



Plug-in Development for the Open Source AADL Tool Environment Part 3: Generation & External Models

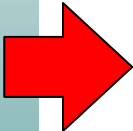


**Peter Feiler / Aaron Greenhouse
Software Engineering Institute
• (phf / aarong)@sei.cmu.edu
412-268- (7790 / 6464)**

ADL



OSATE Plug-in Development Series

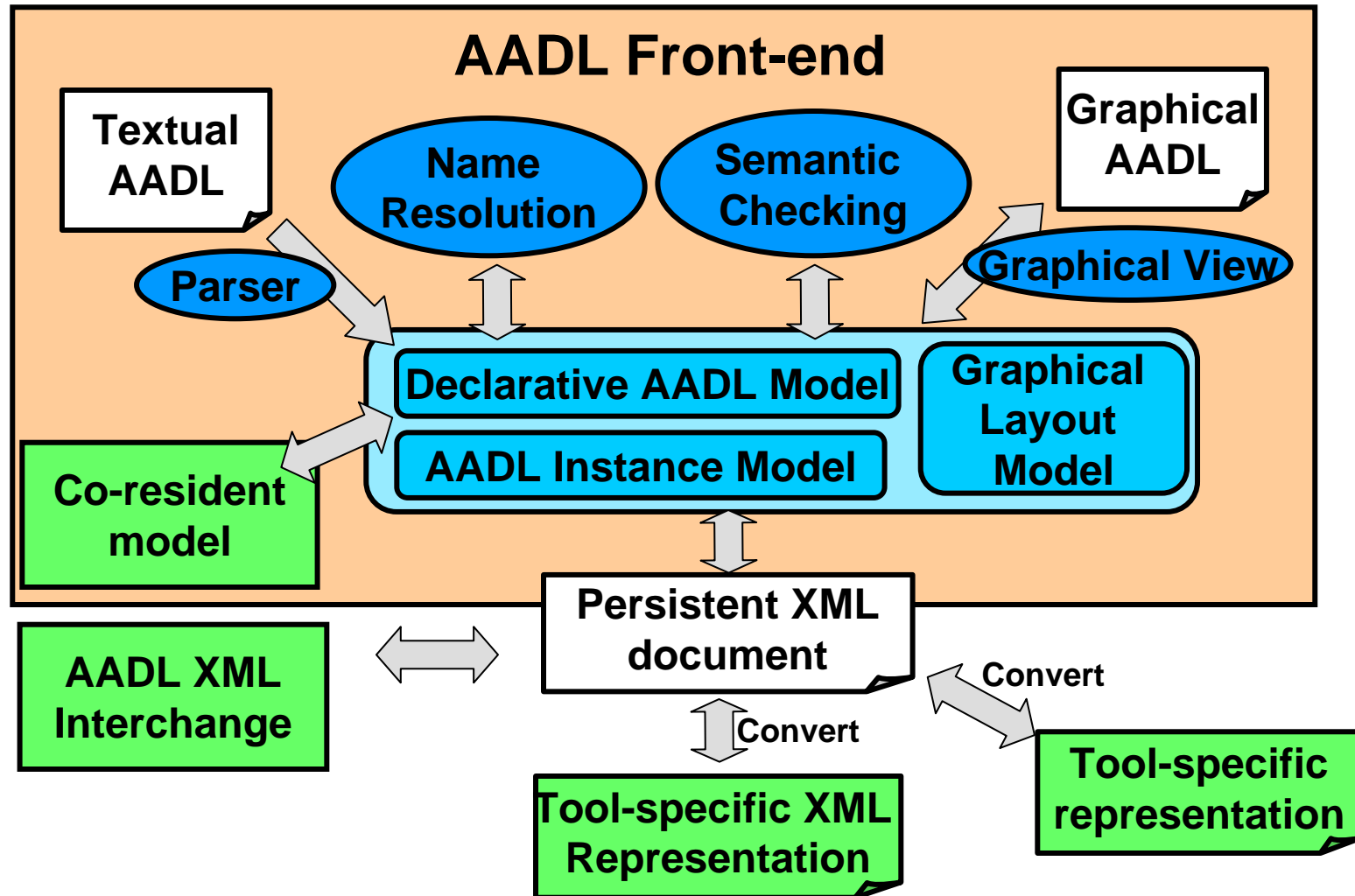
- Introduction to OSATE Plug-in Development
 - OSATE capabilities & plug-in architecture
 - AADL Meta model & example plug-in
- OSATE Plug-in Development Process
 - Plug-in development design approach
 - Model traversal & AADL properties
 - Analysis plug-ins & result management
-  Interfacing with Existing Models & Tools
 - Declarative & instance models
 - Generation & external representations
- OSATE Infrastructure & API
 - Modal system models
 - Persistency
 - Sublanguage extensions

