Dynamic Publish/Subscribe Topics in the Scripted BML Server

Lisa Nicklas, Dr. J. Mark Pullen, and Douglas Corner
C4I Center
George Mason University
Fairfax, VA 22030, USA
+1 703 993 3682
{lnicklas ,mpullen, dcorner}@c4i.gmu.edu

Keywords:
Battle Management Language, Web Services, Publish/Subscribe

Version 2.3 Scripted BML Server (SBMLServer) used JBoss messaging's implementation of Java Messaging Service (JMS) version 1.1 with predefined static topics to which subscribers could subscribe at run time. Under this approach, some topics that a subscriber required might not have been defined when needed. Also, to add a new topic requires updating a topic configuration file read by SBMLServer at startup and adding a new message bean to be deployed by JMS. To allow for dynamic topic definition, we have modified SBMLServer to use one static topic plus the message selectors available in JMS to filter output to clients. This modification provides one Web method that allows clients to retrieve a list of message selectors currently defined within SBMLServer, and another Web method that allows clients to define a new message selector. All clients must subscribe to the one static topic optionally specifying message selectors defined previously with the server. The server keeps track of the different message selectors and tags any outgoing JMS messages with the correct message selector based on the associated query. This second level of filtering for topics allows the JMS middleware to perform message filtering based on the message selectors. Thus the client need have only one static topic that is filtered to clients, based on message properties. This paper describes the design rationale and implementation of the features, and concludes with projection of an enhancement that will allow the use of JMS with an SBML interface available across different possible client languages.

1. Overview

This paper describes an extension to the Scripted BML Server (SBMLServer), past versions of which were presented previously in references [2] through [6]. The purpose for the extension is to support runtime definition of filter topics for publish/subscribe operation.

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 that is 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 input 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; [4] describes a publish/subscribe enhancement of SBML; [5] describes development of the Condensed Scripting Language, which is more programmer-friendly than the original XML scripts; and [6] describes a set of utility and performance enhancements to the SBMLServer.

SBMLServer was used to support NATO MSG-048 experimentation in 2008 and 2009, as described in [7] and [8]. This resulted in a broader understanding of the range of requirements for effective prototyping support of C2-simulation in experimentation.

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 [9], 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.

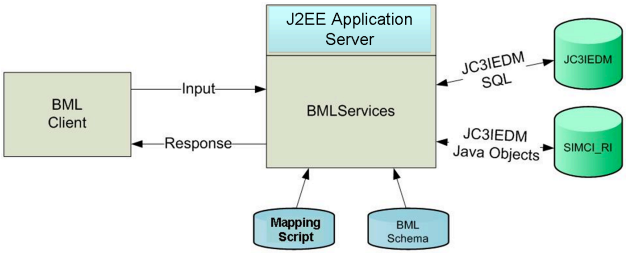

Figure 1. SBML Configuration

This paper reports on an extension to SBMLServer that permits dynamic topic assignment in publish/subscribe operation. The general publish/subscribe architecture employed is shown in Figure 2.

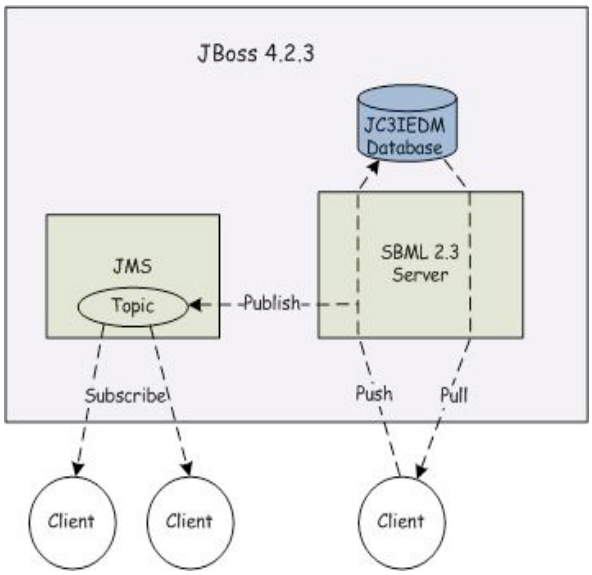

Figure 2. Publish/Subscribe Architecture for SBML

3. Advantages of Publish/Subscribe Capability

Under version 2.3 of SBMLServer, a publish/subscribe capability was added using JBoss messaging's implementation of JMS version 1.1 [4]. In older versions of SBMLServer, it was necessary for clients to poll the server to receive BML that was being submitted by other clients. This was inefficient in several ways:

- The server must re-read 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

The load resulting on the server from polling and the large amount of redundant network traffic clearly is inefficient. It is recognized that, in almost any case, simulation systems must constrain Report generation because they nearly always will be able to produce Reports faster than the C2 systems and infrastructure can process those Reports [8]. In the polling environment, this situation becomes far worse; the server becomes a significant bottleneck.

