Plug-in Development for the Open Source AADL Tool Environment
Part 1: An Introduction

Peter Feiler / Aaron Greenhouse
Software Engineering Institute
(phf / aarong@sei.cmu.edu
412-268- (7790 / 6464)
OSATE Plug-in Development Series

• Introduction to OSATE Plug-in Development
  – OSATE capabilities & plug-in architecture
  – AADL Meta model & example plug-in

• OSATE Plug-in Development Process
  – Plug-in development design approach
  – Model traversal & AADL properties
  – Analysis plug-ins & result management

• Interfacing with Existing Models & Tools
  – Declarative & instance models
  – Generation & external representations

• OSATE Infrastructure & API
  – Modal system models
  – Persistency
  – Sublanguage extensions
Outline

Background: SAE AADL Standard
• Extensible OSATE plug-in architecture
• OSATE Plug-ins for Embedded System Engineering
• AADL Meta model
• Model statistic plug-in example
SAE Architecture Analysis & Design Language

• Notation for specification of task and communication architectures of Real-time, Embedded, Fault-tolerant, Secure, Safety-critical, Software-intensive systems
• Fields of application: Avionics, Automotive, Aerospace, Autonomous systems, …
• Based on 15 Years of DARPA funded technologies
• SAE AADL Standard published Nov 2004
• www.aadl.info
Model-Based Engineering

System Analysis
- Schedulability
- Performance
- Reliability
- Fault Tolerance
- Dynamic Configurability

System Integration
- Runtime System Generation
- Application Composition
- System Configuration

Model the Architecture
Abstract, but Precise

Composable Components
- Ambulatory & Signal Processing
- Information Fusion
- Mechatronics
- Target Recognition
- Guidance & Control
- Supply Chain

SAE AADL

Software System Engineer

Application Software

Execution Platform

GPS | DB | HTTPS | Ada Runtime

Devices | Memory | Bus | Processor

www.aadl.info

© 2004 by Carnegie Mellon University
The AADL Standard

- Requirements document SAE ARD 5296
  - Input from aerospace industry
  - Balloted and approved in 2000
- SAE AADL document SAE AS 5506
  - Core language published by SAE Nov 2004
- In review to be balloted late 2004
  - Graphical AADL notation
  - UML profile of AADL for UML1.4 and UML 2.0
  - AADL Meta model, XMI domain model, XML schema
  - Ada and C Annex
- In development
  - Dependability Modeling Annex
  - Partitioning Annex (ARINC653)
AADL: The Language

Components with precise semantics
- Thread, thread group, process, system, processor, device, memory, bus, data, subprogram

Completely defined interfaces & interactions
- Data & event flow, synchronous call/return, shared access
- End-to-End flow specifications

Real-time Task Scheduling
- Supports different scheduling protocols incl. GRMA, EDF
- Defines scheduling properties and execution semantics

Modal, configurable systems
- Modes to model transition between statically known states & configurations

Component evolution & large scale development support
AADL language extensibility
AADL Language Extensions

• New properties through property sets
• Sublanguage extension
  – Annex subclauses expressed in an annex-specific sublanguage
• Project-specific language extensions
• Language extensions as approved SAE AADL standard annexes
• Examples
  – Error Model
  – ARINC 653
  – Behavior
  – Constraint sublanguage
AADL in Context

Research ADLs

- MetaH
  - Real-time, modal, system family
  - Analysis & generation
  - RMA based scheduling
- Rapide, Wright, ..
  - Behavioral validation
- ADL Interchange
  - ACME

Industrial Strength

- UML 2.0, UML-RT
- HOOD/STOOD
- SDL

© 2004 by Carnegie Mellon University

www.aadl.info
AADL/UML Relationship

System Engineering

Embedded Software System Engineering

AADL Core

AADL Annexes

UML 2.0

Security

UML Working Groups

UML-RT

Timeliness

UML 1.4

Dependability

Detailed design

To Be submitted to OMG for Adoption

AADL UML Profile

SysML
Outline

• Background: SAE AADL Standard
• Extensible OSATE plug-in architecture
• OSATE Plug-ins for Embedded System Engineering
• AADL Meta model
• Model statistic plug-in example
Two-Tier Tool Strategy

• Open Source AADL Tool Environment (OSATE)
  – Developed by SEI
  – Low entry cost solution (no cost CPL)
  – Multi-platform based on Eclipse 3.0
  – Utilizes Eclipse Modeling Framework (EMF) & ANTLR
  – Prototyping environment for project-specific analysis
  – Architecture research platform

