Adding Publish/Subscribe to the Scripted Battle Management Language Web Service

Douglas Corner, Dr. J. Mark Pullen, Samuel Singapogu, and Bhargava Bulusu
C4I Center
George Mason University
Fairfax, VA 22030, USA
+1 703 993 3682
{dcorner, mpullen, ssingapo, bbulusu}@c4i.gmu.edu

Keywords:
Battle Management Language, Web Services, JC3IEDM

The approach to defining a coalition battle management language (BML) now being pursued by SISO requires mapping of BML into a JC3IEDM database, which is accessed via a Web service. In previous SIW papers we have reported on a new approach to implementing such a Web service, based on the notion of an interpreter module. This scripting engine takes as its input the schema of the Web service and a script, coded in XML, that defines the mappings concisely. The interpreter, which will be available as open source software, has the virtues that it is quicker and easier to change than a hard-coded service and also requires a lower level of expertise for development, once the interpreter has been completed. Implementing the BML server capability in a simple Web service results in a bottleneck at the server because the clients must poll the server for updates. This paper provides a detailed description of implementation and use of a publish/subscribe paradigm to improve performance in the Scripted BML Web Service. The implementation uses the JBoss environment and the Java Message Service (JMS). The paper ends with a description of the way this server was used by NATO MSG-048 in its November 2009 experimentation event and some enhancements we plan to implement as a result of that experience.

1. Overview

This paper describes how and why the Scripted Battle Management Web Service (SBML) was extended to provide a publish/subscribe capability, and how the result was used in NATO MSG-048 experimentation 2009.

2. SBML Background

Battle Management Language (BML) and its various proposed extensions are intended to facilitate interoperation among command and control (C2) and modeling and simulation (M&S) systems by providing a common, agreed-to format for the exchange of information such as orders and reports. This is accomplished by providing a repository service that the participating systems can use to post and retrieve messages expressed in BML. The service is implemented as middleware, essential to the operation of BML, and can be either centralized or distributed. Recent implementations have focused on use of the Extensible Markup Language (XML) along with Web service (WS) technology, a choice that is consistent with the Network Centric Operations strategy currently being adopted by the US Department of Defense and its coalition allies [1].

Experience to date in development of BML indicates that the language will continue to grow and change. This is likely to be true of both the BML itself and of the underlying database representation used to implement the BML WS. However, it also has become clear that some aspects of BML middleware are likely to remain the same for a considerable time, namely, the XML input structure and the need for the BML WS to store a representation of BML in a well-structured relational database, accessed via the Structured Query Language (SQL). This implies an opportunity for a re-usable system component: a Scripting Engine, driven by a BML Schema and a Mapping File, that accepts BML *push* and *pull* transactions and processes them according to a script (also written in XML). While the scripted approach may have lower performance when compared to hard-coded implementations, it has several advantages:

- new BML constructs can be implemented and tested rapidly
- changes to the data model that underlies that database can be implemented and tested rapidly
- the ability to change the service rapidly reduces cost and facilitates prototyping
- the scripting language provides a concise definition of BML-to-data model mappings that facilitates review and interchange needed for collaboration and standardization

The heart of SBML is a scripting engine, introduced in [2], that implements a BML WS by converting BML data into a database representation and also retrieving from the database and generating BML as output. It could implement any XML-based BML and any SQL-realized underlying data model. Current SBML scripts implement the Joint Command, Control and Consultation Information Exchange Data Model (JC3IEDM). In the following description, any logically consistent and complete data model could replace JC3IEDM. Reference [3] describes the second generation of SBML.

The current SBML implementation and scripts support two JC3IEDM database interfaces, as shown in Figure 1: one is a direct SQL interface, used with a MySQL database server. The other, SIMCI_RI [4], passes java objects through Red Hat's Hibernate persistence service, which performs the actual database interface function. Version 2 implements a publish/subscribe capability, using the Java Message Service (JMS) as implemented by JBoss in open source (see http://www.jboss.com). Version 2 also implements the XML Path Language (XPATH) (see http://www.w3.org/TR/xpath), wherever a relative path in the XML input is required.

