Maturing Supporting Software for C2-Simulation Interoperation

J. Mark Pullen and Lisa Nicklas
C4I Center
George Mason University
Fairfax, VA, USA
{mpullen, lnicklas}@c4i.gmu.edu

***Abstract*- Battle Management Language (BML) has been developed as an unambiguous representation of information in the form of orders, reports and requests between military Command and Control (C2) systems and simulations. This paper describes development of two critical elements for BML experimentation. The first is a Web service that is used as a repository for the orders, reports, and requests, packaged as XML documents and stored in a relational database in a standard format. The second is a graphic user interface that supports inspection and modification of BML documents using forms that are created from the BML schemas at runtime, combined with a geospatial interface. This paper describes the role of these elements in interoperation of simulation and C2 systems and describes their architecture and features, as motivated by application in NATO Coalition BML experimentation. The software described is available as open source.**

KEYWORDS-command control; simulation interoperability; Web services; interpreted services

I. INTRODUCTION

Today's military operations are assisted by real-time information systems, known as Command and Control (C2) systems. Computer simulations could be very useful components in C2 systems and in a few, custom-crafted cases this has been demonstrated. Example uses are training and mission rehearsal of forces at all echelons. However, no generally accepted means of exchanging information and the systems involved has been developed; thus interoperation of C2 and simulation systems remains a goal of the military distributed simulation community. The most prominent activity in this area since 2001 has been the Battle Management Language (BML) [1]. It provides a capability to facilitate interoperation among such systems by providing a common, agreed-to format for the exchange of information such as orders and reports [2]. Information transfer is accomplished asynchronously by providing a repository service that the participating systems can use to post and retrieve XML documents expressed in BML and synchronously by publish/subscribe operation that forwards documents as they are posted. The essential service is implemented as middleware using Web service (WS) technology and can be either centralized or distributed.

This paper describes significant further development of an innovative solution to implementation of the BML WS middleware, known as the Scripted BML Server (SBML), originally reported in [3]. The approach allows rapid development of the needed WS as BML evolves. This is accomplished in a way that simplifies creating the services and thus reduces opportunities for error and provides clear documentation of the process. Two instances of SBML provided the principal server for the largest BML system assembled to date which was used for NATO Technical Activity MSG-048 Experimentation [4]. Here we describe the rationale and design of a collection of enhancements to the basic capability, largely stimulated by that use, that make the services more robust and useful.

II. BML ARCHITECTURE, THEORETICAL AND APPLIED

Figure 1 shows the general approach used in BML, as described earlier in [3]. The database shown uses the Joint Consultation, Command and Control Information Exchange Data Model (JC3IEDM), developed and maintained by the Multilateral Interoperabiltiy Programme (MIP) [5].

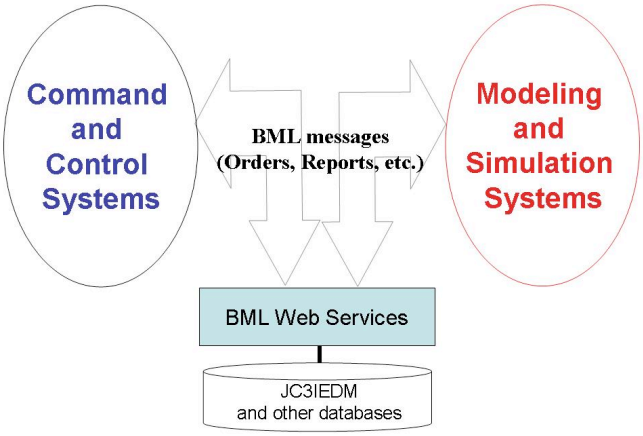

Fig. 1. BML Architcture

Figure 2 shows the elements of SBML client-server operation. The major innovation, as described in [3], is a scripting approach that allows rapid implementation of BML schemas, expressed as JC3IEDM database mappings. The server is capable of storing the database either in open-source relational database MySQL or via a set of Java classes known as the JC3IEDM Reference Implementation.

