiers (OIDs) for pushing to the RI requires implementation of synchronized increment and access methods.
- Implementing the increment attribute on a database column primary key was removed from the SBMLServer and is now performed by MySQL, using the MySQL AUTO INCREMENT attribute on the column field that requires uniqueness.
- Locally developed logging routines that would have required synchronization were replaced by the *log4j* package. *Log4j* components are designed for use in heavily multithreaded systems.

With these changes, the SBMLServer is able to use normal Java multithreading, managed by the JBoss server. This has achieved a measured throughput of over ten messages per second using multithreading, as compared to about one message per second achieved during the MSG-048 experimentation. The test was run on a server with four processors. We believe that at least another factor of two increase in throughput would be possible on a server with more processors.

10. Logging/replay

A primary use of the SBMLServer is to support integration of C2 systems with military simulations. In this context, it often is required that the outputs of a coalition of C2 systems and simulations can be replayed. This capability has been added to the SBMLServer. To allow for replay, the SBMLServer now offers the option to create a log file of all transactions processed. A client has been developed that will read the log file and resubmit the transactions in the same order and with the same timing, with the result that all subscriber clients see the same series of simulation events they would have during the period of operation logged. The replay client offers an option to ignore the timestamps, thus allowing the server to get to the final state of the replay log more quickly.

11. Condensed scripting language (CSL)

The XML-based script used by SBMLServer version 2.3, while simple to implement in the Web service environment, is less than optimal for the human programmer since it suffers from the well-known verbosity of XML. We have initiated use of a front-end translator that can reduce the visual and cognitive burden on the script developer by reducing the script to a condensed representation. This representation is neither more nor less powerful than the XML form, since it can be translated directly into the XML form. It is, however, intended to be more usable in that it is easier to for the human scripter to write and to comprehend the working of a condensed BML script.

To achieve this, we developed a compiler to accept the condensed BML scripting language as input and produce an XML script as output. The whole script is treated as a *BusinessObjectInput* that can contain multiple instances of *BusinessObjectTransaction*. Each BusinessObjectTransaction is a set of database operations, which are intended to leave the database in a consistent state at the end of the transaction if executed without interleaving other BusinessObjectTransactions that operate on the same database tables.

The Condensed Scripting Language (CSL) offers three ways to retrieve from the database depending upon whether to retrieve a row or a list of elements from a column or just one column entry: *GetRow*, *GetList* and *Get*. In all three commands, the first identifier is the table name, the second is the column name and the third is a set of *columnReferences* that constitute the *where* clause of the underlying SQL statement. A *Put* operation is defined as a combination of the table name and a set of *columnReferences* that define the columns that need to be updated.

To invoke the BusinessObject (BO), a *Call* statement can be used to invoke another *BusinessObjectTransaction* or *Routine* by specifying the name of the BO, the *anchorTag*, and lists of optional parameters and optional return values. The *anchorTag* is an XPath statement, which is used to search for the BML document for elements to be used by the called BO. If no matching data is found the *Call* isn't performed. If multiple elements match the XPath statement, the *Call* is performed for each element. Conditional statements are defined as either an *IfThen* or an *IfThenElse*. Both statements make a logical comparison of the identifier with the variable and conditionally execute the statements. The *Assign* command can be used to assign a variable to an identifier.

The *BOReturn* statement returns to the calling script level and optionally can generate output. There can be multiple *BOReturn* statements inside a BO and each *BOReturn* can have an unbounded number of output-generating statements. The scheme used allows for the creation of nested tags and also tags dynamically named using variables. Figure 7 illustrates this concept with a condensed-language script. The XML version of the same script is more than four times as long and is significantly more difficult to scan visually.

```
BOInput
{
 BOTransaction WhatWhenPush(...
 {
  //fragment from WhatWhenPush
  Call TaskeeWhoPush TaskeeWho (task_act_id) () ;
 ...
 }

BOTransaction TaskeeWhoPush (task_act_id) ()
{
  GET unit unit_id (formal_abbrd_name_txt EQ UnitID);
  PUT act_res (
     act_id EQ task_act_id,
     act_res_index EQI act_res_index, cat_code EQ "RI",
     authorising_org_id EQ unit_id) ;
  PUT act_res_item (
     act_id EQ task_act_id,
     act_res_index EQ act_res_index,
    obj_item_id EQ unit_id ) ;
  BOReturn
  {
    BOReturnElement
    {
       Tag Result "OK";
    }
  }
}
```

Figure 7. Condensed script for SBML

12. Conclusions

SBMLServer is intended for rapid, flexible prototyping of BML Web services. It has been developed into a well-rounded capability for generating such services quickly and with a low error rate, based on a simple scripting language. While SBMLServer's design is general enough to accept any XML-based input and work with any data model capable of representing the input, our implementations have focused on BML as the input language and JC3IEDM as the data model. SBMLServer was used for this purpose in support of NATO MSG-2009 in 2008 and 2009.

