

Plug-in Development for the Open Source AADL Tool Environment

Part 2: Plug-in Development Process

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



OSATE plug-in design approach

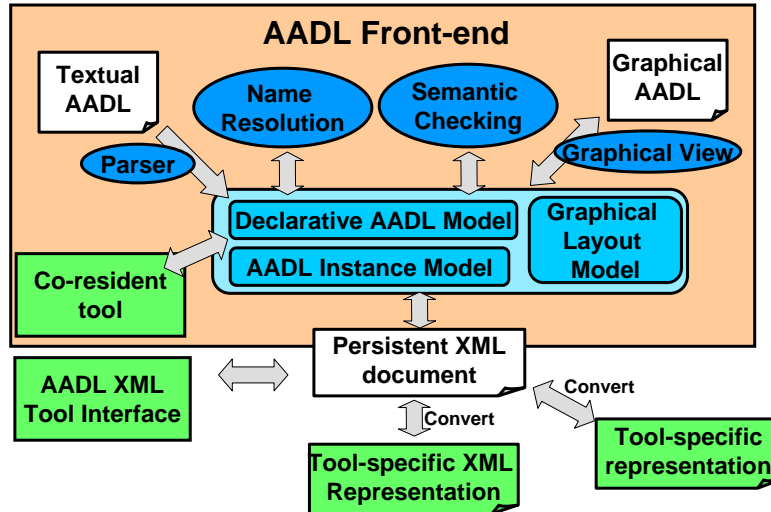
- Model access methods
- Model processing support
- Property processing
- Result recording & reporting
- Flow processing in declarative models



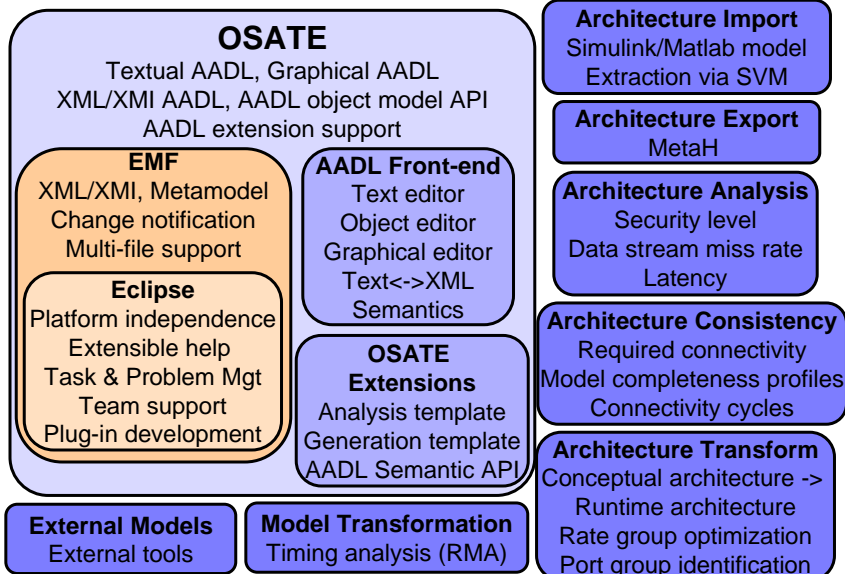
OSATE Plug-in Design Approach

- Type of AADL model processing
- Processing of declarative or instance model
- Relevant parts of AADL (meta) model
- Recording & reporting of processing results
- Execution of model processing plug-in

Tool Interoperability



OSATE Plug-in Extensions





Type of OSATE Plug-ins

- Model analysis
- Model generation
- Model transformation
- Model extraction
- Source code compliance
- Core language extension

**Focus of this session on
Model analysis**

SAE



Types of Model Processing

- Model object specific processing
- Follow containment hierarchy
- Conditional based on model object attributes & AADL properties
- Aggregation of AADL property values
- Component interaction dependency
 - Connections & flows
 - Shared access
 - Subprogram call
- Component impact dependency
 - Above + shared execution platform use
- Mode state reachability

SAE





Declarative or Instance Model

- Declarative model
 - Analyze complete library of component declarations
 - Identify compositional consistency & constraints
 - Process application domain characteristics
 - Propagation along containment hierarchy
 - Connection & flow analysis
 - Mode specific processing per component implementation
- Instance model
 - Runtime architecture of embedded system
 - Process runtime properties
 - Work with
 - instance hierarchy
 - semantic connections
 - Instance specific property values
 - modal system instances (System Operation Mode)



Types of Model Transformation

- In-place transformation
 - Connection aggregation via port groups
 - Component category change
 - Component relocation & rubber-banding of connection
- Translation into new model
 - Template instantiation
 - Redundancy patterns
 - Single threaded processes
 - Model optimizations
 - Rate group transformation
 - Common subgraph elimination

**Deep copy of AADL models
Supported by EMF EcoreUtil**





Execution of OSATE Plug-ins

- Explicit user invocation
 - On specific model objects
 - On all relevant model objects
 - Aaxl<ReadOnly/Modify>Action method
- Automated processing
 - Builder: Resource change propagation & synchronized processing
 - Listener: in-core model object observer notification
 - Impact dependency: forward change propagation
- Multi-user support
 - Team synchronization capability in Eclipse
 - Interfacing to version control system

SAE



Outline

- OSATE plug-in design approach
- ➔ Model access methods
- Model processing support
- Property processing
- Result recording & reporting
- Flow processing in declarative models

SAE



Access to Model Objects

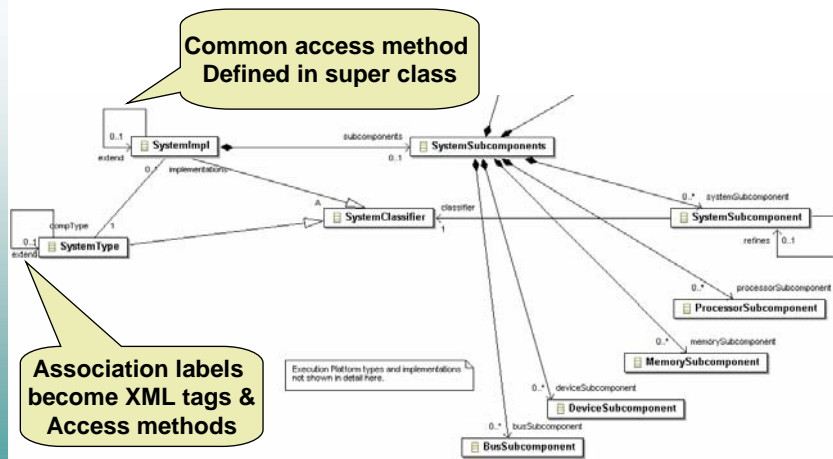
- Generated access methods
 - For all containment associations
 - For all reference associations
 - Multiplicity associations as ELists
- Generic access methods in abstract classes
- AADL inheritance and access methods
 - Feature inheritance by components
 - *Extend/refine* inheritance by component types and implementations
 - Property value inheritance

Association labels for get/set method names

Get method returns EList
Add method



AADL Model Access Methods





Generated Access Methods

- Generated access method

- get<associationlabel>

```
class SystemType {
    public SystemType getExtend() {
        if (extend != null && extend.eIsProxy()) {
            SystemType oldExtend = extend;
            extend = (SystemType)eResolveProxy(
                (InternalEObject)extend);
        }
        if (extend != oldExtend) {
            if (eNotificationRequired())
                eNotify(new ENotificationImpl(this,
                    Notification.RESOLVE,
                    ComponentPackage.SYSTEM_TYPE_EXTEND, oldExtend,
                    extend));
        }
        return extend;
    }
}
```

