Condensed Scripting Language (CSL)
Programming Guide

C4I Center
George Mason University
Fairfax, VA 22030, USA

Table of Contents

1. Introduction

Battle Management Language (BML) is intended to assist with interoperation between command and control (C^2) and modeling and simulation (M&S) systems by supplying a common agreed-to format for exchanging information such as orders and reports [2]. Participating C^2 systems can use a provided repository service to post and retrieve messages that are expressed in BML. This service is centralized or distributed. BML uses Extensible Markup Language (XML) along with web service (WS) technology to store a representation of BML in a well-structured database, which can be accessed via the Structured Query Language (SQL). As BML continues to grow and evolve, some middleware will remain impervious to BML changes, generally the XML input structure and a WS for database transactions. This implies re-usable component structure for a scripting language to control BML transactions.

The Scripted Battle Management Language (SBML) system is a web service that takes BML as input and generates BML as output, persisting information in JC3IEDM (Joint Consultation, Command, Control Information Exchange Data Model) format. SBML is driven by a scripting engine, BML schemas, and an XML Mapping file. It accepts BML transactions and writes them to a database in accordance with the written scripts. In order to help write scripts (mapping files), there is a Condensed Scripting Language (CSL) and a front-end translator that can reduce the tedious burden of reading and writing of XML into a more condensed representation.

CSL is a compact version of the XML SBML scripting language, which is converted to the XML version before being read by the scripting engine. This document describes CSL and will be useful for a handful of applications:

- Implementing new BML constructs for rapid testing
- Making changes to a new data model (only available when using the direct DB access)
- Providing a concise definition of BML-to-data model mappings that facilitates review and interchange needed for collaboration and standardization

1.1 Acronyms

API	Application Programming Interface
BML	Battle Management Language
BO	Business Object
C^2	Command and Control
CSL	Condensed Scripting Language
J2EE	Java 2 Platform Enterprise Edition
JC3IEDM	Joint Consultation, Command, Control Information Exchange Data Model
JMS	Java Message Service
MVWV	Multi-valued Working Variable
RI	Reference Implementation
SDK	Software Development Kit
SBML	Scripted BML
SIMCI	Simulation C^4I Interoperability
SQL	Structured Query Language
WS	Web Service
WV	Working Variable
XML	Extensible Markup Language
XPATH	XML Path Language

2. SBML Operations

The architecture of SBML is described in figure one.

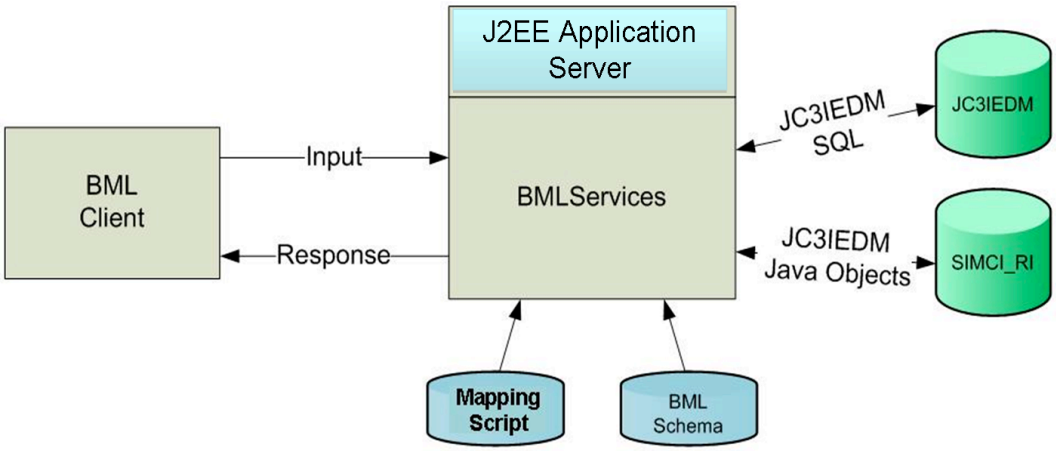

Figure 1. SBML Configuration

The SBML server has a scripting engine that implements a BML web service by converting BML data into the JC3IEDM database representation. Conversely, it can retrieve data from the database and generate BML as an output. This web service is 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. CSL is the dense version of the XML code, which provides the rules for mappings.

The current SBML supports two different interfaces towards a database system: direct MySQL interface and a SIMCI_RI interface. The latter passes data through the JC3IEDM SDK developed by the U.S. Army to Red Hat's Hibernation persistence service as java objects. However, this document concentrates on the details of script writing aspects and processes. Script developers have an ability to create new constructs and facilitate rapid prototyping by defining concise definitions of BML information to data model information. All services in SBML are driven by BML elements. As a result, all transactions must comply with the XML schema definition of BML. Data is sent to the SBML server in two different transactions.

One transaction is *pushing* data (or *push*). This transaction contains data that adds or updates tables to the data model database. For example, this operation could be an order. A *push* will return an indicator message acknowledging a success or failure. The second transaction is *pulling* data (or *pull*). This transaction requests data from the associated database.

This document makes the assumption that the reader will have a reasonable background with command line interpreters, XML, and elementary programming constructs.

2.1 Publish and Subscribe Service

BML Web Services uses a publish/subscribe capability based on the J2EE JMS API. This has been added to the scripting engine that is diagramed in figure two. This capability provides for a number of fixed subscriptions "topics", which are established in advanced based on the contents of BML documents [2] and defined in a configuration file used by the server. XPATH specifies the topic filters being used. Each BML Document to be added to the database is compared to every XPATH statement in the topic list. If the XPATH statement matches, the BML document is published to the corresponding topic. Consequently, the document is forwarded to all subscribing clients immediately after it is validated and posted to the database.

For example, as a military unit moves from one area to another, there are frequent reports that generate updated unit positions within a certain time frame. Furthermore, these updates are a specific topic that other units have subscribed to. As BML messages are received from the moving unit, they are processed by the appropriate script and written to the database. After a successful transaction, that message is then forwarded on to other subscribers

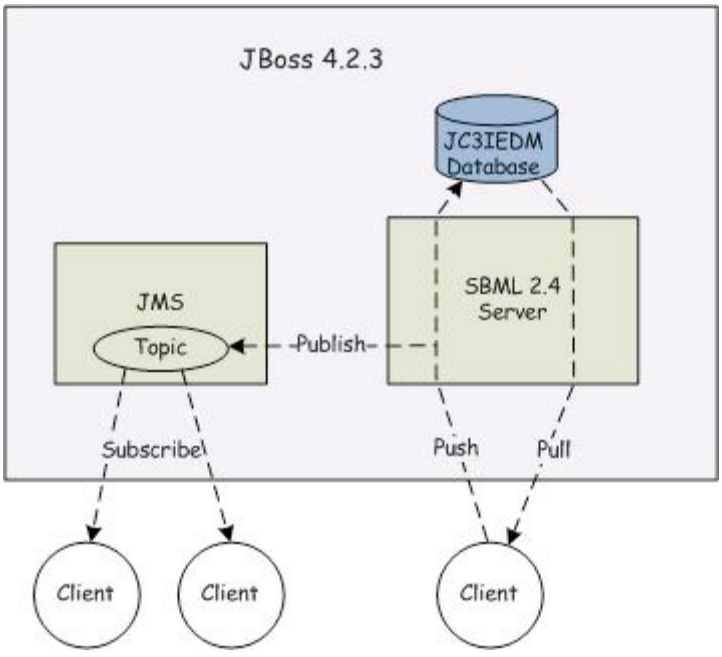

Figure 2. Publish/Subscribe Architecture

Clients receive published documents by subscribing to topics of interest using JMS procedures. Any given document may be published to more than one topic. Since JMS is Java specific, a C++ client also has been developed to use Java specific protocols via the Java Native Interface [2].

