Report on implementing "Complete Thoughts" in SBML

Author: Samuel Singapogu

Introduction: Battle Management Language (BML) and its various 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. Recent implementations have focused on the use of 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.

Scripted BML (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. The BML/JC3IEDM conversion process is accomplished under the control of the scripting language. 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).

A databaseQuery is used for reading from or writing to the database. There are XML tags to denote the name of the database table, the column to be retrieved, the columns to be written etc. (For full description of the SBML and the structure of database queries please see references at the end of this document. Additionally, since there is often a need to perform data manipulation, SBML allows for the creation of "workingVariables" to store user data. A working variable can store a scalar string, a row of values, and a List of rows/scalars.

SBML allows users to use conditional statements to perform data manipulation and database queries. Users can also call other BusinessObjects and SubRoutines in the script (more on this later in the document). SBML also facilitates the creation of output BML documents using Business Object returns. There is flexibility to create tag names on the fly, loop through a set of values and other necessary characteristics for XML document creation.

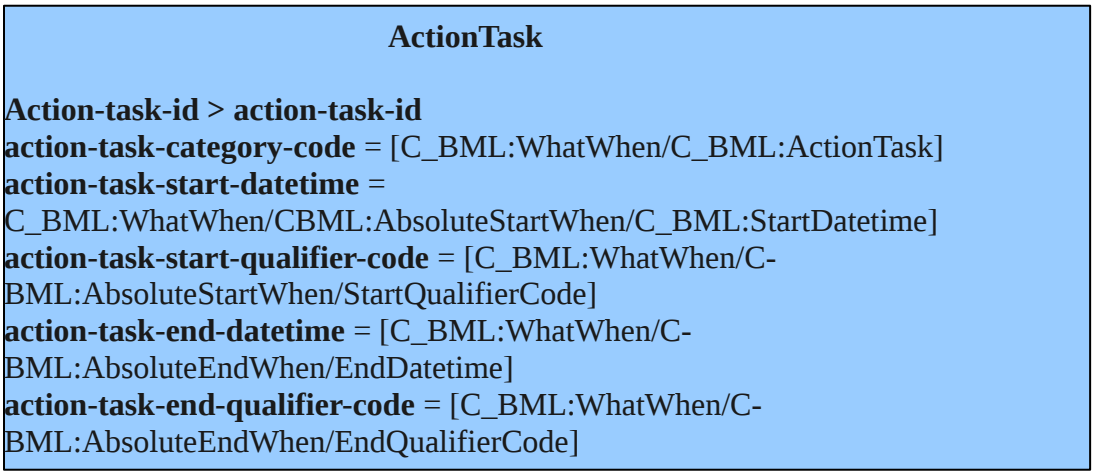
SBML has shown great potential as a basis for rapid development of supporting services for BML. It has been used in the annual NATO experimentation group to test and support interoperability within multiple C2 and M&S systems.

The need to think about Complete Thoughts came when we were working with the JC3IEDM Reference Implementation in association with Northrop Grumman. We realized that the Reference Implementation expects for some Business Objects (like Control Feature) that all the necessary tables and columns be updated before the transaction can be completed. It became evident that SBML needed a similar paradigm to work in tandem. I was part of the team of discussion that realized that a Business Object Transaction provides a framework to represent a Complete Thought. The rest of this document reports on the issue of "Complete Thought" and how SBML implements it.

Complete Thought: JC3IEDM-Annex O-XML-DMWG-Edition_3.0.2_20090514 defines a Complete Thought as a update or a query that constitutes a complete military thought. The document also suggests that a "Complete Thought" can be thought of as a Business Object. So, when it translates into SBML, a Complete Thought contains 'databaseQuery's that update all the mandatory elements to

complete the military thought and keep the database in a consistent state. We have noticed that Simulation systems that are subscribed to different 'Complete Thoughts' receive the updates as they are pushed to the Database using SBML. As an example consider WhatWhen as a Complete Thought:

The root element for WhatWhen is Action-Task. And the columns that need to be updated are illustrated in the figure 1.a.