• Commercial Tool Support
  – UML tool environment extension based on UML profile
  – Extension to existing modeling environment with AADL export/import
  – Analysis tools interfacing via XML or XML to native filter
  – Runtime system generation tools
XML-Based Tool Integration Strategy

AADL Front-end

Textual AADL

Semantic Checking

Graphical AADL

Declarative AADL Model

AADL Instance Model

Graphical Layout Model

Scheduling Analysis

Reliability Analysis

Safety Analysis

AADL Runtime Generator

Commercial Tool

Research prototype

Project-Specific In-House
AADL Models

• **Declarative AADL Model**
  – Reflects semantically processed abstract syntax of AADL
  – Represents declarative nature of AADL
  – Preserves complete content of textual AADL specifications
  – Package-based partitioning of XML documents

• **AADL Instance Model**
  – Represents system instance as application & execution platform
  – Compact representation with locally cached relevant property values
  – Captures modal system instances & system operation modes
  – System operation mode specific of property values
  – Multiple property value sets
Tool Interoperability

AADL Front-end

- Textual AADL
- Semantic Checking
- Parser
- Graphical AADL
- Graphical View

Declarative AADL Model

AADL Instance Model

Persistent XML documents

Co-resident tool

AADL To Text

Graphical Layout Model

Tool-specific XML Representation

Convert

Tool-specific representation

AADL XML Tool Interface

© 2004 by Carnegie Mellon University
OSATE Plug-in Architecture

Eclipse Platform
- Workbench
- JFace
- SWT
- Workspace
- Platform Runtime

Help
- Team
- Marker

Java Development Tools (JDT)
- Plug-in Development Environment (PDE)

Eclipse Environment

AADL Environment
- AADL Parser
- AADL Textual Editor
- AADL Graphical Editor
- AADL Object API
- XML Document Persistence

Standalone Generation Tool
- Analysis Tool Via Java
- Analysis Tool Via XML
OSATE Capabilities

- OSATE Release 0.3.0 based on Eclipse Release 3
- Online AADL help
- Text to XML & XML to text
- Syntax-sensitive text editor
- Parsing & semantic checking of full AADL
- AADL property viewer
- Syntax-Sensitive AADL Object
- Model versioning & team support
- Model instantiation
- Model consistency checking
- AADL to MetaH translator

Over 200 downloads

Processed 21000 line AADL model in 20 sec

Next release Dec 2004
Graphical editor
Multi-file support
Analysis plug-in development
OSATE Plug-in Extensions

OSATE
Textual AADL, Graphical AADL
XML/XMI AADL, AADL object model API
AADL extension support

EMF
XML/XMI, Metamodel
Change notification
Multi-file support

Eclipse
Platform independence
Extensible help
Task & Problem Mgt
Team support
Plug-in development

AADL Front-end
Text editor
Object editor
Graphical editor
Text<->XML Semantics

OSATE Extensions
Analysis template
Generation template
AADL Semantic API

Architecture Import
Simulink/Matlab model
Extraction via SVM

Architecture Export
MetaH

Architecture Analysis
Security level
Data stream miss rate
Latency

Architecture Consistency
Required connectivity
Model completeness profiles
Connectivity cycles

Architecture Transform
Conceptual architecture ->
Runtime architecture
Rate group optimization
Port group identification

External Models
External tools

Model Transformation
Timing analysis (RMA)
OSATE Core Plug-ins

- AADL Model Processing: edu.cmu.sei.aadl.model
- AADL Model Viewing & Edit_cmds: edu.cmu.sei.aadl.model.edit
- AADL Object Editor: edu.cmu.sei.aadl.model.editor
- AADL Text Editor: edu.cmu.sei.aadl.texteditor
- AADL Parser/Semantics: edu.cmu.sei.aadl.model.parser
- AADL Text Generator: edu.cmu.sei.aadl.unparser
- AADL Help: edu.cmu.sei.aadl.help
- AADL Model Instantiation: edu.cmu.sei.aadl.instance
- OSATE project & builder: edu.cmu.sei.osate.core
- OSATE UI Support: edu.cmu.sei.osate.ui
OSATE Community Development

- www.aadl.info website
- OSATE Plug-in update site
- Bugzilla error reporting
- SEI-Hosted CVS Development Server
- Availability of Source Code (CPL)
- Plug-in contributions
  - Syntax-sensitive text editor by York U.
  - Graphical layout editor by USC
  - AADL to MetaH translator by SEI
  - Error modeling support by Embry-Riddle
Outline

- Background: SAE AADL Standard
- Extensible OSATE plug-in architecture
- OSATE Plug-ins for Embedded System Engineering
- AADL Meta model
- Model statistic plug-in example
An Extensible Engineering Environment