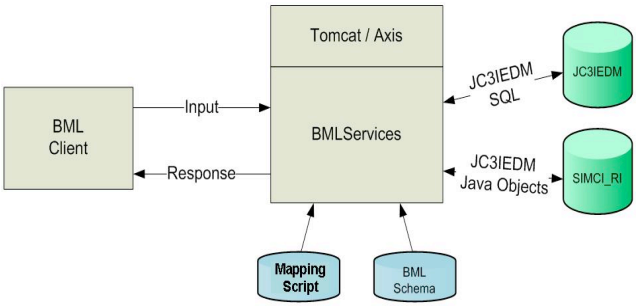

Figure 1: SBML Configuration

The BML/JC3IEDM conversion process is accomplished under control of the scripting language which is described in [3].

3. Polling Interface

M&S systems generally process two kinds of messages, orders and reports. Orders are larger and more complex than reports, however, the frequency of reports is much higher. For example as a military unit moves from one position to another the M&S system may generate updated position reports every few seconds. With the previous release of the SBML service it was necessary for C2 systems to poll the server to determine if updated report information was available. This is inefficient in several ways:

- The server must reread information that may have just been written to the database
- The poll may retrieve no new information
- The request for each client will be processed separately

Client applications have no way to determine what messages might have been posted to the server since sending the last query; they must first determine what reports are available. Scripting was developed within the SBML framework to retrieve a list of reports posted over a specified period of time. It was then necessary for the client to format a query to retrieve those messages that might be of interest. The precision of this method turns out to be low in that many of the messages retrieved tend to be status reports that are then updated by later messages in the same group. An additional problem is that it is necessary for each client using the server to perform this process in parallel.

The load resulting on the server from polling and the large amount of redundant network traffic clearly is inefficient. In this environment it was necessary for M&S systems to limit the rate of report production so that the server was not overloaded.

4. Publish/Subscribe Implementation

A publish subscribe capability was added to the SBML server in order to eliminate the inefficiencies of the polling interface. SBML runs under JBoss 4.2.3 (http://jboss.org). The messaging service provided by this release of JBoss, JBoss Messaging or JBossMQ is an implementation Java JMS 1.1 [6]. JBossMQ provides both point to point messaging between two entities (JMS Queues) and a subscription based distribution mechanism (JMS Topics) for publishing messages to multiple subscribers. JMS provides reliable delivery of messages for all subscribers to a particular topic.

SBML 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. The successful completion of the transaction is an indication that there were no errors in incoming data and that the message can be forwarded to subscribers. There is an XPath [7] statement (see http://www.w3.org/TR/xpath) associated with each Topic which determines if a particular message should be written to that Topic. If application of the XPath statement to the message results in non-null result the message is written to that Topic. Note that a particular BML message may match more than one XPath statement and therefore could be transmitted to more than one Topic. A client then might receive the same message more than once. The publish/subscribe architecture is depicted in Figure 2 below.