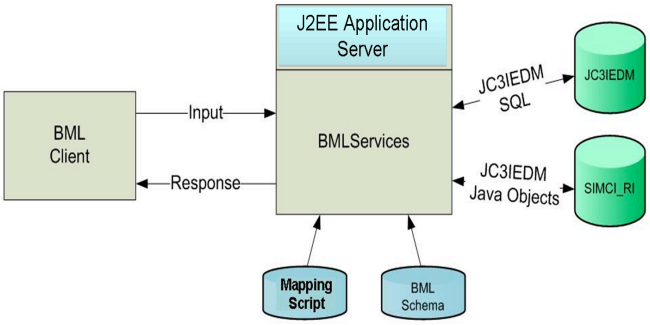

Fig. 2. Scripted BML WS Operating Configuration

III. Programmability Enhancements

The XML-based script used by early versions of the SBMLServer, while simple to implement in the Web service environment, is inconvenient for the human programmer since it suffers from the well-known verbosity of XML. We have developed a Condensed Scripting Language (CSL) and front-end translator to 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. However, we have found it to be more usable in that it is easier for the human scripter to write and to comprehend the working of a condensed BML script. (Shortly after CSL became available, our script writers dropped all use of the XML scripting format.)

The translation is achieved by using well-known compiler techniques to accept the condensed BML scripting language as input and produce an XML script as output. The basic unit of operation is a Business Object (BO); the whole script is treated as a *BusinessObjectInput* that can contain multiple instances of *BusinessObjectTransaction*, each of which consists of a set of database operations. These 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 provides three ways to retrieve from the database. The actions can retrieve a row or a list of elements from a column or a single column entry: *GetRow*, *GetList* and *Get*. All three commands require three identifiers: the table name, the column name, and 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 which need to be updated.

To invoke the BO, a *Call* statement can be used to call 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 pattern statement [16], which is used to search the BML document for elements to be used by the called BO. If no matching data is found the *Call* is not 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.

*BusinessObjectTransaction*s can return parameters via a list of arguments. To generate output, the *BOReturn* statement is used. There can be multiple *BOReturn* statements inside a BO and each *BOReturn* can have an unbounded number of output-generating statements. Nested tags and dynamically-named tag variables are supported. Figure 3 shows a condensed-language script and Figure 4 shows part of its XML counterpart. The XML version of the same script is more than four times as long and significantly more difficult to scan visually. CSL is documented in [7].

IV. Publish-Subscribe for Dynamic Topics

The history of SBMLServer has been one of continuous, incremental improvement, as shown in Table 1. Client C2 and simulation systems have the requirement to receive information in Orders and Reports that the server receives from the other clients [10]. Initially, this was implemented by polling the SBMLServer. The clients had to poll the server to get any information supplied by other clients. Later, a publish/subscribe capability was added to the server in order to eliminate the inefficiencies of the polling interface. This approach greatly improved efficiency over a polling interface in that:

- Each BML message is posted to each Topic at most one time.

TABLE 1
ADDITIONAL FEATURES OF SBMLSERVER BY VERSION

2.0	basic scripted server functions publish/subscribe
2.1	Condensed Scripting Language (CSL) Publish/subscribe with fixed topics
2.2	initial NATO OPORD script namespace capability multithreaded operation complete thought in JC3IEDM logging
2.3	dynamic topics using message selectors associated CSL features for dynamic topics nested complete thoughts in JC3IEDM XML attributes on output elements replay of logfiles
2.4	HORNETQ RESTful interface Multi-language clients document-based push/query SISO C-BML Phase 1 Trial Use scripts C-BML Light NATO OPORD script

```
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";
    }
  }
}
```

