Scripted Battle Management Language Web Service Version 1.0 Operation and Mapping Description Language

Dr. J. Mark Pullen, Douglas Corner and Samuel Singapogu
C4I Center
George Mason University
Fairfax, VA 22030, USA
+1 703 993 3682
{mpullen, dcorner, ssingapo}@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 a JC3IEDM database which is accessed via a Web service. That service must support business objects representing elements of the BML, expressed in terms of the entities of the JC3IEDM. Typical business objects are the classical elements of military plans and orders: who (more specifically, the tasker who, taskee who, and affected who), what, when, where, and why for each task. The service must implement a mapping from the semantics of the BML to the defined information structure of the JC3IEDM. Earlier BML implementations have contained such a mapping, hard-coded in Java programs. The GMU C4I Center has developed a new approach to implementing such a Web service, based on the notion of an interpreter module which takes as its input the schema of the Web service and a script, coded in XML, that defines the mappings concisely. The GMU interpreter will be available as open source software. While it may offer lower performance than the hard-coded implementation, such a scripted service has the virtues that it is quicker and easier to change and also requires a lower level of expertise for development, once the interpreter has been completed. We advocate adopting this style of script and service as the technical definition and reference implementation, respectively, for SISO's BML standards. This paper provides details of the scripting language and its implementation, in support of that advocacy.

1. Overview

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 used of the Extensible Markup Language (XML) and Web service (WS) technology, 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

This paper describes the Scripted BML (SBML) server. SBML is a scripting engine that implements a BML WS by converting BML data into a database representation and also retrieving from the database and generating BML as output. 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.

The current SBML implementation and scripts support two JC3IEDM database interfaces: one is a direct SQL interface, used with a MySQL database server. The other, SIMCI_RI (to be implemented in version 2), passes java objects through Red Hat's Hibernate persistence service, which performs the actual database interface function. Version 2 also will implement a publish/subscribe capability.

The BML/JC3IEDM conversion process is accomplished under control of the scripting language which is described in this document.

2. Usage

The scripted BML WS has been used in two significant demonstrations. In November 2008, BML was demonstrated along with a JC3IEDM Reference Implementation in a C2-Simulation interoperation as part of the 2008 US Army SIMCI Combined Project [2], with the architecture shown in Figure 1. In December 2008, BML was demonstrated at the InterService/Industry Training, Simulation and Education Conference (I/ITSEC-08), by the NATO Modeling and Simulation Group Technical Activity 48 (MSG-048)[3]. Six different national software systems interaction over BML, in different C2-simulation combinations such as Netherlands-France and Norway-France.

