1 July 2015

Design Documentation

SysML to AADL Translator

User’s Manual

Version 1.0

Prepared for DARPA STO

System of Systems Integration Technology and Experimentation

SoSITE

Contract FA8650-14-C-7437

|  |  |
| --- | --- |
| **Technical Point of Contact:**  **Dr. Steven P. Miller**  Rockwell Collins, Inc.  400 Collins Road NE  Cedar Rapids, IA 52498  Telephone: (319) 295-2055  <Steven.Miller@rockwellcollins.com> | **Business Point of Contact:**  **Ms. Kelsey Rwayitare**  Rockwell Collins, Inc.  400 Collins Rd. NE, MS 121-200  Cedar Rapids, IA 52498  Telephone: (319) 263-3101  [kelsey.rwayitare@rockwellcollins.com](mailto:kelsey.rwayitare@rockwellcollins.com) |



The views, opinions, and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

**Table of Contents**

[1 Summary 4](#_Toc420506556)

[1.1 Installing the SysML to AADL Translator 5](#_Toc420506557)

[2 Using the Translator 6](#_Toc420506558)

[2.1 Importing a SysML (XMI) Model 6](#_Toc420506559)

[3 Creating SysML Models for Translation 8](#_Toc420506560)

[3.1 Creating SysML Block Diagrams 13](#_Toc420506561)

[3.1.1 Creating a SysML Type Block 13](#_Toc420506562)

[3.1.2 Creating a SysML Implementation Block 13](#_Toc420506563)

[3.2 Creating SysML Internal Block Diagrams 14](#_Toc420506564)

[3.3 Adding Properties to SysML Elements 14](#_Toc420506565)

[3.4 Adding State Behavior 14](#_Toc420506566)

[3.4.1 Hierarchical State Machine 16](#_Toc420506567)

[4 Translating AGREE Contracts 17](#_Toc420506568)

[5 Additional AADL Constraints on SysML Models 19](#_Toc420506569)

**List of Figures**

[Figure 1 – Successful Installation of the SysML/AADL Translator 5](#_Toc420506570)

[Figure 2 – Navigating to XML file 6](#_Toc420506571)

[Figure 3 - Importing Agree contracts 7](#_Toc420506572)

[Figure 4 – Relationship of SysML and AADL 9](#_Toc420506573)

[Figure 5 – SysML Profile for AADL Components 10](#_Toc420506574)

[Figure 6 – SysML Profile for AADL Features 11](#_Toc420506575)

[Figure 7 – Example SysML Block Diagram 12](#_Toc420506576)

[Figure 8 – Example SysML Internal Block Diagram 12](#_Toc420506577)

[Figure 33 – Text for AGREE Contract 17](#_Toc420506578)

[Figure 34 – Inserted AGREE Contract 18](#_Toc420506579)

|  |  |  |  |
| --- | --- | --- | --- |
| **Release History** | | | |
| **Date** | **Version** | **Author** | **Description** |
| July 1, 2015 | 1.0 | Siddhartha Bhattarcharyya | Initial Release |
|  |  |  |  |
|  |  |  |  |

# Introduction

The SysML to AADL translator translates system architectural models specified in a subset of the Systems Modeling Language (SysML) to models specified in a subset of the Architectural Analysis and Description Language (AADL). SysML is an open standard published by the Object Management Group (OMG) for the specification of system architectures. AADL is an open standard published by the Society of Automotive Engineers (SAE) for the specification of system architectures for embedded systems.

The SysML models are created using the Sparx Systems Enterprise Architect® (EA) tool and then exported as an Extensible Markup Language (XML) Metadata Interchange (XMI) file using the export option provided by EA. The translator currently supports the 2.1 version of XMI. A SysML profile for AADL extends SysML with constructs commonly used in AADL. These include AADL component categories such as system, process, thread, data, device, processor, bus and memory and AADL features such as port, data access, and bus access. These constructs are provided on an AADL toolbar palette in Enterprise Architect to simplify the development of SysML models that can be translated to AADL.