Fig. 3. Condensed script for SBML

```
<!-- Fragment of code from WhatWhenPush -->
<call>
   <boName>TaskeeWhoPush</boName>
   <anchorTag>TaskeeWho</anchorTag>
   <parameter>
          <workingVariable>task_act_id</workingVariable>
</parameter>
</call>
. . .
<BusinessObjectTransaction>
          <transactionName>TaskeeWhoPush</transactionNam
e>
          <parameter>task_act_id</parameter>
          <tableQuery>
                <databaseTable>unit</databaseTable>
                <queryAction>GET</queryAction>
                <resultName>unit_id</resultName>
                <columnReference>
                      <columnName>
                          formal_abbrd_name_txt
                </columnName>
                      <businessObjectTag>
                      UnitID
                </businessObjectTag>
                                  </columnReference>
                                   </tableQuery>
…
```

Fig. 4. XML equivalent to first few lines of CSL script

- 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.

The publish/subscribe capability was built on top of JBoss Messaging. JBoss Messaging is the default JMS 1.1 provider in the JBoss application server version 4.2.3 [12]. JBoss Messaging 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 can provide reliable delivery of messages for all subscribers to a particular topic.

Effective with SBMLServer 2.3, the server provides a set of preconfigured JMS 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 is available to be published to one or more Topics.

Within the server, there is an XPath [16] statement associated with each Topic, which is matched against the input BML data to determine if that particular BML should be published to that Topic.

A particular BML message may match more than one XPath statement and therefore could be published to more than one Topic. A client that is subscribed to multiple Topics might therefore receive the same BML message more than once. The SBML publish/subscribe architecture is depicted in Figure 5.

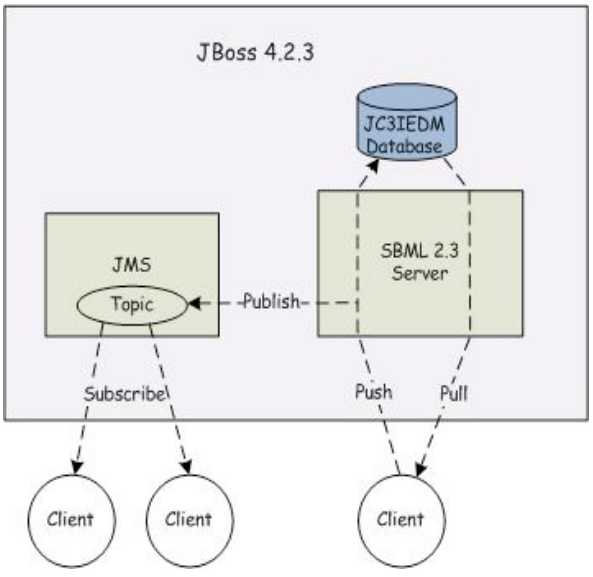

Fig. 5. Publish/Subscribe Architecture for SBML

JMS is built for the Java environment; thus, interfacing with JMS presents a problem for clients written in other programming languages. We provide an interface for C++ users, built under the Java Native Interface (JNI) framework that adds another layer of complexity to C++ clients.

Analysis by the MSG-048 technical subgroup indicated that a more flexible publish/subscribe mechanism is desirable [4]. As used by MSG-048, the SBML server implementation of publish/subscribe predefined static topics to which subscribers could subscribe dynamically. However, this approach had a major drawback in that Topics were statically defined. A client could not use a Topic that was not predefined in an SBML Topic configuration file. Under version 2.3 of the SBMLServer, adding a new topic to the server requires two manual steps: (1) Updating the SBML Topic configuration file with the new topic name and XPath formatted search criteria and (2) Creating a new message bean definition in another SBML configuration file with the same topic name.

Version 2.4 of SBMLServer improves on publish/subscribe by providing dynamic topics. This is accomplished by making use of JMS message selectors with a single static topic. JMS message selectors provide a way for the clients to be more selective about the messages that they receive for a given topic. Figure 6 gives a visual of how message selectors are used for publish/subscribe with SBMLServer. In the example there is one static topic defined, named *SBMLTopic*.

