Performance and Usability Enhancements to the Scripted BML Server

Dr. J. Mark Pullen, Douglas Corner, and Lisa Nicklas
C4I Center
George Mason University
Fairfax, VA 22030, USA
+1 703 993 3682
{mpullen, dcorner, lnicklas}@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 Scripted BML Server (SBML), which is 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. Previous SIW papers have described the development of the SBML concept and its various enhanced capabilities. This paper describes recent enhancements including BML namespaces, BML message push/pull logging/replay, threaded and parallel operation for improved performance, a Condensed Scripting Language (CSL), and a NATO OPORD schema.

1. Overview

This paper describes a set of enhancements to the Scripted BML Server (SBML), which was presented previously in references [2] through [5].

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. In recent implementation, this has been 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 (or mapping file, 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 the database can be implemented and tested rapidly
- the ability to change the service rapidly reduces cost and facilitates prototyping
- the script 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; reference

[4] describes a publish/subscribe enhancement of SBML, and [5] describes development of the Condensed Scripting Language, which is more programmer-friendly than the original XML scripts. (The Appendix of this paper updates Appendix 1 of [5].) SBML was used to support NATO MSG-048 experimentation in 2008 and 2009, as described in [6] and [7].

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 [8], passes java objects through Red Hat's Hibernate persistence service, which performs the actual database interface function. Version 2 implements a publish/subscribe capability as shown in figure 2, 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.

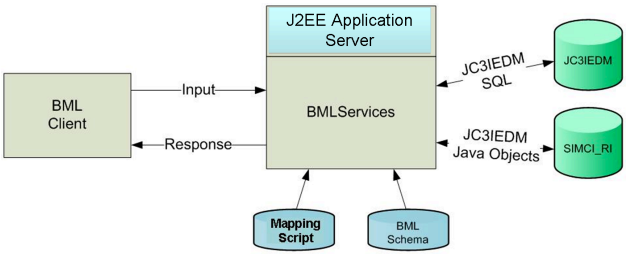

Figure 1. SBML Configuration

The BML/JC3IEDM conversion process is accomplished under control of the scripting language, which is described in [3] and [5].

This paper reports on recent enhancements to SBML including modular JC3IEDM representation, BML message push/pull logging/replay, threaded and parallel operation for improved performance, and a NATO OPORD schema. It also updates the description of the CSL.

3. Back-to-Back Client for Hybrid BML

The environment under study consists of two BML domains, one based on exchange of native JC3IEDM; the other based on BML. The interface to the native JC3IEDM domain is via the JC3IEDM SDK Reference Implementation (RI) developed as part of OneSAF by Northrop Grumman and part of the SIMCI Combined Project described in [8]. The native JC3IEDM system supports both Web Services and the JBoss remoting interface. The BML domain supports a Web Services interface and has been described in [2] and [3].

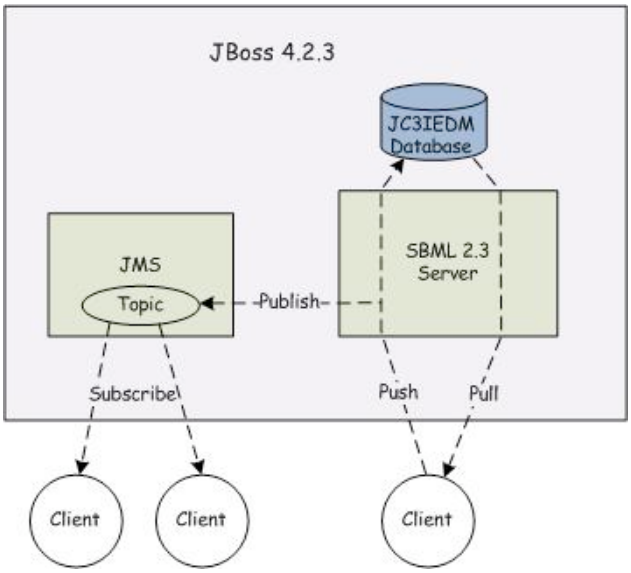

Figure 2. Publish/Subscribe Architecture for SBML

Earlier parts of this project developed an interface to transmit orders to the RI. A BML order is submitted by a client to the SBML Server via web services. The SBML service translates the BML to a series of JC3IEDM database transactions. These are either written directly to a JC3IEDM database via SQL calls or indirectly through the RI, represented as java objects used with Hibernate.