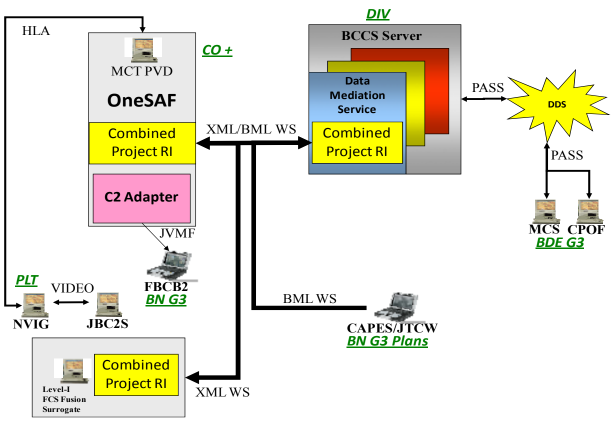

Figure 1. SIMCI Combined Project Demonstration

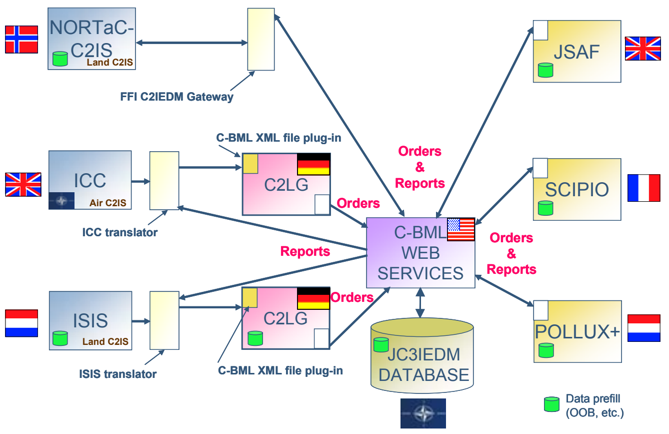

Figure 2. MSG-048 Demonstration at I/ITSEC'08

3. Operation

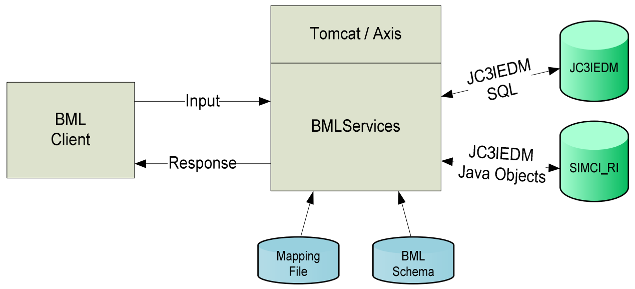

Figure 3. Scripted BML WS Operating Configuration

The BML Input may be a *push* containing data (e.g. an Order) or may be a *pull* request for data. If successful, a push returns a response indicating success; a pull returns the requested data, formatted in BML per the script. If unsuccessful, either push or pull will return an error message. The SBML service is driven by elements of the BML that are individually processed by the script. These elements are XML aggregates, known as *BusinessObjects* (BO). (Alternately, they may be described by their grammatical role; they are *constituents* of the BML grammar [4].)

The SBML service is named BMLServices as it runs under Axis/Tomcat. Eight methods are available through the Web service. These are distinguished by names indicating:

Push or pull
Type of data: Report or Order
Database: Native SQL or SIMCI RI

String	bmlOrderPushDB(String bml);
String	bmlReportPushDB(String bml);
String	bmlOrderPullDB(String bml);
String	bmlReportPullDB(String bml);
String	bmlOrderPushRI(String bml);
String	bmlReportPushRI(String bml);
String	bmlOrderPullRI(String bml);
String	bmlReportPullRI(String bml);

There are two files that control the BML/JC3IEDM conversion. The mapping file contains scripting to process each BO, as an XML subtree rooted at a particular XML tag in the BML input. A BO script contains all the variable definitions and processing instructions needed for that subtree and may be thought of as a subroutine with parameters passed in and return variables passed back. The first phase of BML operation identifies the tags and the BO names with which they are associated. A BML transaction input may cause the invocation of multiple BO. The root of the BML input document is also the name of the root BO. All other BOs are invoked by <call> in the script.

3.1 Working Variables

The basic data element in SBML is a character string. The SBML service creates a number of *workingVariables* (WV) which are associated with strings or aggregates of strings. WVs are used to hold parameters and or intermediate results. They are referred to by name in the script. There are three types of WV:

- Scalar string
- Row, containing multiple named Scalars (to be implemented in version 2)
- List of Scalars or Rows

Scalars are simply referred to by name. List WVs are defined by attribute like this:

```
<workingVariable type="List">
     unitNameTable
</workingVariable>
```

Members of Lists are not explicitly referenced; they are used within a List context and are selected implicitly in the process of iterating over the List (see 3.2 below).

3.2 Iteration

Some BO are repeated multiple times (for example, points in a path). SBML supports two features for repeated processing.

1. A BO may also be executed iteratively through the use of a multi value WV (MVWV). A MVWV contains a sequence of working variables known as a *List*. Iteration is implicit in the sense that the BO is executed once for each element in the List. For each execution, the current element of the List is associated with an *assignTo* variable.
2. When a BO is executed via a <call> a reference to a point in the input data is created. This reference is relative to the calling BO. If the relative reference is satisfied by multiple tags (e.g. a series of points in a line) then the called BO will be executed once for each tag found.

3. A named WV defined as an *iterator* may be associated with a BO. The first time the BO is executed the iterator is initialized to "1". Each time it is executed after that it is incremented. Like all SBML data it is stored as a character string; however the server will return an error if it does not contain an integer value.

4. When generating XML output, if any of the working variables within a <BusinessObjectReturnElement> are of type List that <BusinessObjectReturnElement> will be executed iteratively using each element in the List WV until it is exhausted.