Proxy resolution for cross XML document references

Tailorable Change propagation via Notification & Listeners

SAE



Generic Access Methods

- Generic access methods in abstract classes

- getX<associationlabel>

- Java 1.4 does not permit superclass return value

```
class ComponentClassifier {
    public ComponentClassifier getXExtend() {
        EClass compclass = this.eClass();
        if (compclass == null) return null;
        EStructuralFeature providesFeature =
            compclass.getEStructuralFeature("extend");
        Object o = this.eGet(providesFeature);
        return (ComponentClassifier) o;
    }
}
```

Runtime interpretation of meta model

SAE





AADL Inheritance & Access Methods

- Inheritance transparency through access methods

- getAll<associationlabel>
- getXAll<associationlabel>

```
class ComponentType {
public EList getXAllFeature() {
    ComponentType more = (ComponentType)
    this.getXExtend();
    if (more == null) {
        BasicEList returnlist = new BasicEList();
        returnlist.addAll(this.getXFeature());
        return returnlist;
    } else {
        -- recurse on extends
        EList result = more.getXAllFeature();
        < merge the lists >
    }
}
```

Return modifiable list
of features

SAE



Features & Feature Refinement

- Include the refinement as feature representative

```
EList result = more.getXAllFeature();
EList local = this.getXFeature();
if (local != null) {
    for (Iterator i = local.iterator();
        i.hasNext();)
    {
        Feature fe = (Feature) i.next();
        Feature rfe = fe.getRefinesFeature();
        if (rfe != null)
            result.remove(rfe);
        result.add(fe);
    }
}
return result;
```

Replace feature
with refinement;
Otherwise add feature

SAE





Outline

- OSATE plug-in design approach
- Model access methods
- ➔ Model processing support
 - Property processing
 - Result recording & reporting
 - Flow processing in declarative models

SAE



Model Processing Support

API in package `edu.cmu.sei.aadl.model.util`

- Basic model processing support
 - Default processing of visited model objects
 - Filtered processing
 - User-defined actions
 - List-based processing
- Containment hierarchy traversal methods
- Content-driven processing
- Dependency processing

Covered in next session

SAE





Basic Model Processing

- Class ForAllAObject
 - Provides containment traversal methods
 - Supports result reporting via Eclipse markers
- Redefinable default processing methods
 - suchThat: model object specific condition (default:true)
 - action: conditionally executed method (default: append visited model object)
 - process: called on every visited model object (default: calls action if suchThat is true)
 - Default implementation returns all conditionally visited model objects



Filtered Model Processing

- User-defined filter condition
- Any condition on model objects
 - Port direction or number of ports
 - Component name
 - AADL property of component
- Returns filtered list of model objects

Condition redefinition via anonymous subclass

```
ForAllAObject filteredProcessing = new ForAllAObject(){  
    protected boolean suchThat(AObject obj){  
        return obj instanceof Port &&  
            ((Port)obj).getDirection() == PortDirection.IN_LITERAL ;  
    }  
};
```





User-defined Action Processing

- User-defined action for visited model object
- Can be combined with filtering
- Replaces or complements creation of visited model object list

```
ForAllAObject filteredProcessing = new ForAllAObject(){  
    protected void action(AObject obj){  
        <user-defined action> ;  
        super.action(obj);  
    }  
};
```

Optional inclusion of visited model object list creation



List Processing Methods

- Processing of ELists
 - Utilizes condition, action processing methods
 - EList processEList(EList list)
 - Returns (filtered) list of model objects

- Sorted ELists
 - Quicksort on user-defined comparison criteria
 - Default: comparison of toString results

```
QuickSort quick = new QuickSort(){  
    protected int compare(Object obj1, Object obj2){  
        String a = obj1.toString();  
        String b = obj2.toString();  
        if ( a > b ) return 1;  
        if ( a == b ) return 0;  
        return -1;  
    }  
};
```



Containment Hierarchy Traversal

- Traversing the declarative model
 - Declaration hierarchy
- Traversing the Instance model
 - Corresponds to system component hierarchy
- Hierarchy traversal methods
 - Prefix and postfix order for declarative & instance models
 - Declaration/use order for declarative models
 - Category-specific component instance processing

Does not represent system hierarchy



Traversal Methods

- Full model traversal
 - processPreOrderAll(AObject root)
 - processPostOrderAll(AObject root)
- Declarative model processing
 - processAllComponentImpl(AObject root)
 - processTopDownComponentImpl(AObject root)
 - processBottomUpComponentImpl(AObject root)
- Instance model processing
 - processPreOrderComponentInstance(AObject root)
 - processPostOrderComponentInstance(AObject root)
 - Process<Pre/Post>OrderComponentInstance(AObject root, ComponentCategory cat)





Traversal Driven Switch Processing

- AadlProcessingSwitch
 - Meta model class specific processing
 - Redefinition of EMF generated "case" methods
 - Subclass of ForAllIAObject
 - Redefines process method to invoke switch

```
public Object  
  caseComponentInstance(ComponentInstance ci) {  
    Subcomponent sub = ci.getSubcomponent();  
    ComponentImpl cii = sub.getComponentImpl();  
    self.processPreOrderAll(cii);  
    return DONE;  
  }
```

Self-reference of switch

SAE



Outline

- OSATE Plug-in Design Approach
- Model access methods
- Model processing support
- ➔ Property processing
- Result recording & reporting
- Plug-in examples

SAE





AADL Properties

- A property
 - Provides information about an AADL element
 - E.g., period, latency, size
 - Has
 - A name
 - A type
- Properties are declared in ***property sets***
- Components have ***property associations***
 - Associates a value with a property for that component



Chapter 10 in AADL Spec



Properties Outline



- Property Sets
 - Property Type declarations
 - Property Name declarations
- Property Associations
- Properties in the Meta Model
- Property Lookup
- Using properties in a Plug-in





Property Sets

- A property set contains declarations of
 - Property types
 - Property names
 - Property constants

```
property set Example is
-- Unit Type declaration
English_Units: type units
  (inch, foot => inch * 12, yard => foot * 3);

-- Integer type declaration
Length: type aadlinteger units Example::English_Units;

-- Constant declaration
One_Foot: constant Example::Length => 1 foot;

-- Property declaration
Documentation_Thickness:
  Example::Length applies to (processor, bus, system)
  => value(Example::One_Foot);
end Example;
```



Property Types (1)

- Booleans: `aadlboolean`
 - Values: `true` and `false`
- Strings: `aadlstring`
 - Values: "A String Value", etc.
- Enumerations:
`enumeration (read_only, write_only, read_write);`
 - Values: literals — `read_only`, etc.
- Units: `units (base, a => base * 3, b => a * 4);`
 - Values: none
- Integers:
`aadlinteger 0..10 units Example::English_Units;`
 - Values: `0 inch, 2 feet, 15 yard`, etc.





Property Types (2)

- Reals: `aadlreal -5.0..5.0;`
 - Values: `-5.0, -4.77734, 2.1`, etc.
- Range: `range of aadlinteger -10..10;`
 - Values: `-10..3, 3..4, 0..10`, etc.
- Classifier: `classifier (processor, memory);`
 - Values: Processor and memory component classifiers;
e.g., `Intel_x86`
- Reference: `reference (data, system);`
 - Values: References to data and system components;
e.g., `reference sub_system1.shared_data`



Property Names

Two property name declarations:

```
property set Example2 is
-- A single-valued property
Stack_Size:
  inherit Size
  applies to (system, process, thread)
  => 1 KB;

-- A multi-valued property
Source_Files:
  list of aadlstring
  applies to (all);
end Example2;
```



Property Names: Types

Two property name declarations:

```
property set Example2 is
-- A single-valued property
Stack_Size:
  inherit Size
  applies to (system, process, thread)
  => 1 KB;

-- A multi-valued property
Source_Files:
  list of aadlstring
  applies to (all);
end Example2;
```

Reference to standard type
AADL_Properties::Size.

