AGREE Simulator Developer Guide

Version 0.1

Philip Alldredge, University of Alabama in Huntsville  
Ryan McIlnay, University of Alabama in Huntsville

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 Introduction 1](#_Toc458503491)

[2 High Level Design 1](#_Toc458503492)

[2.1 User Interface 2](#_Toc458503493)

[2.1.1 UI Simulation Service 2](#_Toc458503494)

[2.1.2 Eclipse Launch Shortcut 3](#_Toc458503495)

[2.1.3 Eclipse Commands 3](#_Toc458503496)

[2.1.4 Views 3](#_Toc458503497)

[2.1.5 Charts 3](#_Toc458503498)

[2.1.6 OSATE Graphical Editor Extensions 3](#_Toc458503499)

[2.2 Simulation Backend 3](#_Toc458503500)

[2.2.1 Simulation Service 4](#_Toc458503501)

[2.2.2 Eclipse Launch Configuration Type 4](#_Toc458503502)

[2.2.3 AGREE Simulation Engine 4](#_Toc458503503)

**List of Figures**

[Figure 1 – Conceptual Overview 1](#_Toc458498618)

[Figure 2 - Structural Overview 2](#_Toc458498619)

[Figure 3- Simplified AGREE Simulation Program Transformation Process 5](#_Toc458498620)

[Figure 4 – AGREE Simulation Process 6](#_Toc458498621)

|  |  |  |  |
| --- | --- | --- | --- |
| **Release History** | | | |
| **Date** | **Version** | **Author** | **Description** |
| 2016-03-28 | 0.1 | Philip Alldredge | Initial Release |
|  |  |  |  |
|  |  |  |  |

# Introduction

The AADL/AGREE Simulator is an Eclipse-based tool that is designed to add AADL simulation capabilities to the Open Source AADL Tool Environment (OSATE). It integrates with the AGREE analysis tools to simulate Architecture Analysis and Design Language (AADL) models which contains AGREE contracts.

This document describes the technical implementation of the simulator.

# High Level Design

A high-level design for the AADL Simulator is described in the following sections. A conceptual and structural overview of the AADL simulator’s components as well as external components with which the simulator interfaces is shown in Figure 1 and Figure 2 .



Figure 1 – Conceptual Overview



Figure - Structural Overview

## User Interface

The simulator user interface consists of extensions of the Eclipse platform. The details regarding the implementation of the user interface is described in the following sections. The details of the operation of the user interface is described in the simulator User’s Manual. The simulator defines an Eclipse perspective which contains the display the views and menu items needed to control the simulator.

### UI Simulation Service

An OSGi (Open Service Gateway initiative) service is provided by the user interface to allow user interface components to retrieve the state of the simulator and to easily listen to state changes. The service implements the *edu.uah.rsesc.aadlsimulator.ui.services. SimulationUIService* interface.

### Eclipse Launch Shortcut

The simulator implements an Eclipse launch shortcut. When requested, the launch shortcut finds a launch configuration based on the current selection. If a launch configuration does not exist, the launch shortcut automatically creates a temporary launch configuration. Automatically created launch configurations are not saved. When a simulator launch configuration is launched, the AADL Simulator perspective is automatically be activated. The previously active perspective is reactivated when the simulator is stopped. Switching perspectives is handled by a launch listener in the *edu.uah.rsesc.aadlsimulator.ui.Activator* class.

### Eclipse Commands

Simulator controls and other menu items are implemented as Eclipse commands. While most menu items are available only in the AADL Simulator perspective, toolbar items for simulator controls are available in both the AADL Simulator and the AADL perspectives.

### Views

The tabular display is implemented as an Eclipse view. The view displays the names and values of variables in a tabular form. The tabular display is arranged in a hierarchical manner so that variables are grouped by their containing subcomponent. The view allows the user to specify input constraints which are simulation engine independent.

### Charts

The simulator allows the user to visualize the value of simulation variables by view the values as a chart. One such chart shows simulation variable values in relation to the simulation step. The other allows the user to visualize the value of multiple versions of the next simulation steps. These charts are implementing using the Jzy3d library

### AGREE Counterexample Extractors

The simulator implements two AGREE counterexample extractor extensions. These extensions allow the user to simulate AGREE counterexamples using the simulator. One extractor handles monolithic simulation and the other extractor handles single layer simulation.

A counterexample is simulated by starting the simulation in the appropriate mode and stepping forward the appropriate number of steps while constraining the simulation inputs for each step to the values contained in the counterexample. If a step of the counterexample cannot be simulated because of an unsatisfied property, the user is given the opportunity to disable simulation properties and continue counterexample simulation.

### OSATE Graphical Editor Extensions

The graphical display is provided by the OSATE Graphical Editor. The simulator implements several extensions to better integrate with the OSATE Graphical Editor.

#### Tooltips

The simulator uses the graphical editor’s tooltips (org.osate.ge.tooltips) extension point to register *SimulatorTooltipContributor* as a tooltip contributor. When a simulation is active, the tooltip contributor adds variable names and values to the tooltip of appropriate diagram elements.

#### Commands

The simulator uses the graphical editor’s commands extension point (org.osate.ge.commands) to register the *ShowInSimulationVariablesViewCommand*. This command is available when the simulator is active. When activated, it opens the *Simulation Variables* view and selects and expands the variable corresponding to the selected shape.

## Simulation Backend

The Simulation Backend is responsible for simulating the AADL model.