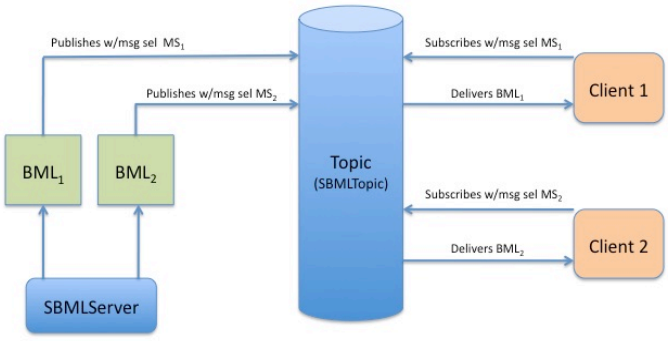

Fig. 6. Message Selectors in SBML

To support dynamic topics, two new web services are available to clients. They are *getMsgSelectors*() and *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 returns a server generated message selector name that is to be associated with the supplied query.

With this approach, creating clients that set up subscriptions in languages other than Java is still not straightforward. In the next section, we will describe a RESTful version of SBMLServer that will allow subscribers to be written in any programming language that have access to a HTTP client library.

V. PERFORMANCE ISSUES: NAMESPACES, THREADING, AND RESTFULNESS

One shortcoming of the SBML server used to support MSG-048 was that it ignored namespaces: BML input and output did not use XML namespaces. Namespaces provide a way for independently developed element and attribute names to be unique, by appending a namespace identifier at the front of the tag. Lack of namespace usage made validation of input and output difficult and required that there be no conflicting names across the multiple namespaces used to create a BML document. Members of the MSG-048 Technical Group pointed out the constraints on experimentation that this shortcoming caused. Therefore, the SBML server has been modified with version 2.3 to support XML namespaces. With this version, SBML server expects that (1) any input BML is specified with the correct namespaces and (2) any generated output BML also has namespaces correctly specified.

This version allows for easier input and output BML validation. To implement namespaces, changes were made 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 IBML 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. With version 2.5 of the SBML server, the mapping of BML root nodes to corresponding schemas has moved out of the server code and into a configuration file that is read in during initialization.

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, which defines 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 content. The correct prefix defined in the namespace mapping file is used in any references to BML elements within the script's body.

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 military unit would report 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.

It is easy to foresee that near-term experimental use of BML might need higher server performance. Therefore, performance of the SBML web services has been improved by providing for multithreaded operation. In previous versions, client requests made to the web service were serialized by the server. In version 2.4, multiple requests are processed simultaneously to the greatest extent possible in order to improve performance. 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 performance testing was run on a server with four processors. Since much of the server's work can proceed in parallel, we expect very good efficiency from the addition of more processors.

With SBMLServer version 2.5, a Representational State Transfer (RESTful) web server is available along with the Simple Object Access Protocol (SOAP) based web server. The RESTful approach avoids the need to convert the XML input to the serialized remote procedure call which is the basis of SOAP. To help implement the RESTful version of SBML server, we used the RESTEasy implementation [17] of JAX-RS. We continue to use JBoss 4.2.3 but have replaced JBoss Messaging with HornetQ messaging [18]. Because the RESTful approach requires less overhead, we expected it would give better performance. When we compared the performance of the two servers in processing pushes of position status reports, we saw a 15% improvement with the RESTful server versus the SOAP-based server when using HornetQ for messaging. We do welcome this performance improvement; however, the major advantage to using the RESTful server has been the ability for clients written in programming languages other than Java to create subscriptions in a straightforward way.

Both the SOAP-based and RESTful versions of the server publish BML to the single JMS topic, SBMLTopic, as described in the section 4 of this paper. Subscribing clients written in Java essentially work the same way as before but now can receive messages published from both/either server. HornetQ provides the HornetQ REST Interface [19], which can be used to create subscribers more easily in other programming languages. The only requirement is that the programming language has access to a HTTP client library.