SAE





Tool Interoperability





Model Generation Approaches

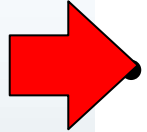
- Text-based model representations
- XML-based model representations
- In-core model representations

SAE





Outline



- Generation of Textual Models
- Generation of XML-based Models
- AADL Instance Meta Model
- AADL Instance Model Generation
- Instance Model Analysis: Priority Inversion
- In-core External Model Generation
- Resource Binding & Scheduling Analysis

SAE





Generation of Textual Models

- Template-based generation
 - XSLT
 - XML -> XML
 - XML -> text
 - Java Emitter Templates (JET)
 - Generator compilation
 - Use by EMF generator
 - No recursive template invocation
- Formatted text generation
 - OSATE support
 - Used by textual AADL generator & MetaH generator

SAE





A JET Template Example

```
<%if (genFeature.isChangeable()) {%>
```

```
/**
```

```
 * <!-- begin-user-doc -->
```

```
 * phf: add element to multiplicity EList
```

```
 * <!-- end-user-doc -->
```

```
 * @generated
```

```
 */
```

```
public void
```

```
add<%=genFeature.getAccessorName()%>(<%=genFeature.getListItem  
Type()%> new<%=genFeature.getCapName()%>)
```

```
{
```

```
<%if (!genFeature.isVolatile() || genFeature.hasDelegateFeature()) {%>  
    if (new<%=genFeature.getCapName()%> != null) {
```

```
        this.get<%=genFeature.getAccessorName()%>().add(new<%=genF  
eature.getCapName()%>);
```

```
    }
```

Target language text

Control code accessing model object attributes



OSATE Text Formatting Support

- Class UnparseText in edu.cmu.sei.aadl.model.util
- Output into StringBuffer
 - addOutput(String str) : append text to current line
 - addOutputNewline(String str): append text and add newline
- Indentation control
 - incrementIndent() & decrementIndent()
 - Affects next indentation
 - Indentation added before first string of a new line
- Result retrieval
 - String getParseOutput()

SAE





Content-Driven Model Processing

- Limitations of traversal-based switch processing
 - Subclass before superclass processing
 - One parent visit only
 - Prefix only, postfix only
- Need for content-driven processing
 - Different content orderings
 - Optional list processing
 - List element separator & terminator processing
 - Multi-pass processing
 - Property value driven output
 - Context-sensitive output

SAE





AADL Text Generator

`edu.cmu.sei.aadl.unparser`

- Use of class hierarchy
 - Category-specific processing before common processing
 - Component types, implementations, subcomponents, features
- Processing of optional subclauses
 - Subclause processing: Non-existent, empty, non-empty lists
 - Property value lists: list element separators
 - Lists of declarations: list element terminators
 - Automatic indentation
- Context-sensitive processing
 - Property associations as subclause vs. bracketed list
 - Package qualification for non-local references

SAE





Use of Class Hierarchy

```
public Object caseProcessImpl(ProcessImpl object) {  
    processComments(object);  
    aadlText.addOutput("process implementation ");  
    return NOT_DONE;  
}
```

Process ComponentImpl case method

```
public void processComments(AObject obj) {  
    if (obj == null) return;  
    EList el = obj.getComment();  
    if (el.isEmpty()) return;  
    for (Iterator it = el.iterator(); it.hasNext(); ) {  
        String comment = (String) it.next();  
        aadlText.addOutputNewline(comment);  
    }  
}
```

Processing of model object attribute





Multiple Model Object Actions

```
public Object caseAadlPackage(AadlPackage  
object) {  
    processComments(object);
```

Prefix processing

```
    aadlText.addOutputNewline("package " +  
object.getName());
```

Switch-based content processing

```
    self.process(object.getAadlPublic());  
    self.process(object.getAadlPrivate());
```

```
    aadlText.addOutputNewline("end " +  
object.getName()+ " ;");
```

```
    return DONE;
```

```
}
```

Postfix processing



Processing of Optional Clauses

- processOptionalSection method
 - Subclause label
 - Empty list label
 - List element separator label

```
public Object caseFlowSpecs(FlowSpecs object) {  
    processComments(object);  
    EList ftl = object.eContents();  
    // do as flow specs  
    processOptionalSection(ftl, "flows", NONESTMT);  
    return DONE;  
}
```