### Simulation Service

It provides a Simulation Service that serves as an interface for the creating and disposal of simulation engine instances. Simulation engine instances represent an active simulation. The Simulation Service supports multiple simulation engine types which are registered using the Eclipse extension mechanism. Each simulation engine type provides a factory class which is used by the simulation service to create an instance of the simulation engine. Each simulation engine instance implements the *SimulationEngine* interface which enables the user interface to access the simulation state and control the simulation without concern for the implementation of the current simulation engine. The Simulation Service also contains a mechanism for listening for the creation and disposal of simulation engines. This mechanism is used by the user interface service to track simulation engines.

### Eclipse Launch Configuration Type

The backend provides an Eclipse Launch Configuration Type that utilizes the Simulation Service to create simulation engine instances based on a launch configuration.

### AGREE Simulation Engine

The AGREE Simulation Engine is responsible for simulating an AADL System Instance based on the AGREE language. The factory for the AGREE Simulation Engine uses classes provided by the AGREE Eclipse plugin to generate the Lustre specification that is used by the AGREE tools for verification. The factory then transforms the Lustre specification into a Simulation Program used to create the AGREE Simulation Engine. An overview of the process by which the AGREE Simulation Engine Factory creates the AGREE Simulation Program used to create the AGREE Simulation Engine is shown in Figure 3.



Figure - Simplified AGREE Simulation Program Transformation Process

#### Transformation Process

The transformation process transforms the assume guarantee Lustre program produced by the AGREE tools into a Lustre specification for simulating a single step. The transformation process works on either the monolithic or single layer version of the Lustre program. The transformed Lustre specification is such that it contains a single Lustre property for which when a counterexample is produced, the counterexample contains the values for the next step of the simulation. The specification is such that all the guarantees included in subcomponents which are component types are satisfied.

##### Populate Metadata

The produced simulation program contains metadata to associate elements within the AADL, AGREE, and Lustre specifications with one other.

The simulator stores metadata which associates AADL, AGREE, Lustre, and simulation objects with each other. For each input of the main Lustre node, the simulation creates simulation variable objects which are assigned a name and contain references to the AADL component instance and feature instance, the AGREE declaration, and the Lustre variable type and variable identifier.

The simulator also stores a mapping between the AADL component instances and the AGREE node.

##### Create Local Variables for Property Expressions

The simulator creates variable declarations and equations from expressions associated with AGREE variable declarations. Other uses of the expression are replaced with references to the created variables. These variables are marked as simulation property variables and are used to create simulation properties later in the transformation.

##### Require Valid Subcomponent State

The simulator transforms the Lustre to require that the nodes for each subcomponent produces results regardless of whether its assumptions are true. This is required to force subcomponent nodes to capture status of assumptions and ensure guarantees are fulfilled. In normal agree analysis, this is not needed due to the nature of the analysis.

##### Inline Lustre Nodes

The simulator transforms the Lustre specification so that it contains a single Lustre node. This simplifies later transformations.

##### Transform Temporal Logic

The transformation process replaces all temporal operators in the Lustre language with expressions which reference variables to allow simulating individual steps. Additional variables are added to the Lustre program to allow the simulator to transfer internal variables between steps and to track the current simulation step.

##### Create Simulation Properties

The simulator create simulator property objects based on the simulation property variables created earlier in the process and local variables related to AGREE properties. The status of all simulation properties are shown while running the simulation. Inductive Validity Core (IVC) statements are added to the Lustre specification to request that associated variables be included when determining the IVC when a counterexample cannot be generated by the model checker. This allows the simulator to show the properties which are preventing the simulator from taking a step forward.

##### Remove Lustre Properties

The AGREE assume guarantee Lustre program contains property statements which indicate to the model checker which properties to check. The simulator removes all of these property statements.

##### Create Simulation Lustre Property

The simulator creates a new Lustre property which evaluates to false when all simulation constraints are satisfied. Additionally, it creates additional variables and expressions which force the model checker to produce values for each variable in the simulation rather than just the values critical to the counterexample. This ensures that each counterexample contains the complete state of the simulation.

#### Simulation Process

Each instance of the AGREE Simulation Engine owns a thread in which simulation is performed. It uses the transformed Lustre specification in conjunction with the JKind model checker to perform the simulation. For each step of the simulation, the engine will add constraints to the Lustre specification based on the current and previous model states. After adding constraints, the specification will be passed to JKind to determine a new model state that meet the AGREE guarantees specified in the component types. This process is shown in Figure 4.



Figure – AGREE Simulation Process

The AGREE simulation engine transforms the simulation engine independent input constraints into Lustre expressions which are added to the Lustre specification.

#### Early Evaluation

The AGREE simulation engine has support for attempting to determine the next value for a simulation variable before calling the model checker to simulate the next frame. The evaluated value is displayed in the user interface and indicates that the simulation variable is fully constrained. Although it is not limited to these cases, the capability is intended to prevent over constraining the simulation in cases such as variables being passed down the subcomponent hierarchy and variables being constant between simulation steps. The AGREE simulation engine determines the value for a simulation variable by solving related equations using user specified input constraints and the state of the simulation.

To evaluate a variable, the simulator builds a collection of unique variables. A unique variable is a single object representing a set of Lustre identifiers which are known to evaluate to the same value. Creating such a collection prevents re-evaluating variables which are known to be equivalent. The simulator builds this collection by evaluating expressions based on the simulation program and the current simulation constraints. The simulator attempts to solve equations related to the variable being evaluated algebraically. When expressions involving multiple literal values are encountered, the expressions are evaluated. It evaluates the equation recursively until the equation being solved is a linear equation of one variable or the maximum recursion depth is reached. To allow the user interface to show values interactively, the solve depth is currently set to 4. Once the related equation has been simplified to a linear equation of one variable involving the variable being solved, the simulator solves the equation. The simulator attempts to solve all related equations until the equations have been exhausted or a conflict between equations is found. If each non-null value from the related equations are consistent, it is returned as the result of the evaluation.