However, analysis by the MSG-048 technical subgroup has indicated that a more flexible publish/subscribe mechanism is desirable. Previously, SBMLServer's implementation of publish/subscribe predefined static topics to which subscribers could subscribe to dynamically. During initialization, SBMLServer reads in a configuration file *topicDefinitions.xml* that contains statically defined topics. This configuration file includes topic names and associated XPath query strings. The

SBMLserver then uses each XPath query string in this configuration file to determine whether to publish BML to a particular topic. Clients can create a subscriber to the static topics they want, thereby filtering the BML that they would receive. This approach greatly improved efficiency in that:

- each message is posted to each topic at most one time
- poll requests to the server are not required
- database queries responding to polling were eliminated.

However, there was a major drawback to this approach 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 SBMLServer, adding a new topic to the server requires the following manual steps:

- updating the *topicDefinitions.xml* file with the new topic name and 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 application server JBoss must be restarted for the new topic to be available. A restart of SBMLServer was required to pick up the new topic since the topic definition file is only read during server initialization, while a restart of JBoss was needed to create a message bean for the newly defined topic.

4. Implementation of Dynamic Topics

To implement dynamic topics, version 2.4 of SBMLServer makes use of JMS message selectors with a single static topic. This allows SBMLServer to provide client controlled filtering of BML. The JMS message selectors provide a way for the clients to be more selective about the messages that they receive for a given topic. Figure 3 shows how message selectors are used for publish/subscribe with SBMLServer. Note that there is just one static topic defined named SBMLTopic.

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 in by SBMLServer at initialization. A sample *msgSelectors.xml* is shown in figure 4. Each message selector has a name and an associated search string that must be a valid XPath query.

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

- getMsgSelectors()
- addMsgSelector(String search)

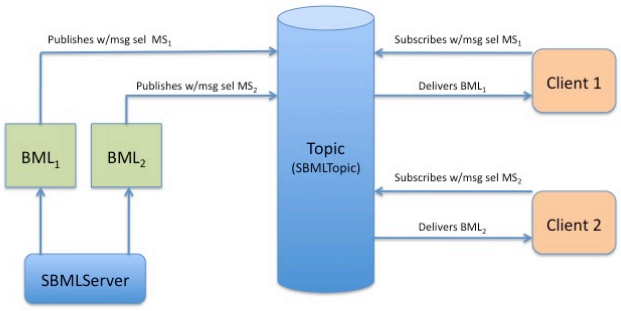

Figure 3. Message Selectors in SBML

```
<?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 4. Sample msgSelectors.xml

The *getMsgSelectors* routine allows clients to get a list of currently defined message selectors. SBMLServer returns to the client a list of message selector names and associated XPath queries. Figure 5 shows sample Java code that requests a list of current message selectors and then outputs the message selectors' names and associated search strings.

```
import edu.gmu.c4i.sbmlclientlib.SBMLClient;
import edu.gmu.c4i.sbml.MsgSelector;
...
// create a client to the SBMLserver webservice
SBMLClient sbmlClient = new SBMLClient(host);

// get the list of Message Selectors
List<MsgSelector> selectors =
sbmlClient.getMsgSelectors();

// Print out each message selector defined on SBMLServer
for (int i = 0; i < selectors.size(); i++)
{
   MsgSelector ms = selectors.get(i);
   String name = ms.getName();
   String search = ms.getSearch();
   System.out.println("Message Selector " + i + ": name = " +
name + " with search = " + search);
}
```

Figure 5. Java code to get message selectors

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. If the XPath formatted query provided by the client is the same as a query that was already requested by a previous client, the server does not generate a new message selector name but instead returns the name generated by the previous client request.

```
...

import edu.gmu.c4i.sbmlclientlib.SBMLClient;

...