2.2 BML Schemas

The BML schemas are located in the SVN repository under the SBML_HOME directory. There is a different top-level schema dependent on whether the request is an order or a report with one top-level schema for status reports and another for all the other type of reports. Each BML transaction is described by an XML schema and can be validated by the server.

3. Condensed Scripting Language

Script writing for SBML was previously done using XML (see appendix B for an example XML script). While XML is a well defined and a machine-readable format, it is also very verbose. The Condensed Scripting Language (CSL) allows script developers to write scripts in a more reader friendly format than the wordy XML format. Basically, CSL is a compact version of the XML script format and a user can inscribe scripts in a condensed format and has an interpreter translate it to an XML formatted script.

NOTE: Appendix B has a sample CSL scripts and their resulting SBML (XML) scripts.

3.1 CSL Test Settings

While deploying newly developed CSL scripts, the user will have to store them under the *CSL_Scripts* directory. When the SBML Server starts it will compile all scripts under this directory. Currently scripts are stored in two subdirectories under *CSL_Scripts:* *IBML* and *CBML*. This directory location will be a sub-directory within the SBML_HOME home directory:

SBML_HOME/CSL_Scripts

There is also a *CSL.jar* file available under *CSL_Scripts* that a user can employ to test whether or not a script will compile successfully. The jar file expects the CSL script file as the first argument and the resulting XML script file as the second argument. Here's an example using a Unix shell command:

```
java –jar CSL.jar <scriptFileName.csl>  <outputFileName.xml>
```

Each time the SBML server is started all scripts are compiled for server operations. For editing and test purposes you can set the server properties to compile CSL scripts each time a transaction occurs. To enable this option, edit the *sbml.properties* file that is located in:

$JBOSS_HOME/server/default/conf

Edit the *sbml.properties* file by adding the line: *testMode=y*.

NOTE*: sbml.properties* file is found in the setup directory of the SBMLServer repository and is copied to the above location as part of installation.

3.2 Debugging Server Log

SBMLServer uses log4j to output to the JBoss server log located in:

$JBOSS_HOME/server/default/log/server.log

The level of logging that appears is controlled by the jboss-log4j.xml configuration file located in:

```
$JBOSS_HOME/server/default/conf
```

A sample jboss-log4j.xml configuration file is included in the SBMLServer repository under the setup sub-directory. This configuration file can be customized to increase or decrease the amount of logging by SBMLServer as it executes.

3.3 CSL Description

All CSL scripts consist at the outermost level of a single BO_INPUT{} block. The CSL's condensed script main components are Business Objects (BOs) which are represented by BO_TRANSACTION{} blocks. Consequently this means each *BusinessObjectInput* can have multiple *BusinessObjectTransaction* elements within the XML version. All transactions contain a set of database operations. The following subsections give the significant details for writing CSL scripts.

3.3.1 CSL File Contents

The file contents are framed with *BO_INPUT* which may contain multiple *BO_TRANSACTION*s. Below is a general outline of what every CSL file should look like:

```
BO_INPUT
{
    BO_TRANSACTION  boName1
    {
          Statements;
    }

    BO_TRANSACTION  boName2
    {
          Statements;
    }
}
```

1. All single line statements are terminated with a semicolon ";"

2. Statements that include other statements frame those statements with braces { . . . }.

3. Working Variables (WV)

   There are three types of string-type working variables available in the scripting engine:

   Scalar – the variable refers to a single element

   List – The variable is a list of strings (i.e., scalars)

   Row – The variable is a named collection of scalars. Individual elements are referred to by WVRowName!elementName. For example, action!act_id refers to the act_id element in the action Row variable.

4. When referring to variables, the following formats are used:

   a. Identifier - Working variable name if the field must be a WV name
   b. Variable – Can be a working variable, a literal, or a BusinessObjectTag
   c. A WV name can contain:

      A-Z, a-z, 0-9, special characters other than ";"

   d. Literal is framed by quotes e.g. "abc".
   e. A businessObjectTag refers to a position in the input XML. It is an XPATH statement and is framed with brackets, e.g. [/input/orderid]

In addition to the three WV types parameters to the various statements may be of three types:

Working Variables – Written without punctuation. E.g. act_id

Literal Values (Constants) - Written in quotes. E.g. "ABCD"

Business Object Tags – These refer to data in the input XML relative to the anchor tag used to call the current BO. These in turn are written as XPATH statements and are framed by square brackets. For example,

[./When/OrderIssuedWhen]

When a Business Object is called, only the working variables that are passed as parameters will be available to the called BO. The exceptions to this are multiVariableWorkingVariables (MVWV) and iterator working variables that are injected into the called BO's space. Any other WV's used by the calling BO are saved and restored when the called BO returns.

3.3.2 CSL BO_TRANSACTION Description

The general outline for a BO_TRANSACTION:

```
BO_TRANSACTION boName [attributes] (parm1 parm2 ….) (return1 return2 …)
	{
		Statements;
	}
```

This is the start of a businessObject. Here is a description of the outline above:

boName the name of this BO – which usually corresponds to an element within a known schema. An exception is making an immediate call to a known element.

attributes indicated special variables passed to this BO. There are two forms:

mvwv=identifier and *assignTo=identifier*
mvwv is the name of a List Working Variable. If this is specified, the BO will be called once for each member of the list. The individual values will be assigned to the assignTo value for each call.

Iterator=identifier
The variable is a string representation of an integer. When the anchorTag results in a list of input XML elements the BO will be called one time for each instance. The iterator will be initialized to one and incremented each time the BO is called.

parm1, *parm2*, etc.
These are parameters passed by the calling BO. These are working variable (WV) names.

return1, *return2,* etc.
These are the names of working variables that are to be returned to the caller.

Example:

```
BO_TRANSACTION WhatWhenPush [mvwv=refs assignTo=ref_id] (act_id order_id)
(obj_item)
{
        .
        .
}
```

In this example the "WhatWhenPush" *BusinessObjectTransaction* is passed a list multi-value working variable named refs. Each element of the list refs is assigned to the variable named ref_id for each execution of the transaction. In addition, there are two parameters: "act_id" and "order_id" that are passed in and the other parameter "obj_item" is returned.