Reference to primitive
type aadlstring.

Optional "list of" modifier.
Values are lists of strings,
e.g., ("a.c", "b.c").



Property Names: Applies to

Two property name declarations:

```
property set Example2 is
-- A single-valued property
Stack_Size:
  inherit Size
  applies to (system, process, thread)
  => 1 KB;

-- A multi-valued property
Source_Files:
  list of aadlstring
  applies to (all);
end Example2;
```

Optional "inherit" modifier. If not
specified, property value is inherited from
containing component.

The component categories that
the property can be applied to.

Applies to all categories.





Property Names: Default Values

Two property name declarations:

```
property set Example2 is
-- A single-valued property
Stack_Size:
  inherit Size
  applies to (system, process, thread)
  => 1 KB;

-- A multi-valued property
Source_Files:
  list of aadlstring
  applies to (all);
end Example2;
```

Optional default value.
Property value is 1 KB
when not explicitly set.

No default value. Property value is
"not present" unless explicitly set.

SAE



Properties Outline

- Property Sets
 - Property Type declarations
 - Property Name declarations
- ➔ Property Associations
- Properties in the Meta Model
- Property Lookup
- Using properties in a Plug-in

SAE





Property Associations

- Associate a value with a property for a given component
- Are declared in
 - Component properties clauses
 - Subcomponent declarations
- May be “contained”
 - Association “reaches down” the subcomponent hierarchy
- May be modal
 - Association applies only in specified modes
 - Defer to future discussion on modes

SAE



Property Associations Example (1)

```
thread implementation MyThread.Impl
  properties
    Example2::Source_Files => ("MyThread.c", "Helper.c");
  end MyThread.Impl;

process implementation MyProcess.Impl
  subcomponents
    t1: thread MyThread.Impl;
    t2: thread MyThread.Impl;
  properties
    Example2::Stack_Size => 2 KB;
  end MyProcess.Impl;

system implementation Main.Impl
  subcomponents
    p: process MyProcess.Impl
      { Example2::Stack_Size => 4 KB applies to t1; };
  properties
    Example2::Source_files +=> ("Main.c");
  end Main.Impl;
```

SAE



Properties clauses



Property Associations Example (2)

```

thread implementation MyThread.Impl
  properties
    Example2::Source_Files => ("MyThread.c", "Helper.c");
end MyThread.Impl;

process implementation MyProcess.Impl
  subcomponents
    t1: thread MyThread.Impl;
    t2: thread MyThread.Impl;
  properties
    Example2::Stack_Size =>
end MyProcess.Impl;

system implementation Main.Impl
  subcomponents
    p: process MyProcess.Impl
      { Example2::Stack_Size => 4 KB applies to t1; };
  properties
    Example2::Source_files +=> ("Main.c");
end Main.Impl;

```

Property association is for the subcomponent of "p" named "t1"

Contained property association

41



Property Associations Example (3)

```

thread implementation MyThread.Impl
  properties
    Example2::Source_Files => ("MyThread.c", "Helper.c");
end MyThread.Impl;

process implementation MyProcess.Impl
  subcomponents
    t1: thread MyThread.Impl;
    t2: thread MyThread.Impl;
  properties
    Example2::Stack_Size => 2 KB;
end MyProcess.Impl;

system implementation Main.Impl
  subcomponents
    p: process MyProcess.Impl
      { Example2::Stack_Size => 4 KB applies to t1; };
  properties
    Example2::Source_files +=> ("Main.c");
end Main.Impl;

```

Associate a list of strings

Associate an integer value.

Append to the existing list of strings.
Interacts with the property lookup algorithm



© 2004

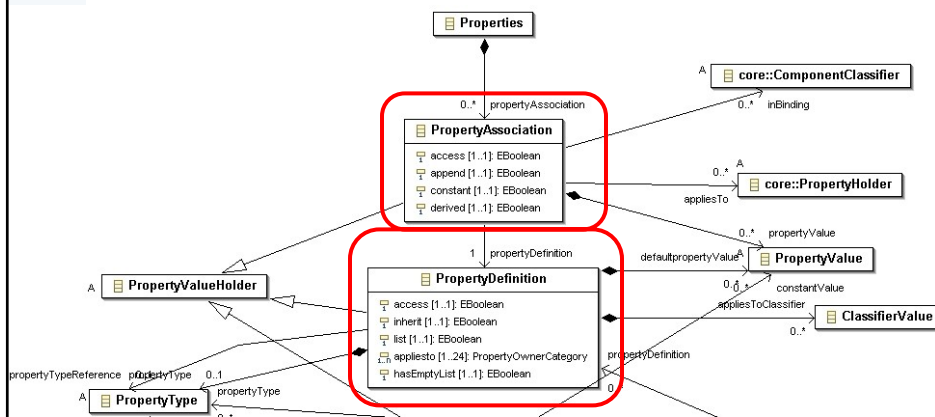
42

Properties Outline

- Property Sets
 - Property Type declarations
 - Property Name declarations
- Property Associations
- ➔ Properties in the Meta Model
- Property Lookup
- Using properties in a Plug-in