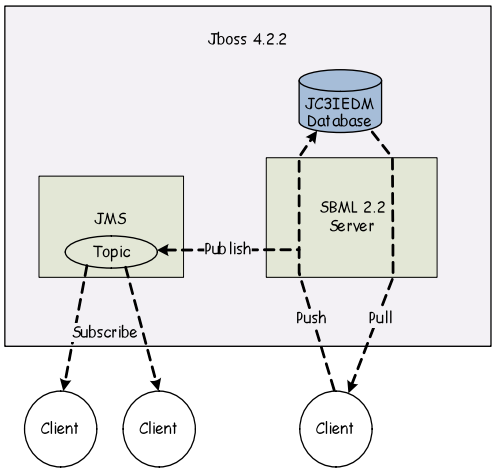

Figure 2. Publish/Subscribe Architecture for SBML

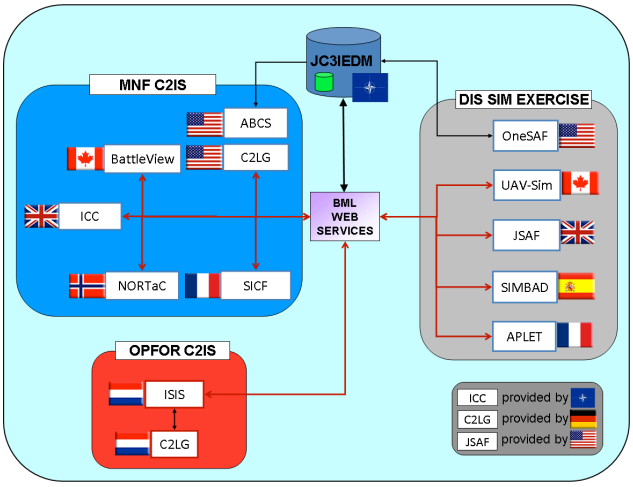

Figure 3. NATO MSG-048 Experimentation Configuration 2009

JMS is built for the Java environment and uses Java Remote Procedure Calls (RPC). This presents an additional requirement for programs written in C++ (and other languages) clients. We provide an interface for C++ users, built under the Java Native Interface (JNI) framework. This interface works well, however it does separate the actual client code from a direct interface with the messaging service.

5. SBML in NATO MSG-048

The NATO Modeling and Simulation Group Technical Activity 48 (MSG-048) was chartered in 2006 to investigate the potential of a Coalition Battle Management Language for multinational and NATO interoperation of command and control systems with modeling and simulation. The 2009 experimentation phase of MSG-048, described in [5], was performed at George Mason University's Manassas, Virginia Campus in November 2009 and included six C2 System and five M&S systems from eight NATO nations, as shown in Figure 3. This was the first large scale use of the SBML Publish/Subscribe service. The service performed well and enabled a much larger set of systems to interact as well as increasing the precision of reports by permitting a higher status update rate. BML messages were batched, to reduce the impact of Web service overhead. The processing time supported by our server for MSG-048 experiments was 50 ms per transaction.

As shown in Figure 2, the publish/subscribe we implemented requires a client module to open a connection to the server, which is used to transmit Web service transactions matching the subscription. The C2 systems and simulations used by MSG-048 in 2009 were implemented in Java and C++ languages. In order to support publish/subscribe under the latter, we provided an interface module made on the Java Native Interface (JNI).

6. Publish/Subscribe Enhancements

Topics. The current implementation, SBML 2.3, the Publish Subscribe service has the topics hard coded along with corresponding XPath statements used for matching messages with JMS Topics. While acceptable in a prototyping environment, environment this necessitates the rebuilding of the service as requirements change. We therefore plan that, at a minimum, in the next version of SBML the Topics will be configurable through an external file. A more sophisticated enhancement would be a dynamic topic capability such that clients could create a topic in an XML message by specifying the topic name and specifying an XPath statement, used to match against incoming messages. The topic would then be active until the next time the application server was restarted. Some planning is necessary to prevent name collisions and might be beneficial to provide for the publication of new topics are they are created to allow sharing between clients.

Client Language. Release 5 of JBoss is in common use and, among other things, provides for a choice of protocols to be used with the messaging subsystem. This provides for a more direct interface for C++ clients.

7. Scripting Enhancements

In the process of developing the scripts described above, we learned some lessons about how to make scripting more practical.

7.1 Nested *if-then-else* construct

The logic required in scripts turns out to entail some decisions that are not easy to encode with the single level of if-then-else described in [3]. Therefore, we are adding a nested *if* to the XML scripting. Our nested ifs mimic the use of "curly braces" { } in progamming languages such as Java. Thus a BO could contain:

```
<BusinessObjectTransaction>
    <transactionName>transName</transactionName>
    [Other BOT header elements]
    <Block>
        <SBML elements>
    </Block>
</BusinessObjectTransaction>
```

If statements within the top level <Block> appear as:

```
<ifThenElse>
    <compare>someWV</compare>
    <relation>EQ</relation>
    <literalValue>xxx</literalValue
</ifThenElse>
<Block>
    <SBML elements executed if condition was true>
</Block>
<Block>
    <SBML elements executed if condition was false>
</Block>
```

<Block> thus serves as the parent of all language elements. As positive side effect of this is that formatting the script with XML editing tools results in automatic indentation of the elements in each block, making the script much more readable.