Optional flows subclause processing



Context-Sensitive Processing

Processing of model object attribute

```
public Object caseSubcomponent(Subcomponent object) {  
    if ( object.isRefined() )  
        aadlText.addOutput("refined to ");  
    aadlText.addOutput(  
        ownPackage(object, object.getQualifiedClassifierName  
            ( ) ) );  
    processComments(object.getProperties());  
    processCurlyList(object.getProperty());  
    processModeMembers(object);  
    aadlText.addOutputNewline(";");  
    return DONE;  
}
```

Property associations as curly bracket list





MetaH Text Generator

edu.cmu.sei.aadl.toMetaH

- Multi-pass processing
 - All data types must be declared as port types
 - Data type properties in port type package implementation
 - Data component declarations as port types, property associations, monitors
- Component declaration before use
 - Component type before component implementation
 - Component implementation before subcomponent classifier reference
- Property-based reserved words
 - Thread dispatch protocol
- Content-dependent reserved words
 - System vs. macro reserved word

SAE





Multi-Pass Processing

- Option 1: Multiple switches
 - Separate switch for each pass
- Option 2: conditional processing with single switch

```
public Object caseDataType(DataType object) {  
    if (doData) {  
        if (doDataProperties){  
            EList plist = object.getProperty();  
            processEList(plist);  
        } else {  
            processComment(object);  
            aadlText.addTextNewline(object.getName()+" :  
type;");  
        }  
    } else {  
        // monitor processing of subprogram features  
    }  
}
```

Condition set before processing pass



Sorted Component Declarations

```
EList getDeclarationOrderedComponentClassifiers(  
    AObject aobj){  
    AadlSpec as = aobj.getAadlSpec();  
    final EList uniqueClassifiers = new  
        UniqueEList();  
    ForAllAObject collect = new ForAllAObject(){  
        };  
    collect.processBottomUpComponentImpl(as);  
    return uniqueClassifiers;  
}
```

Ensures single declaration per component classifier

Utilize existing traversal method



Sorted Component Declarations - 2

```
EList getDeclarationOrderedComponentClassifiers(  
    AObject aobj){  
    ForAllAObject collect = new ForAllAObject(){  
        public void process(EObject theEObject) {  
            ComponentImpl ci = (ComponentImpl)  
theEObject;  
            ComponentType ct = ci.getComponentType();  
            uniqueClassifiers.add(ct);  
            if (uniqueClassifiers.contains(ci)){  
                uniqueClassifiers.remove(ci);  
            }  
            uniqueClassifiers.add(ci);  
        }  
    }  
}
```

Handle multiple visits by traversal method
Move component implementation declaration to end



Property-Based Reserved Words

```
EnumLiteral e1 = object.getDispatchProtocol();
```

Get DispatchProtocol property value

```
if ( e1 == PredeclaredProperties.PERIODIC)
    aadlText.addOutput("periodic ");
else if (e1 == PredeclaredProperties.APERIODIC)
    aadlText.addOutput("aperiodic ");
else if (e1 == PredeclaredProperties.SPORADIC){
    aadlText.addOutputNewline("-- Sporadic thread
modeled as aperiodic process");
    aadlText.addOutput("aperiodic ");
}
```

Property value specific output



Content-Dependent Reserved Words

- MetaH macro for application system
- MetaH system for system containing execution platform components
- Propagation of marking up containment hierarchy

Bottom up marking pass

```
MarkSystemSwitch sw = new MarkSystemSwitch();  
sw.processBottomUpComponentImpl(as);  
// now we process the component classifiers  
EList uniqueClassifiers =  
    AadlUtil.getDeclarationOrderedComponentClassifiers(as)  
    ;  
processEList(uniqueClassifiers,NEWLINE);
```

Output conditional on adapter marking



Recording of Temporary Results

- Use of adapter to mark AADL model objects
- ExternalModelAdapter class in edu.cmu.sei.aadl.util

Associate adapter with model object

```
ExternalModelAdapter ad =  
    (ExternalModelAdapter)ExternalModelAdapterFactory.INST  
ANCE.adapt(aobject,ExternalModelAdapter.class);
```

```
ad.setExternalModelObject(DONE);
```

Set attribute in adapter

```
ExternalModelAdapter ad =  
    (ExternalModelAdapter)ExternalModelAdapterFactory.INST  
ANCE.adapt(aobject,ExternalModelAdapter.class);
```

```
if (ad.getExternalModelObject() != null)
```

```
    aadlText.addOutput("system ");
```