The SysML translator only translates a subset of the SysML constructs. A mapping of SysML constructs and the AADL constructs to which they translate is shown in Table 1.

The translator is invoked from within the OSATE (Open Source Architectural Tool Environment). OSATE is an open-source Eclipse-based framework for constructing and analyzing AADL models. The translator is written in the Java programming language and is packaged as plug-ins for the Eclipse development environment containing Java source code, executable byte code, and supporting XML files. It can be installed using the standard Eclipse facilities for installing new software. Once installed, new menu items are provided to the OSATE user to import a XMI-based XML model generated by Enterprise Architect.

Instructions for installing the translator are given in Section 1.1. Instructions for using the translator are given Section 2. Details on how to construct SysML and AADL models suitable for translation are provided in Section 3.

Table 1 - SysML to AADL Objects

|  |  |
| --- | --- |
| **SysML Structure** | **AADL Object** |
| Block | ComponentType |
| Block as a Client in Realization Relation | ComponentImplementation |
| Flowports | Ports |
| Binding Connector | Connection |
| SysML part (block property) | Subcomponent |
| State | Boolean variable in an AGREE annex |
| Transition | Boolean variable in AGREE |
| Transition Guard | Boolean expression (in a transition) |
| Transition Action | Guarantee in a ComponentType  Assertion in a ComponentImplementation |
| Tagged Value | AADL property |

## Installing the SysML to AADL Translator

Install a production version of OSATE if you do not already have it installed. The translator has been tested on OSATE 2.1.1. This version of OSATE can be obtained from:

<https://wiki.sei.cmu.edu/aadl/index.php/OSATE_2_download_page>

Click on the Products link then 2.1.1 followed by clicking on products. Now download the OSATE version for your system.

Start OSATE. Select Help → Install New Software … Select Add… to add a new update site. Enter “SysML to AADL EA 21” as the name of the update site repository and select Local … to navigate to the location of the update site in the Distribution file. Be sure to use

../Distribution/SysML to AADL EA21 Update Site/  
 com.rockwellcollins.atc.sysml2aadl.ea21.updatesite

as the location of the update site. Uncheck the “Group Items by Category” box, select SysML Translator (EA XMI 21) in the main window and select Next. Select Next on the Install Details screen, accept the license terms, and select Finish. Select “OK” on the Security Warning pop-up, and select to restart OSATE. When OSATE restarts, you will see a menu item for SysML at the top of the window if the translator has installed correctly (Figure 1).