// create a client to the SBMLserver webservice
SBMLClient sbmlClient = new SBMLClient(host);

// add a new Message Selector
String s = null;
try
{
   s = "//newwho:ListWho";
   String selectorName = sbmlClient.addMsgSelector(s);
   System.out.println("added msg selector " + selectorName);
}
catch (Exception e)
{
   System.out.println("Unable to add Message Selector " + s +
" " +
                  e.getMessage());
}

...
```

Figure 6: Java code to add msg selector

Sample code to define a new message selector is shown in When the client creates a subscriber, it always subscribes to the one static topic, SBMLTopic. When creating a subscriber, the client has the option of specifying one or more message selector names each associated with a specific search string. This allows the client to filter output BML that it will receive. With these two new web services, a client can define a set of new search strings and then subscribe to BML with a specification matching only these locally defined searches. Sample code to create a subscriber with a single message selector is shown in the Appendix.

Internally, the server manages a cross reference of message selector names with XPath queries. The server tags any outgoing BML transactions (JMS messages) with the correct message selector, based on the associated search string. In addition, other fields are set within the message properties that are available to client subscribers. These currently include the type of BML (at present, either IBML or C-BML) and the root tag of the output BML. Under this approach, SBMLServer only publishes each message (BML) once to SBMLTopic. The message is passed over to clients, based on their configured subscription. This is a second level of filtering for topics that will allow for the JMS middleware to handle the message filtering based on the message selectors. In this way you can have one static topic that is then filtered to clients based on message properties.

5. Planned Improvements to SBMLServer

For the initial implementation of dynamic topics, we have continued to use the JBoss 4.2.3 implementation of JMS directly. However, we are planning an enhancement that would abstract away the use of a Java specific messaging library. A drawback of the current implementation is that creating clients in languages other than Java is not straightforward. We are experimenting with a RESTful version of SBMLServer using the RESTEasy implementation of JAX-RS for JBoss [9]. With this RESTful version of SBMLServer and the HornetQ RESTful interface [10], clients could be written in any language that has access to a HTTP client library.

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 coding 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-048 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.

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 Interoperability Workshop, Ottawa, Canada, 2010

[6] 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

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

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

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

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
Java Code for Client Using Dynamic Publish/Subscribe Topics

```
...
import javax.jms.*;
import javax.naming.*;
import java.text.SimpleDateFormat;
...
   // Imbedded callback class that will receive published messages.  One instance per subscription
   public static class ExListener implements MessageListener {

      String id;           //  msg selector name

      // Constructor
      ExListener(String id) throws Exception {
         this.id = id;
      }  // ExListener()

      public void onMessage(Message msg) {
…
            // Print contents from returned BML string.
            Date sendingTs = new Date(msg.getJMSTimestamp());
            String bmlType = msg.getJMSType();
            String rootNode = msg.getStringProperty("content");
            Date dateNow = new Date();
            SimpleDateFormat df = new SimpleDateFormat("yyyy'/'mm'/'dd HH:mm:ss");
            System.out.printf(
               "%4d  %s  BMLType:%s Sent:%s MsgSelector:%-15s \n",
               ++count, df.format(dateNow), bmlType, sendingTs.toString(), id);
            System.out.printf("\tContent dump:%s\n", msg);
…
      }  // onMessage()
   }  // class MessageListener
...
      TopicConnection conn = null;
      TopicSession session = null;
      Topic topic = null;
      TopicSubscriber recv = new TopicSubscriber();

      // Set up connectivity to JNDI Service
      Properties prop = new Properties();
      prop.put(Context.INITIAL_CONTEXT_FACTORY, "org.jnp.interfaces.NamingContextFactory");
      prop.put(Context.PROVIDER_URL, "jnp://" + host + ":1099");
      InitialContext iniCtx = new InitialContext(prop);
      if (iniCtx != null)
         System.out.println("InitialContext created");
      TopicConnectionFactory tcf = (TopicConnectionFactory) iniCtx.lookup("ConnectionFactory");
      conn = tcf.createTopicConnection();
      topic = (javax.jms.Topic) iniCtx.lookup("topic/SBMLTopic");

      session = conn.createTopicSession(false, TopicSession.AUTO_ACKNOWLEDGE);
      recv = session.createSubscriber(topic, name + " = 'true'", false);
      recv.setMessageListener(new ExListener(name));
      System.out.println("Subscribing to " + name + " with search = " + search);

      conn.start();
      System.out.println("Waiting for messages");

      while (true) { // Wait for messages to be delivered to the onMessage method

}
```