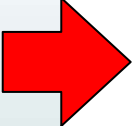
```
else
```

```
    aadlText.addOutput("macro ");
```

Check attribute in adapter



Outline

- Generation of Textual Models
-  Generation of XML-based Models
- AADL Instance Meta Model
- AADL Instance Model Generation
- Instance Model Analysis: Priority Inversion
- In-core External Model Generation
- Resource Binding & Scheduling Analysis

SAE





XML-Based Model Generation

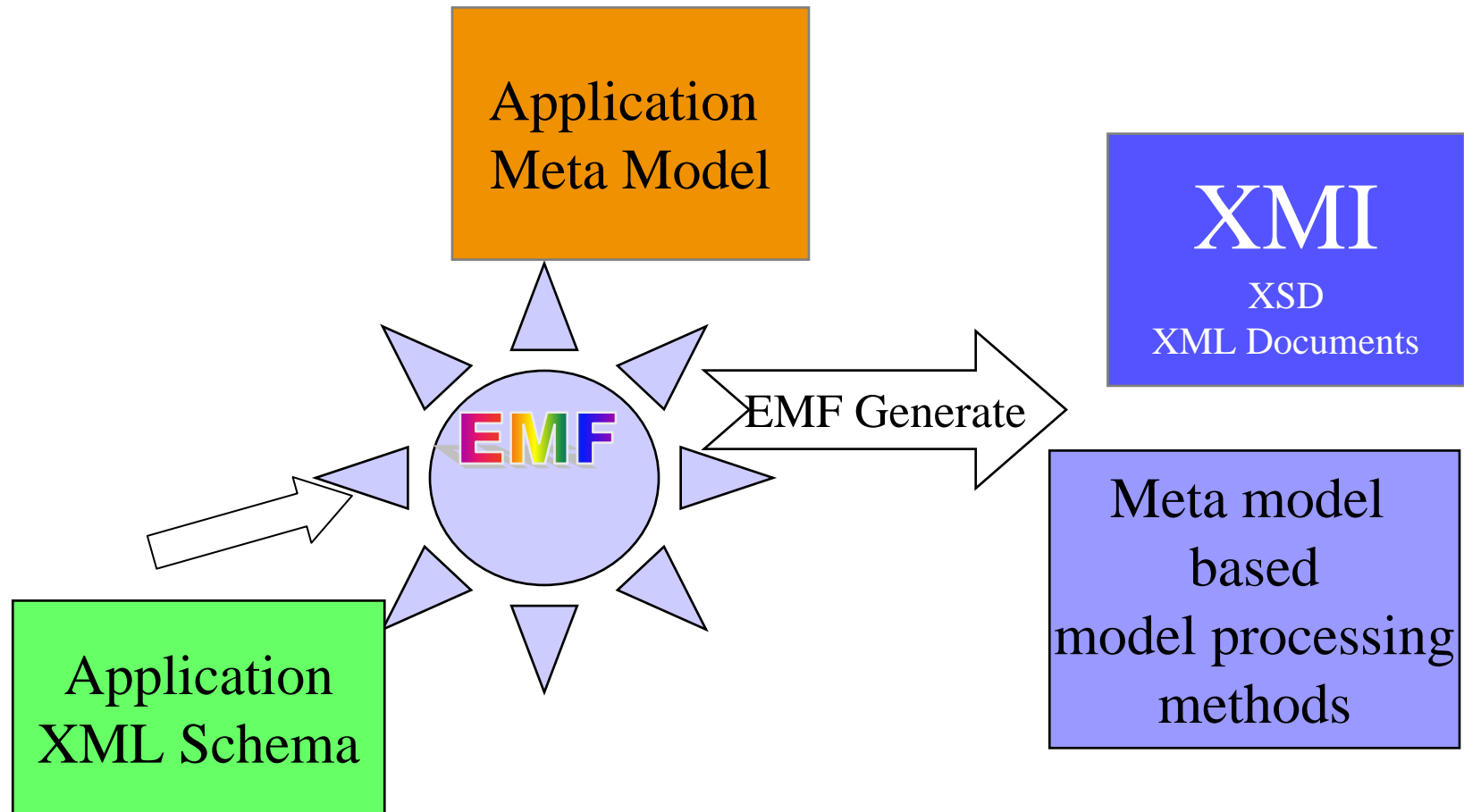
- Textual XML generation
- XML -> XML transformation
- In-core model generation with XML Persistence
 - Based on EMF Meta model
 - Meta model generation from XML Schema or interactive editor
 - EMF generated model creation & manipulation methods from Meta model