3.3 WV Scope

A BO is invoked in the context of the XML node where it is used. The scope of a WV is local to the BO in which it is used unless it has been defined as global. When a BO is invoked it will have available the following WV:

- Any WVs defined as parameter, with values passed from <parameter> in the invoking BO
- Any WVs defined as global
- Data from the input XML at this node, referenced by <businessObjectTag>

When the BO completes it returns to the invoking BO values of WVs defined by <Return> in the BO's script. These are associated with WVs in the invoking BO by <return> under the <call>

4. Mapping File

The mapping file must be a well formed XML file with root element <BusinessObjectInput>, consisting of multiple <BusinessObjectTransaction> elements. Each <BusinessObjectTransaction> defines a BO. Its UML representation is shown in Figure 4.

4.1 Contents of BO

- Declaration of variables.

- An optional *iterator* attribute which is used to name a scalar WV containing an integer in string format; it is incremented each time the BO is invoked and is used as an index. If it is necessary that a series of elements be indexed from one (1) the WV can be set to one with <assign> otherwise the WV in index from transaction to transaction.

- Optional MVWV elements (MVWV and assignTo) used to specify iterated processing

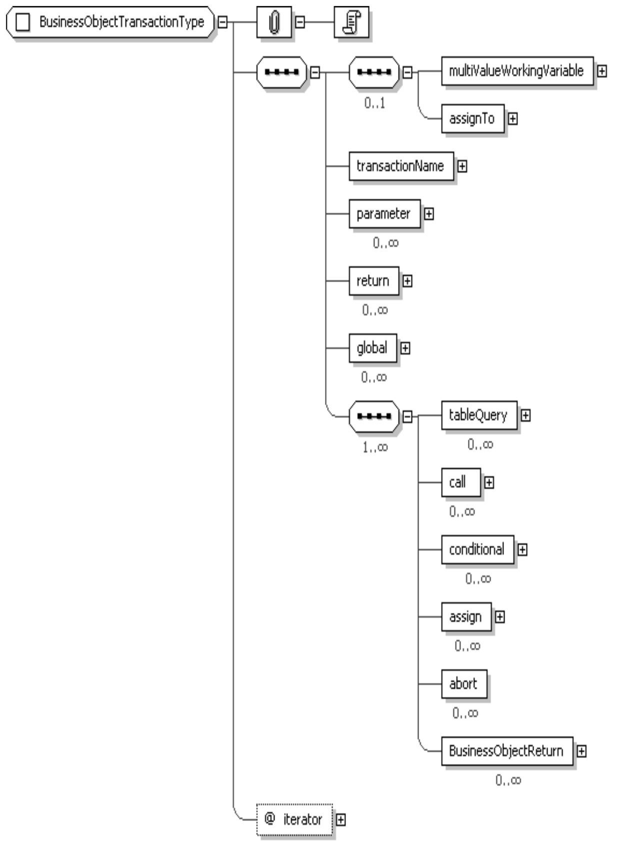

Figure 4. BusinessObjectTransactionType

- One or more processing description elements. These may be in any sequence and may appear any number of times within a BO. Each processing element constitutes a single step in the operation of the SBML WS. Within the BO, the steps of the script are executed sequentially. Iteration may occur within a step, as described in 2.2, and execution of steps may be conditional. However, there is no provision for looping within the BO.

Elements under <BusinessObjectTransaction> are:

<transactionName>	Name of this BO
<parameter>	Names of WVs to be passed to the BO
<return>	Names of WVs to be returned to the invoking BO
<global>	Names of global WVs. If the WV doesn't exist it will be created and set to a null string.
Processing Elements	Scripting Operations (described below)

The processing elements are:

<tableQuery>	Describes a database transaction
<call>	Calls another BO
<conditional>	Makes a conditional branch forward in the processing elements based on a comparison between a WV and another entity
<assign>	Makes an assignment to a WV
<abort>	Aborts this transaction
<commit>	Commits to database; not implemented
<BusinessObjectReturn>	Formats XML output using the results of previous tableQueries

Certain tags occur in multiple places in the mapping file and always have the same meaning:

<workingVariable>	The name of an existing working variable
<businessObjectTag>	The name of an element in the input XML. The element contains the path to the data starting at the root of the current BO.
<literalValue>	A constant string
<relation>	Possible values for comparison in where clause of database query. Possible values: EQ, NQ, GT, LT, GE, LE

4.2 <tableQuery>

The tableQuery, shown in Figure 5, is used for reading from and writing to the database; its elements are are:

<databaseTable> provides the name of the table.