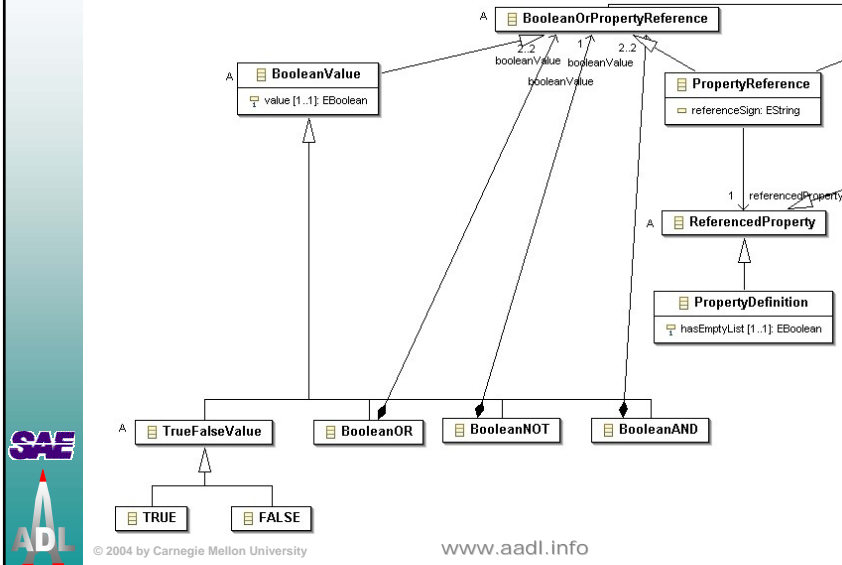


PropertyAssociation & PropertyDeclaration

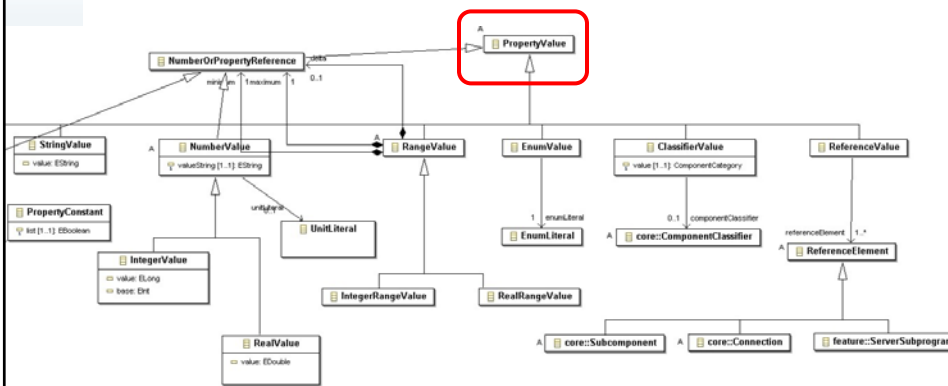




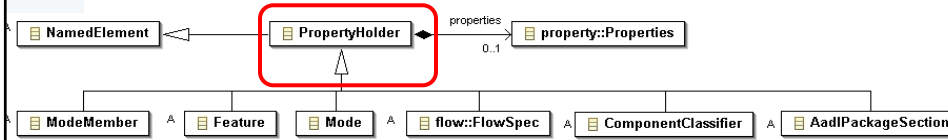
Property Values: Boolean Expressions



Property Values: The Rest



PropertyHolder



- Root class of elements that can have property associations
 - Also parent of `InstanceObject`
 - So includes `ComponentInstance`, `ConnectionInstance`, etc.
- Property associations from the declarative model can be cached in the instance model

Properties Outline

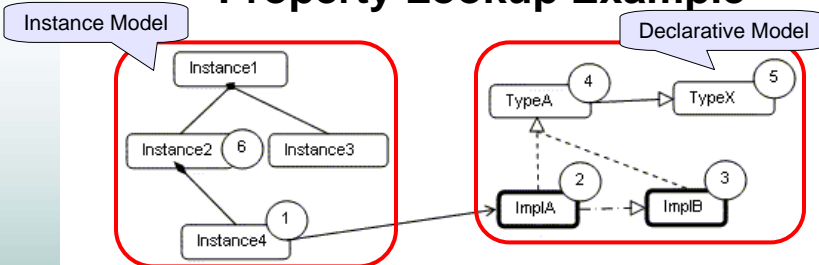
- Property Sets
 - Property Type declarations
 - Property Name declarations
- Property Associations
- Properties in the Meta Model
- ➔ Property Lookup
- Using properties in a Plug-in

Property Lookup

- Property lookup follows both the component and the containment hierarchies.
- In general,
 - Search the element itself for the property association.
 - Recursively search the category of the element.
 - Recursively search the container of the element.
 - If the property is inherit.
 - Use the default property value, if any.
- List values are built using the append operator (+=>) following the lookup rules.



Property Lookup Example



Looking up a property value in component instance **instance4**

- Search component instance **instance4** (1)
- Search **instance4**'s component implementation **ImplA** (2)
 - Search **ImplA**'s ancestor component implementation **ImplB** (3)
 - Search **ImplA**'s ancestor component type **TypeA** (4)
 - Search **TypeA**'s ancestor component type **TypeX** (5)
- Search **instance4**'s containing component instance **instance2** (6)
 - Etc.



“AADL Property Values” View (1)

The screenshot displays the AADL Property Values view for a file named TestFeature.aadl. The main editor shows the following code:

```

20 features
21 -- prop1 => 0, prop2 => "from Type1"
22 foo: in data port in data port ;
23 -- prop1 => 5, prop2 => "from Type1"
24 bar: out data port
25 { MyProperties::prop1 => 5; };
26 properties
27 MyProperties::prop2 => "from Type1";
28 end Type1;
29
30 -- Setting properties by refinement, and inheriting
31 -- closest type of the container
32 system Type2 extends Type1
33 features
34 -- prop1 => 10, prop2 => "local"
35 foo: refined to in data port
36 { MyProperties::prop1 => 10;
37   MyProperties::prop2 => "local"; };
38 -- prop1 => 5, prop2 => "from Type2"
39 bar: refined to out data port ;
40 properties
41 MyProperties::prop2 => "from Type2";
42 end Type2;

```

The tree view on the right shows the project structure, with a callout box pointing to the 'bar : out Data Port' entry under 'System Features'.

Show properties of feature bar in Type2

The Properties view at the bottom shows the following values:

- MyProperties
 - prop1 => 5
 - prop2 => "from Type2"
- AADL_Properties
 - Required_Connection => true

“AADL Property Values” View (1)

This screenshot is similar to the first one but includes annotations to explain the property lookup process. Red boxes highlight the 'bar' feature definition in Type1 (lines 24-25), the 'bar' feature definition in Type2 (lines 39-40), and the 'MyProperties::prop2' property definition in Type2 (line 41). A callout box explains:

Feature bar refined in Type2, so property lookup starts in Type2.

The tree view on the right also has a callout box pointing to the 'bar : out Data Port' entry.

The Properties view at the bottom shows the same values as in the first screenshot:

- MyProperties
 - prop1 => 5
 - prop2 => "from Type2"
- AADL_Properties
 - Required_Connection => true

Modal AADL Property Values

ModalComponents.aadl

```

16 system implementation S.I1
17 subcomponents
18 -- x => 1 in m1, m3; y => 7 in m
19 sc: system Foo
20 { ps::x => 1;
21 ps::y => 7; }
22 in modes (m1, m3);
23 modes
24 m1: initial mode;
25 m2: mode;
26 m3: mode;
27 end S.I1;
28
29 system implementation S.I2 extends S
30 subcomponents
31 -- x => 1 in m3; x => 2 in m2; x
32 sc: refined to system Foo
33 { ps::x => 2 in modes (m2); }
34 in modes (m2, m3);
35 end S.I2;

```

platform:/resource/AADLModel...yLookup/ModalComponents.aadl

- Aadl Spec ModalComponents
 - Property Set PS
 - Aadl Package TEST
 - Public
 - System Type S
 - System Type Foo
 - System Impl S.I1
 - System Impl S.I2
 - System Subcomponents
 - System Subcomponent sc: TEST::Foo

Tasks Problems Error Log AADL Property Values X Properties

PS

- 1 x =>
- 2 in modes {(m2)}
- 1 in modes {(m3)}
- 1 y =>
- undefined in modes {(m2)}
- 7 in modes {(m3)}

ADL

53

Show properties of subcomponent *sc* in *S.I2*

Modal AADL Property Values - 2

ModalComponents.aadl

```

16 system implementation S.I1
17 subcomponents
18 -- x => 1 in m1, m3; y => 7 in m
19 sc: system Foo
20 { ps::x => 1;
21 ps::y => 7; }
22 in modes (m1, m3);
23 modes
24 m1: initial mode;
25 m2: mode;
26 m3: mode;
27 end S.I1;
28
29 system implementation S.I2 extends S
30 subcomponents
31 -- x => 1 in m3; x => 2 in m2; x
32 sc: refined to system Foo
33 { ps::x => 2 in modes (m2); }
34 in modes (m2, m3);
35 end S.I2;

```

platform:/resource/AADLModel...yLookup/ModalComponents.aadl

- Aadl Spec ModalComponents
 - Property Set PS
 - System Subcomponents
 - System Subcomponent sc: TEST::Foo