3.3.3 CSL ROUTINE Description

A ROUTINE is exactly the same as a BusinessObject transaction. It is used to help represent higher-level functions distinguishing between BusinessObject transactions that call another transaction as part of an operation.

Example:

```
BO_TRANSACTION AtWherePush () (obj_item_id)
{
        .
        .
        CALL ObjectItemPush . (cat)(obj_item_id)
        .
        .


}
        .
        .


ROUTINE ObjectItemPush (type) (obj_item_id)
{

}
```

3.3.4 CSL Statement ASSIGN Description

The ASSIGN statement is declared as follows:

```
ASSIGN variableName identifierName
```

The declaration above is assigning *variableName* to the identifierName.

Here a few examples:

```
ASSIGN “123” cat_code;
ASSIGN temp_act_id act_id;
```

3.3.5 CSL Statement ASSIGN_ITEM Description

```
ASSIGN_ITEM  A B I;
REMOVE_ITEM  A B I;
```

A is a list (WV)
B is a target working variable (Scalar)
I is a scalar WV containing an integer value or one of two codes (F or L).

ASSIGN_ITEM assigns the entry in A indicated by I to B. REMOVE_ITEM performs the same assign function as ASSIGN_ITEM, in addition it removes the entry from the list. The index (I) can contain F (First entry), L (Last entry) or an integer.

Examples:

```
// Assign the third entry in Alist to Bwv ;

ASSIGN “3” Indx;
ASSIGN_ITEM Alist Bwv Indx;

// Remove the first entry in Alist;

ASSIGN “F” Indx2;
REMOVE_ITEM Alist Bwv Indx2;

// Remove the first entry in Alist;

ASSIGN “F” Indx2;
REMOVE_ITEM  Alist Bwv Indx2;
```

3.3.6 CSL Incrementing an Integer Stored In a WV Description

```
INC  I;
```

Increments an integer stored as a scalar WorkingVariable

Example:

// Increment wva;

INC wva;

Note: Incrementing an index that has the value "L" will return an error.

3.3.7 CSL Setting a DEFAULT Value

```
DEFAULT Src Trgt "defaultVal";
```

Src If *Src* exists, *Src* is assigned to *Trgt*. If not, then assign *defaultVal* to *Trgt*.

Examples:

```
// If A exists assign it to result.
DEFAULT A result "abc";
```

The result of this statement produces a *IF_THEN_ELSE* processing element (see Appendix A). Here is a sample:

```
<csl>defTest.csl:Test1:18 Default A result "abc"</csl>
            <ifThenElse>
                <compare>A</compare>
                <relation>NE</relation>
                <literalValue/>
            </ifThenElse>
            <codeBlock>
                <assign>
                    <workingVariable>A</workingVariable>
                    <to>result</to>
                </assign>
            </codeBlock>
            <codeBlock>
                <assign>
                    <literalValue>abc</literalValue>
                    <to>result</to>
                </assign>
            </codeBlock>
```

```
// Example using a BO Transaction
DEFAULT [cbml:Thing/cbml:/OtherThing] result2 "def";
```

The prior example produces the following *IF_THEN_ELSE* (see appendix A) processing elements:

```
<csl>defTest.csl:Test1:19 Default [cbml:Thing/cbml:/OtherThing] result2
```

```
   "def"</csl>
            <assign>

<businessObjectTag>cbml:Thing/cbml:/OtherThing</businessObjectTag>
                <to>tempDefault</to>
            </assign>
            <ifThenElse>
                <compare>tempDefault</compare>
                <relation>NE</relation>
                <literalValue/>
            </ifThenElse>
            <codeBlock>
                <assign>
                    <workingVariable>tempDefault</workingVariable>
                    <to>result2</to>
                </assign>
            </codeBlock>
            <codeBlock>
                <assign>
                    <literalValue>def</literalValue>
                    <to>result2</to>
                </assign>
            </codeBlock>
```

3.3.8 CSL ASSERT a Conditon is True Description

```
ASSERT (condition) “Abort message”;
```

If the condition is true continue else *ABORT* with the message. This is a shorter way of writing an equivalent *IF_THEN* statement (section 3.3.18) in conjunction with a BOR_MSG to the client (section 3.3.8). The condition can compare a WV or a BO Transaction to a WV, BO Transaction, or LiteralValue.

This Example uses a BO Transaction:
// Assert that the contents of BO Transaction C is less than the WV D;

ASSERT ([C] LT D) "Assert message 1";

3.3.9 CSL Return a Message to Client Description

```
BOR_MSG "msg";
```

Return a specific message to the client.

Example:

```
// Return OK to client;

BOR_MSG "Transaction OK";
```

3.3.10 CSL ABORT Description

```
ABORT Message;
```

The ABORT statement is used to terminate a process when a statement fails. It stops all processing and prints a message in the debug log.

Example:

```
. . .

IF_THEN (reporter_org_id EQ NULL)
        {
            ABORT  Invalid or missing ReporterWho in report;
        }
. . .
```

3.3.11 CSL DEBUG Description

```
DEBUG;
```

This statement prints all current working variables to the debug log.

3.3.12 CSL Comment Description

```
// this is a comment;
```

This comments text and it is generated as an XML comment by the translator. **NOTE**: each comment line must be terminated with a semicolon.

3.3.13 CSL COMMIT Description

```
COMMIT;
```

Commit the current transaction to the database and start a new one.

NOTE: this applies to SQL databases only.

3.3.14 CSL *RI_START* Description

```
RI_START topLevelObjectName topLevelObjectPrimaryKey;
```

Start a new RI transaction. Specify the object (table) that will be the root when the transaction is written and the primary key of that object. Note that the full object names are used for the object. RI_START commands may be nested.

NOTE: this applies only to RI databases.

Example:

```
RI_START   Action   act_id;
```

3.3.15 CSL RI_END Description

```
RI_END;
```

This statement completes the current transaction and writes the pending transaction to the RI. This does not start a new transaction. An `RI_START` must be executed to start a new one.

NOTE: this applies only to RI databases.

3.3.16 CSL CALL Description

```
CALL boName  anchorTag  (parm1 parm1 ...) (ret1 ret2 ...);
```

This statement calls another BO passing parameters and indicating what variables will be returned. Here is the statement description:

boName	the name of the called BO.
anchorTag	this is an XPATH statement and passes a position on the input XML to the called BO.
parmX	these are variables: WV names, literals, or BusinessObjectTags. See section 3.2
retX	these returns are WV names

Example:

```
BO_INPUT

        {

            BO_TRANSACTION WhatWhenPush [mvmv=list_act_ids,
            assignTo=list_act_id]( task_id act_id)( cat_code name_txt)

            {

                  CALL someSub ./point ([parm1] "parm2") (ret1 ret2);

            }


         }
```

In this CALL example, the transaction someSub is called with a pointer into the input bml xml at the point element off the current position pointer.