Based on recommendations from MSG-048 participants, SBMLServer has been extended. It now supports dynamic publish/subscribe topics, RESTful services with multiple-programming-language client capability, standardized namespaces, complete thoughts in JC3IEDM, multithreading, logging/replay, and a Condensed Scripting Language (CSL).

References

[1] Carey, S., M. Kleiner, M. Hieb, and R. Brown, "Standardizing Battle Management Language – A Vital Move Towards the Army Transformation," IEEE Fall Simulation Interoperability Workshop, Orlando, FL, 2001

[2] Sudnikovich, W., J. Pullen, M. Kleiner, and S. Carey, "Extensible Battle Management Language as a Transformation Enabler*," SIMULATION,* 80:669-680, 2004
[3] Levine, S. *et al.*, "Joint Battle Management Language (JBML) Phase 1 Development and Demonstration Results," IEEE Fall Simulation Interoperability Workshop, Orlando, FL, 2007
[4] Tolk, A. and J. Pullen, "Using Web services and Data Mediation/Storage Services to Enable Command and Control to Simulation Interoperability," 9th IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2005), Montreal, Canada, 2005
[5] Schade, U. and M. Hieb, "Development of Formal Grammars to Support Coalition Command and Control: A Battle Management Language for Orders, Requests, and Reports, 11th International Command and Control Research and Technology Symposium*,* Cambridge, UK, 2006
[6] Schade, U. and M. Hieb, "A Linguistics Basis for Multi-Agency Coordination," 12th International Command and Control Research and Technology Symposium*,* Newport, RI, 2007
[7] Pullen, J., K. Makineni, and P. McAndrews. "A Grammar-Based Web Service Enabling Multi-domain Distributed Interoperation of Command/Control and Simulation Systems, 11th IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2007), Chania, Greece, October 2007
[8] The Multilateral Interoperability Program (MIP) website: http://ww.mip-site.org
[9] Tolk, A, M. Hieb, K. Galvin, L. Khimeche, and J. Pullen, "Developing a Coalition Battle Management Language to facilitate Interoperability between Operation CIS and Simulations in support of Training and Mission Rehearsal", 10th Command and Control Research and Technology Symposium*,* McLean, VA, 2005
[10] Pullen, J., M. Hieb and S. Levine, "Using Web Service-Based Command and Control to Support Coalition Collaboration in C2 and Simulation," 13th International Command and Control Research and Technology Symposium*,* Seattle, WA, 2008
[11] de Reus, N., R. de Krom, O. Mevassvik, A. Alstad, U. Schade and M. Frey, "BML-enabling national C2 systems for coupling to Simulation,", IEEE Spring Simulation Interoperability Workshop, 2008, Newport, RI
[12] Heffner, K. *et al.*, "NATO MSG-048 C-BML Final Report Summary," SCS/SISO Euro-Simulation Interoperability Workshop, Ottawa, Canada, 2010
[13] Pullen, J., D. Corner and S. Singapogu, "Scripted Battle Management Language Web Service Version 1.0: Operation and Mapping Description Language," IEEE Spring 2009 Simulation Interoperability Workshop, San Diego CA, 2009
[14] Pullen, J., D. Corner and S. Singapogu, "Scripted Battle Management Language Web Service Version 2," IEEE Fall 2009 Simulation Interoperability Workshop, Orlando, FL, 2009
[15] Corner, D. J. Pullen, S. Singapogu, and B. Bulusu, "Adding Publish/Subscribe to the Scripted Battle Management Language Web Service," IEEE Spring 2010 Simulation Interoperability Workshop, Orlando, FL, 2010
[16] Pullen, J., D. Corner, S. Singapo, B. Bulusu, and M. Ababneh, "Implementing a Condensed Scripting Language in the Scripted Battle Management Language Web Service," SCS/SISO Euro-Simulation Interoperability Workshop, Ottawa, Canada, 2010
[17] Pullen, J., D. Corner and L. Nicklas, "Performance and Usability Enhancements to the Scripted BML Server," IEEE Fall 2010 Simulation Interoperability Workshop, Orlando, FL, 2010
[18] Nicklas, L., J. Pullen, and D. Corner, "Dynamic Publish/Subscribe Topics in the Scripted BML Server," IEEE Spring 2011 Simulation Interoperability Workshop, Boston, MA, 2011