7.2 Condensed Scripting Language

The XML-based script used by SBML2.3, 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 are investigating 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 is intended to be no more nor less powerful than the XML form, since it will translate directly into the XML form. It is intended to be more usable in that it is easier to write and to comprehend the working of a condensed BML script. We are developing a compiler that will accept the condensed BML scripting language as input and produce an XML script as output.

Appendix 1 provides a Backus-Naur Form (BNF) representation of the condensed scripting language. 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 of other BusinessObjectTransactions that operate on the same database tables. Each BusinessObject is identified by a unique identifier and may optionally include a definition of a *multivalueworkingVariable* and the variable name that it should be assigned to at every iteration of the *BusinessObjectTransaction*. The definition is then followed by a declaration of the parameters and the Return values. The parameters are declared as an ordered list of *variables* where a variable is either a string literal enclosed in quotation marks or a String literal which maps to a *workingVariable* in the XML script. Every *businessObjectTag* is treated as *workingVariable* by the SBML engine. The BusinessObjectTransaction thus can contain any number of elements.

The condensed scripting language 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 call another *BusinessObjectTransaction* by specifying the name of the BO, the *anchorTag*, and lists of parameters and optional return values. Conditional statements are defined as either an *IfThen* or an *IfThenElse*. Both statements make a logical comparision of the identifier with the variable and conditionally execute the statements. The *Assign* command can be used to assign a variable to an identifier.

There can be multiple *BOReturn* statements inside a BO and each BOReturn can have a unbounded number of the following statements: *HigherTagStart, HigherTagEnd*, Tag. HigherTagStart creates a opening tag corresponding to the variable name that is specified and HigherTagEnd creates a closing tag. The Tag statement can be used to create a tag with the name specified after the first whitespace and the value specified after the second whitespace. This scheme allows for the creation of nested tags and also tags dynamically named using variables.

Figure 4 illustrates this concept with a condensed-language script. Appendix 2 contains the XML version of the same script, which is more than four times as long..

```
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 4. Condensed script for SBML

8. Conclusions

The SBML publish/subscribe capability worked well in a demanding environment that included a large variety of clients and development environments. Future enhancements of SBML should increase usability and performance. SBML will be available as open source.

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] 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

[3] Pullen, J., D. Corner and S. Singapogu, "Scripted Battle Management Language Web Service Version 2," IEEE Fall 2009 Simulation Interoperability Workshop, Orlando, FL, 2009

[4] 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

[5] Pullen, J. *et al.*, "An Expanded C2-Simulation Experimental Environment Based on BML," IEEE Spring Simulation Interoperability Workshop, Orlando, FL, 2010

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

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

Author Biographies

DR. J. MARK PULLEN is Professor of Computer Science at George Mason University, where he serves as Director of the C4I Center and also heads the Center's Networking and Simulation Laboratory. He has served as Principal Investigator of the XBML and JBML projects.

DOUGLAS CORNER is a member of the staff of the George Mason University C4I Center. He is the lead software developer on the SBML scripting engine.

SAMUEL SINGAPOGU is a member of the staff of the George Mason University C4I Center. He is the lead script developer on the SBML scripting engine.

BHARGAVA BULUSU is a member of the staff of the George Mason University C4I Center. He is a script developer on the SBML scripting engine.

Appendix 1: BNF Representation of Condensed Scripting Language

<BOInput> ::= BOInput { <BOTransaction> [<BOInput>] | }

<BOTransaction> ::= BOTransaction <identifier> [mvwv=<identifier>,assignTo=<identifier>][iterator=<identifier>]
(<parameterList>)([parameterList]){ <BOTTransactionElements> }

<BOTTransactionElements> ::= <BOTTransactionElements><BOTTransactionElement> <BOReturnElements> ;

<BOTTransactionElement> ::= <getRow>
| <getList>
| <getColumn>
| <put>
| <call>
| <conditional>
| <assign>
| <singleStatement>
| <abort>
| <BoReturn>
| <comment> | Debug; | Commit;

<columnReference> ::= <identifier> <relation> <variable> [,<columnReference>]

<parameterList> ::= <variable> [, <parameterList>] | ;

<literalValue> ::= "<identifier>"

<relation> ::= EQ|NE|GT|LT|GE|LE|EQI

<variable> ::=<literalValue>|<identifier>

<getRow> ::= GetRow <identifier> <identifier> (<columnReference>) ;

<getList> ::= GetList <identifier> <identifier> (<columnReference>) ;

<getColumn> ::= Get <identifier> <identifier> (<columnReference>) ;

<put> ::= Put <identifier> (<columnReference>) ;

<call> ::= Call <identifier> <identifier> (<parameterList>)([parameterList]) ;

<conditional> ::= <ifThen> | <ifThenElse>

<ifThen> ::= IfThen(<identifier><relation><variable>) { <BOTTransactionElements> }

<ifThenElse> ::= IfThenElse(<identifier><relation><variable>) { <BOTTransactionElements> }
{ <BOTTransactionElements> }

<assign> ::= Assign <identifier> <variable>

<abort> ::= Abort <literalValue>;

<BOReturn> ::= BOReturn { <multipleBOReturnElements>

<multipleBOReturnElements> ::= <multipleBOReturnElements> <BOReturnElement> }

<BOReturnElements> ::= BOReturnElement { <BOReturnElements> | }

<BOReturn> ::= BOReturn { <multipleBOReturnElements>

<BOReturnElement> ::= BOReturnElement HigherTagStart <variable>;
| BOReturnElement HigherTagEnd <variable>;
| BOReturnElementTag <literalValue> <identifier>;
| ;

<Identifier> ::= <lowerCaseAndOtherCharacters>[<Identifier>] | <upperCaseCharacter>[<Identifier>]

<lowerCaseAndOtherCharacters > ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z|->|0|1|2|3|4|5|6|7|8|9|.

<upperCaseCharacter> ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

<comment> ::= // "… text …"

NOTE: Whitespace (any number of blanks and carriage returns) is the delimiter, unless other delimiter is indicated.

Appendix 2: XML Script Example