3.3.17 CSL GET_XXX Description

```
GET_XXX tableName assignTo=result orderBy=sortCol (columnref1) (columnref2). …
```

This statement performs a read transaction from the (JC3IEDM) database. Getxxx may be: GET, GET_LIST, GET_ROW, GET_MAX. Here are the explanations:

GET	sets the result from the first record satisfying the query
GET_LIST	will try to read a collection of records and create a list from the result column for each record retrieved.
GET_ROW	takes the first row retrieved and creates a Row working variable from the columns in the record
GET_MAX	this will perform a GET, sorts the result by the orderBy column Descending and returns the result column from the first row returned. This returns the result column from the row having the highest value in the orderBy column. This is useful, for example, when a set of rows has the same values in the SELECT clause except for a date and the one with the latest date is needed.

tableName	the name of the table or RI Object. This corresponds to the table name within the JC3IEDM database.

assignTo=result	retrieve the column specified by result. "assignTo=" is optional; if present, assign the value of the result to the variable named by assignTo.
orderBy= sortCol	used only by the GET_MAX instruction
columnrefx	specify the contents of the "Where" clause of the SQL query. Each columnref provides a column name, logical operator and value for the query. Logical operators are EQ, NE, GT, LT, GE, LE. The values are any valid variable, i.e. working variable, literal or BusinessObjectTag.

If the working variable specified in the first column reference is a List WV, multiple queries will be made, one for element in the list.

Example:

```
BO_TRANSACTION TRKREP()
{

  GET  UNIT unit_id(formal_abbrd_name_txt EQ [ReporterWho/bml:NameText]);
.
.
.

}
```

3.3.18 CSL PUT Description

```
PUT tableName (columnref1, columnref2, …)
```

This statement performs an insert transaction to the database.

tableName	the name of the table or RI object.
columnrefX	specify the data to be written in the following form:

```
columnName = variable
```

variable can be any valid variable. If ">" is specified instead of "=" this indicates the column of the primary key of this table. The database will create the primary key and the value will be assigned to the variable name specified.

3.3.19 CSL IF_THEN and IF_THEN_ELSE Description

```
IF_THEN (identifier OP variable)
       {
            Statements executed if true
       }
```

OR

```
IF_THEN_ELSE (identifier OP variable)
       {
            Statements executed if true
       }
       {
            Statements executed if false
       }
```

identifer	this is a WV
variable	this is any valid variable(WV, literal or BO_TRANSACTION)
OP	this is the operator, e.g. EQ,LE, etc.

3.3.20 CSL CONT and NOTCONT Description

To add easier control and handling of output data, you can take advantage of two additional relations to the contiditonal: CONT (contains) and NOTCONT (does not contain).

Example:

```
IF_THEN (catCode CONT “END”)
{
       Statements executed if catCode = “END”
}
```

3.3.21 CSL BO_RETURN(s) Description

BO_RETURN and various child statements are used to create XML output from the contents of the database and from Working Variables.

The structure is:

```
BO_RETURN {
    BO_RETURN_ELEMENT {
         HIGHER_TAG_START    tag;
         BO_RETURN_ELEMENT_TAG  tag  variable;
         HIGHER_TAG_END      tag;
    }

    BO_RETURN_ELEMENT {
         HIGHER_TAG_START    tag;
         BO_RETURN_ELEMENT_TAG  tag  variable;
         HIGHER_TAG_END      tag;
    }
}
```

BO_RETURN	Sets up the return
BO_RETURN_ELEMENT	Contains specific output directives
BO_RETURN_ELEMENT_TAG	Specifies a tag and data value to be contained in the tag
HIGHER_TAG_START	Specifies a start tag only. The tag name may be in a literal or a variable
HIGHER_TAG_END	Specifies an end tag, no data. Like `HIGHER_TAG_START` the tag name be in a literal for a variable.

Example:

```
BO_RETURN {
    BOElement {
         HIGHER_TAG_START "Result";
    }
    BOElement {
         BOElementTag "UnitID" unit_name;
    }
    BOElement {
         HIGHER_TAG_END "Result";
    }
}
```

If the contents of a working variable was "3rd Battalion" the resulting XML returned for thie transaction would be:

```
<?xml version="1.0" encoding="UTF-8"?>
<Result>
    <UnitID>3rd Battalion</UnitID>
</Result>
```

3.4 Interfacing with the RI

There are additional rules that must be followed in order for a script to be usable with both the SQL database and the RI. In order to be able to push objects to the RI in "Complete Thoughts" and in order to have full control of the transaction in the script:

1) Full transactions from the client (One BML transaction) will be broken into multiple standalone independent JC3IEDM objects, known as "Complete Thoughts".

2) Each Complete Thought will be delineated by two new script elements.

   a. `<ri_start table="TopLevelTableName" key="Name of primary key for TopLevelTable">`

   b. `<ri_end>`

   The *TopLevelTableName* is the top level object that will be pushed, Reference for an order, *ControlFeature* for a control feature, Point, Line, etc for a Location. It is not necessarily the first object to be pushed by the script to be the TopLevelTableName. There can only be one object of a given type in a JC3 transaction.

3) When an <ri_end> is executed all the objects since the last <ri_start> will be pushed to the RI as a single object.

4) After the <*ri_end*> the RI_Interface will return the primary key named in the <*ri_start*> and will inject it into the current set of working variables. This will be the actual MIP key created by the RI.

5) PUTs will return OID's as the current implementation does now. These OIDs can be used as primary keys within a transaction but cannot be used after an <*ri_end*>. The script doesn't need to be concerned about what kind of key is being used, MIPKey or OID.

6) When there is a set of tables consisting of parent and sub classes they must be created in order from child to parent as they are now. It is the OID of the parent object (Lowest level) that is used in association tables. For example it is the OID of the Unit that is used rather than the OID of the ObjectItem. In fact, the PushObjectItem returns a 0 for the obj_item_id.

7) Care must be taken so that if the child object is created in a called BO (Not recommended!) that the key is returned to the caller.

8) <*ri_start*> .. <*ri_end*> statements can be nested.

9) Independent objects are tied to parents through association tables. Examples of this are:

10) ControlFeature is tied to the Task via ActionObjectiveItem

11) Task is tied to Order via ActionFunctionalAssociation

12) The association table is part of the parent task, not the child. When an independent object is pushed the actual MIP key is created (See 4) above). The MIP key is used in the association table to connect the parent to the child. The association table, therefore, must be PUT after the <ri_end> so that the actual MIP key is available. Note – This is only true if the child is to be pushed as an independent object. If a Location is imbedded inside a ControlFeature the ObjectItemLocation association table can be pushed at any time in accordance with the above.

Appendix A XML Scripting Language Elements

The basic elements in SBML are character strings. SBML services are driven by elements of BML. The SBML service creates a number of *workingVariables* (WV) that are associated with strings and aggregates of strings. WV's hold intermediate results or parameters. These are referred to by name within a script.