[19] http://wjboss.org

[20] Levine, S., L. Topor, T. Troccola, and J. Pullen, "A Practical Example of the Integration of Simulations, Battle Command, and Modern Technology," IEEE European Simulation Interoperability Workshop, Istanbul, Turkey, 2009

[21] http://jaxp.java.net

[22] Sun Microsystems, "JAVA Message Service 1.1" April 12, 2002.

[23] World Wide Web Consortium, "XPath Language Version 1.0", November 16, 1999

[24] http://www.w3.org/TR/xpath

[25] www.jboss.org/resteasy, *RESTEasy JAX-RS: RESTFul Web Services for Java 2.0.1.GA*, August 10, 2010

[26] www.jboss.org/hornetq/rest.html, *HornetQ REST Interface 1.0-beta-3*, August 9, 2010

[27] Multilateral Interoperability Programme, *The Joint C3 Information Exchange Data Model (JC3IEDM Main)*, Greding, Germany, 24 April 2009

Appendix
BML Order from MSG-048 (Abridged)

```
<OrderPush>
   <bml:TaskersIntent>Secure Area 12099</bml:TaskersIntent>
   <bml:Task>
      <bml:GroundTask>
         <bml:TaskeeWho>
            <bml:UnitID>CDR-TRK-A-8SQ10CAV</bml:UnitID>
         </bml:TaskeeWho>
         <bml:What>
            <bml:WhatCode>MOVE</bml:WhatCode>
         </bml:What>
         <bml:Where>
            <bml:WhereID>A-8SQ10CAV-AXIS_OF_ADVANCE</bml:WhereID>
            <bml:RouteWhere>
               <bml:Along>
                  <bml:Coords>
                     <bml:GDC>
                        <bml:Latitude>40.1363</bml:Latitude>
                        <bml:Longitude>47.5369</bml:Longitude>
                        <bml:ElevationAGL>0</bml:ElevationAGL>
                     </bml:GDC>
                  </bml:Coords>
…
                  </bml:Along>
            </bml:RouteWhere>
         </bml:Where>
         <bml:StartWhen>
            <bml:WhenTime>
               <bml:WhenQualifier/>
               <bml:DateTime>20080808080808.000</bml:DateTime>
            </bml:WhenTime>
         </bml:StartWhen>
         <bml:Why>
            <bml:Effect>KILL</bml:Effect>
         </bml:Why>
         <bml:TaskID>TASK_LABEL</bml:TaskID>
      </bml:GroundTask>
   </bml:Task>
   <bml:OrderIssuedWhen >20080808080808.000</bml:OrderIssuedWhen>
   <bml:OrderID>206A</bml:OrderID>
   <bml:TaskerWho >
      <bml:UnitID>CDR-8SQ10CAV</bml:UnitID>
   </bml:TaskerWho>
   <bml:ControlMeasures >
      <bml:ControlMeasure>
         <bml:WhereID>OBJ_TIGER</bml:WhereID>
         <bml:AtWhere>
            <bml:JBMLAtWhere>
               <bml:WhereLabel>OBJ_TIGER</bml:WhereLabel>
               <bml:WhereCategory>OBJECTIVEAREA</bml:WhereCategory>
               <bml:WhereClass>SURFAC</bml:WhereClass>
               <bml:WhereValue>
                  <bml:WhereLocation>
                     <bml:GDC>
                        <bml:Latitude>39.8227</bml:Latitude>
                        <bml:Longitude>47.9478</bml:Longitude>
                        <bml:ElevationAGL>0</bml:ElevationAGL>
                     </bml:GDC>
   …
                        </bml:WhereValue>
               <bml:WhereQualifier>AT</bml:WhereQualifier>
            </bml:JBMLAtWhere>
         </bml:AtWhere>
      </bml:ControlMeasure>
   </bml:ControlMeasures>
</OrderPush>
```