Tasks Problems Error Log AADL Property Values X Properties

PS

- 1 x =>
- 2 in modes {(m2)}
- 1 in modes {(m3)}
- 1 y =>
- undefined in modes {(m2)}
- 7 in modes {(m3)}

ADL

54

Originally in modes *m1* and *m3*

Refined in *S.I2* to be in modes *m2* and *m3* instead

Never given value for mode *m2*

Instance Property Values

Thread instance selected

Transparent access to cached property values or property values from declarative model

List of values

```

Library Instance
├── SEI
│   ├── SecurityLevel => 4
│   └── AADL_Properties
│       ├── Active_Thread_Handling_Protocol => abort
│       ├── Active_Thread_Queue_Handling_Protocol => flush
│       ├── Allowed_Processor_Binding => ()
│       ├── Allowed_Processor_Binding_Class => (processor Pentium_Type.Pentium)
│       ├── Compute_Execution_Time => 1 Us..1 Us
│       ├── Deadline => 10 Ms
│       ├── Device_Dispatch_Protocol => Aperiodic
│       ├── Dispatch_Protocol => Periodic
│       ├── Period => 10 Ms
│       └── Source_Text => ("sunseekerplant_ports.ads", "sunseekerplant_ports.adb", "sunseekerplant_process.adb")
    
```

Property Lookup Methods

PropertyHolder declares property lookup methods

- `getSimplePropertyValue(PropertyDeclaration pd)`
`getSimplePropertyValue(String ps, String pn)`
 - For non-modal association of a single-valued property
 - Returns a `PropertyValue` object, or null if association is modal, multi-valued, or not found.
- `getPropertyValueList(PropertyDeclaration pd)`
`getPropertyValueList(String ps, String pn)`
 - For non-modal association of a multi-valued property
 - Returns a List of `PropertyValue` objects, or null if association is modal or not found
- `getPropertyValue(PropertyDeclaration pd)`
`getPropertyValue(String ps, String pn)`
 - General lookup method
 - Returns a `ModalPropertyValue` object
 - (More on this in future session)



Property Setting Methods

`PropertyHolder` declares property setting methods

- `setPropertyValue(`
 `PropertyDeclaration pd, PropertyValue pv)`
 - Create/replace a single-valued property association
 - Returns a `PropertyAssociation` object
 - Modify this object to set the modes of the association
 - Throws `IllegalArgumentException` if the property doesn't apply to the `PropertyHolder`, or if the value is incorrect for the type

- `setPropertyValue(PropertyDeclaration pd, List pv1)`
 - Create/replace a multi-valued property association
 - (Otherwise, same as above)



Class `PredeclaredProperties`

References the properties, types, and constants declared in property sets `AADL_Properties` and `AADL_Project`

- (package `edu.cmu.sei.aadl.model.property.predeclared`)

- E.g., `PredeclaredProperties.PERIOD`
 - Property name `AADL_Properties::Period`
- E.g., `PredeclaredProperties.NANOSEC`
 - Unit literal from type `AADL_Properties::Time_Units`
- E.g., `PredeclaredProperties.MAX_QUEUE_SIZE`
 - Constant `AADL_Project::Max_Queue_Size`





Properties Outline

- Property Sets
 - Property Type declarations
 - Property Name declarations
- Property Associations
- Properties in the Meta Model
- Property Lookup



Using properties in a Plug-in



The Security Level Plug-in

- Associate a “security level” with each component
- Prevent high-level components from sending data to low-level components
 - Contained components must have lower security level
 - Connections must go from lower level to higher level
- Infer least upper bound on security level for components with no explicit setting





Security Level Plug-in

- Declare the security level property
- Write analysis for components
- Write analysis for connections

We analyze the components and connections in 2 passes so that we can correct component security levels **before** checking connections.



The Security Level Property

We declare a new property in a new property set

```
property set SEI is
  SecurityLevel:
    aadlinteger
    applies to (data, subprogram, thread, thread group,
               process, memory, processor, bus, device,
               system);
end SEI;
```





ComponentSecuritySwitch Class (1)

```

public class ComponentSecuritySwitch extends AadlProcessingSwitch { . . .
protected final void initSwitches() {
    componentSwitch = new ComponentSwitch() {
        public Object caseComponentImpl(ComponentImpl ci) {
            PropertyValue cipv =
                ci.getSimplePropertyValue("SEI", "SecurityLevel");
            final long cilv =
                (cipv instanceof IntegerValue)
                ? ((IntegerValue) cipv).getValue() : 0;

            // Get the max security level of my subcomponents
            long maxslv = 0;
            final EList subs = ci.getAllSubcomponent();
            for (Iterator it = subs.iterator(); it.hasNext(); ) {
                final Subcomponent sub = (Subcomponent) it.next();
                final ComponentImpl sci = sub.getComponentImpl();
                if (sci != null) {
                    PropertyValue scipv =
                        sci.getSimplePropertyValue("SEI", "SecurityLevel");
                    if (scipv != null) {
                        if (scipv instanceof IntegerValue) {
                            long slv = ((IntegerValue) scipv).getValue();
                            if (slv > maxslv) maxslv = slv;
                        }
                    }
                }
            }
        }
    }
}
. . .

```

SAE



ComponentSecuritySwitch Class (1)

```

public class ComponentSecuritySwitch extends AadlProcessingSwitch { . . .
protected final void initSwitches() {
    componentSwitch = new ComponentSwitch() {
        public Object caseComponentImpl(ComponentImpl ci) {
            PropertyValue cipv =
                ci.getSimplePropertyValue("SEI", "SecurityLevel");
            final long cilv =
                (cipv instanceof IntegerValue)
                ? ((IntegerValue) cipv).getValue() : 0;

            // Get the max security level of my subcomponents
            long maxslv = 0;
            final EList subs = ci.getAllSubcomponent();
            for (Iterator it = subs.iterator(); it.hasNext(); ) {
                final Subcomponent sub = (Subcomponent) it.next();
                final ComponentImpl sci = sub.getComponentImpl();
                if (sci != null) {
                    PropertyValue scipv =
                        sci.getSimplePropertyValue("SEI", "SecurityLevel");
                    if (scipv != null) {
                        if (scipv instanceof IntegerValue) {
                            long slv = ((IntegerValue) scipv).getValue();
                            if (slv > maxslv) maxslv = slv;
                        }
                    }
                }
            }
        }
    }
}
. . .

```

Extract the integer value

Look up the property value for the current component

SAE





ComponentSecuritySwitch Class (1)

```

public class ComponentSecuritySwitch extends AadlProcessingSwitch { . . .
protected final void initswitches() {
    componentSwitch = new ComponentSwitch() {
        public Object caseComponentImpl(ComponentImpl ci) {
            PropertyValue cipv =
                ci.getPropertyValue("SEI", "SecurityLevel");
            IntegerValue cilv =
                ((IntegerValue) cipv).getValue();

            // Get the max security level of my subcomponents
            long maxslv = 0;
            final EList subs = ci.getAllSubcomponent();
            for (Iterator it = subs.iterator(); it.hasNext(); ) {
                final Subcomponent sub = (Subcomponent) it.next();
                final ComponentImpl sci = sub.getComponentImpl();
                if (sci != null) {
                    PropertyValue scipv =
                        sci.getPropertyValue("SEI", "SecurityLevel");
                    if (scipv != null) {
                        if (scipv instanceof IntegerValue) {
                            long slv = ((IntegerValue) scipv).getValue();
                            if (slv > maxslv) maxslv = slv;
                        }
                    }
                }
            }
        }
    };
}

```

Get all the subcomponents

Get the subcomponent's component implementation

Look up the property value for the subcomponent

Extract the integer value and update the max security level of the subcomponents.