<queryAction> provides the action to be performed, which may be:
GET – read an element from a row or a List of elements from sequential rows in the table
PUT – write a row to the table
UPDATE – update a Row in the table

<resultName> names WV to receive the result of the GET, which may be a scalar or MVWV; attribute List indicates the GET may retrieve multiple rows, in which case the WV created to hold the result will be of type List

<columnReference> (one or more) specifies the contents of a column; usage depends on the queryAction:

If <queryAction> is GET or UPDATE, columnReference elements provide the data for the SQL WHERE clause. If queryAction is PUT, columnReference elements provide the data to be written.

If <queryAction> is PUT, each columnReference specifies the contents of one column. The row to be written consists of all the contents of the columnReferences in this tableQuery.

If <queryAction> is UPDATE, a row is updated with the contents of the elements under updateColumnReference. The row to be updated is specified by <columnReference> elements.

<updateColumnReference> There may be multiple <updateColumnReference> elements. There are used only with UPDATE actions. Each element describes the contents of an update to be made to a column.

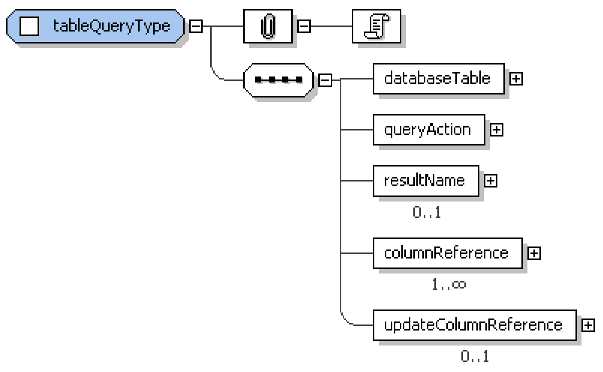

Figure 5. tableQueryType

4.2.1 Elements of columnReference and updateColumnReference

Figure 6 shows how reference to table columns is made.

<columnName> is the column in the database table, as described in 4.2

<relation> is used only with <queryAction> GET and UPDATE; its value specifies the comparison in the query, which may be EQ, NE, GT, LT, GE, or LE

If <queryAction> is GET or UPDATE, one (and only one) of the following is required to specify a value which is used in the SQL WHERE clause:

<businessObjectTag> gives the tag, at the current node in the XML file, associated with the data value to be used

<literalValue> gives a value to be used

<workingVariable> gives the name a WV containing the value to be used. If the WV is of type a List, it must be the first <columnReference>, and only one List WV may be specified in a GET. List WVs are not allowed with PUT or UPDATE. (NOTE: Using a List WV in a GET causes an SQL SELECT to be executed for each element in the List.) If the increment='yes' attribute of workingVariable is used in a PUT a SELECT will be executed to determine the maximum value for this column. That value will be incremented (it must be an integer) and used when writing the new row. This is generally used for table keys that are unique integers but have no other meaning.

.

