**Developer User’s Guide**

**For THE**

**ARRoW Model Interconnection Language (AMIL) LIbrary**

Contract No.: HR0011-10-C-0108

**June 22, 2011**

DRAFT

**Prepared by:**

BAE Systems

6 New England Executive Park

Burlington, MA 01803

(781) 273-3388

Table Of ContentsTable Of Contents i

1 AMIL Library Developer Concepts and Usage 2

1.1 AMIL Overview 2

1.2 AMIL Graph User Interface 2

1.3 AMIL Library / AMIL Manager 3

1.4 AMIL Object 4

1.5 AMIL Graph Node 4

1.5.1 Node Value Types 5

1.5.2 Node Parameters 5

1.5.3 Node Value Retrieval 5

1.6 AMIL Graph Link 6

1.6.1 Parsing Links During Node Value Retrieval 7

2 JAVADOC Production 8

# AMIL Library Developer Concepts and Usage

## AMIL Overview

AMIL is the ARRoW Model Interconnection Language. It is used to model interconnections between heterogeneous models while making minimal assumptions about the nature of these models. For example, AMIL makes no assumption about shared meta-models as is the case in many modeling tool chains such as Eclipse Modeling Framework. The strategy in AMIL is to only assume that external models and modeling tools provide some sort of access interface and that they are capable of exporting unique identifiers for modeling elements that are relevant across multiple models. AMIL creates proxies for these elements as nodes in an attributed graph. These nodes can be associated with related nodes (proxies of other model elements) with attributed links. The approach is fundamentally *lazy*, *i.e.* represent only the proxies that are required and only introduce the interconnections that are useful. The underlying attributed graph semantics provides a general and extremely flexible foundation in support of the model interconnection semantics. Both nodes and links of the interconnection network contain lists of key-value pairs that can be extended without arbitrary limit. New relationship types can be introduced on demand without compromising performance.

## AMIL Graph User Interface

The ArrowWebServices component provides a user interface (UI) for viewing the AMIL data, see Figure 1 Sample AMIL Graph. The UI is presented via HTML web pages and can be viewed using a browser.

The UI depicts “immediate” nodes, whose attributes are stored directly in the database, as magenta circles and “dynamic” nodes, whose attributes are obtained by calling external functions, as blue stars. Links are shown as arrows connecting the nodes. All links and nodes are labeled with an identifier derived from their internal name data. Moving the mouse over nodes provides a pop-up with detailed node information. Zooming in and out of the graph can be performed using the mouse wheel. The detail information for dynamic nodes reflects what would be passed to its associated function, not what was returned from it.

The UI is available when the ArrowWebServices are deployed. When run locally the ArrowWebServices pages are available at <http://localhost:8080/ArrowWebServices/>. From the main web page select the “View Arrow Graph” link.