ComponentSecuritySwitch Class (2)

```

        if (maxslv > cilv) {
            if (cipv != null) {
                reportWarning(ci, "Security level set");
                ((IntegerValue) cipv).setNewValue(maxslv);
            } else {
                reportWarning(ci, "Security level set");
                // Create new property value: An Integer value
                final IntegerValue newpv =
                    PropertyFactory.eINSTANCE.createIntegerValue();
                // Set to max security level
                newpv.setNewValue(maxslv);
                // Get the property definition
                final PropertyDefinition pd =
                    ci.getPropertyDefinition("SEI", "SecurityLevel");
                if (pd != null) {
                    // Set the property association
                    ci.setPropertyValue(pd, newpv);
                }
            }
        }
    }
    return DONE;
}
}
} // End class

```

Uh oh! Subcomponents are more secure than container.



ComponentSecuritySwitch Class (2)

If the security level was set, update it to the needed level.

```

if (maxslv > cilv) {
    if (cipv != null) {
        reportWarning(ci, "Security level updated");
        ((IntegerValue) cipv).setNewValue(maxslv);
    }
    else {
        reportWarning(ci, "Security level set");
        // Create new property value: An Integer value
        final IntegerValue newpv =
            PropertyFactory.eINSTANCE.createIntegerValue();
        // Set to max security level
        newpv.setNewValue(maxslv);
        // Get the property definition
        final PropertyDefinition pd =
            ci.getPropertyDefinition("SEI", "SecurityLevel");
        if (pd != null) {
            // Set the property association
            ci.setPropertyValue(pd, newpv);
        }
    }
}
return DONE;
}
}
} // End class
    
```

SAE

ADL

ComponentSecuritySwitch Class (2)

Otherwise, we create a new property association.

(1) Create a new IntegerValue

(2) Init the IntegerValue

(3) Lookup the property definition

(4) Create the property association on the component

```

} else {
    reportWarning(ci, "Security level set");
    // Create new property value: An Integer value
    final IntegerValue newpv =
        PropertyFactory.eINSTANCE.createIntegerValue();
    // Set to max security level
    newpv.setNewValue(maxslv);
    // Get the property definition
    final PropertyDefinition pd =
        ci.getPropertyDefinition("SEI", "SecurityLevel");
    if (pd != null) {
        // Set the property association
        ci.setPropertyValue(pd, newpv);
    }
}
return DONE;
}
}
}
    
```

SAE

ADL



ConnectionSecuritySwitch Class

```

public class ConnectionSecuritySwitch extends AadlProcessingSwitch { . . .
    protected final void initSwitches() {
        connectionSwitch = new ConnectionSwitch() {
            public Object caseConnection(final Connection conn) {
                if (conn instanceof DataAccessConnection
                    || conn instanceof BusAccessConnection) return DONE;

                final PropertyHolder scxt =
                    (PropertyHolder) conn.getAllSrcContextComponent();
                final PropertyHolder dcxt =
                    (PropertyHolder) conn.getAllDstContextComponent();
                if (scxt == null || dcxt == null) return DONE;

                final PropertyValue spv =
                    scxt.getSimplePropertyValue("SEI", "SecurityLevel");
                final PropertyValue dpv =
                    dcxt.getSimplePropertyValue("SEI", "SecurityLevel");
                final long slv = (spv instanceof IntegerValue)
                    ? ((IntegerValue) spv).getValue() : 0;
                final long dlv = (dpv instanceof IntegerValue)
                    ? ((IntegerValue) dpv).getValue() : 0;

                if (slv > dlv)
                    reportError(conn, "Security level violation: Source has level "
                        + slv + " and destination has level " + dlv);
                return DONE;
            }
        };
    }
}

```



ConnectionSecuritySwitch Class

Ignore access connections

Get the security levels of the endpoints

Get the endpoints

Report an error if the destination is less secure than the source

```

. . .
    protected final void initSwitches() {
        connectionSwitch = new ConnectionSwitch() {
            public Object caseConnection(final Connection conn) {
                if (conn instanceof DataAccessConnection
                    || conn instanceof BusAccessConnection) return DONE;

                final PropertyHolder scxt =
                    (PropertyHolder) conn.getAllSrcContextComponent();
                final PropertyHolder dcxt =
                    (PropertyHolder) conn.getAllDstContextComponent();
                if (scxt == null || dcxt == null) return DONE;

                final PropertyValue spv =
                    scxt.getSimplePropertyValue("SEI", "SecurityLevel");
                final PropertyValue dpv =
                    dcxt.getSimplePropertyValue("SEI", "SecurityLevel");
                final long slv = (spv instanceof IntegerValue)
                    ? ((IntegerValue) spv).getValue() : 0;
                final long dlv = (dpv instanceof IntegerValue)
                    ? ((IntegerValue) dpv).getValue() : 0;

                if (slv > dlv)
                    reportError(conn, "Security level violation: Source has level "
                        + slv + " and destination has level " + dlv);
                return DONE;
            }
        };
    }
}

```





ConnectionSecurityAction Class

```

public class CheckSecurity extends AaxlModifyAction {
    public void doAaxlAction(AObject obj) {
        if (obj == null) return;
        final AObject as = obj.getAObjectRoot();

        if (as instanceof AadlSpec) {
            final ComponentSecuritySwitch componentSecuritySwitch =
                new ComponentSecuritySwitch(getMarkerReporter());

            componentSecuritySwitch.processBottomUpComponentImpl((AadlSpec)
                as);

            final ConnectionSecuritySwitch connectionSecuritySwitch =
                new ConnectionSecuritySwitch(getMarkerReporter());
            connectionSecuritySwitch.processPreOrderAll(as);
        }
    }
}

```



ConnectionSecurityAction Class

Extends AaxlModifyAction so that changes to the property associations are saved.

```

public class CheckSecurity extends AaxlModifyAction {
    public void doAaxlAction(AObject obj) {
        if (obj == null) return;
        final AObject as = obj.getAObjectRoot();

        if (as instanceof AadlSpec) {
            final ComponentSecuritySwitch componentSecuritySwitch =
                new ComponentSecuritySwitch(getMarkerReporter());
            componentSecuritySwitch.processBottomUpComponentImpl((AadlSpec) as);

            final ConnectionSecuritySwitch connectionSecuritySwitch =
                new ConnectionSecuritySwitch(getMarkerReporter());
            connectionSecuritySwitch.processPreOrderAll(as);
        }
    }
}

```

*(1) Process **components**—not model elements—bottom up to propagate security levels.*

(2) Process connections.





Outline

- OSATE plug-in design approach
- Model access methods
- Model processing support
- Property processing
- ➔ Result recording & reporting
- Flow processing in declarative models

SAE



Management of Plug-in Results

- Eclipse Markers (temporary & persistent)
- SWT Dialog Box (temporary)
- AADL Properties (persistent)
- AADL Model Adapters (temporary)
- Analysis-specific files (persistent)
- Analysis-specific in-core models (temporary)
- Meta Model-Based XML Documents (persistent)

SAE





Recording Results as Eclipse Markers

- Associate messages with model objects in a resource
- 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
- Markers are accessible programmatically via `org.eclipse.core.resources`.