The HornetQ REST Interface is installed as a Web Archive. As part of the SBML open source distribution, we provide SBMLHornetqREST.war that will install the HornetQ REST interface if dropped into JBoss' deploy directory. This interface allows a client to set up either a push or pull subscription to JMS topics using HTTP HEAD and POST commands. Figure 7 shows the different components and how they connect for publish/subscribe.

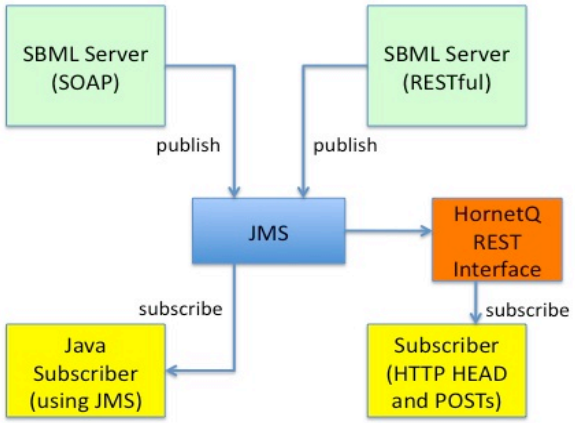

Fig. 7. Publish/Subscribe with HornetQ REST Interface

To set up a pull subscription, the client must first do a HTTP HEAD on the topic resource. For example, if the JMS Topic is named SBMLTopic as for both SBML Servers, the HTTP command would be:

```
HTTP HEAD /SBMLHornetqREST/topics/jms.topic.SBMLTopic
```

The next step is to POST to the resource, msg-pull-subscriptions, returned by the HEAD command. The return from the POST command as well as successive POST commands would give a resource for msg-consume-next. POSTs to msg-consume-next will retrieve any messages.

To set up a push subscription, the client must first do a HTTP HEAD on the topic resource in the same way for a pull subscription. Instead of a POST to the resource msg-pull-subscription, the next step is to POST to the resource msg-push-subscriptions. With a push subscription, the client must have the ability to process incoming HTTP commands from the HornetQ REST interface. With either pull or push subscriptions through the HornetQ REST interface, clients have the option to specify message selectors to filter the messages that they receive.

VI. Example of Application: MSG-048 Experimentation

The Scripted BML Server was used in the largest BML activity to date: NATO MSG-048 experimentation. Coalition military operations have a need for interoperability that is even greater than that of national Service and Joint operations. Because coalitions must function under greater complexity due to significant differences among doctrine and human language barriers; the ability to train and rehearse rapidly before the actual operation is highly important [1]. The NATO RTO Modeling and Simulation Group (MSG) recognized this need and chartered Technical Activity MSG-048 to explore the promise of BML in coalitions. Figure 8 shows the architecture of the experiment, which was considered to be ambitious.

The MSG-048 2009 effort improved over previous work by expanding the number of C2 and simulation systems interoperating. In order to do this, it expanded the Service Oriented Architecture (SOA) communication paradigm, as implemented in Web services, to include publish/subscribe, so that the various C2 systems could subscribe to Reports of interest and the simulation systems could subscribe to Orders of interest, avoiding the need to poll the BML Web service for updates and thus increasing both computational and communications efficiency. The primary purpose of this activity was to evaluate the effectiveness of BML in maintaining common state to the degree required for effective interoperation among the C2 and simulation systems. Six C2 systems and five simulations achieved interoperability with the support of a Web service repository and a middleware graphical user interface (GUI).