Embry-Riddle Reliability Analysis

Concurrency Analysis

AADL Extensions
- Error model
- Concurrency behavior

Architecture Import & Extraction

Architecture Export
- MetaH, TTA

MetaH Toolset (Honeywell)
- Scheduling analysis
- Reliability analysis
- Isolation analysis
- Runtime system generation

TimeWiz Commercial Tool
- Scheduling analysis
- Execution trace analysis

TimeWeaver (CMU)
- Distributed resource allocation
- Multi-platform runtime system generation

Object Model Interface
- Network model

Model Export Filters
- Timing model

OMNET++
- Network simulation

© 2004 by Carnegie Mellon University

www.aadl.info
Architecture Consistency

Systemic issues discovered in architecture analysis
Issue detection codified as OSATE extensions

- Expected component connectivity
  - Required and optional port connections
- Miss rate of data stream
  - Accommodates incomplete sensor readings
  - Allows for controlled deadline misses
- State vs. state delta communication*
  - Data reduction technique
  - Requires guaranteed delivery protocol

Use in models of different fidelity
Architecture Consistency - Impact

• Safety criticality
  – Authority over high criticality components

• Security levels & information flow
  – Port-based flow, shared access, subprogram call
  – Security level of containing component
  – Security level of execution platform

• Impact dependency*
  – Partition isolation in safety-critical systems
  – Fault propagation in reliability models

Use in models of different fidelity
Architecture Consistency – Data Flow

• Domain data typing of ports
  – Connection-based data type checking
  – Measurement units*
  – Value range*

• Data stream constraints
  – Delta value bound*
  – Acceptable miss rate
  – Assumed latency

• Data accuracy
  – Reading accuracy
  – Computational error accumulation

Utilize property range value
Represents controller set-point constraint
Compositional & configuration consistency
Architecture Consistency - Flows

• Flow latency in partitioned systems
  – Use of flow specification
  – Partition execution semantics
  – Low fidelity, high precision models

• Flow specification across data and flow types
  – Logical application flow

• Spec-based flow analysis
  – Early latency analysis for integrated systems and systems of systems
  – Latency budgeting
  – Specification validation
Architecture Consistency - Performance

• Priority inversion
  – Manually assigned thread priorities

• Scheduling analysis
  – Earliest deadline first: Java-based
  – Rate-monotonic Analysis: Java-based
  – External timing model*

• Distributed system resource allocation
  – TimeWeaver binpacking
    • Load balance vs. resource minimization
  – Allocation constraints
    • Processor, memory, bus types & instances
    • Co-location
Generators

• Textual AADL
  – Fully preserved AADL model content
  – Preserved ordering, number precision
  – Formatting by generator

• MetaH
  – Sorted declarations
  – Multi-pass processing
  – Port group unfolding

• AADL instance model
  – XML-based object model
  – EMF-based meta model

• Model transformation*
  – Deep copy cloning & undo history
  – Examples
    • Connection rubber-banding
    • Rate group optimization
    • Single threaded processes
    • Redundancy template instantiation
Outline

• Background: SAE AADL Standard
• Extensible OSATE plug-in architecture
• OSATE Plug-ins for Embedded System Engineering
• AADL Meta model
• Model statistic plug-in example
Defining the XML Representation

- XML Data Type Definition (DTD)
  - Define tags & hierarchical structure
- XML Schema
  - Better handling of references & typing
- XMI
  - Meta model definition
  - XML documents with document change information

Record of content additions/changes by different tools
Leverage of Existing Technology

- **UML**
  Model & Design

- **XMI**
  XSD
  XML Documents

- **Annotated Java**
  XSD, XMI…

- **Java**

- **EMF**
  - Integration Foundation of Eclipse
  - Transformation Tools
  - Database Tools
  - Testing Tools
  - Model Driven App Development
  - MetaData Management
  - Java Tools Etc..

EMF - The Data Integration Foundation of Eclipse
AADL Meta Model

• Defined in Eclipse Modeling Framework (EMF)
  – Collection of meta model packages with graphical views
  – Separate from, but close to UML profile of AADL

• XML as persistent storage
  – XMI specification from Ecore meta model
  – Generated XML schema

• In-core AADL model
  – Generated methods for AADL model manipulation
  – Edit history, deep copy, object editor, graphical editor
  – Methods to support
    • AADL extends hierarchy
    • feature “inheritance”
    • property value “inheritance”
AADL Meta Model Packages

• Core: defines the concepts of component type, implementation, subcomponent, AADL packages and modes.
• Component: defines the concrete classes for the different categories of components, including the constraints on their containment.
• Feature: defines the features of component types.
• Connection: defines the connections between component features.
• Flow: defines flow related elements of the AADL.
• Property: defines the elements for associating property values and for introducing new property types and properties via property sets.
AADL Meta Model Class Hierarchy