Reporting Markers: MarkerReporter

- Class `MarkerReporter` manages a plug-ins markers
 - Associates markers with Eclipse `IResource` or EMF `Resource`
 - Removes existing markers from resource
 - In package `edu.cmu.sei.aadl.model.plugin-support`
- Can be associated with a specific type of marker
 - Plug-in-defined marker type
 - Default marker: “`edu.cmu.sei.aadl.model.AadlObjectMarker`”
 - `MarkerReporter.createMarkerReporter(
Resource rsrc, String markertype)`
- Markers associate message with model object:
 - `reportError(AObject location, String message)`
 - `reportWarning(AObject location, String message)`
 - `reportInfo(AObject location, String message)`





Reporting Markers: MarkerReporter - 2

- Marker goes with containing resource
 - Associated with file that contains the model object
 - Location recorded as URI
 - Editor provides *goto* method implementation for location
- Limit on the number of markers per resource
 - Predefined limit per markerReporter
- Secondary model marking
 - Example: instance model generation
 - Tracking of “external” markers

Allows for message with original model



Defining a New Marker Type

Defined in the `plugin.xml` file:

```
<extension
  id="YourAadlObjectMarker"
  name="Your Plugin Object Marker"
  point="org.eclipse.core.resources.markers">
  <super
    type="edu.cmu.sei.aadl.model.AadlObjectMarker"/>
  <persistent value="true"/>
</extension>
```

Plug-in specific marker types allow for results from different plug-in to be managed separately.





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.

SAE



Results as AADL Property Values

- New properties can be introduced through the property set construct in AADL
- Kept persistently in the AADL model
- AADL property values are accessible programmatically via the AADL model API for AADL properties
 - Results of one plug-in can be used by other plug-ins
- AADL property values are visible via AADL model presentation mechanisms
 - AADL Property Viewer
 - Graphical editor
 - Etc.

SAE





Saving Results in Model Adapters

- Allows model object specific information to be passed in context
 - Between different passes of one OSATE plug-in
 - Between plug-ins executed in succession.
- In-core addition of information to AADL model
 - Without extensions to the model representation itself
- Information is accessible programmatically
 - Via adapter concept
 - EMF generated adapter factory meta model class specific adapters
 - Refine adapters for plug-in-specific uses
- ExternalModelAdapter class in edu.cmu.sei.aadl.util



Saving Results in the File System

- Persistent record in file system
- Visible as resource in “Navigator” view
- Permits use of representation appropriate for specific analysis or generation tool; e.g.,
 - Generation of textual AADL
 - Generation of textual MetaH
- Supports interchange representations of external tools





Saving Results in Meta Model-Based XML Documents

- Define an EMF meta model for output
 - Import an existing XML schema of an object model
 - Create the meta model interactively
- Generate the output object model in-core
 - Process the AADL object model
 - Add the generated model root object as content of an EMF resource
 - Call save method of resource to store as XML document

This technique is used to generate the AADL instance model

SAE



Outline

- OSATE plug-in design approach
- Model access methods
- Model processing support
- Property processing
- Result recording & reporting
- ➔ Flow processing in declarative models

SAE





Flow Processing

- Specification-based analysis
 - One hierarchy layer at a time, one component at a time
 - Flow implementation against flow specification
- Single layer end-to-end flow analysis
 - Based on component characteristics
 - Based on component specifications
- Multi layer flow analysis
 - Higher fidelity models
 - Propagation of aggregate property values
 - Refinement of specification value

SAE



Two Illustrative Examples

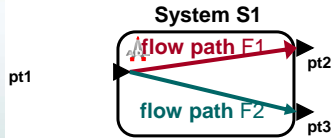


- Flow specification validation
 - Flow latency property
 - Validation one component implementation at a time
 - Aggregation of implementation latency
 - Propagation up the system hierarchy
 - Processing of declarative model
- Single layer end-to-end flow
 - Pilot display response time
 - Determined by flow latency
 - Leverage timing characteristics of subsystems
 - Subsystems as partitions
 - Partition execution semantics
 - Processing of declarative model

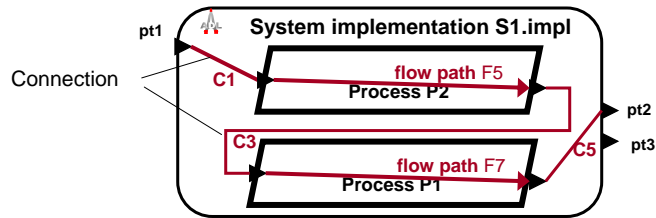
SAE



Flows in AADL



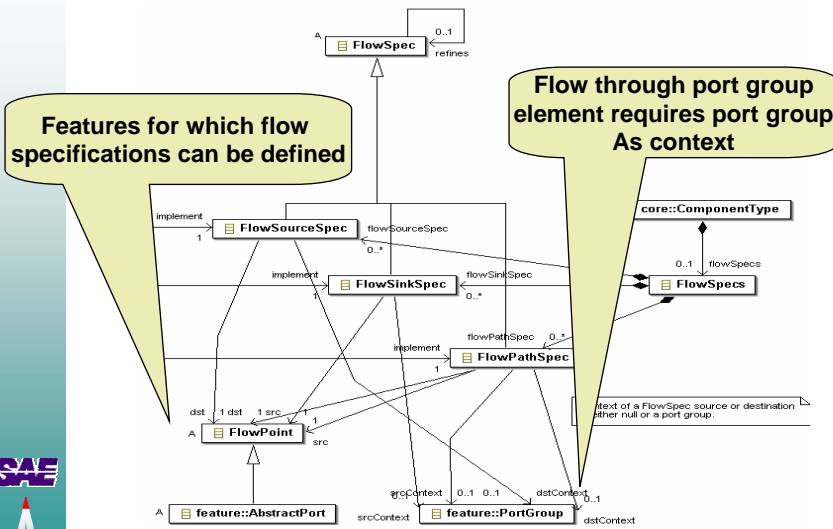
Flow Specification
 F1: flow path pt1 -> pt2
 F2: flow path pt1 -> pt3



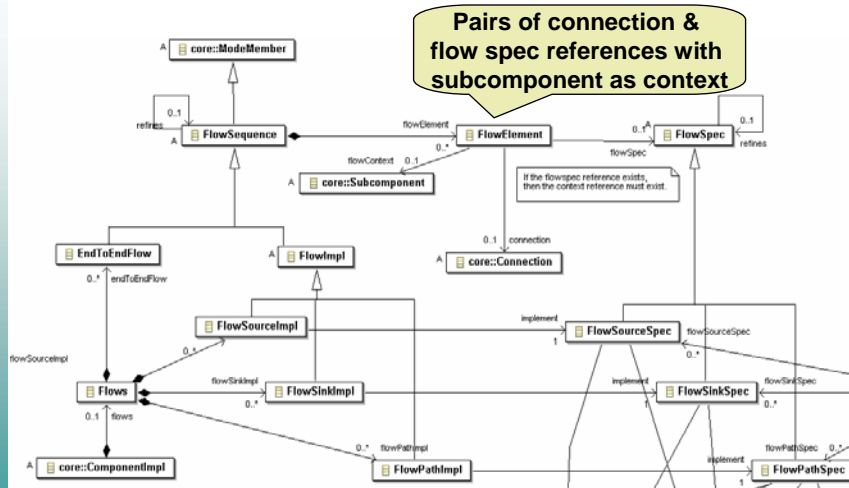
Flow Implementation
 F1: flow path pt1 -> C1 -> P2.F5 -> C3 -> P1.F7 -> C5 -> pt2



Flow Specification Meta Model



Flow Sequence Meta Model



Meeting Flow Requirements

- Requirement as expected latency property of flow specification
- Flow implementation latency determined by
 - Subcomponent flow specification latency
 - Thread execution properties
- Actual latency as computed flow specification property
- Analysis of instance model as analysis of declarative model





Flow Specification Analysis - 1

```

public Object caseFlowImpl(FlowImpl fi) {
    FlowSpec fs = fi.getXImplement();
    EList fel = fi.getFlowElement();
    double result = 0;
    for (Iterator it = fel.iterator(); it.hasNext();){
        FlowElement fe = (FlowElement)it.next();
        Connection conn = fe.getConnection();
        if ( conn != null) {
            IntegerValue cpv = (IntegerValue)conn.getSimplePropertyValue(
                PredeclaredProperties.LATENCY);
            if ( cpv != null ) {
                result +=
                cpv.getScaledValue(PredeclaredProperties.MICROSEC);
            }
        }
    }
}

```

Use of property definition is more efficient than use of name as string.