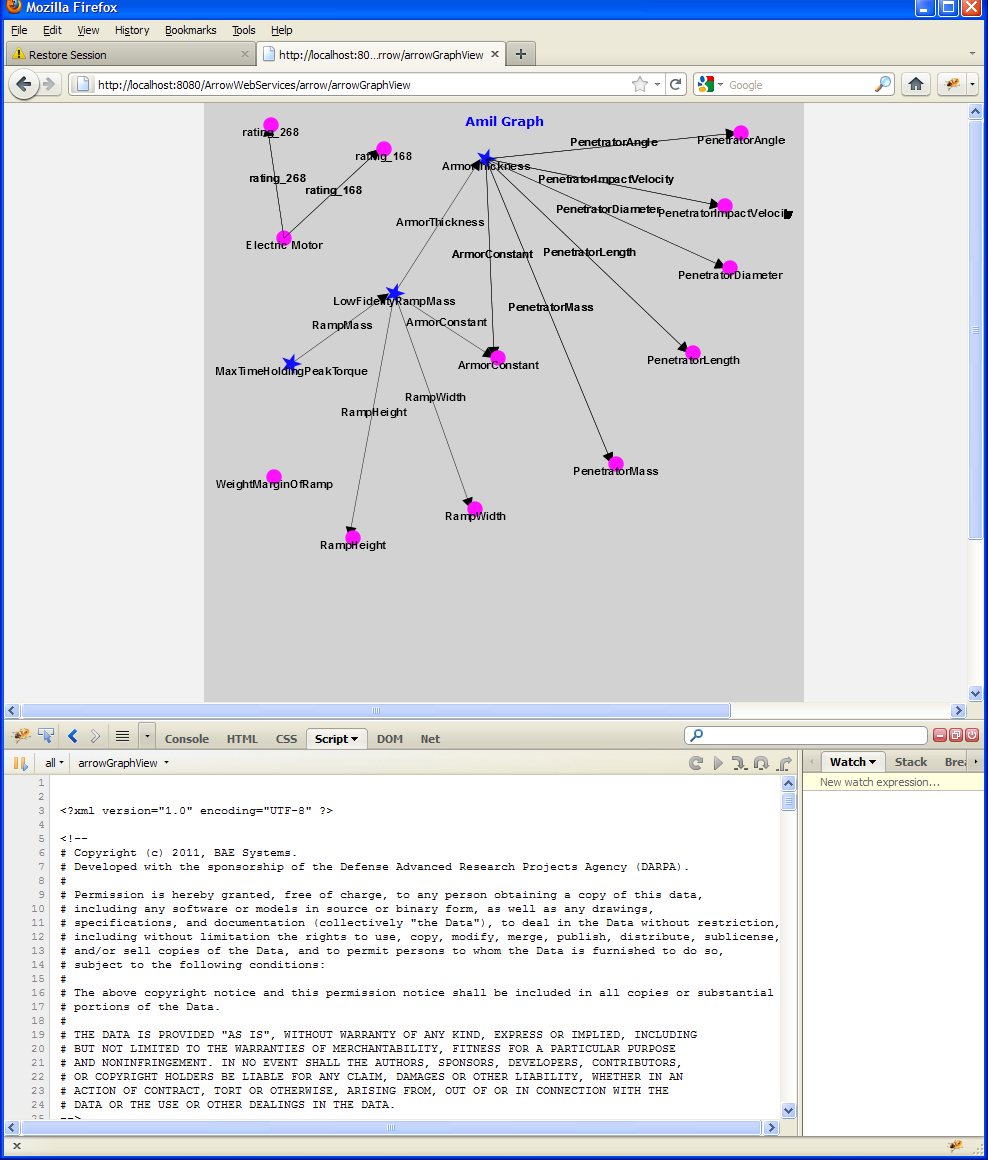


Figure Sample AMIL Graph

## AMIL Library / AMIL Manager

The AMIL Manager and AMI Lib class (com.bae.meta.amillib.internal.AmilManager & com.bae.meta.amillib.AmilLib) provide the primary application interface for AMIL data management. The methods provided by these classes allow for creation, deletion, retrieval and modification of AMIL graph data stored in the AMIL database. AmilManager methods are used relatively infrequently.

**Retrieval**

getLinksFromDB()

getLink()

getNode()

getAllNodes()

getEntireGraph()

getByQuery()

**Deletion**

markDeleted()

**Creation**

addObject()

createNode()

**Modification**

commit() – Updates AMIL database with any direct modification to AMIL Node and Link objects

The AMIL database is in general very light on semantics (just nodes and links). There are a few special fields that it interprets during node retrieval. Those working of those mechanisms are described in the sections that follow.

## AMIL Object

AmilObject (com.bae.meta.amillib.AmilObject) is the base class used for defining AMIL Graph Nodes and Links. This base class provides a structure in which an array of name / value pairs (“properties”) is used to define attributes of objects.

## AMIL Graph Node

AmilNode (com.bae.meta.amillib.AmilNode) provides the structure for defining AMIL graph node objects. AMIL graph nodes represent modeled entities and provide a value. Table 1 AMIL Node Structure provides a summary of the significant elements of an AmilNode.

|  |  |
| --- | --- |
| **Field** | **Usage** |
| name | Uniquely identifies the node. |
| nodeType | Identifies the type of node (IMMEDIATE\_VALUE or DYNAMIC\_VALUE). |
| properties | Specific attribute name/value pairs the node provides as part of its value. When a node is retrieved these values are provided. |
| outboundLinks | A list of links to nodes from which this node pulls data. |
| inboundLinks | A list of links to nodes for which this node provides data. |
| foreignName | When this attribute is provided its value overrides the value of “name”. |

Table AMIL Node Structure

### Node Value Types

Node values can be provided from static data or can be computed dynamically by incorporating data values from other nodes in the graph. Nodes that use static data to provide a value are identified as type “IMMEDIATE\_VALUE”. Nodes that compute their values dynamically are identified as type DYNAMIC\_VALUE. The node type is maintained within the node’s internal data as “nodeType” attribute.

Entity Width

Entity Height

Entity Depth

Entity Volume

Figure 2 Node Type Example

Figure 2 Node Type Example illustrates the use of both types of nodes. The nodes (“Entity Width”, “Entity Height” and “Entity Depth”) are IMMEDIATE\_VALUE type nodes with fixed values of 2, 3 & 4 meters respectively. Whereas “Entity Volume” is an DYNAMIC\_VALUE node and its value is computed dynamically using the current values from the other 3 nodes to produce a value of 24 meters3.

Note: The direction of the arrows in the graph indicates control flow rather than data flow. “Entity Volume” actively retrieves data from “Entity Width”, “Entity Height” and “Entity Depth” in order to perform its value computation.