```
<?xml version="1.0" encoding="UTF-8"?>
<BusinessObjectInput xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="BusinessObjectTransactionInterpreterSchema.v0.18.xsd">
<!-- Fragment of code from WhatWhenPush -->
    <call>
        <boName>TaskeeWhoPush</boName>
        <anchorTag>TaskeeWho</anchorTag>
        <parameter>
            <workingVariable>task_act_id</workingVariable>
        </parameter>
    </call>
. . .
    <BusinessObjectTransaction>
        <transactionName>TaskeeWhoPush</transactionName>
        <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>
        <tableQuery>
            <databaseTable>act_res</databaseTable>
            <queryAction>PUT</queryAction>
            <columnReference>
                <columnName>act_id</columnName>
                <workingVariable>task_act_id</workingVariable>
            </columnReference>
            <columnReference>
                <columnName>act_res_index</columnName>
                <workingVariable increment="Yes">act_res_index</workingVariable>
            </columnReference>
            <columnReference>
                <columnName>cat_code</columnName>
                <literalValue>RI</literalValue>
            </columnReference>
            <columnReference>
                <columnName>authorising_org_id</columnName>
                <workingVariable>unit_id</workingVariable>
            </columnReference>
        </tableQuery>
        <tableQuery>
            <databaseTable>act_res_item</databaseTable>
            <queryAction>PUT</queryAction>
            <columnReference>
                <columnName>act_id</columnName>
                <workingVariable>task_act_id</workingVariable>
            </columnReference>
            <columnReference>
                <columnName>act_res_index</columnName>
                <workingVariable>act_res_index</workingVariable>
            </columnReference>
            <columnReference>
                <columnName>obj_item_id</columnName>
                <workingVariable>unit_id</workingVariable>
            </columnReference>
        </tableQuery>
        <BusinessObjectReturn >
            <BusinessObjectReturnElement>
                <tag>Result</tag>
                <literalValue>OK</literalValue>
            </BusinessObjectReturnElement>
        </BusinessObjectReturn>
    </BusinessObjectTransaction>
</BusinessObjectInput>
```