(Fig 1.a) : *Figure illustrating the IDFX (slightly modified- please see notes below) representation for the Complete Thought 'WhatWhen'*

Notes on the representation above:

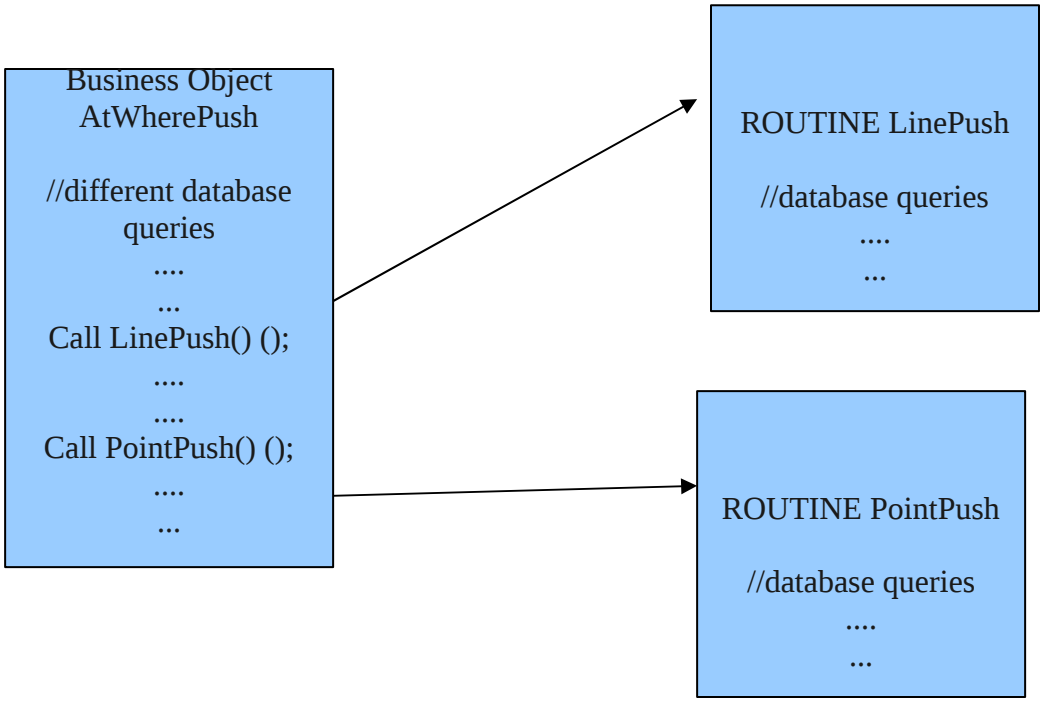
1) *The Tableb name and the column names are in bold.*
2) *If the value to be put in the column is from the input file then that is represented using [].*
3) *Actio-Task-id is the primary key for the Complete Thought and it is created as a new value using the '>' operator.*

Action task is a child entity of Action and so the creation of Action-Task involves the implicit creation of Action in the Database. The End time is optional in the input BML (as of Oct 2010) and SBML does not update the relevant columns if the input file does not have the optional values.

The mandatory columns are the primary key (Action-task-id), the Category Code, the Start Date Time and the StartTimeQualifier which are updated in this Buisness Object which makes it a complete military thought. Updating the Database using the above figure will result in a consistent Database snapshot and any retrieval of these set of data will make complete military sense for this Business Object.

Use of Business Objects for Complete Thoughts: In SBML, Business Objects are used to represent a set of Database queries (Gets and/or puts). This allows for a modular view of scripting which even helps in debugging and isolation of errors. SBML also allows for the creation of Sub Routines (using the keyword ROUTINE indead of BusinessObject) which have the same structure as Business Objects but do not allow the definition of <ri_start> and <ri_end>. This is done to separate the ideas of Business Objects and helper methods. For example, a AtWhere (CONTROL-FEATURE) and as such should be defined in a Business Object while any necessary reusable set of database queries should be defined in sub-routines. This is recommend use of Business Objects and Sub-routines. The following

figure (fig 1.b) is intended to illustrate the above principle.