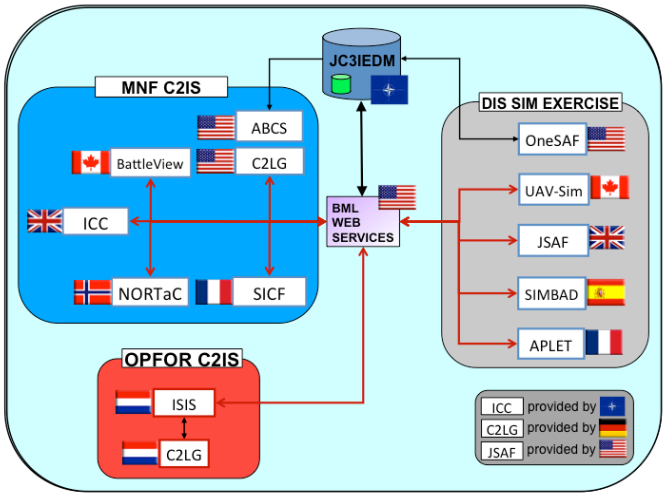

Fig. 8. BML Architecture as Implemented in MSG-048 Experimentation

The MSG-048 final experiment was a "warfighter experiment," allowing military personnel to evaluate a BML capability and, in order to do so, expanding what was used in earlier technical experiments. The purpose of the experiment was to get an indication of the military benefits of BML and to evaluate the current capability in order to generate future requirements. In those terms it must be considered successful, in that the military participants endorsed further experimental evaluation of BML, which now is ongoing as NATO MSG-085.

VII. CLIENT SUPPORT MIDDLEWARE: BMLC2GUI

All teams developing BML interfaces have one need in common: the need to inspect BML passing between client and server and in order to verify that the right information is flowing, and in some cases to edit that information and view its geospatial elements in a map context. During MSG-048 experimentation, the Command and Control Lexical Grammar Graphical User Interface (C2LGGUI) system element in Figure 7, developed by Fraunhofer FKIE, proved a valuable system element in that it supported review and, where necessary, modification of BML orders in the experimental environment. In consultation with Fraunhofer FKIE, we determined that an open-source interface with similar functionality would provide a valuable capability for the BML community as it engages in developing, prototyping, and experimentation. Accordingly, we have developed the BMLC2GUI, based on inspiration from Fraunhofer FKIE's C2LG GUI with additional functionality described in this paper, as part of the open source tools associated with SBML.

The BMLC2GUI is an open-source user interface tool that represents information flowing to/from C2 and simulation systems in text and image formats. Its main purpose is to provide an easy-to-use graphical user interface to BML users and developers that can serve as a surrogate input/output C2 GUI or alternately to monitor (and, if necessary, revise) BML documents flowing to/from BML client systems. The GUI is a Java application that generates an interface using other open-source tools: Xcentric's JAXFront and BBN's OpenMap. Figure 9 shows a screen capture of the GUI.

The GUI provides an easy and efficient alternative for the end user to edit, validate, and push BML orders to the SBML Web Services and also to pull and view BML reports from the services. The GUI also can subscribe to the SBML subscription service so that the GUI will be updated whenever a new report is published. The map will depict geospatial information from the BML document the user is editing or revising, displaying the correct symbols representing the objects or units in addition to all of the mapping capabilities supported by OpenMap. BMLC2GUI uses the OpenMap implementation of military standard MilStd2525b for unit and object symbol representation [21].

VIII. CONCLUSIONS

BML is a promising approach to interoperation of C2 and Simulation systems. The supporting middleware described in this paper is essential to facilitate development and operation of working BML systems. The evolution of server and GUI software reported above has met an important need in military simulation. We expect this capability will continue to evolve for some years to come. A good measure of computer science, combined with an eye to practical effectiveness, has enabled these advances in supporting software for BML. It is very likely that more of the same will be required as BML capabilities advance.