SAE





Leverage of EMF



EMF – Eclipse Modeling Framework





AADL Instance Model Generation

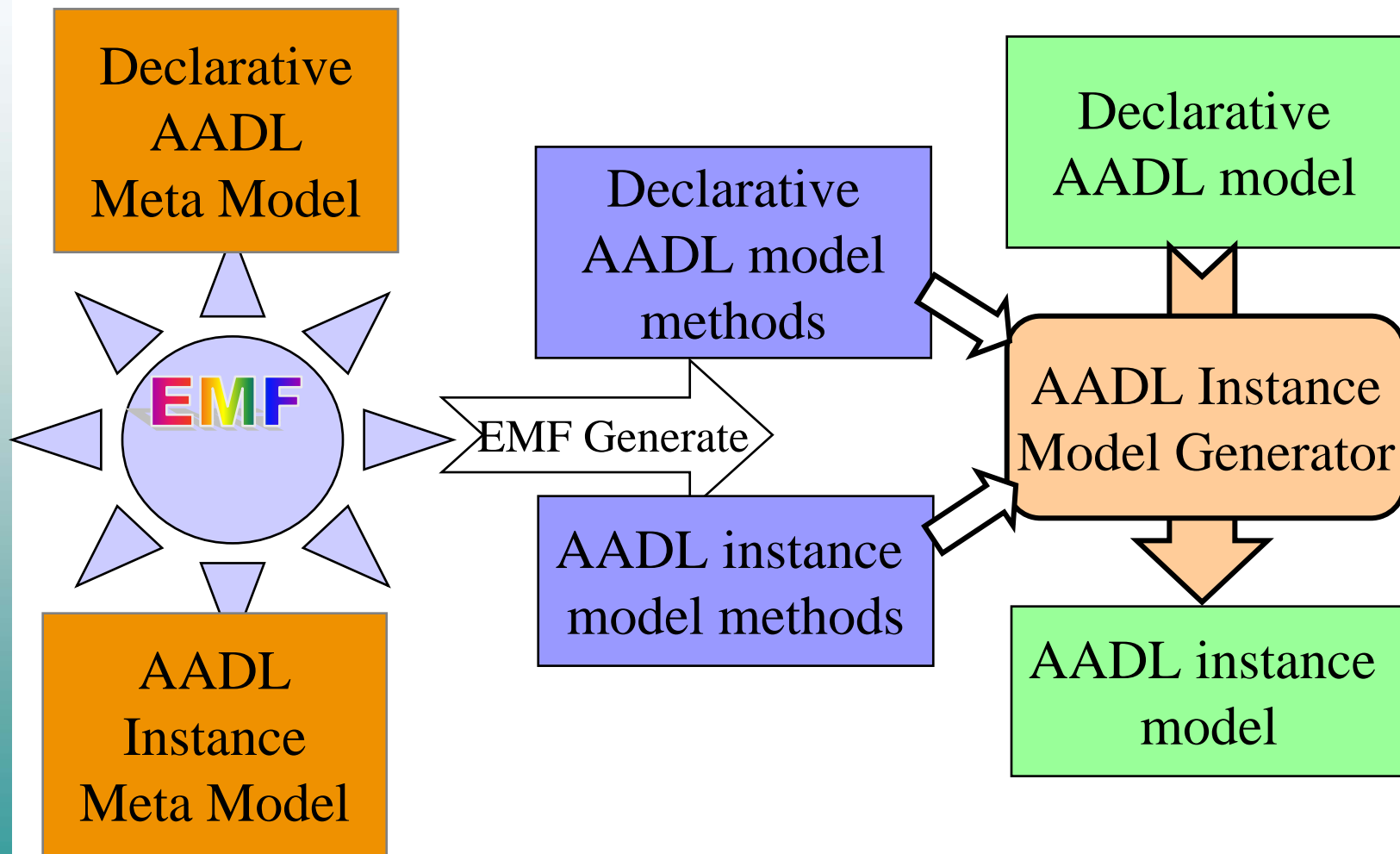
- Instance model defined by AADL Instance Meta model
- Instance model derived from declarative AADL model

SAE





EMF-Based Instance Model Generator



EMF – Eclipse Modeling Framework





Outline

- Generation of Textual Models
- Generation of XML-based Models
- ➔ AADL Instance Meta Model
- AADL Instance Model Generation
- Instance Model Analysis: Priority Inversion
- In-core External Model Generation
- Resource Binding & Scheduling Analysis