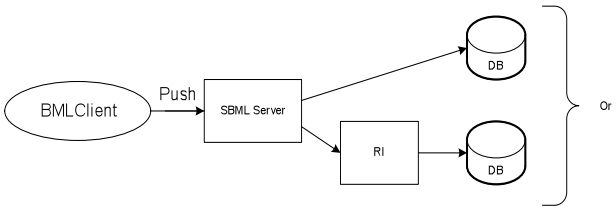

Figure 3. Back-to-Back SBML Client

All transactions once received and submitted successfully are transmitted to other clients via an asynchronous publish/subscribe service previously described in [3]. SBML supports two general types of transactions: Orders and Reports. Orders are complex and infrequent. Reports are generally simpler and, as they report information such as position, may occur quite frequently. The ultimate consumer of an Order is a simulation system, which will then carry out the order. There may be multiple consumers the Order; it is even more likely that there will be multiple receivers of Reports, because a number of command and control (C2) systems may need to display their contents for situational awareness. Therefore, use of publish/subscribe is important for distributing Reports because of their frequency (which otherwise would require frequent polling by all consuming systems) and the potential that they may be required by multiple clients.

Clients of one of the two domains are not subscribers to the other domain because they use different format for the storage function. Therefore, transmission of Reports from one domain to the other is done through a proxy client, known as the Back to Back (B2B) client, that subscribes to both domains and transmits BML format documents from one to the other. The B2B client receives a publication from one domain, translates it from one format to the other (e.g. JC3IEDM XML to BML XML) and submits it as a transaction to the other domain. B2B therefore has the role of being a subscriber to one domain and a client to the other. This capability enabled SBML to play a hybrid role in supporting MSG-048's 2009 experimentation, whereby US systems using the JC3IEDM and RI interoperated with NATO systems using a MySQL database, as shown in figure 4.

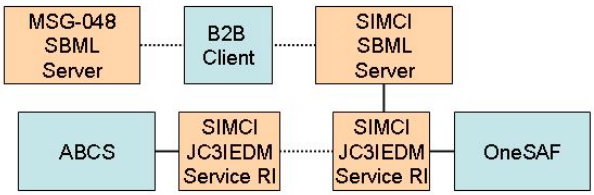

Figure 4. Back-to-Back Client used with MSG-048

4. Pushing a Complete Thought in JC3IEDM

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 will be committed and written in its final form to the database. The OneSAF RI interface [8] supports a different approach to transaction control, where all tables updates are combined into a single "push" to the RI interface. This approach is in consonance with the MIP documentation [10], 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 the SBML server in the form of the SBML `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 an `ri_end` command, causing the set of linked objects is passed to the RI client code which translates the linked objects into a JC3IEDM XML document. This document then is passed to subscribers of any objects 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.

5. BML Namespaces

A shortcoming of the previous version of SBML web services was that it ignored namespaces in the BML input and did not return output BML that used namespaces. This made validation of the BML difficult and required that there exist no coinciding names in the various namespaces used. The current version of SBML web services expects that (1) any input BML is specified with the correct namespaces and (2) any generated output BML also has namespaces specified. Thus we now expect all input and output BML to validate against BML schemas. This improvement required changes to the SBML web service as well as the scripting.

Within the server, an option has been added to the input properties file that will turn on validation of all input BML. Depending on whether the client interface provides validation of BML input, the user may request that SBML services perform the validation. 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 the existing scripts to support this capability were one-time and straightforward. Achieving the improvement required 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. The correct prefix then must used in any references to BML elements within the script's body. Figure 5 shows a sample namespace mapping file and a section of CSL that shows how to specify namespaces within the script. In this example, the `OrderID` element is part of the BML namespace URN http://netlab.gmu.edu/IBML. The "bml" prefix defined at the beginning of the script must be used with every mention of `OrderID.`

```
<?xml version="1.0" encoding="UTF-8"?>
<BusinessObjectInput …
   <!-- Define URI to prefix mapping for namespaces -->
   <Namespace>
      <uri>http://netlab.gmu.edu/IBML</uri>
      <prefix>bml</prefix>
   </Namespace>
   <Namespace>
      <uri>http://netlab.gmu.edu/JBML/BML</uri>
      <prefix>newwho</prefix>
   </Namespace>
   <Namespace>
      <uri>http://netlab.gmu.edu/JBML/MSDL</uri>
      <prefix>msdl</prefix>
   </Namespace>
   <Namespace>
      <uri>urn:int:nato:standard:mip:jc3iedm:3.1a:oo:2.0</uri>
      <prefix>jc3iedm</prefix>
   </Namespace>
</BusinessObjectInput>

...

   // LowerOrderPush ******************************************
   BOTransaction LowerOrderPush()()
   {
      Assign [bml:OrderID] OrderIDwv;
      IfThen (OrderIDwv EQ "")
      {
         Abort OrderID does not exist in the input file;
      }
      GET  ACT act_id (name_txt EQ [bml:OrderID]);
      Assign act_id order_act_id;
      IfThen (act_id NE "")
      {
         Abort OrderID already exists;
      }
      GET  UNIT unit_id assignTo=taskerWhoUnitID
         (formal_abbrd_name_txt EQ
               [bml:TaskerWho/bml:UnitID]);
      IfThen (taskerWhoUnitID EQ "")
      {
            Abort Invalid or Absent TaskerWho in the Order;
      }
...
```