(fig 1.b) *Figure illustrating the use of Business Objects and Routines in Complete Thoughts.*

Complete Thoughts in SBML: In SBML, Complete Thoughts are perceived still as a collection of database queries that are all anchored on a root table. Therefore, there needs to be a way to designate the start of the Complete Thought, the root table and the end of the Complete Thought.

The start of a Complete Thought and the root table can be represented using:

<ri_start table="" key="">

The 'key' specifies the primary key of the root table.

The end of the Complete Thought can be represented using:

<ri_end>

Within, the start and end of SBML allows databaseQueries.

Business Object Transactions being implicit Complete Thoughts:

SBML facilitates the representation of Complete Thoughts using Business Object Transaction (BOT). BOT's are the basis for molecularity in SBML- they represent a collection of data base queries and they can be called from other BOT's thus achieving re usability. A Complete Thought in the JC3IEDM reference implementation also has the property of being a collection of related and mandatory database operations. Therefore a BOT can be the representation of a Complete Thought in SBML. Additionally, SBML also has the command “commit” (works the same way a SQL commit works) that will push all recent database operations to the database. In this way, a script can, conceivably, have all database operations for a Control Feature Push and at the BOT have the commit statement. This takes away the need for <ri_start> and <ri_end>. In essence, a BOT has a implicit <ri_start> and <ri_end> since the collection of database operations are all done simultaneously.

Contrasting Business Object Transactions with Sub-routines.

There are often a collection of tables that need to be updated that do not account towards a Complete Thought (e.g. creation of a Point). Additionally, there are also a collection of tables that need to be updated together and often. To facilitate the representation of a re-usable collection of databaseQueries which do not form a Complete Thought, SBML has the notion of a sub-routine. Sub Routines can be created by the <Routine> tag (and ROUTINE keyword in CSL). A Sub routine can contain all the elements of a Business Object Transaction except <ri_start> and <ri_end>.

Routines are designed to (as the name suggests) to have useful pieces of script that can be called repeatedly without having to create Complete Thoughts.

As an example:

A Control-Feature has been identified as a Complete Thought which means that when a Control Feature needs to be created there are a set of tables (and columns) that all need to be updated and the database consistency needs to be maintained. And so the script needs to create a BusinessObjectTransaction for the Control feature Push. A Control Feature can have multiple points and each Point has about ten tables that need to be updated. So, a Routine can be created for a Point Push which can be called from Control Feature Push (or other Business Object Transaction). In this way, the Business Object Transaction can use the helper sub routines but the engine knows that since its a Business Object Transaction a Complete Thought is created.

Notes: As of now the only element that we know as a Complete Thought is a Control feature. The analysis here looks for all the tables that are needed to update the element in the JC3IEDM.

The following illustrates the possible Complete Thoughts in SBML Orders and the relevant Root Nodes and Primary Keys

1) Order:
   Root Table: REF
<ri_start table=”REF Key=”ref_id”>

Tables that are needed to perform a OrderPush are:
ACT, REF, ACT_TASK,ACT_REF_ASSOC, ORG_ACT_ASSOC

Name of the BOT:LowerOrderPush

2) Task (What+When)

   Root Table: ACTION-TASK, primary key: action-task-id

   Tables needed to perform a Task Push are: ACT, ACT_FUNCTL_ASSOC, ACT_RES

Name of the BOT:WhatWhenPush

Questions/Points to consider: Since Tasks can have both WhenTime and relativeWhen, the BOT right now checks if there are RelativeWhens and calls other BOT's for each relative When. Each of those BOT's update the tables ACT_TASK and ACT_TEMPRL_ASSOC. So, WhatWhenPush is not a Complete Thought until the RelativeWhen's are done in another Routine.

3) Where
   RootTable: 'ControlFeature' or 'Route' depending upon whether it is a Route or not.