There are also XML aggregates called *BusinessObjects* (BO). To successfully write scripts, you must create a well-formed XML file with the correct associated elements of a BO. Appendix B has a complete XML script example that is sampled throughout the following subsections.

All descriptions is section three can be found in [2] and [3].

A.1 Business Object Elements

The root element is <*BusinessObjectInput*>, which may contain multiple <*BusinessObjectTransaction*> elements. The following is a list of elements under a <*BusinessObjectTransaction*>:

<*transactionName*>	The Name of this BO
<*parameter*>	Names of WVs passed to the BO
<*return*>	Names of the WVs to be returned to BO
<*global*>	Names of the global WVs

Here's a sample of using some of the root elements:

```
<!--This is a fragment of code from a WhatWhenPush message -->
<BusinessObjectInput>
        <BusinessObjectTransaction multiValueWorkingVariable="list_act_ids" assignTo="list_act_id">
                <transactionName>WhatWhenPush</transactionName>
                <parameter>
                        <workingVariable>task_id</workingVariable>
                <parameter>
                <return>cat_code</return>
                <return>name_text</return>

                .       .       .
```

Since global element weren't used, it will be set to a null valued string automatically

A.1.1 Iteration

Some BOs are repeated multiple times. SBML supports a few features for repeated processing.

1. A BO may be executed iteratively through the use of multi-valued WV (MVWV). The MVWV contains a sequence of WVs known as a *List*. Iteration is implicit so that each 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 (sections 3.1, 4.3.2, and 4.3.11 for examples)
2. When a BO is executed via a <*call*>, a reference to a point in the input bml data is created. The reference is relative to the calling BO current pointer into the input bml data.
3. A named WV defined as an *iterator* may be associated with a BO. The first time the BO is executed, the iterator is initiated to "1". Each execution is incremented by one. This integer is stored as a string 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 unitil it is exhausted.

A.2 Processing Elements

Here is a list of processing elements:

<*tableQuery*>	Describes a database transaction
<*call*>	makes a call to another BO
<*ifThen*>	This starts a conditional branch for processing elements
<*ifThenElse*>	
<*assign*>	makes an assignment to a WV
<assignItem>	assigns a specific element from a list
<removeItem>	same as <assignItem> with the element removed from the list
<increment>	increments a working variable
<*abort*>	terminates a transaction
<*BusinessObjectReturn*>	Formats XML output using results of previous table queries

A.2.1 *<tableQuery>*

The tableQuery element is used for reading and writing to the Jc3IEDM database. It has its own elements to complete a database transaction.

<*dataBaseTable*>	provides the name of a table
<*queryAction*>	actions to be performed: **GET** – read elements from a table **PUT** – write a row to a table **UPDATE** - update a row in a table
<resultName>	Names the WV to the received result of GET
<columnReference>	specified contents of a column (see 3.2.4)

A.2.2 *<call>*

<*call*> is used to call upon another BO. Parameters are passed from the calling BO to the called BO. If there is any returned data, then it is stored into the calling BO's list of working variables. The elements of <call> are:

<*boName*>	The name of called BO
<*businessObjectTag*>	The path to a point in the input XML
<*anchorTag*>	Point in the XML relative to the current base used as a base for the called BO. XPATH notation is used.
<*parameter*>	WVs to be passed to the called BO.
<*return*>	WVs to be passed back to the called BO.

Here's a an example:

. . .

```
<transactionName>WhatWhenPush</transactionName>
<parameter>
    <workingVariable>task_id</workingVariable>
</parameter>
<parameter>
    <workingVariable>act_id</workingVariable>
</parameter>
<return>cat_code</return>
<return>name_txt</return>
<codeBlock>
    <call>
        <boName>someSub</boName>
        <anchorTag>./point</anchorTag>
        <parameter>
            <businessObjectTag>parm1</businessObjectTag>
        </parameter>
        <parameter>
            <literalValue>parm2</literalValue>
        </parameter>
        <return>ret1</return>
        <return>ret2</return>
    </call>
```

A.2.3 *<ifThen>* and *<ifThenElse>*

These are conditional processing elements that specify conditions under which the processing elements that follow them will be executed. The conditions are based on the contents of an existing WV. The elements of this conditional are:

<*compare*>	The name of the WV to be compared
<*relation*>	Comparison to made EQ = equals NE = not equals GT = greater than LT = less than GE = greater than equal to LE = less than equal to
<*compare*>	used for a scalar value WV
<*literalValue*>	value to compare; if null, that value will be a null string.
<*businessObjectTag*>	tag at current XML node; its associated value to compare

Here is a sample illustrating conditional processing:

. . .

```
<ifThenElse>
      <compare>act</compare>
      <relation>EQ</relation>
      <workingVariable>0</workingVariable>
  </ifThenElse>
  <codeBlock>
      <ifThen>
          <compare>c</compare>
          <relation>EQ</relation>
          <workingVariable>d</workingVariable>
      </ifThen>
      <codeBlock>
          <assign>
              <literalValue>123</literalValue>
              <to>act_id</to>
          </assign>
      </codeBlock>
  </codeBlock>
```

A.2.4 <*columnReference*> and <*updateColumnReference*>

The *columnReference* specifies the contents of a column. This will depend on the queryAction used:

- If <*queryAction*> is GET or UPDATE, the *columnReference* elements provide the data for the SQL WHERE clause.
  - For UPDATE, a row is updated with the contents of the elements under the element *updateColumnReference*. The row that's updated is specified by <*columnReference*>.
- If <*queryAction*> is PUT, then the *columnReference* elements provide the data to be written.
  - Each *columnReference* specifies the contents of one column. The row that written is the total of all contents of *columnReferences* within the <*tableQuery*>.

The *updateColumnReference* can be multiple <*updateColumnReference*> elements. ***These are used only with UPDATE transactions***. Each element describes the contents of an update being made to a column.

Here's an example of a tableQuery using PUT:

. . .

```
<tableQuery>
       <databaseTable>Org</databaseTable>
            <queryAction>PUT</queryAction>
                 <columnReference>
                     <columnName>org_id</columnName>
                     <workingVariable increment="Yes">org_id</workingVariable>
                 </columnReference>
                 <columnReference>
                     <columnName>act_id</columnName>
                     <workingVariable>temp_act</workingVariable>
                 </columnReference>
                 <columnReference>
                     <columnName>ref_id</columnName>
                     <literalValue>ABC</literalValue>
                 </columnReference>
</tableQuery>
```

. . .

A.2.5 <*assign*>

In order to assign one WV to another WV, use <*assign*>. Elements of <*assign*> are:

<*workingVariable*> The source WV.

<to>	Follows a WV or literal value.
<*businessObjectTag*>	The input BML element associated with value to assign.
<*literValue*>	Literal value to assign.
<*transformMethod*>	Names a method to be used to transform data.