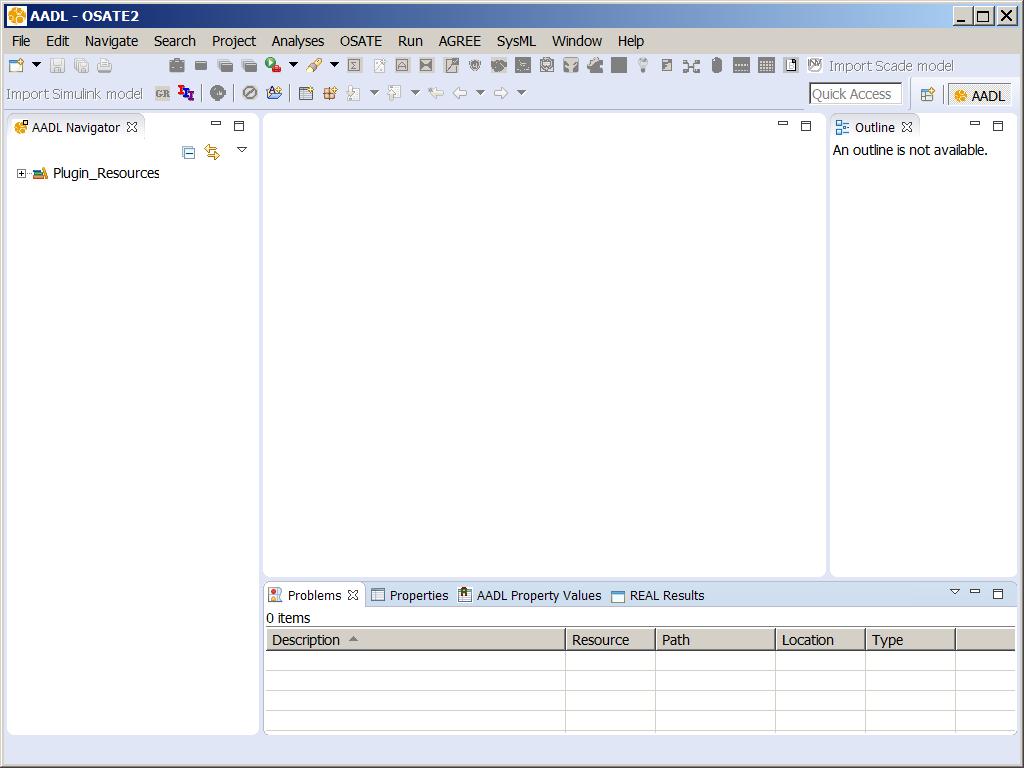


Figure 1 – Successful Installation of the SysML/AADL Translator

# Using the Translator

This section provides a quick guide on how to import a SysML model. Detailed information on how to build a SysML model that can be translated is given in Section 3 and Section 4.

## Importing a SysML Model

To import an Enterprise Architect SysML model into OSATE, it must first be exported to an XMI 2.1 file using the export capability of Enterprise Architecture.[[1]](#footnote-1) The XMI file can then be imported into OSATE by selecting SysML → Import Enterprise Architect XMI 2.1 from the menu bar in OSATE. A dialog window should appear indicating that the import has started along with window allowing you to select the appropriate XMI file as shown in (Figure 2).