Add connection latency if connection & latency property exist



Flow Specification Analysis - 2

```

public Object caseFlowImpl(FlowImpl fi) {
    FlowSpec fs = fi.getXImplement();
    EList fel = fi.getFlowElement();
    double result = 0;
    for (Iterator it = fel.iterator(); it.hasNext();){
        <connection latency>
        FlowSpec fefs = fe.getFlowSpec();
        if (fefs != null){
            IntegerValue fefspv =
            (IntegerValue)fefs.getSimplePropertyValue(PredeclaredProperties.
            LATENCY);
            if (fefspv != null){
                result =
                result+fefspv.getScaledValue(PredeclaredProperties.MICROSEC);
            } else {

```

Add flow specification latency if it exists





Flow Specification Analysis - 3

```

Subcomponent sc = fe.getFlowContext();
if (sc instanceof ThreadSubcomponent){
  ThreadClassifier tc = (ThreadClassifier) sc.getXClassifier();
  EnumLiteral dp = tc.getDispatchProtocol();
  if ( dp == PredeclaredProperties.PERIODIC){
    IntegerValue period =
    (IntegerValue)sc.getSimplePropertyValue(PredeclaredProperties
    .PERIOD);
    if (period != null){
      result =
      result+period.getValue()*PredeclaredProperties.MICROSEC;
    } else {
      reportInfo(sc,"Thread subcomponent has no flowspec latency
      or periodic thread deadline");
    }
  } else {
    IntegerValue deadline =
    (IntegerValue)sc.getSimplePropertyValue(PredeclaredProperties
    .DEADLINE);
    if (deadline != null){
      result = result+deadline.getValue();
    } else {
      reportInfo(sc,"Thread subcomponent has no flowspec latency
      or thread deadline");
    }
  }
}

```

Use period for periodic threads

Use deadline for aperiodic threads



Application to Instance Model

- Analysis defined for declarative model
- Instance model visitation defers processing to declarative model methods

```

public Object
caseComponentInstance(ComponentInstance ci) {
  subcomponent sub = ci.getSubcomponent();
  ComponentImpl cii = sub.getComponentImpl();
  self.processPreOrderAll(cii);
  return DONE;
}

```

Call on processing method in declarative model

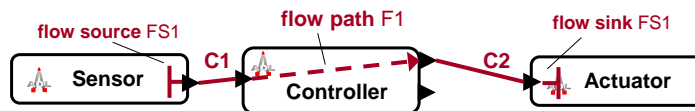


Two Illustrative Examples

- Flow specification validation
 - Flow latency property
 - Validation one component implementation at a time
 - Aggregation of implementation latency
 - Propagation up the system hierarchy
 - Processing of declarative model
- ➔ Single layer end-to-end flow
 - Pilot display response time
 - Determined by flow latency
 - Leverage timing characteristics of subsystems
 - Subsystems as partitions
 - Partition execution semantics
 - Processing of declarative model



End-To-End Flows in AADL



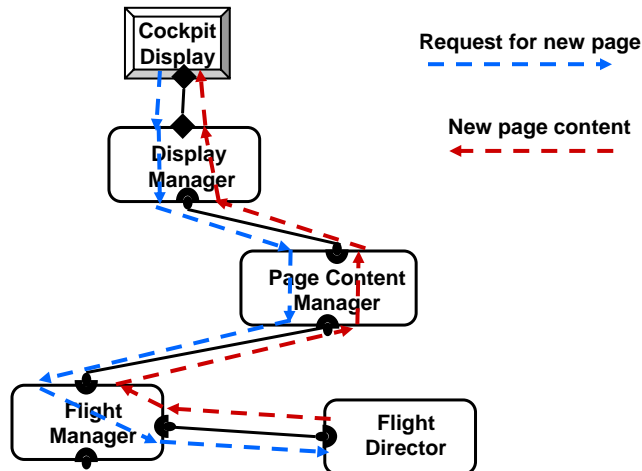
End-To-End Flow Declaration

SenseControlActuate: **end to end flow** Sensor.FS1 -> C1 ->
Controller.F1 -> C2 -> Actuator.FS1





End-To-End Command Response Time



Response Time Analysis

- Worst-case scenario
 - Period delay per partition communication
 - DM sampling latency (max. = partition period)
 - Six periods of partition communication latency
 - DM execution latency (max. = partition period)
- Single processor static timeline
 - One direction immediate, opposite direction phase delayed
 - Reduces partition communication latency to three periods
- Multiple processor synchronous system
 - Take into consideration bus/network latency
- Multiple processor asynchronous system
 - Asynchronous sampling with max. sampling latency = period



Response Time Plug-in - 1

Iterate over flow elements, i.e., connection /subcomponent flow spec pairs

```

public Object caseEndToEndFlow(EndToEndFlow e) {
    int responsetime = 0;
    for(Iterator it = etef.getFlowElement().iterator(); it.hasNext()){
        FlowElement fe = (FlowElement) it.next();
        Subcomponent sub = fe.getFlowContext();
        IntegerValue ppv =
        sub.getSimplePropertyValue("SEI", "Partition_Period");
        if (ppv == null )
            reportError(sub, "Subcomponent is missing partition period");
        else
            responsetime +=
            ppv.getScaledValue(PredeclaredProperties.MICROSEC);
    }
}

```

Get the partition period of the subcomponent



Response Time Plug-in - 2

```

public Object caseEndToEndFlow(EndToEndFlow etef) {
    int responsetime = 0;
    for(Iterator it = etef.getFlowElement().iterator(); it.hasNext()){
        <get the property value>
        if (ppv == null )
            reportError(sub, "Subcomponent is missing partition period");
        else
            responsetime +=
            ppv.getScaledValue(PredeclaredProperties.MICROSEC);
    }
}

```

Aggregate latency in units of micro seconds





Response Time Plug-in - 3

```
public Object caseEndToEndFlow(EndToEndFlowAnalysisContext context) {
    <calculate aggregate responsetime>
    etef.setSimplePropertyValue("Actual_Latency",responsetime);
    IntegerValue epv =
        etef.getSimplePropertyValue("Expected_Latency");
    if (responsetime >
        epv.getScaledValue(context.getDeclaredProperties.MICROSEC))
        reportWarning(etef,"Actual latency exceeds expected
        latency");
    return DONE;
}
```

Store result as property value

Compare against expected latency



Summary

- Model analysis made simple
 - Categories of plug-ins with different complexity
 - Tailored model processing infrastructure
- Property retrieval made simple
 - Property viewer for full property value complexity
 - Property value retrieval and setting for simple values
 - Property values processing for modal values and lists
- Processing of declarative models goes a long way
 - Connection & flow based analysis
 - Via containment traversal methods
 - Tailored processing patterns
 - Persistent and temporary result recording & reporting