Here's a few samples on how to use <*assign*>:

. . .

```
<assign>
     <workingVariable>task_act_id</workingVariable>
     <to>act_id</to>
</assign>
```

. . .

```
<assign>
     <literalValue>123</literalValue>
        <to>act_id</to>
 </assign>
```

A.2.6 <*assignItem*>

In order to assign an element of a WV list to another WV, use <*assignItem*>. Elements of <*assignItem*> are:

<*workingVariable*>	The source WV – must be of type LIST
<to>	destination WV
<*index*>	This specifies first entry (F), last entry (L), or an integer (I)

Here's a sample on how to use <*assignItem*>:

```
<assignItem>
                <workingVariable>Alist</workingVariable>
                <to>Bwv</to>>
                <index>Indx</index>
</assignItem>
```

A.2.7 <*removeItem*>

In order to assign an element of a WV list to another WV and then remove the assigned item from the original list, use <*removeItem*>. Elements of <*removeItem*> are:

<*workingVariable*>	The source WV – must be of type LIST
<to>	destination WV
<*index*>	This specifies first entry (F), last entry (L), or an integer (I)

Here's a sample on how to use <*removeItem*>:

```
<removeItem>
                <workingVariable>Alist</workingVariable>
                <to>Bwv</to>>
                <index>Indx</index>
<removeItem>
```

A.2.8 *<increment>*

In order to increment a WV, use <*increment*>. There are no sub-elements of <*increment*>. It only specifies a WV as text.

Note that the WV supplied to <*increment*> must hold a numeric or the letter "F" that corresponds to the value zero.

Here's a sample on how to use <*increment*>:

```
<increment>Indx</increment>
```

A.2.9 *<abort>*

If a script finds a fatal error and can't continue with its current implementation, then the script should be terminated. Use <*abort*> so the entire transaction is rolled back and no changes are made to the database.

NOTE: If <abort> happens inside an invoke BO, the calling BO is also aborted.

A.2.10 *<BusinessObjectReturn>*

<*BusinessObjectReturn*> is used to generate output XML to be returned to the client, generally using data retrieved by database query. This element always has one or more returned elements of <*BusinessObjectReturnElement*>. The elements within <*BusinessObjectReturnElement*> are :

<*higherTagStart*>	This generates a start tag with no data, used to frame results (zero or more).
<*higherTagEnd*>	This generates an end tag with no data, used to fram actual results (zero or more). The <higherTagEnd> and <higherTagStart> tags must balance.
<*tag*>	Generates start and end tags using any of the following elements as defined within section 3.2.5: <*businessObjectTag*>, <*workingVariable*>, and <*literalValue*>. Each element may be specified any number of times in a sequence.

NOTE: The order of the output will coincide the order in which the elements are presented within the script.

Here's a sample:

```
<BusinessObjectReturn>
        <BusinessObjectReturnElement>
            <higherTagStart>
                <workingVariable>order</workingVariable>
            </higherTagStart>
            <tag>action</tag>
            <workingVariable>act_id</workingVariable>
            <higherTagEnd>
                 <workingVariable>order</workingVariable>
            </higherTagEnd>
        </BusinessObjectReturnElement>
        <BusinessObjectReturnElement>
            <tag>ref</tag>
            <literalValue>123</literalValue>
        </BusinessObjectReturnElement>
```

Appendix B CSL to XML Example

Example 1

```
BO_INPUT
{
  // This is a comment;
  BO_TRANSACTION WhatWhenPush [mvmv=list_act_ids, assignTo=list_act_id](task_id act_id)
       (cat_code name_txt)
  {
       CALL someSub ./point ([parm1] "parm2") (ret1 ret2);
       PUT Org  (org_id > org_id), (act_id = temp_act), (ref_id = "ABC");
       PUT Act  (cat_code = [./example]) ;

       IF_THEN (act_id EQ 1)
       {
              Assign task_act_id act_id;
       }

       ASSIGN [./OrderID] b;
       DEBUG;
       COMMIT;
       RI_START Action act_id;
       RI_END;
       GET_ROW act x=cat_code (act_id EQ task_act_id),(name_txt EQ abc);
       GET act cat_code (act_id EQ task_id),(name_txt EQ sample);
       IF_THEN_ELSE (act EQ 0)
       {
              IF_THEN (c EQ d)
              {
                     ASSIGN "123" act_id;
              }
       }

       {
              ASSIGN act_id dtemp_act_id;
       }
       CALL displayAll . ;
       BO_RETURN
       {
              BO_RETURN_ELEMENT
              {
                     HIGHER_TAG_START    order;
                     BO_RETURN_ELEMENT_TAG            action act_id;
                     HIGHER_TAG_END      order;
              }
              BO_RETURN_ELEMENT
              {
                     BO_RETURN_ELEMENT_TAG            ref    "123";
              }
              BO_RETURN_ELEMENT
              {
                     HIGHER_TAG_START    stuff;
                     BO_RETURN_ELEMENT_TAG            id     [./order/id];
                     HIGHER_TAG_END      stuff;
              }
       }
   }
}
```

Resulting SBML Script

```
<?xml version="1.0" encoding="UTF-8"?>
<BusinessObjectInput>
    <!-- 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>
        <!--      0     GET unit formal_abbrd_name_txt = TaskerWho result <-
         unit_id -->
        <tableQuery>
            <databaseTable>unit</databaseTable>
            <queryAction>GET</queryAction>
            <resultName>unit_id</resultName>
            <columnReference>
                <columnName>formal_abbrd_name_txt</columnName>
                <businessObjectTag>UnitID</businessObjectTag>
            </columnReference>
        </tableQuery>
        <!--      1     PUT act_res act_id -> act_id
            ++act_res_index "RI" -> cat_code org_id -> authorising_org_id -->
        <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>
        <!--      2     PUT act_res_item act_id -> act_id
            act_res_index -> act_res_index
            unit_id -> obj_item_id  -->
        <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>
```

Example 2

```
BO_INPUT
{

      BO_TRANSACTION TaskerWhoPush( task_act_id obj_item_id)()
            {
            PUT  ACT_RES
           (act_id = task_act_id)
           (act_res_ix > act_res_ix)
           (cat_code = "RI")
           (creator_id = "0")
           (update_seqnr = "0");

            PUT  ORG_ACT_ASSOC
           (org_id = obj_item_id)
           (act_id = task_act_id)
           (org_act_assoc_ix > org_act_assoc_ix)
           (cat_code = "INIT")
           (creator_id = "0")
           (update_seqnr = "0");
            }

      BO_TRANSACTION TaskeeWhoPush( task_act_id obj_item_id)()
            {
            PUT  ACT_RES_ITEM
           (act_id = task_act_id)
           (act_res_ix > act_res_ix)
           (obj_item_id = obj_item_id)
           (creator_id = "0")
           (update_seqnr = "0");

            PUT  ORG_ACT_ASSOC
           (org_id = obj_item_id)
           (act_id = task_act_id)
           (org_act_assoc_ix > org_act_assoc_ix)
           (cat_code = "PLAN")
```

```
            (creator_id = "0")
            (update_seqnr = "0");

             }

        BO_TRANSACTION TaskWhatPush( taskerWhoUnitID task_act_id)()
            {
            PUT  ACT_TASK
              (act_task_id = task_act_id)
              (actv_code = [What])
              (cat_code = "ORD")
              (creator_id = "0")
            (update_seqnr = "0");

          PUT  ORG_ACT_ASSOC
            (org_id = taskerWhoUnitID)
            (act_id = task_act_id)
            (org_act_assoc_ix > org_act_assoc_ix)
            (cat_code = "CONTRL")
            (creator_id = "0")
         (update_seqnr = "0");
        }



}
```