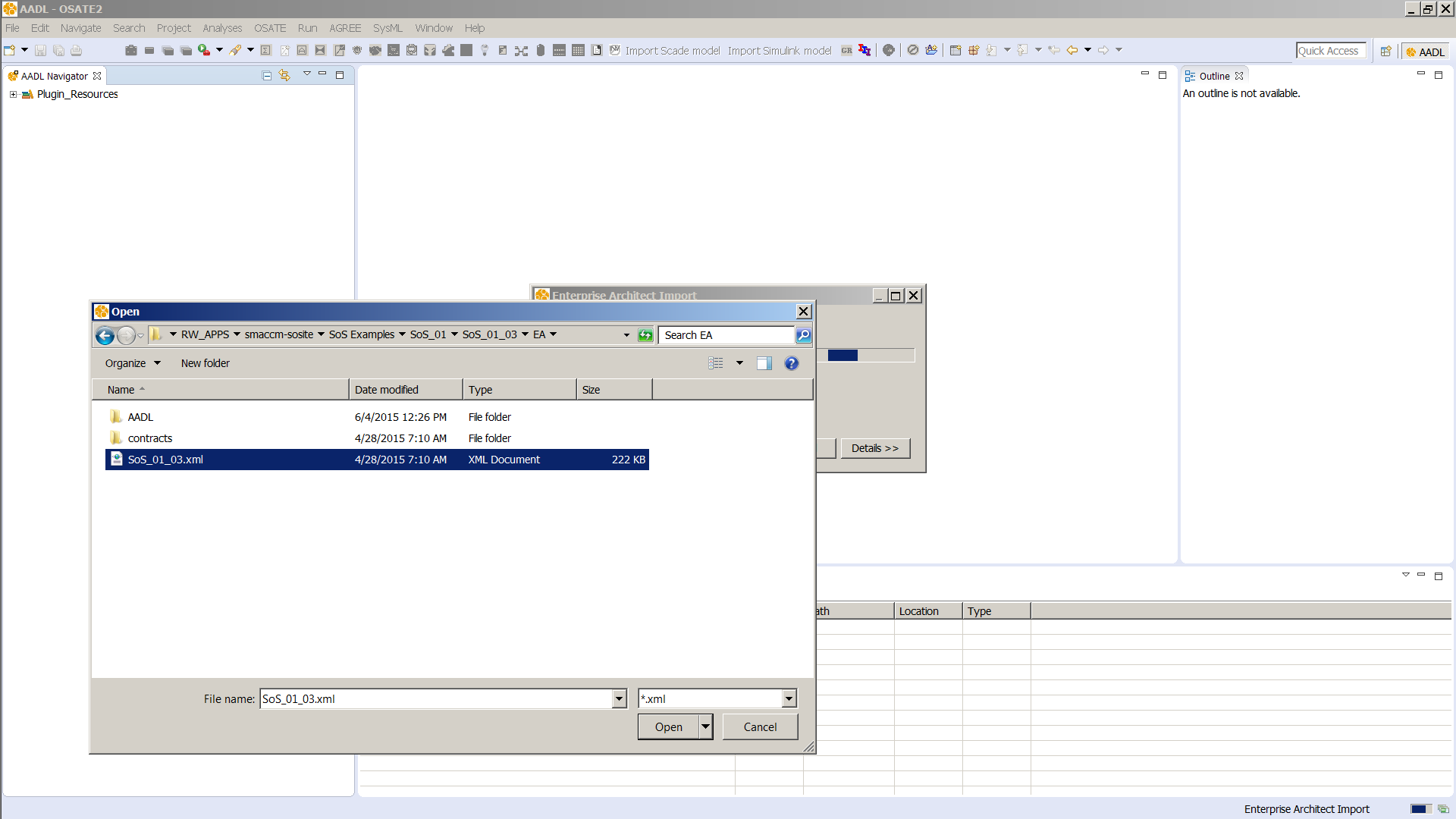


Figure 2 – Navigating to the XMI file

Navigate to the desired XMI file and open it. The importation of the SysML model should begin. If a folder named “contracts” exists in the same folder as the XMI file, AGREE contracts found in that folder will be merged with the generated AADL. If the contracts folder doesn’t exist in the same folder as the XMI file then a windows dialog box will open up for you to browse to a file within the “contracts” folder. If there are, no contracts to be included just click the Cancel button on the dialog box.

The first time a SysML model is imported, it may be necessary to add hand-coded AADL files to the AADL model by hand. For example, the Pilot Flying example references the hand-coded QS\_Properties.aadl property file and the Agree\_Nodes.aadl file containing common AGREE subroutines. After importing the SysML model the first time, copy these files to the appropriate location (usually the propertysets directory for property files and the packages directory for other files) , then import the SysML model again as described above. This should resolve any errors caused by the absence of these files during the translation. If any error messages remain, select the model root it in the AADL Navigator window (left side of Figure 2), then select Project → Clean … from the menu bar. This will rebuild the internal representation of the model in OSATE and flag any remaining errors in the Problems Window. Any remaining errors should be corrected in the SysML model, exported to XMI 2.1 version and the model reimported.

# Creating SysML Models for Translation

SysML is a general-purpose graphical modeling language for representing systems. These systems may include hardware, software, data, people, facilities and natural objects. SysML is based on UML and has many similarities to UML. SysML can represent the following aspects of systems, component and other entities:

* Structural composition
* Behavioral representation: state based, function based, message based
* Constraints: physical, performance properties
* Allocation: behavior, structure and constraints
* Relationships of requirements to other requirements, design elements, test cases

SysML includes nine diagram types as enumerated below:

* Package diagrams: Organization of a model in terms of packages
* Requirement diagrams: Text based requirements and relationship to requirements, design elements and test cases to support requirements traceability
* Behavior diagrams: Representing the behavior, functionality of a system
  + Activity diagrams: order in which actions execute to transform input to output
  + Sequence diagrams: sequence of messages exchanged between systems
  + State Machine diagrams: behavior of an entity in terms of transitions between states
  + Use case diagrams: how a system is used by external entities
* Parametric diagrams: constraints on property values
* Structure diagrams: representing the structural elements of a system
  + Block definition diagrams: represent structural elements called blocks and their composition
  + Internal Block Diagrams: represent interconnection and interfaces between parts of blocks

Each diagram represents a particular aspect of the system model.

*The current translator only supports block, internal block, and state machine diagrams.*

AADL was designed to provide modeling and analysis capability for engineering systems. These models are architecture models of systems with software, hardware, and physical system components, their interactions, and properties of these elements. AADL was designed for modeling real-time embedded systems and thus lacks many of the more general modeling diagrams of SysML such as requirement and use case diagrams. On the other hand, SysML lacks many of the constructs needed to model embedded systems such as processes, threads, processors, buses, and memory. While UML, SysML, and AADL can all be extended to fit almost any domain, UML was originally developed for modeling software, SysML was originally developed for modeling systems, and AADL was developed for modeling real-time embedded systems. Due to its more rigorous semantics and more limited domain, AADL typically provides greater support for model analysis. Figure 4 illustrates one way of viewing the intended domains and analytic capabilities of SysML, AADL, and UML.



Figure 4 – Relationship of SysML and AADL

The area of overlap between SysML and AADL is the region where translation between the two notations is feasible. For SysML this consists primarily of the blocks, ports, and connections depicted on block diagrams and internal block diagrams and behavior specifications such as state machines, activity diagrams, and message sequence charts. For AADL, this region consists of systems, features (such as ports), subcomponents, connections, and annexes for specifying behavior such as AGREE or the AADL Behavior Annex.

SysML does not provide specific constructs to model embedded systems the same way that AADL does. To enable SysML developers to create models in which blocks and ports can be translated into specific AADL constructs, a SysML profile was created as shown in Figure 5 and Figure 6 for AADL. Note that it is not required to use this profile when creating SysML models. By default, a SysML block is translated into an AADL system, a SysML part is translated into an AADL subcomponent, and a SysML port is translated into an AADL port. However, SysML constructs that adhere to this profile are automatically translated into the appropriate AADL constructs.

In the AADL profile of Figure 5, AADL components are modeled as SysML *blocks* (which are in turn UML *Class* objects). Every AADL component is stereotyped with the *component* stereotype. An AADL component is further stereotyped as one of *system*, *thread*, *process*, *data*, *bus*, *memory*, *device*, or *processor*.[[2]](#footnote-2) These stereotypes are automatically added when a component is selected from the AADL palette in SysML.



Figure 5 – SysML Profile for AADL Components

*The AADL components Abstract, Subprogram and Thread Group are not supported.*

The SysML profile for AADL features is shown in Figure 6. AADL features are modeled as SysML *FlowPorts* (which are in turn UML *port* objects). Every AADL feature is stereotyped with the *feature* stereotype. An AADL feature is further stereotyped as one of *bus\_access*, *data\_access*, or *port*, and ports can be further stereotyped as *dataPort*, *eventPort*, or *dataEventPort*. Direct access features such as *bus\_access* or *data\_access* also have *requires* and *provides* attribute. These stereotypes are automatically added when a feature is selected from the AADL palette.



Figure 6 – SysML Profile for AADL Features

*AADL Port Groups are not supported.*

## Creating SysML Block Diagrams

Not all SysML models will be correctly translated by the translator. This section provides guidance on the SysML constructs that can be translated and the conventions that need to be followed in the modeling process. A SysML model can be organized in packages. These packages represent the structure in which the SysML components are organized. Packages can be nested within packages and this structure will be recreated in the AADL model.