Programmatically available for common methods & operations

Inheritance of meta model attributes and associations

- AObject
  - comment: EString

- NamedElement
  - name: EString

- PropertyHolder

- ComponentClassifier

- ComponentType

- ComponentImpl

- ModeMember

- feature::PortGroupType
AADL Meta Model Fragment

Reference associations
Can be cross XML document

Association labels
become XML tags &
Access methods
AADL Inheritance

Port references require Context due to inheritance

AADL inheritance handled by methods
package edu::cmu::sei::XMIExample
public
  system GPS
features
    init: in event port;
    signal: out data port GPS_Signal;
end GPS;
system implementation GPS.basic
end GPS.Basic;
data GPS_Signal
end GPS_Signal;
end edu::cmu::sei::XMIExample;
AADL XML Example

<?xml version="1.0" encoding="UTF-8"?>
<core:AadlSpec xmi:version="2.0" ………>
  <packageSpec name="edu::cmu::sei::XMLExample">
    <aadlPublic>
      <systemType name="GPS">
        <features>
          <eventPort name="init"/>
          <dataPort name="signal" direction="out" dataClassifier="#//packageSpec[@name=edu::cmu::sei::XMLExample]/aadlPublic/dataType[@name=GPS_Signal]"/>
        </features>
      </systemType>
      <systemImpl name="GPS.basic" compType="#//packageSpec[@name=edu::cmu::sei::XMLExample]/aadlPublic/systemType[@name=GPS]"/>
      <dataType name="GPS_Signal"/>
    </aadlPublic>
  </packageSpec>
</core:AadlSpec>
Outline

• Background: SAE AADL Standard
• Extensible OSATE plug-in architecture
• OSATE Plug-ins for Embedded System Engineering
• AADL Meta model
  Model statistic plug-in example
A First Plug-in: Model Statistics

Task: Plug-in that counts the components in a model

Topics Covered

- How to set up a plug-in project
  - Creating a new OSATE Plug-in project
  - Creating a new OSATE Action class
- How to traverse AADL models
  - Declarative vs. instance model processing
  - Meta model’s class hierarchy
  - Using “switch” classes for processing
- How to report analysis results
  - Reporting via markers
  - Reporting via dialog boxes
Constructing the “Model Statistics” Plug-in

- Create a plug-in project

- Create the analysis
  - Walk the AADL model to count the components

- Create an Eclipse action
  - Invokes the analysis
  - Reports analysis results
Creating a Plug-in Project (1)

1. Select "Project..." from the Eclipse Platform menu.
2. In the New Project dialog, select "Plug-in Project".
3. Click "Next" to proceed.
Creating a Plug-in Project (2)

4. Project name: edu.cmu.sei.osate.statistics

5. [Next button highlighted]
Creating a Plug-in Project (3)

6. Enter the data required to generate the plug-in.

- **Plug-in ID**: edu.cmu.sei.osate.statistics
- **Plug-in Version**: 1.0.0
- **Plug-in Name**: Statistics Plug-in
- **Plug-in Provider**:
- **Runtime Library**: statistics.jar

- **Plug-in Class**
  - Generate the Java class that controls the plug-in's life cycle (recommended)
    - **Class Name**: edu.cmu.sei.osate.statistics.StatisticsPlugin
  - This plug-in will make contributions to the UI

- **Intended for use with older Eclipse platforms (prior to 3.0)**

Click **Finish** to complete the process.
Generated Plug-in Project

- Java package with same name as project
- Class that manages plug-in lifecycle. (Don’t edit)
- Properties file that influences build process. (Don’t edit)
- **Plug-in Manifest File**: Describes appearance and structure of plug-in. (Must edit to define plug-in action)
Plug-in Dependencies

- Plug-ins export the Java packages they contain
- Plug-ins **depend** on other plug-ins for their packages
  - Dependencies are declared in `plugin.xml`

- A new plug-in already declares that it depends on
  ```java
  org.eclipse.ui
  org.eclipse.core.runtime
  ```

- Our “Model Statistics” plug-in also depends on
  ```java
  org.eclipse.emf.ecore
  edu.cmu.sei.aadl.model
  ```
  - All OSATE plug-ins will depend on these plug-ins
Setting the Plug-in’s Dependencies

Edit the plugin.xml file
– Use the “Dependencies” tab

1. Go to the Dependencies tab.
2. Add the required plug-ins.
3. Select the appropriate plug-ins.
4. Click OK.
Setting the Plug-in’s Dependencies in XML