### Node Parameters

AMIL graph nodes of the DYNAMIC\_VALUE type may require input parameters in order to compute their value as described above. The node data structure includes a list of all required input parameters. Outbound links whose type is “parameter” cause values to be pulled into the node from the link destination, for the use during the computation of the node’s attribute list. All input parameter data needs must be satisfied in order for the node to properly compute a value. The satisfaction of node parameters is achieved with AMIL graph links. The links identify the source node of the required input parameter. Each node maintains a list of its links. AMIL graph links are further described below.

### Node Value Retrieval

In order for nodes to provide a value upon retrieval some processing must occur. The following sections describe the retrieval process for the various node types.

#### IMMEDIATE\_VALUE Node Value Retrieval

Nodes of the IMMEDIATE\_VALUE type maintain their value within the node data. When an IMMEDATE\_VALUE node is retrieved its value can be obtained by simply reading its stored data.

#### DYNAMIC\_VALUE Node Value Retrieval

Nodes of the DYNAMIC\_VALUE type require a separate executable to be invoked in order to compute a value upon retrieval. The name and path of the executable are maintained in the “executableName” attribute of the node. When a DYNAMIC\_VALUE node is retrieved the executable is invoked with values for the required input parameters. Obtaining the value of the input parameters first requires execution of the foreign nodes that provide the data. The foreign nodes are identified in the local node’s link data. Since the foreign nodes could also be of DYNAMIC\_VALUE type the local node’s retrieval and subsequent execution may trigger a cascade of node executions in order to complete. For example, in Figure 1 Sample AMIL Graph, the retrieval of the “MaxTimeHoldingPeakTorque” node first requires retrieval of the “LowFidelityRampMass” node which in turn first requires retrieval of the “ArmorThickness” node.

Note: Currently there is no guard against circular dependencies where two or more DYNAMIC\_VALUE nodes depend upon each other.

#### DYNAMIC\_VALUE Node Value Retrieval With Parameter Override

DYNAMIC\_VALUE nodes can also be executed with user provided input parameters. User provided input parameters will override the need for foreign node values. For example, in Figure 1 Sample AMIL Graph, if during retrieval of the “MaxTimeHoldingPeakTorque” node a value is provided for “LowFidelityRampMass” then the link for the parameter will not be followed and the “LowFidelityRampMass” will not be retrieved. Some or all parameters can be overridden. Parameter override is useful for localized testing and prototyping.

## AMIL Graph Link

AmilLink (com.bae.meta.amillib.AmilLink) provides the structure for defining AMIL graph link objects. AMIL links associate data from one node to another. Links contain a “from” attribute identifying the node that is requesting the data and a “to” attribute defining the node that is providing the data. A single link can be used to import some data or all data from one node to another. The specific data to be imported is identified via the link’s name field. If this field contains the special value of “multi” all data is imported from one node to the other, otherwise only the single data item identified by OBJECT\_ID is imported. In the case where a single data item is identified and the link contains a “foreignValueName” attribute, the value of this attribute is used to override the data item identified by OBJECT\_ID. A summary of a links structure is provided in Table 2 AMIL Link Structure.

|  |  |
| --- | --- |
| **Field / Attribute** | **Usage** |
| name | Identifies the specific data the link refers to. A value of “multi” is used to refer to all data. |
| type | Identifies the type of link. Currently the only value supported is “parameter”. |
| from | The node which is importing the data. |
| to | The node from which data is to be pulled. |
| foreignName | When this attribute is provided its value overrides the value of “name”. |

Table AMIL Link Structure

### Parsing Links During Node Value Retrieval

When a DYNAMIC\_VALUE node is retrieved the list of links data stored with the node is parsed to obtain input parameter data. For each individual link the node providing data is retrieved. The specific data identified by the link is then imported from the retrieved node and matched to the corresponding input parameter.

# JAVADOC Production

Javadoc can be obtained for “AMILLib” by performing the following:

1. From a Windows, cygwin or linux command prompt set the current directory to the META Projects SVN trunk/AmilLib directory.
2. Execute the following maven command

**mvn javadoc:javadoc**

1. The output is placed in the trunk/AmilLib/target/site/apidocs
2. From a browser open the trunk/AmilLib/target/site/apidocs/index.html file. This is the root of the Javadoc HTML pages.

Javadoc can be obtained for “ArrowWebServices” by a similar process to the above:

1. From a Windows, cygwin or linux command prompt set the current directory to the META Projects SVN trunk/ArrowWebServices directory.
2. Execute the following maven command

**mvn javadoc:javadoc**

1. The output is placed in the trunk/ ArrowWebServices /target/site/apidocs
2. From a browser open the trunk/ArrowWebServices/target/site/apidocs/index.html file. This is the root of the Javadoc HTML pages.