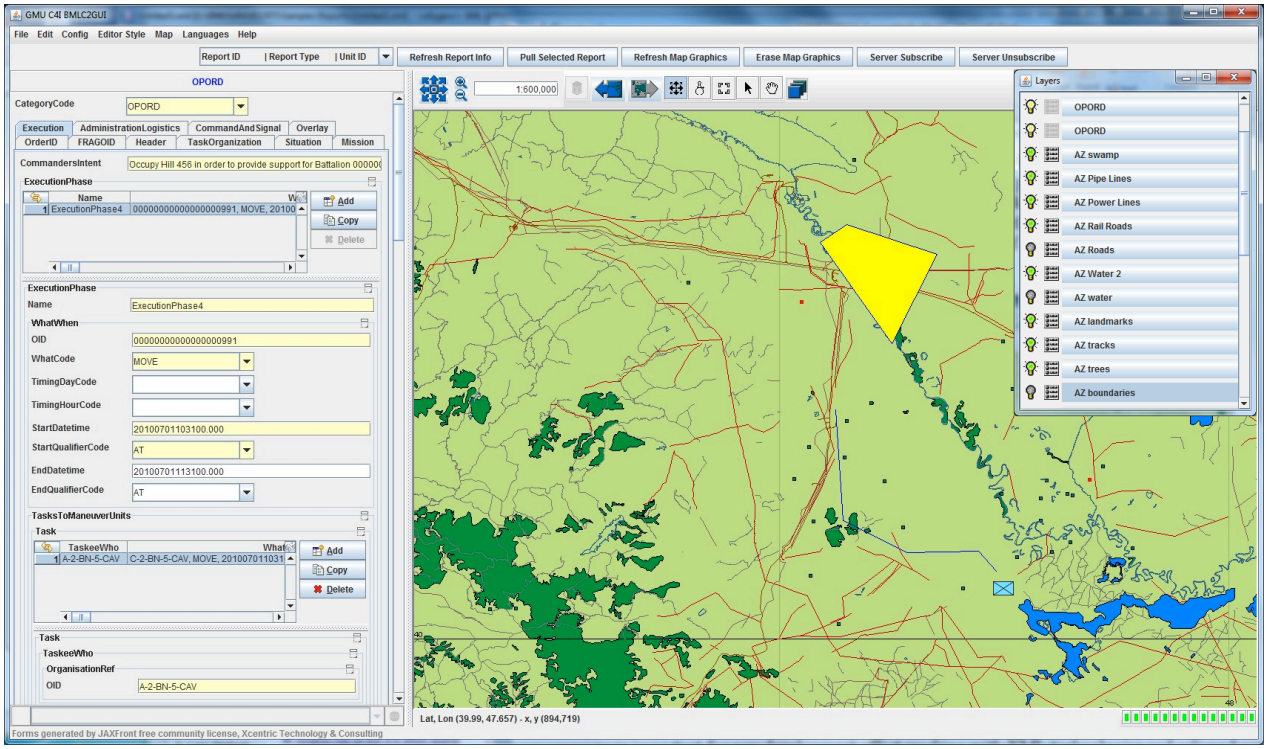

Fig. 9. Screen Capture of the BMLC2GUI

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," in *SIMULATION,* 80:669-680, 2004

[3] Pullen, J., D. Corner, S. Singapogu, and P. McAndrews, "Interpreted Web Services as a Tool for Development of Command and Control Interoperability with Simulations," 13th IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2009), Singapore, October 2009

[4] Heffner, K. *et al.*, "NATO MSG-048 C-BML Final Report Summary," SCS/SISO Euro-Simulation Interoperability Workshop, Ottawa, Canada, 2010

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

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

[7] Crawford, T., "Condensed Scripting Language Programming Guide," George Mason University C4I Center, available through http://c4i.gmu.edu/BML

[8] 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

[9] 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

[10] 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

[11] Pullen, J., D. Corner, and L. Nicklas, "Supporting NATO C2-Simulation Experimentation with Scripted Web Services," International Command and Control Research and Technology Symposium, Quebec, Canada, June 2011

[12] http://wjboss.org

[13] 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

[14] http://jaxb.java.net

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

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

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

[18] www.jboss.org/hornetq HornetQ 2.1 User Manual: Putting the buzz in messaging, June 15, 2010

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

[20] Ababneh, M. and J. Pullen, "A Open Source Graphical User Interface Surrogate C2 System for Battle Management Language Experimentation," International Command and Control Research and Technology Symposium 2011, Quebec, Canada, June 2011

[21] US Department of Defense, "Interface Standard Common Warfighting Symbology MIL-STD-2525B w/Change 1," July 2005