Figure 5. Namespace Example

6. Multithreaded Operation

The performance of SBML web services has been improved by providing for multithreaded operation. In previous versions, client requests of the web service were serialized. In the current version, multiple requests are processed simultaneously as much as possible 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 the SBML services now has its own connection to the MySQL database.
  - Each instance of the SBML services 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 the SBML web services must parse the input scripts.
- Semaphores exist to insure serial access to the remaining global resources.
  - Since there can only be one connection to the RI, that connection to the RI must now be shared between all instances requiring a semaphore to control access.
  - Initialization of SBML is now protected by a semaphore.
- Setting and using OIDs for pushing to the RI required synchronized increment and access methods.
- Implementing the increment attribute on a database column table was removed from SBML server 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 to be used in heavily multithreaded systems.

Table 1 shows the speedup in using multiple threads when pushing reports to a local MySQL database. For these measurements, three hundred reports were pushed with a 0.2 sec interval between pushes. The timings were collected on a machine with four cores. Note that the expected best speed-up occurred around four threads.

Table 1. Multithreaded Speedup

Threads	Speedup	Avg Exec Time for One Report Push
1	1.0	0.51 s
2	1.86	0.27 s
4	2.38	0.21 s
6	2.37	0.22 s
60	1.73	0.29 s

7. Logging/replay in SBML

A primary use of the SBML server is to support integration of C2 systems with military simulations. In this context, often it is required that the outputs of a coalition of C2 systems and simulations be replayed. This capability has been added to the SBML server. To allow for replay, the SBML server can now optionally create a log file of all transactions. A client has been developed that will read the log file and resubmit the transactions in the same order and with the same timing. The result is that all SBML subscriber clients will see the same series of simulation events as it did during the initial run.

8. OPORD Schemas for SBML

Under subcontract to L3-Comm Inc., the GMU C4I Center and Atlantic Consulting Services, Inc. (ACS) produced an Integrated BML Architecture for the US Army CIO/G6 [9]. An important part of this effort is a US Army OPORD, developed as completely as practical, along with an SBML script implementing it. This in turn is being used to create a NATO OPORD and associated SBML script for presentation to the new NATO Technical Activity MSG-085 described in [7] and [8]. We expect to report on these activities in greater detail, in future papers.

9. Conclusions

SBML is intended for rapid, flexible prototyping of BML services. It has been developed into a well-rounded capability for generating Web services quickly and with a low error rate, based on a simple scripting language. While SBML 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. SBML was used for this purpose in support of NATO MSG-2009 in 2008 and 2009.

SBML has been extended to support publish/subscribe and a condensed scripting language (CSL). Most recently it has been enhanced to support full XML namespaces for greater flexibility and multithreaded operation for improved performance, as well as logging/replay.

This paper also introduced the concept of OPORD schemas; we plan a more extensive future paper on that topic.

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] 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

[5] 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 Interoper-ability Workshop, Ottawa, Canada, 2010

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

[7] Heffner, K. *et al.*, "NATO MSG-048 C-BML Final Report Summary," IEEE Fall 2010 Simulation Interoperability Workshop, Orlando, FL, 2010

[8] 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

[9] George Mason University C4I Center, *Integrated Battle Management Architecture Final Report*, contract deliverable to US Army CIO/G6, February 2010, available at http://c4i.gmu.edu/BML

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

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 is a Fellow of the IEEE, Fellow of the ACM and license Professional Engineer. He has served as Principal Investigator of the XBML, and JBML, IBML and SBML 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.

LISA NICKLAS is a member of the staff of the George Mason University C4I Center. She is the lead implementer for SBML interoperability with the US Army Battle Command and OneSAF systems.

Appendix: Updated BNF Representation of Condensed Scripting Language

The SBML server was implemented using an XML based scripting language. The available of software for parsing XML made the implementation of the scripting interpreter straightforward. Approximately 10,000 lines of script code have been developed for a variety of applications. The language, however, is verbose and therefore difficult to code and to read. Consequently a more concise language, Condensed Scripting Language (CSL) was developed as a replacement for SBML XML scripting. A compiler was developed implemented using JavaCC, a Java based parser generator now supported by Oracle. We will use CSL for all future SBML scripting.