Edit the `plugin.xml` file in raw XML

- Use the “plugin.xml” tab
Constructing the “Model Statistics” Plug-in

• Create a plug-in project

• Create the analysis
  – Walk the AADL model to count the components

• Create an Eclipse action
  – Invokes the analysis
  – Reports analysis results
The “Model Statistics” Plug-in

Counts the components in a model

- **Declarative model**
  - Component types
  - Component implementations
  - Data types and port group types
  - Flows
  - End-to-end flows

- **Instance model**
  - Thread, process, processor, memory, bus, device *instances*
  - Semantic connections
Gathering Model Statistics

Extend class AadlProcessingSwitch

- (In package edu.cmu.sei.aadl.model.util)
- Extends ForAllAObject
  - *Concrete* model traversal methods: e.g., processPreOrderAll
  - *Abstract* model processing method: process
- Uses EMF “switch” classes
  - One switch per EMF package
    - E.g., edu.cmu.sei.aadl.model.component.util.ComponentSwitch
  - One “case” method per class
    - E.g., Object caseBusType(BusType obj)
- Processing “falls-through” based on the class hierarchy
  - Return DONE — do not fall through
  - Return NOT_DONE — fall through

Extend & provide task-specific “switch” implementations
public class ModelStatistics extends AadlProcessingSwitch {
    /* Counters to keep track of occurrences of different 
     * objects in the model. 
     */
    private int typeCount = 0;
    private int componentTypeCount = 0;
    private int compImplCount = 0;
    private int threadCount = 0;
    private int processCount = 0;
    private int processorCount = 0;
    private int busCount = 0;
    private int deviceCount = 0;
    private int memoryCount = 0;
    private int componentCount = 0;
    private int connectionCount = 0;
    private int flowCount = 0;
    private int endToEndFlowCount = 0;

    public ModelStatistics() { super(); }

    protected void initSwitches() {
        /* We overwrite the case method for a class in the meta model 
        * specific switches. 
        */
        ...
    }
}

Extend OSATE utility class
Declare fields to hold component counts
Override initSwitches to define behavior
protected final void initSwitches() {
    componentSwitch = new ComponentSwitch() {
        public Object caseComponentType(ComponentType obj) {
            typeCount++;
            // Only count those that are not DataType or PortgroupType
            if (!(obj instanceof DataType || obj instanceof PortGroupType))
                componentTypeCount++;
            return DONE;
        }
        public Object caseComponentImpl(ComponentImpl ci) {
            compImplCount++;
            return DONE;
        }
    };

    flowSwitch = new FlowSwitch() {
        public Object caseFlowSpec(FlowSpec obj) {
            flowCount++;
            return DONE;
        }
        public Object caseEndToEndFlow(EndToEndFlow obj) {
            endToEndFlowCount++;
            return DONE;
        }
    };
    ...
}
ModelStatistics Class: Declarative Model

```java
protected final void initSwitches() {
    componentSwitch = new ComponentSwitch() {
        public Object caseComponentType(ComponentType obj) {
            typeCount++;
            // Only count those that are not DataType or PortgroupType
            if (!(obj instanceof DataType || obj instanceof PortGroupType))
                componentTypeCount++;
            return DONE;
        }
        public Object caseComponentImpl(ComponentImpl ci) {
            compImplCount++;
            return DONE;
        }
    };

    flowSwitch = new FlowSwitch() {
        public Object caseFlowSpec(FlowSpec obj) {
            flowCount++;
            return DONE;
        }
        public Object caseEndToEndFlow(EndToEndFlow obj) {
            endToEndFlowCount++;
            return DONE;
        }
    };
    ...}
```
ModelStatistics Class: Declarative Model

protected final void initSwitches() {
    componentSwitch = new ComponentSwitch() {
        public Object caseComponentType(ComponentType obj) {
            typeCount++;
            // Only count those that are not DataType or PortgroupType
            if (!(obj instanceof DataType || obj instanceof PortGroupType))
                componentTypeCount++;
            return DONE;
        }
        public Object caseComponentImpl(ComponentImpl ci) {
            compImplCount++;
            return DONE;
        }
    };
    flowSwitch = new FlowSwitch() {
        public Object caseFlowSpec(FlowSpec obj) {
            flowCount++;
            return DONE;
        }
        public Object caseEndToEndFlow(EndToEndFlow obj) {
            endToEndFlowCount++;
            return DONE;
        }
    };
    // Override handling of specific model elements
}
ModelStatistics Class: Instance Model

protected final void initSwitches() {
...