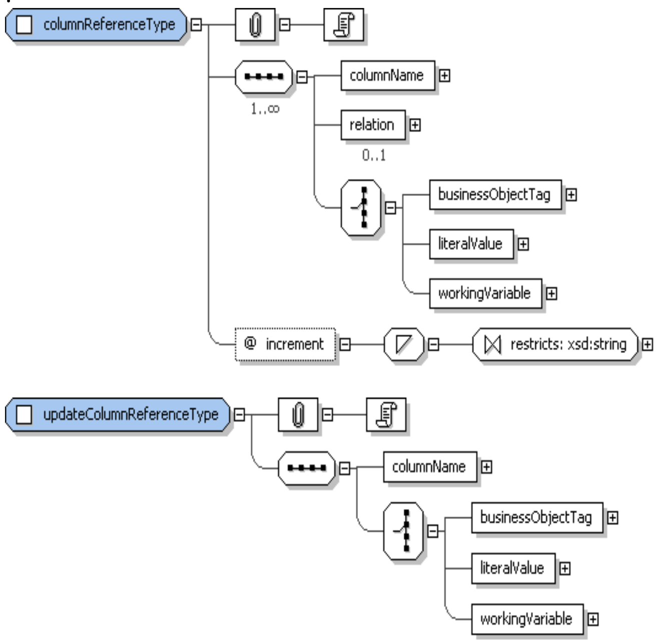

Figure 6. columnReferenceType and updateColumnReferenceType

4.3 <call>

This is used to invoke another BO as shown in Figure 7. Parameters are passed from the calling BO to the called BO. The called BO may also return data which is put into the calling BO's list of workingVariables. When the called BO is invoked it is associated with a position in the input XML data. This position is specified by the businessObjectTag and is relative to the position in the input data for the calling BO. If there are multiple instances of the specified businessObjectTag then the called BO will be invoked once for each instance. The workingVariables that will exist at the time the called BO is invoked are:

- Any global WVs
- WVs specified as parameters for the called BO
- businessObjectTags referenced in any of the tableQuery's specifed for the called BO

Elements of <call> are:

<boName>	The name of the called BO
<businessObjectTag>	The path to a point in the input XML
<parameter>	Working Variables to be passed to the called BO (zero or more)
<return>	Working variables to be passed back to the called BO (one or more)

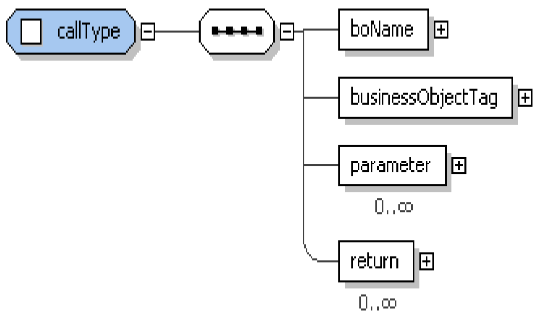

Figure 7. callType

4.4 <conditional>

A conditional processing element is used specify conditions under which processing elements that follow it will be executed. the conditions are based on the contents of an existing WV. (In programming terms, this is a simple "if then" statement with no "else" clause.)

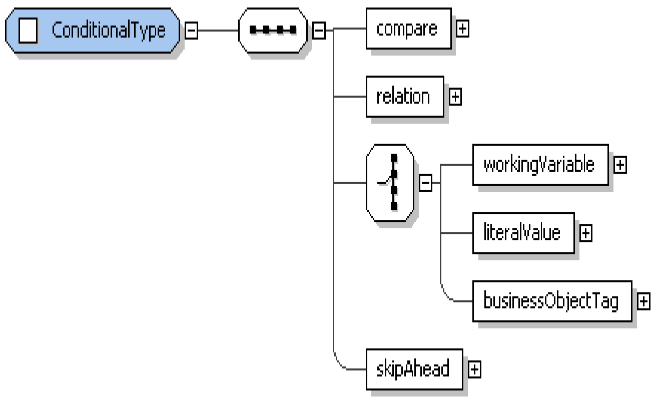

Figure 8. Conditional Type

Elements of <conditional> are:

<compare>	The name of the WV to be compared
<relation>	Comparison: EQ, NE, GT, LT, GE, LE

One of the following:

<workingVariable> a scalar WV; its associated value will be compared to <compare>

<literalValue> value to compare; if null, the value will be assumed to be a null String ("").

<businessObjectTag> tag at current XML node; its associated value will be compared

<skipAhead> how many <mappingSequence> steps to skip if the condition is true

4.4 <assign>

Assigns the value associated with one WV to another WV, as shown in Figure 9. Elements of <assign> are:

<to>	The receiving WV

One or two of the following (if <literalValue> is specified with either <workingVariable> or <BusinessObjectTag> the <literalValue> is concatenated to the other variable; <workingVariable> and <BusinessObjectTag> may not both be specified.):

<workingVariable>	The source WV
<businessObjectTag>	Input BML element associated with value to assign
<literalValue>	Literal value to assign

<transformMethod>	Names a method to be used to transform the data. Current transform methods are used with coordinate systems: DTGToDTTM DTTMToDTG