AADL models contain *Component Types* and their *Implementations*. A component type defines the external interfaces of a component, while a component implementation defines the internal structure of the component.

SysML does not make such a distinction between type and implementation, so this is accomplished in SysML by creating two blocks with a Realization relation between them, one representing the component type and the other the implementation as shown in Figure 7.



Figure 7 – Example SysML Block Diagram

A SysML block to be translated into an AADL component type should have only externally visible features, such as ports. An example is shown in Figure 7 for the *Pilot\_Flying\_Block*. A SysML block to be translated into an AADL component implementation should inherit all its externally visible features from its type as shown in Figure 7 for the *Pilot\_Flying\_Impl* block. An implementation block may have internal parts specified as SysML properties with connections between them. Figure 8 shows an internal block diagram depicting the internal structure of the *Pilot\_Flying\_Impl* block.



Figure 8 – Example SysML Internal Block Diagram

### Creating SysML Type Blocks

For each AADL type in a package, create a SysML type block by selecting the appropriate AADL component (system, data, process, thread, bus, device, memory, or processor) from the AADL toolbox or by simply using a SysML block (SysML blocks without AADL stereotypes are translated into AADL system blocks). Next, add the features for the type by dragging the appropriate AADL feature (port, data access, or bus access) from the AADL toolbox or by simply using SysML 1.3 flowports (which are translated into AADL data ports). Finally set the direction of the ports and their types. Ports can also be defined as arrays by associating multiplicity for a port in Sysml. This can be done by selecting the port and then opening the corresponding property window and then clicking onto the property menu. This view provides the multiplicity option which can be set by providing values to the *lower* and *upper bound*.

### Creating SysML Implementation Blocks

To create a SysML block to be translated as an AADL implementation, open the block diagram containing the component’s type block. Select the appropriate component type from the AADL palette and drag the component to the block diagram canvas. The examples follow the convention of naming the implementation the same as its type block and attaching the suffix “Impl”. For example, to create an implementation of the *Pilot\_Flying* type, the name of the implementation block would be *Pilot\_Flying\_Impl*.

Next, make the block an implementation of its type by selecting the *Realization* relation on the toolbox and then dragging from the implementation block to the type block.

*The current version of the translator only supports one implementation per type.*

## Creating SysML Internal Block Diagrams

The internal structure of a component such as its subcomponents and their connections can be specified by creating a SysML internal block diagram (ibd). Create an ibd diagram for the implementation block for which the internal structure is to be defined. For each SysML part (AADL subcomponent) to be added, drag a SysML type block (i.e., a SysML block defining the component type) onto the ibd. This will add the type as a SysML property (part) of the implementation block.

Array of parts in ibd can be implemented by associating multiplicity with that part. Each part is selected by opening the property window and then clicking onto the property menu. This view provides the multiplicity option which can be set by providing values to the *lower* and *upper bound*.

In order to represent interaction among the components add connections between the ports, by selecting the Connector option on the toolbox. Each connection should be named. Each connection can also be associated with a connection pattern that signifies the representation of arrays or multiplicity of subcomponents or ports. The connection pattern is included as a tag *AADL::Property* and the value is represented as *Connection\_Pattern::NameofPattern* in AADL.

AADL “direct access connections” can be made by adding an AADL *Data Access* or *Bus Access* port at both endpoints and then making the connection as normal. In addition, a direct access connection can be made directly from an AADL Data or Bus component (i.e. without a port) if the port being connected to is a *Data Access* or *Bus Access* port.

## Adding Properties to SysML Elements

In AADL, an element within a model can have properties associated with it that provide additional information about that element. For example, a connection may be assigned a latency property that has a value of 50 msec. Properties typically have a name, a type, and constraints on which AADL elements they can be applied to. AADL comes with a large number of predefined properties and user defined properties can be specified in a property set. When a user defined property is applied to an AADL element, its name must include the name of the property set and the name of the property. For example, the AADL property “Clock\_Name” defined in the “QS\_Properties” property set would be assigned to an AADL element using the property name “QS\_Properties::Clock\_Name”.