    instanceSwitch = new InstanceSwitch() {
        public Object caseComponentInstance(ComponentInstance obj) {
            componentCount++;
            switch (obj.getCategory().getValue()) {
                case ComponentCategory.THREAD:
                    threadCount++; return DONE;
                case ComponentCategory.PROCESS:
                    processCount++; return DONE;
                case ComponentCategory.PROCESSOR:
                    processorCount++; return DONE;
                case ComponentCategory.MEMORY:
                    memoryCount++; return DONE;
                case ComponentCategory.BUS:
                    busCount++; return DONE;
                case ComponentCategory.DEVICE:
                    deviceCount++; return DONE;
            }
            return DONE;
        }

        public Object caseConnectionInstance(ConnectionInstance ci) {
            connectionCount++;
            return DONE;
        }
    };
}
ModelStatistics Class: Instance Model

protected final void initSwitches() {
    
    instanceSwitch = new InstanceSwitch() {
        public Object caseComponentInstance(ComponentInstance obj) {
            componentCount++;
            switch (obj.getCategory().getValue()) {
                case ComponentCategory.THREAD:
                    threadCount++;  return DONE;
                case ComponentCategory.PROCESS:
                    processCount++; return DONE;
                case ComponentCategory.PROCESSOR:
                    processorCount++; return DONE;
                case ComponentCategory.MEMORY:
                    memoryCount++; return DONE;
                case ComponentCategory.BUS:
                    busCount++; return DONE;
                case ComponentCategory.DEVICE:
                    deviceCount++; return DONE;
            }
            return DONE;
        }
    
    public Object caseConnectionInstance(ConnectionInstance ci) {
        connectionCount++;
        return DONE;
    }
    
    };
}
public String getModelResult() {
    return "Model Statistics: " + componentTypeCount
        + " component type declarations, " + compImplCount
        + " component implementation declarations, "
        + (typeCount - componentTypeCount)
        + " data/port group type declarations. ";
}

public String getFlowResult() {
    return "Flow Statistics: " + flowCount + " flow specifications, "
        + endToEndFlowCount + " end-to-end flows. ";
}

public String getApplicationResult() {
    return "Application statistics: " + threadCount + " threads, "
        + processCount + " processes, " + connectionCount
        + " semantic connections. ";
}

public String getExecutionPlatformResult() {
    return "Execution platform statistics: " + processorCount
        + " processors, " + memoryCount + " memory units, "
        + busCount + " buses, " + deviceCount + " devices. ";
}
} // End of class ModelStatistics
Constructing the “Model Statistics” Plug-in

• Create a plug-in project

• Create the analysis
  – Walk the AADL model to count the components

• Create an Eclipse action
  – Invokes the analysis
  – Reports analysis results
“Model Statistics” Eclipse Action

• We need an Action to invoke our analysis
  – Appears as
    • Button on Eclipse toolbar
    • Item in Eclipse menu
  – Reports results to user
    • Via Eclipse markers on model
    • Via dialog box

• Action implemented in two parts
  – Java class implements action behavior
  – plugin.xml installs action into UI
    • Buttons, menu items, etc.
Implementing Action Class

Extend class `AaxlReadOnlyAction`

- (In package `edu.cmu.sei.aadl.model.pluginsupport`)
- Implements `IWorkbenchWindowActionDelegate`
  - Required interface for Eclipse actions
- Ensures needed Eclipse/EMF resources are initialized
- Action defined by method `doAaxlAction(AObject)`
  - Passed the currently selected model object
- **Read-only**: changes made to models by action are **not** saved
  - Use `AaxlModifyAction` instead to automatically save changes

Extend and implement `doAaxlAction`
public class DoModelStatistics extends AaxlReadOnlyAction {

    public void doAaxlAction(AObject obj) {
        // Get the root object of the model
        AObject root = obj.getAObjectRoot();

        // Get the associated AadlSpec
        AadlSpec as = obj.getAadlSpec();

        // Get the system instance (if any)
        SystemInstance si = obj.getSystemInstance();

        // Get the data
        ModelStatistics stats = new ModelStatistics();
        stats.processPreOrderAll(as);
        if (si != null) stats.processPreOrderAll(si);

        ...
    }
}
public class DoModelStatistics extends AaxlReadOnlyAction {

    public void doAaxlAction(AObject obj) {
        // Get the root object of the model
        AObject root = obj.getAObjectRoot();

        // Get the associated AadlSpec
        AadlSpec as = obj.getAadlSpec();

        // Get the system instance (if any)
        SystemInstance si = obj.getSystemInstance();

        // Get the data
        ModelStatistics stats = new ModelStatistics();
        stats.processPreOrderAll(as);
        if (si != null) stats.processPreOrderAll(si);