In order to sustain the existing scripts, a reverse compiler was developed that converts XML script to CSL scripts. This was developed using XSLT and is available along with the CSL compiler on the SBML web site at http://www.netlab.gmu.edu/OpenBML. A user's guide for writers of CSL scripts also is available there.

Using JavaCC, the implementer first defines first the lexical elements in terms of tokens and then defines the syntactical elements using a form of Backus-Naur Form (BNF). A previous paper discussed CSL but the BNF definition of the language was incomplete. It is included here.

```
<program> ::= <BOInput> <LBRACE> (<CSLCode>)+ <RBRACE> <EOF>

   <CSLCode> ::= (<BOTransaction> | <Routine>) <boName> <attrs>
      (<LPAREN> (<boparameter>)* <RPAREN> )
      ?(<LPAREN> (<boReturn>)* <RPAREN> )? <LBRACE> (<parseLine)+ <RBRACE>

   <boparameter> ::= <identifier>

   <boReturn> ::= <identifier>

   <attrs> ::= ("mvwv=" <identifier> "assignTo=" <identifier>)? | ("iterator=" <identifier>)?

   <parseLine> ::= <assign>
      | <call>
      | <abort>
      | <debug>
      | <commit>
      | <ri_start>
      | <ri_end>
      | <comment>
      | <assert>
      | <put>
      | <get>
      | <ifthen>
      | <ifthenelse
      | <BOReturn>
      | <BOTransaction>
      | <BOReturnElement>
      | <BOReturnElementTag>
      | <HigherTagStart>
      | <HigherTagEnd

   <assign> ::= "Assign" <variable> <identifier> (<transformRoutine>)? <br>

   <transformRoutine> ::= <identifier>

   <call> ::= "Call" <boName> <anchorTag>(<LPAREN> (<callParameter>)* <RPAREN>)
               ? (<LPAREN> (<boReturn>)* <RPAREN>)? <br>

   <boName> ::= <identifier>

   <anchorTag> ::= <idenetifier>

   <callParameter> ::= <variable>

   <boReturn> ::= <identifier>

   <abort> ::= "Abort" (~["\n","\r"])* <br>
```

```
<debug> ::= "Debug" <br>

<commit> ::= "Commit" <br>

<ri_start> ::= "ri_start" <rootTableName> <key_id> <br>

<rootTableName> ::= <identifier>

<key_id> ::= <identifier>

<ri_end> ::= "ri_end" <br>

<assert> ::= "Assert" <LPAREN> <var1> <rel> <var2> <RPAREN> <quote> (~["\n","\r"])* <br>

<comment> ::= "//" (~["\n","\r"])* <br>

<put> <tableName> <putColRef> <br>

<tableName> ::= <identifier>

<putColRef> ::=   (( <LPAREN> <colName> (<incOp>|<assignOp>) <val> <RPAREN>))+

<colName> ::= <identifier>

<incOp>  ::= ">"

<assignOp> ::= "="

<val> ::= <variable>

<get> ::= <gettoken> <tableName> <resultName> ("assignTo=" <identifier>)
           ? ("orderBy=" <identifier>)? <getColRef> <br>

<getToken> ::= "GET" | "GETList" "GETRow" "GETMax"

<getColRef> ::= (<LPAREN> <colName> <rel> <val><RPAREN>)+

<rel> ::=  "EQ"|"NE"|"LT"|"GT"|"LE"|"GE"

<ifthen> ::= <LPAREN> <var1> <rel> <var2> <RPAREN> <LBRACE> <<parseLine>)+ <RBRACE>

<var1> ::= <identifier>

<var2> ::= <variable>

<ifthenelse> ::= <LPAREN> <var1> <rel> <var2> <RPAREN>
                 <LBRACE> <<parseLine>)+ <RBRACE> <LBRACE> <<parseLine>)+ <RBRACE>

<BOReturn> ::= <BOReturn> <LBRACE>(BOReturnElement)+ <RBRACE>

<BOReturnElement> ::= <LBRACE> (<BOElementTag> | <HigherTagStart> | <HigherTagEnd>)+ <RBRACE>

<BOReturnElementTag> ::= <tag> <tagVal> <br>

<tag> ::= <identifier>

<tagVal> ::= <variable>

<HigherTagStart> ::= <variable>

<HigherTagEnd> ::= <variable>

<variable> ::= <identifier> | <quote> <identifier <quote> | "[" <identifier> "]"
<identifier> ::= (<lowercaseAndOtherCharacters>)+

<lowercaseAndOtherCharacters> ::= ["A"-"Z","a"-"z","0"-"9",".","_","/",":","<","-","*", "'"]

<quote> ::= [\"]

<br> ::= [;]
```

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