AADL properties can be added to a SysML element using “tagged values”. The name of the tag must be “AADL::Property” and the value of the tag must be the property set name, the property name, and the property value, where the value of strings include quotes. For example, the property described above would be specified in SysML as a tagged value with the tag name “AADL::Property and the value QS\_Properties::Clock\_Name::"CLK1”. For a list of properties associated with a component the tag value is a comma separated list such as QS\_Properties::Clock\_Name::"CLK1", QS\_Properties::Primary\_Side::True.

## Adding State Behavior

The behavior of a block can be modeled in SysML as a state machine diagram. State machines are owned by a block and execute within the context of an instance of that block.

A SysML state machine contains one or more states and transitions between these states. Transitions include an optional guard that must be satisfied for the transition to be taken and an optional set of actions affecting local variables and output ports when the transition is taken. A state machine begins execution in its *Initial State*. At each step, each transition from the current state is evaluated to determine if it can be taken. While it may be possible to take several transitions on a step, one will be selected non-deterministically from the possible transitions. When a transition is taken, its actions are executed to update the local state and the destination state becomes the new current state. If no transition can be taken, no change occurs in the component’s local state or outputs on that step. Some states may be designated as *Final States*, meaning execution terminates if it is reached.

Local variables can be defined as attributes of the enclosing block. Local variables can be interrogated by a transition guard and set by a transition action. Each state may also have an optional “entry behavior” that is executed on entry to the state, an optional “exit behavior” that is executed in exit from a state, and an optional “during behavior” that is executed on each step while in the state. Each such behavior specifies a set of actions that can change local variables or output ports associated with the block.

State machines can be hierarchical in that they can contain other state machines. When a transition is made into a state machine, its initial state becomes the current state. When a transition is made out of a state machine, the current state within the nested state machine is exited.

Concurrent state machines (i.e., state machines that execute in parallel) are not supported in the current translator.

The translator will create an AGREE annex for each component that contains a state machine. The AGREE annex defines the behavior of a state machine as an assume/guarantee contract specifying how the outputs of the component change in response to changes in its inputs. The following section describe conventions that should be followed for a SysML state machine to be correctly translated into a valid AGREE annex.

1. State Machines

A state machine can be added to a SysML block for a type or an implementation. A state machine may also be added to an existing state machine as a nested state machine.

1. Local Variables

Local variables can be added as SysML attributes of the parent SysML block of a state machine. Local variables are accessible globally within the state machine and its descendent state machines.

The local variable name should start with *local::* as the identifier followed by the variable name. This is required for the translator to distinguish between local variables and other attributes like the subcomponents.

1. States

One or more states can be added to each state machine. Each state must have unique name within the state machine and one state must be identified as the initial state.

1. Entry, Exit, and During Actions

Entry actions may be added to a state as SysML “entry” operations, exit actions may be added to a state as SysML “exit” operations, and during actions may be added to a state as SysML “do” operations. The SysML “behavior” associated with each operation is translated directly into AGREE as constraints that must hold after the state is entered, exited, or is active, respectively. For this reason, the SysML behavior must be valid AGREE syntax.

1. Transitions

A SysML transition can be added from a state to another state or another state machine. Each transition may have an optional SysML guard and an optional set of actions specified as the SysML effect. The guard is translated directly into an AGREE predicate that must hold for the transition to occur and must be valid AGREE syntax. A guard may reference local variables and input ports. The actions are translated into AGREE constraints that must hold when the transition is completed and must be valid AGREE syntax. Actions may set local variables and output ports.