        ...
    }
}
public class DoModelStatistics extends AaxlReadOnlyAction {

    public void doAaxlAction(AObject obj) {
        // Get the root object of the model
        AObject root = obj.getAObjectRoot();

        // Get the associated AadlSpec
        AadlSpec as = obj.getAadlSpec();

        // Get the system instance (if any)
        SystemInstance si = obj.getSystemInstance();

        // Get the data
        ModelStatistics stats = new ModelStatistics();
        stats.processPreOrderAll(as);
        if (si != null) stats.processPreOrderAll(si);
    }
}
public class DoModelStatistics extends AaxlReadOnlyAction {

    public void doAaxlAction(AObject obj) {
        // Get the root object of the model
        AObject root = obj.getAObjectRoot();

        // Get the associated AadlSpec
        AadlSpec as = obj.getAadlSpec();

        // Get the system instance (if any)
        SystemInstance si = obj.getSystemInstance();

        // Get the data
        ModelStatistics stats = new ModelStatistics();
        stats.processPreOrderAll(as);
        if (si != null) stats.processPreOrderAll(si);
        ...
    }
}

Get the instance model associated with the selected object. If the object is from a declarative model, null is returned.
Class **DoModelStatistics**: Running Analysis

```java
public class DoModelStatistics extends AaxlReadOnlyAction {

    public void doAaxlAction(AObject obj) {
        // Get the root object of the model
        AObject root = obj.getObjectRoot();

        // Get the associated AadlSpec
        AadlSpec as = obj.getAadlSpec();

        // Get the system instance (if any)
        SystemInstance si = obj.getSystemInstance();

        // Get the data
        ModelStatistics stats = new ModelStatistics();
        stats.processPreOrderAll(as);
        if (si != null) stats.processPreOrderAll(si);
        ...
    }
}
```

1. Create a new analysis
2. **Invoke it on the declarative model**
3. **Invoke it on the instance model (if it exists)**
public class DoModelStatistics extends AaxlReadOnlyAction {
    public void doAaxlAction(AObject obj) {
        final StringBuffer msg = new StringBuffer();
        final String modelStats = stats.getModelResult();
        final String flowStats = stats.getFlowResult();
        reportInfo(root, modelStats);
        reportInfo(root, flowStats);
        msg.append(modelStats);
        msg.append(flowStats);

        if (si != null) { // Do we have instance statistics?
            final String appStats = stats.getApplicationResult();
            final String epStats = stats.getExecutionPlatformResult();
            reportInfo(root, appStats);
            reportInfo(root, epStats);
            msg.append(appStats);
            msg.append(epStats);
        }

        // Also report the results using a message dialog
        MessageDialog.openInformation(getShell(), "Model Statistics", msg.toString());
    }
}
Class DoModelStatistics: Reporting Results

public class DoModelStatistics extends AaxlReadOnlyAction {
    public void doAaxlAction(AObject obj) {
        ...
        final StringBuffer msg = new StringBuffer();
        final String modelStats = stats.getModelResult();
        final String flowStats = stats.getFlowResult();
        reportInfo(root, modelStats);
        reportInfo(root, flowStats);
        msg.append(modelStats);
        msg.append(flowStats);

        if (si != null) { // Do we have instance statistics?
            final String appStats = stats.getApplicationResult();
            final String epStats = stats.getExecutionPlatformResult();
            reportInfo(root, appStats);
            reportInfo(root, epStats);
            msg.append(appStats);
            msg.append(epStats);
        }

        // Also report the results using a message dialog
        MessageDialog.openInformation(
            getShell(), "Model Statistics", msg.toString());
    }
}

2. Record results using Eclipse markers
Class DoModelStatistics: Reporting Results

```java
public class DoModelStatistics extends AaxlReadOnlyAction {
  public void doAaxlAction(AObject obj) {
    ...
    final StringBuffer msg = new StringBuffer();
    final String modelStats = stats.getModelResult();
    final String flowStats = stats.getFlowResult();
    reportInfo(root, modelStats);
    reportInfo(root, flowStats);
    msg.append(modelStats);
    msg.append(flowStats);

    if (si != null) { // Do we have instance statistics?
      final String appStats = stats.getApplicationResult();
      final String epStats = stats.getExecutionPlatformResult();
      reportInfo(root, appStats);
      reportInfo(root, epStats);
      msg.append(appStats);
      msg.append(epStats);
    }

    // Also report the results using a message dialog
    MessageDialog.openInformation(getShell(), "Model Statistics", msg.toString());
  }
}
```

3. Report results using dialog box
Recording Results as Eclipse Markers

• Record markers using the methods
  – reportInfo(AObject obj, String message)
  – reportWarning(AObject obj, String message)
  – reportError(AObject obj, String message)
  Associates message with the given model object

• Some properties of markers
  – Separate recording from reporting
  – Persist beyond the plug-in execution
  – Auto-reset on plug-in execution
  – Decoration icon on the resource in “Navigator” view
  – Marker location indicators on the sidebar of an open editor
Reporting Results in a Dialog Box

• Use factory methods in `MessageDialog`
  – (In package `org.eclipse.jface.dialogs`)
  – `openInformation(Shell s, String title, String msg)`
  – `openWarning(Shell s, String title, String message)`
  – `openError(Shell s, String title, String message)`
  Opens synchronous dialog box.