Resulting SBML Script

```
<?xml version="1.0" encoding="UTF-8"?>
<BusinessObjectInput>
    <csl>example.csl:TaskerWhoPush:4 BO_TRANSACTION TaskerWhoPush</csl>
    <BusinessObjectTransaction>
        <transactionName>TaskerWhoPush</transactionName>
        <parameter>task_act_id</parameter>
        <parameter>obj_item_id</parameter>
        <codeBlock>
            <csl>example.csl:TaskerWhoPush:6 PUT ACT_RES</csl>
            <tableQuery>
                <databaseTable>ACT_RES</databaseTable>
                <queryAction>PUT</queryAction>
                <columnReference>
                    <columnName>act_id</columnName>
                    <workingVariable>task_act_id</workingVariable>
                </columnReference>
                <columnReference>
                    <columnName>act_res_ix</columnName>
                    <workingVariable increment="Yes">act_res_ix
                    </workingVariable>
                </columnReference>
                <columnReference>
                    <columnName>cat_code</columnName>
                    <literalValue>RI</literalValue>
                </columnReference>
                <columnReference>
                    <columnName>creator_id</columnName>
                    <literalValue>0</literalValue>
```

```
                </columnReference>
                <columnReference>
                    <columnName>update_seqnr</columnName>
                    <literalValue>0</literalValue>
                </columnReference>
            </tableQuery>
            <csl>example.csl:TaskerWhoPush:13 PUT ORG_ACT_ASSOC</csl>
            <tableQuery>
                <databaseTable>ORG_ACT_ASSOC</databaseTable>
                <queryAction>PUT</queryAction>
                <columnReference>
                    <columnName>org_id</columnName>
                    <workingVariable>obj_item_id</workingVariable>
                </columnReference>
                <columnReference>
                    <columnName>act_id</columnName>
                    <workingVariable>task_act_id</workingVariable>
                </columnReference>
                <columnReference>
                    <columnName>org_act_assoc_ix</columnName>
                    <workingVariable
increment="Yes">org_act_assoc_ix</workingVariable>
                </columnReference>
                <columnReference>
                    <columnName>cat_code</columnName>
                    <literalValue>INIT</literalValue>
                </columnReference>
                <columnReference>
                    <columnName>creator_id</columnName>
                    <literalValue>0</literalValue>
                </columnReference>
                <columnReference>
                    <columnName>update_seqnr</columnName>
                    <literalValue>0</literalValue>
                </columnReference>
            </tableQuery>
        </codeBlock>
    </BusinessObjectTransaction>
    <csl>example.csl:TaskeeWhoPush:22 BO_TRANSACTION TaskeeWhoPush</csl>
    <BusinessObjectTransaction>
        <transactionName>TaskeeWhoPush</transactionName>
        <parameter>task_act_id</parameter>
        <parameter>obj_item_id</parameter>
        <codeBlock>
            <csl>example.csl:TaskeeWhoPush:24 PUT ACT_RES_ITEM</csl>
            <tableQuery>
                <databaseTable>ACT_RES_ITEM</databaseTable>
                <queryAction>PUT</queryAction>
                <columnReference>
                    <columnName>act_id</columnName>
                    <workingVariable>task_act_id</workingVariable>
                </columnReference>
                <columnReference>
                    <columnName>act_res_ix</columnName>
                    <workingVariable
increment="Yes">act_res_ix</workingVariable>
                </columnReference>
                <columnReference>
```

```
                    <columnName>obj_item_id</columnName>
                    <workingVariable>obj_item_id</workingVariable>
                </columnReference>
                <columnReference>
                    <columnName>creator_id</columnName>
                    <literalValue>0</literalValue>
                </columnReference>
                <columnReference>
                    <columnName>update_seqnr</columnName>
                    <literalValue>0</literalValue>
                </columnReference>
            </tableQuery>
            <csl>example.csl:TaskeeWhoPush:31 PUT ORG_ACT_ASSOC</csl>
            <tableQuery>
                <databaseTable>ORG_ACT_ASSOC</databaseTable>
                <queryAction>PUT</queryAction>
                <columnReference>
                    <columnName>org_id</columnName>
                    <workingVariable>obj_item_id</workingVariable>
                </columnReference>
                <columnReference>
                    <columnName>act_id</columnName>
                    <workingVariable>task_act_id</workingVariable>
                </columnReference>
                <columnReference>
                    <columnName>org_act_assoc_ix</columnName>
                    <workingVariable
increment="Yes">org_act_assoc_ix</workingVariable>
                </columnReference>
                <columnReference>
                    <columnName>cat_code</columnName>
                    <literalValue>PLAN</literalValue>
                </columnReference>
                <columnReference>
                    <columnName>creator_id</columnName>
                    <literalValue>0</literalValue>
                </columnReference>
                <columnReference>
                    <columnName>update_seqnr</columnName>
                    <literalValue>0</literalValue>
                </columnReference>
            </tableQuery>
        </codeBlock>
    </BusinessObjectTransaction>
    <csl>example.csl:TaskWhatPush:41 BO_TRANSACTION TaskWhatPush</csl>
    <BusinessObjectTransaction>
        <transactionName>TaskWhatPush</transactionName>
        <parameter>taskerWhoUnitID</parameter>
        <parameter>task_act_id</parameter>
        <codeBlock>
            <csl>example.csl:TaskWhatPush:43 PUT ACT_TASK</csl>
            <tableQuery>
                <databaseTable>ACT_TASK</databaseTable>
                <queryAction>PUT</queryAction>
                <columnReference>
                    <columnName>act_task_id</columnName>
                    <workingVariable>task_act_id</workingVariable>
                </columnReference>
```

```
                <columnReference>
                    <columnName>actv_code</columnName>
                    <businessObjectTag>What</businessObjectTag>
                </columnReference>
                <columnReference>
                    <columnName>cat_code</columnName>
                    <literalValue>ORD</literalValue>
                </columnReference>
                <columnReference>
                    <columnName>creator_id</columnName>
                    <literalValue>0</literalValue>
                </columnReference>
                <columnReference>
                    <columnName>update_seqnr</columnName>
                    <literalValue>0</literalValue>
                </columnReference>
            </tableQuery>
            <csl>example.csl:TaskWhatPush:50 PUT ORG_ACT_ASSOC</csl>
            <tableQuery>
                <databaseTable>ORG_ACT_ASSOC</databaseTable>
                <queryAction>PUT</queryAction>
                <columnReference>
                    <columnName>org_id</columnName>
                    <workingVariable>taskerWhoUnitID</workingVariable>
                </columnReference>
                <columnReference>
                    <columnName>act_id</columnName>
                    <workingVariable>task_act_id</workingVariable>
                </columnReference>
                <columnReference>
                    <columnName>org_act_assoc_ix</columnName>
                    <workingVariable
increment="Yes">org_act_assoc_ix</workingVariable>
                </columnReference>
                <columnReference>
                    <columnName>cat_code</columnName>
                    <literalValue>CONTRL</literalValue>
                </columnReference>
                <columnReference>
                    <columnName>creator_id</columnName>
                    <literalValue>0</literalValue>
                </columnReference>
                <columnReference>
                    <columnName>update_seqnr</columnName>
                    <literalValue>0</literalValue>
                </columnReference>
            </tableQuery>
        </codeBlock>
    </BusinessObjectTransaction>
</BusinessObjectInput>
```