# Translating AGREE Contracts

AGREE is an OSATE plugin that allows assume/guarantee contracts to be associated with AADL component types or implementations. AGREE also supports formal verification that an implementation satisfies its contract. Details on installing and using AGREE can be found at

<https://github.com/smaccm/smaccm>

As already described, an AGREE annex will be generated for state machines associated with a SysML block. It is also possible to associate a hand-code AGREE contract with any SysML block translated into an AADL component.

Contracts for individual components should be placed in a directory “contracts” located in the same directory as the SysML XMI file being translated. To associate an AGREE contract with an AADL type with name TYPENAME, place a text file with name PATHNAME.TYPENAME.txt in this directory, where PATHNAME is a list of the enclosing packages separated by periods. To associate an AGREE contract with an AADL implementation with name IMPLNAME, place a text file with name PATHNAME.TYPENAME.IMPLNAME.txt in this directory, where TYPENAME is the name of the implementation’s AADL type. These files should contain the body of the AGREE contract without the enclosing

agree {\*\* …. \*\*}

For example, the file contracts/Pilot\_Flying.Cross\_Channel\_Bus.txt containing the following text

eq ffT: bool = Agree\_Nodes.fT(2);

guarantee "O\_Is\_Pre\_I" : O =

if ffT then pre(I) else Get\_Property(this, QS\_Properties::Init\_Bool);

Figure 33 – Text for AGREE Contract

will insert an AGREE contract into the Cross\_Channel\_Bus component type as shown in Figure 37.

**system** Cross\_Channel\_Bus

**features**

I: **in** **data** **port** Base\_Types::Boolean;

O: **out** **data** **port** Base\_Types::Boolean;

**annex** agree {\*\*

-- Inserted from ../contracts/Pilot\_Flying.Cross\_Channel\_Bus.txt

**eq** ffT**:** **bool** **=** Agree\_Nodes**.**fT**(**2**);**

**guarantee** "O\_Is\_Pre\_I" **:** O **=**

**if** ffT **then** **pre(**I**)** **else** **Get\_Property(this,** QS\_Properties**::**Init\_Bool**);**

\*\*};

Figure 34 – Inserted AGREE Contract

If the AGREE plugins are not installed, the contract will be treated as comments by OSATE. If the AGREE plugins are installed, the text will be parsed by the plugins for correct syntax and used to construct an internal representation of the contract in OSATE.

# Additional AADL Constraints on SysML Models

In addition to the constraints identified in the preceding sections, there are also several constraints that one needs to impose on SysML by the semantics of AADL to support representation of more constructs from AADL. For example, a thread subcomponent must be contained within a process component. These constraints are specified in Table 2 which defines the features, parts, parent, and direct access connections allowed for each AADL construct.

Table 1 – AADL Constraints

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Category** | **Features** | **Parts** | **Parent** | **Notes** |
| System | Port  Portgroup†  Subprogram†  Data Access  Bus Access | System  Data  Process  Processor  Memory  Bus  Device | System |  |
| Data | Subprogram†  Data Access (Provides) | Data | System Process Thread Data | Direct Access allowed |
|  |  |  |  |  |
| Process | Port  Subprogram†  Data Access | Thread  Data  Threadgroup† | System |  |
| Thread | Port  Subprogram†  Data Access | Data | Process |  |
| Abstract† |  |  |  |  |
| Threadgroup† |  |  |  |  |
| Subprogram† |  |  |  |  |
|  |  |  |  |  |
| Processor | Port  Portgroup†  Subprogram†  Bus Access (Requires) | Memory | System |  |
| Memory | Bus Access (Requires) | Memory | System  Processor  Memory |  |
| Bus | Bus Access |  | System | Direct Access allowed |
| Device | Port  Portgroup†  Subprogram†  Bus Access (Requires) |  | System |  |

† Not Supported

1. The XMI file created by Enterprise Architect will be of type .xml. [↑](#footnote-ref-1)
2. If the translator encounters a SysML block without a stereotype, it will translate it into an AADL system component. [↑](#footnote-ref-2)