• Some properties of dialog boxes
  – Immediate reporting
  – Synchronous or asynchronous dialog
    • Synchronous — requires user confirmation (ok button)
    • Asynchronous — lets user change focus without closing window.
  – Messages only exist for the duration of the popup.
Describing the Action in `plugin.xml`

- Actions reside in **action sets**
  - An extension point to Eclipse
  - Groups together related actions

- **Action set description includes**
  - An internal `id` — used to identify the set in XML
  - Human readable `label`

- **Action description includes**
  - An internal `id`
  - Human readable `label`
  - Reference to `class` that implements the action
  - An `icon` — programmer provides images
  - Locations in toolbar and menu bar
Adding Icons to the Project

We add “gif” files to use as icons. We create an *icons* directory to hold them.
The “Model Statistics” Action Set

Eclipse extension point

Our new action set

<plugin ...>

...<extension point="org.eclipse.ui.actionSets"><actionSet id="edu.cmu.sei.osate.statistics.actionSet" label="Statistics Action Set" visible="true"><menu id="analysisMenu" label="Analysis & Menu"/>
</action>
  </actionSet>
</extension>
</plugin>
The “Model Statistics” Action Set

We create an “Analysis” menu

Our action description

```
<plugin ...>
...
<extension point="org.eclipse.ui.actionSets">
  <actionSet id="edu.cmu.sei.osate.statistics.actionSet"
    label="Statistics Action Set"
    visible="true">
    <menu id="analysisMenu"
      label="Analysis &Menu"/>
    <action id="edu.cmu.sei.osate.statistics.DoModelStatistics"
      label="&Model statistics"
      class="edu.cmu.sei.osate.statistics.DoModelStatistics"
      icon="icons/stats.gif"
      disabledIcon="icons/noStats.gif"
      tooltip="Determine model statistics"
      enablesFor="1"
      toolbarPath="edu.cmu.sei.osate.statistics.actionSet"
      menubarPath="analysisMenu/statisticsGroup">
    </action>
  </actionSet>
</extension>
</plugin>
```
The “Model Statistics” Action Set

<plugin ...>
...
<extension point="org.eclipse.ui.actionSets">
  <actionSet id="edu.cmu.sei.osate.statistics.actionSet" label="Statistics Action Set" visible="true">
    <menu id="analysisMenu" label="Analysis & Menu"/>
    <action id="edu.cmu.sei.osate.statistics.DoModelStatistics" label="& Model statistics" class="edu.cmu.sei.osate.statistics.DoModelStatistics" icon="icons/stats.gif" disabledIcon="icons/noStats.gif" tooltip="Determine model statistics" enablesFor="1" toolbarPath="edu.cmu.sei.osate.statistics.actionSet" menubarPath="analysisMenu/statisticsGroup">
    </action>
  </actionSet>
</extension>
</plugin>

Reference to our action implementation

Action icons and tooltip

Location in toolbar and menu bar. References menu id analysisMenu.
“Model Statistics” in Eclipse

Icon in toolbar

Menu item
“Model Statistics” Results

Recorded markers:

- Application statistics: 4 threads, 2 processes, 0 semantic connections.
- Execution platform statistics: 2 processors, 0 memory units, 0 buses, 0 devices.
- Flow Statistics: 0 flow specifications, 0 end-to-end flows.
- Model Statistics: 7 component type declarations, 12 component implementation declarations, 0 data/port group type declarations.
Outline

• Background: SAE AADL Standard
• Extensible OSATE plug-in architecture
• OSATE Plug-ins for Embedded System Engineering
• AADL Meta model
• Model statistic plug-in example

Summary
Final Observations

• Industry-standard architecture modeling notation for embedded systems
  – Abstract but precise modeling
  – Early and repeated predictable analysis

• Industry-standard XML interchange format
  – Model interchange between contractors
  – Interoperability of architecture design, analysis & generation tools

• Low entry cost open source tool set
  – Prototyping environment for project-specific needs
  – Architecture research platform
  – Codification of engineering experience