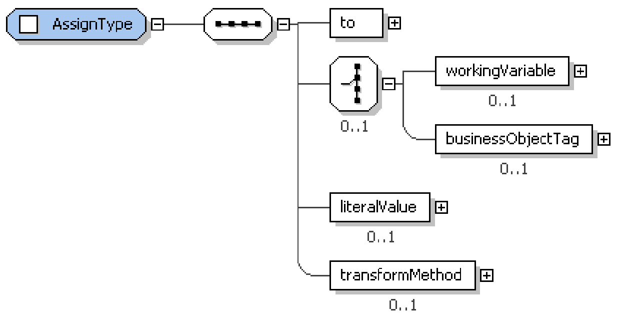

Figure 9. Assign Type

4.5 <abort>

This is used when the script finds a fatal error and needs to abort the entire transaction. The entire transaction will be rolled back and no changes will be made to the database. If the <abort> happens inside an invoked BO, the calling BO also is aborted.

4.6 <BusinessObjectReturn>

<BusinessObjectReturn> is used to generate output XML to be returned to the client, generally using data retrieved by database query. A <BusinessObjectReturn> consists of one or more of <BusinessObjectReturnElement>, which is shown in Figure 10.

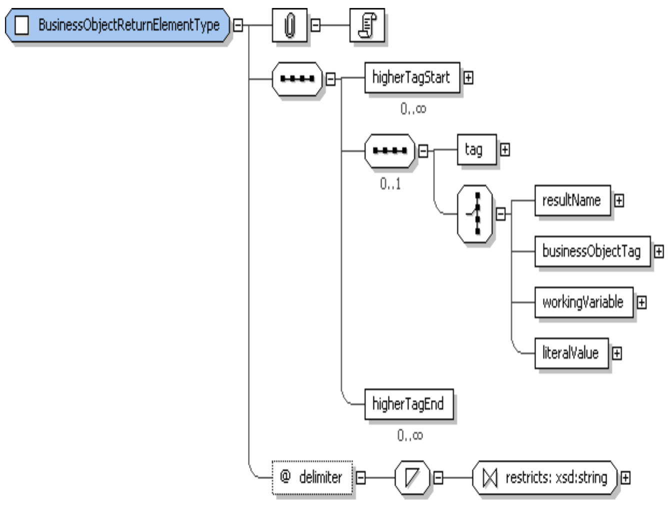

Figure 10. BusinessObjectReturnElementType

The elements are:

<higherTagStart> (zero or more) generates a start tag with no data, used to frame actual results; multiple instances must represent properly nested XML tags (this is not checked by SBML).

<higherTagEnd> (zero or more) generates an end tag with no data, used to frame actual results; multiple instances must represent properly nested XML tags (this is not checked by SBML).

<tag> generates start and end tags using any of the following, as defined in section 4.1.

<businessObjectTag>
<workingVariable>
<literalValue>

Each of these elements may be specified any number of times in any sequence. The order of the output will match the order in which the elements are present in the script.

If any of the <workingVariable> elements is of type List all the elements in the current <BusinessObjectReturnElelment> will be repeated for each element in the list. If there is more than one list WV, the first one to be completed will terminate the sequence.

5. Conclusions

The Scripted BML Web Service shows great potential as a basis for rapid development of supporting services for BML. Our experience shows that scripts can be developed much more rapidly than hard-coded services, and generally are less susceptible to subtle bugs because they are limited to the functionality required in mapping BML to the JC3IEDM. Moreover, the scripting language provides a highly concise representation for the mapping definition. We encourage SISO's C-BML Product Development Group to adopt the Mapping Description Language in this document as the representation for the mapping required in the Version 1 Standard [5].

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] Levine, S., L. Topor, T. Troccola, and J. Pullen, "A Practical Example of the Integration of Simulations, Battle Command, and Modern Technology", IEEE

Fall Simulation Interoperability Workshop, Orlando, FL, 2008

[3] Pullen, J. *et al.*, NATO MSG-048 Coalition Battle Management Initial Demonstration Lessons Learned and Way Forward, IEEE Spring Simulation Interoperability Workshop, Providence, RI, 2008

[4] Schade, U. and M. Hieb, “Development of Formal Grammars to Support Coalition Command and Control: A Battle Management Language for Orders, Requests, and Reports, 11th International Command and Control Research and Technology Symposium*,* Cambridge, UK, 2006

[5] Blais, C., K. Galvin and M. Hieb, “Coalition Battle Management Language (C-BML) Study Group Report,” Paper presented at IEEE Fall Simulation Interoperability Workshop, Orlando FL, 2005

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. Heis the lead developer of scripting for the SBML project.