Appendix C Business Object Script Examples

This example script can be found in [2].

BML Input Data Extract

```
<?xml version="1.0" encoding="UTF-8"?> ...
<!-- Fragment of <OrderPush> transaction --> <Task>
   ...
   <GroundTask> <TaskeeWho>
     <UnitID>UIE9 FA</UnitID> </TaskeeWho>

. . .
```

Mapping File Extract

```
<?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>
        <!--      0     GET unit formal_abbrd_name_txt = TaskerWho result <-
        unit_id -->
        <tableQuery>
            <databaseTable>unit</databaseTable>
            <queryAction>GET</queryAction>
            <resultName>unit_id</resultName>
            <columnReference>
                <columnName>formal_abbrd_name_txt</columnName>
                <businessObjectTag>UnitID</businessObjectTag>
            </columnReference>
        </tableQuery>
        <!--      1     PUT act_res act_id -> act_id
            ++act_res_index "RI" -> cat_code org_id -> authorising_org_id -->
        <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>
        <!--      2     PUT act_res_item act_id -> act_id
            act_res_index -> act_res_index
            unit_id -> obj_item_id  -->
        <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>
```

Appendix D BNF Representation of CSL

```
/*The SKIP section identifies the characters we want to skip*/
SKIP : { " " }
SKIP : { "\t"| "\r"| "\n"}

/*Define the tokens of the language in the TOKEN section. We
define numbers and digits as tokens.
  JavaCC differentiates between definitions for tokens and
definitions for other production rules*/

      TOKEN :

{
     <assign: "ASSIGN">
|      <assign_item: "ASSIGN_ITEM">
|      <remove_item: "REMOVE_ITEM">
|      <inc:              "INC">
|    <BOTransaction: "BO_TRANSACTION">
|    <Routine: "ROUTINE">
|    <BOInput: "BO_INPUT">
|    <abort: "ABORT" (~["\n","\r"])* <br> >
|    <call: "CALL">
|    <commit: "COMMIT">
|      <debug: "DEBUG">
|      <ri_start: "RI_START">
|      <ri_end: "RI_END">
|    <IfThen: "IF_THEN">
|    <IfThenElse: "IF_THEN_ELSE">
|      <def: "DEFAULT">
|    <assrt: "ASSERT">
|    <put: "PUT">
|    <get: "GET">
|    <getList: "GET_LIST">
|    <getRow: "GET_ROW">
|    <getMax: "GET_MAX">
|    <BOReturn: "BO_RETURN">
|    <BOReturnElement: "BO_RETURN_ELEMENT">
|    <BOReturnElementTag: "BO_RETURN_ELEMENT_TAG">
|      <bormsg:   "BOR_MSG">
|    <HigherTagStart: "HIGHER_TAG_START">
|    <HigherTagEnd: "HIGHER_TAG_END">
|      <TopTagStart: "TOP_TAG_START">
|    <relation: "EQ"|"NE"|"LT"|"GT"|"LE"|"GE"|"CONT"|"NOTCONT">
|      <incOp: ">">
|      <assignOp: "=">
|    <Identifier: (<lowercaseAndOtherCharacters>)+>
|    <literalValue: <Identifier>>
|    <lowercaseAndOtherCharacters: ["A"-"Z","a"-"z","0"-
"9",".","_","/",":","<","-","*", "'", "!"]>
|      <quote: "\"">
|      <squote: "\'">
```

```
|      <quotedString: <quote>(<lowercaseAndOtherCharacters> | [" "] |
["="])* <quote>>
|      <SINGLE_LINE_COMMENT: "//" (~["\n","\r"])* (<br>)?
(["\n","\r"])*>
}

TOKEN : /* SEPARATORS */
{
  < LPAREN: "(" >
| < RPAREN: ")" >
| < LBRACE: "{" >
| < RBRACE: "}" >
| < LBRACKET: "[" >
| < RBRACKET: "]" >
| < br: ";" >
| < COMMA: "," >
| < DOT: "." >
}

<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>
    | <assign_item>
      | <remove_item>
      | <increment>
    | <inc>
    | <call>
      | <abort>
      | <debug>
      | <commit>
      | <ri_start>
      | <ri_end>
      | <comment>
      | <assert>
      | <put>
      | <get>
      | <ifthen>
      | <ifthenelse
      | <BOTransaction>
      | <BOReturn>

      <assign> ::= ìASSIGNî <variable> <identifier>
(<transformRoutine>)? <br>
      <transformRoutine> ::= <identifier>
      <assign_item> ::= "ASSIGN_ITEM" <identifier> <identifier>
<variable><br>
      <remove_item> ::= "REMOVE_ITEM" <identifier> <identifier>
```

```
<variable><br>
      <increment> ::= "INC" <identifier><br>
      <call> ::= ìCallî <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

References

[1] Dr. Mark Pullen, Douglas Corner, Samuel Singapogu, Bhargava Bulusu, and Mohammad Ababneh, "Implementing a Condensed Scripting Language in the Scripted Battle Management Language Web Service," IEEE/SISO Simulation Interoperability Workshop 2010.

[2] Dr. J. Mark Pullen, Douglas Corner, Samuel Singapogu , "Scripted Battle Management Language Web Service Version 2," IEEE Fall 2009 Simulation Interoperability Workshop, Orlando, FL, 2009.

[3] Dr. J. Mark Pullen, Douglas Corner, Samuel Singapogu , "Scripted Battle Management Language Web Service Version 1.0 Operation and Mapping Description Language," European Simulation Interoperability Workshop, Istanbul, Turkey, 2009.

[4] Dr. J. Mark Pullen, Douglas Corner, Lisa Nicklas, "Performance Usability Enhancements to the Scripted BML Server".