Tables needed to perform a AtWherePush:
OBJ_ITEM,FEAT,CTRL_FEAT,CTRL_FEAT,OBJ_TYPE,FEAT_TYPE, CTRL_FEAT_TYPE, OBJ_ITEM_TYPE, LOC, OBJ_ITEM_LOC,
If it is a ControlMeasure does CONTXT_OI_ASSOC if not it does ACT_OBJVE and ACT_OBJVE_ITEM

Name of the BOT: AtWherePush calls BOT's: SurfacePush, which in turn calls

LinePush : LINE, POLYGON_AREA (if the 'WhereClass' is a 'SURFAC') which in turn calls

PointPush (for as many 'Locations' there are in the input file: LOC, POINT, VER_DIST, ABS_POINT, GEO_POINT, LINE POINT

Notes:
1) The <ri_end> has to be at the end of AtWherePush but since PointPush uses information from LinePush, we'll need to make sure that tables from LinePush are pushed before PointPush. Will we need a ri_start and ri_end in LinePush

4) *Who*

   a) TaskeeWho:

   Name of the BOT: TaskeeWhoPush- called by WhatWhenPush

   Tables: ACT_RES_ITEM

   Questions/Points to consider: The table ACT_RES is created in WhatWhenPush and ACT_RES_ITEM is created in TaskeeWhoPush. Will this cause a problem ?

b) AffectedWhoPush:

Name of the BOT: AffectedWhoPush- called by AffectedWhoPush

Tables: ACT_OBJVE and ACT_OBJVE_ITEM

5) *Why*

Name of the BOT : WhyPush- called by WhatWhenPush

Tables: ACT_EFFECT : uses the act_id of the Task

6) *NewWho*

Root Element: Unit, primary key= unit_id

Tables: UNIT, OBJ_ITEM, ORG, OBJ_TYPE, UNIT_TYPE, ORG_TYPE, GOVT_ORG_TYPE, MIL_ORG_TYPE, OBJ_ITEM_TYPE

If there is a Location then it calls NewWhoAtWherePush that pushes the Location as a Control feature

Conclusions:

1) SBML facilitates the creation of Complete Thoughts using Business Object Transactions. They have a implicit <ri_start> at the beginning and a <ri_end> at the end. Sub-routines can be created to have helper groups of database queries. Sub Routines can be called from Business Object Transactions.

Notes on W's and their relationship to Complete Thoughts
1) A task is not complete without all five W's but there are tables like ACT, ACT_FUNCTL_ASSOC, ACT_RES that need to be pushed to provide the “skeleton” of the ActionTask that the other W's can use to update JC3IEDM tables. A task does not really have a Root element and so we cannot use a ri_start and ri_end for it. Since the BOT's that do the W Push need the “skeleton” tables, we should make sure they are pushed into the Database before going to the W Push

1) For a Control feature, since there can be multiple points we need to call the BOT PointPush for each Point. This can be treated as a Sub-routine and having a ri_start at the start of AtWherePush and a ri_end at the end of AtWherePush will make AtWherePush a “Complete Thought”

3) For all the W's every table that is necessary to perform a valid composite is being updated. Every needed column is also being updated.

References:

Pullen, J., D. Corner and S. Singapogu, “Scripted Battle Management Language Web Service Version 1.0: Operation and Mapping Description Language,” IEEE Spring 2009 Simulation Interoperability Workshop, San Diego CA, 2009

Pullen, J., D. Corner and S. Singapogu, “Scripted Battle Management Language Web Service Version 2,” IEEE Fall 2009 Simulation InteroperabilityWorkshop, Orlando, FL, 2009

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

Dr. Mark Pullen, Douglas Corner, Samuel Singapogu, et al., “Adding Reports to Coalition Battle Management Language for NATO MSG-048”, IEEE/SISO Simulation Interoperability Workshop, 2009

Blais, C., Brown, D., Diallo, S., Heffner, K., Levine, S., Singapogu, S., St-Onge, M., and Scolaro, D.: “Coalition Battle Management Language (C-BML) Phase 1 Specification Development: An Update the M&S Community,” Paper 09F-SIW-001, Proceedings of the Fall Simulation Interoperability Workshop, Simulation Interoperability Standards Organization, Orlando, September 2009.