Appendix
Why Business Object_Example

Input XML Data

```
<?xml version="1.0" encoding="UTF-8"?>
<GetWhy>
   <OrderID>Order123</OrderID>
   <TaskLabel>GroundTask1</gTaskLabel>
</GetWhy>
```

Mapping File

```
<?xml version="1.0" encoding="UTF-8"?>
<BusinessObjectInput xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file:/home/ssingapo/Desktop/BusinessObjectTransactionInterpreterSc
hema.v0.18.xsd">

    <BusinessObjectTransaction>
        <transactionName>GetWhy</transactionName>
         <parameter>OrderId</parameter>
         <return>whyCode</return>
         <return>
        <!--  First three tableQuery's get the act_id for the Order
            (order_id) and for the task  (task_id -->

        <tableQuery>
            <mappingSequence>0</mappingSequence>
            <databaseTable>act</databaseTable>
            <queryAction>GET</queryAction>
            <resultName>act_id</resultName>
            <columnReference>
                <columnName>name_txt</columnName>
                <workingVariable>OrderID</workingVariable>
            </columnReference>
        </tableQuery>

        <assign>
            <to>order_act_id</to>
            <workingVariable>act_id</workingVariable>
        </assign>

       <tableQuery>
            <mappingSequence>1</mappingSequence>
            <databaseTable>act_functl_assoc</databaseTable>
            <queryAction>GET</queryAction>
            <resultName type="List">obj_act_id</resultName>
            <columnReference>
                <columnName>subj_act_id</columnName>
                <workingVariable>order_act_id</workingVariable>
            </columnReference>
        </tableQuery>

        <tableQuery>
            <mappingSequence>2</mappingSequence>
            <databaseTable>act</databaseTable>
            <queryAction>GET</queryAction>
            <resultName>act_id</resultName>
            <columnReference>
                <columnName>act_id</columnName>
                <workingVariable>obj_act_id</workingVariable>
            </columnReference>
            <columnReference>
                <columnName>name_txt</columnName>
                <businessObjectTag>TaskLabel</businessObjectTag>
            </columnReference>
        </tableQuery>
        <assign>
            <to>task_act_id</to>
            <workingVariable>act_id</workingVariable>
        </assign>

        <tableQuery>
            <mappingSequence>3</mappingSequence>
            <databaseTable>act_effect</databaseTable>
            <queryAction>GET</queryAction>
            <resultName>descr_code</resultName>
            <columnReference>
                <columnName>act_id</columnName>
                <workingVariable>task_act_id</workingVariable>
            </columnReference>
        </tableQuery>

        <tableQuery>
            <mappingSequence>4</mappingSequence>
            <databaseTable>act_effect_descr_code</databaseTable>
            <queryAction>GET</queryAction>
            <resultName>display_value</resultName>
            <columnReference>
                <columnName>code</columnName>
                <workingVariable>descr_code</workingVariable>
            </columnReference>
        </tableQuery>

        <assign>
            <to>whyCode</why>
            <workingVariable>code</workingVariable>
        </assign>

        <BusinessObjectReturn>
            <BusinessObjectReturnElement>
                <higherTagStart>WhyResult</higherTagStart>
                <higherTagStart>Why</higherTagStart>
            </BusinessObjectReturnElement>
            <BusinessObjectReturnElement>
                <tag>OrderID</tag>
                <workingVariable>OrderID</workingVariable>
            </BusinessObjectReturnElement>
            <BusinessObjectReturnElement>
                <tag>Label</tag>
                <workingVariable>Label</workingVariable>
            </BusinessObjectReturnElement>
            <BusinessObjectReturnElement>
                <tag>display</tag>
                <workingVariable>display_value</workingVariable>
            </BusinessObjectReturnElement>
            <BusinessObjectReturnElement>
                <higherTagEnd>Why</higherTagEnd>
            </BusinessObjectReturnElement>
        </BusinessObjectReturn>
    </BusinessObjectTransaction>
</BusinessObjectInput>
```