SAE





AADL Instance Model Objectives

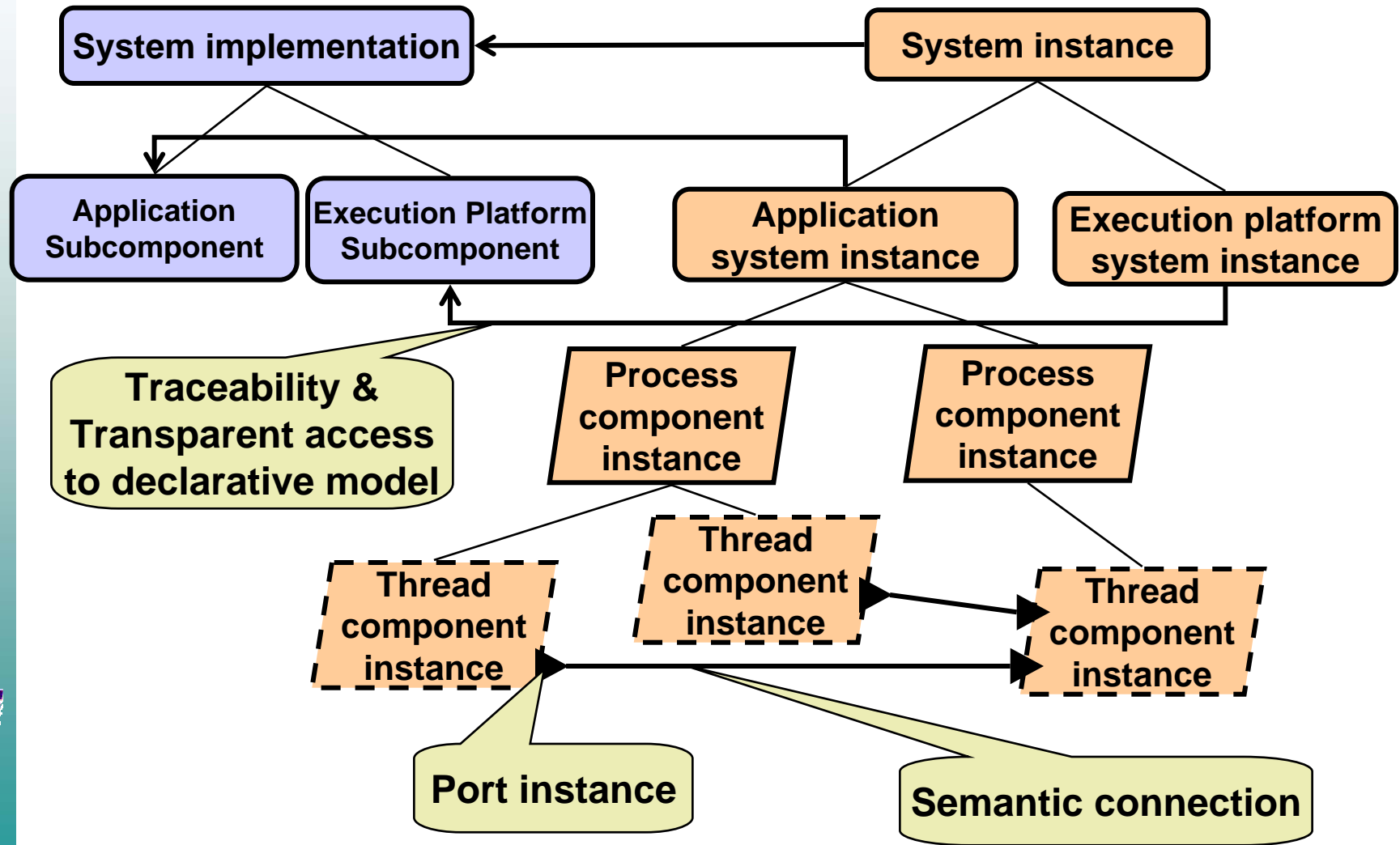
- Derivable from declarative AADL model
 - System implementation as root
 - Application & execution platform as subcomponents
 - Traceability to declarative model
- Self-contained compact system model
 - Compact representation
 - Separately loadable XML document
 - Semantic connections
 - Profile of locally cached property values
- Modal system instances
 - Legal mode combinations for system operation modes
 - System operation mode specific property values
- Recording of instance analysis results

SAE



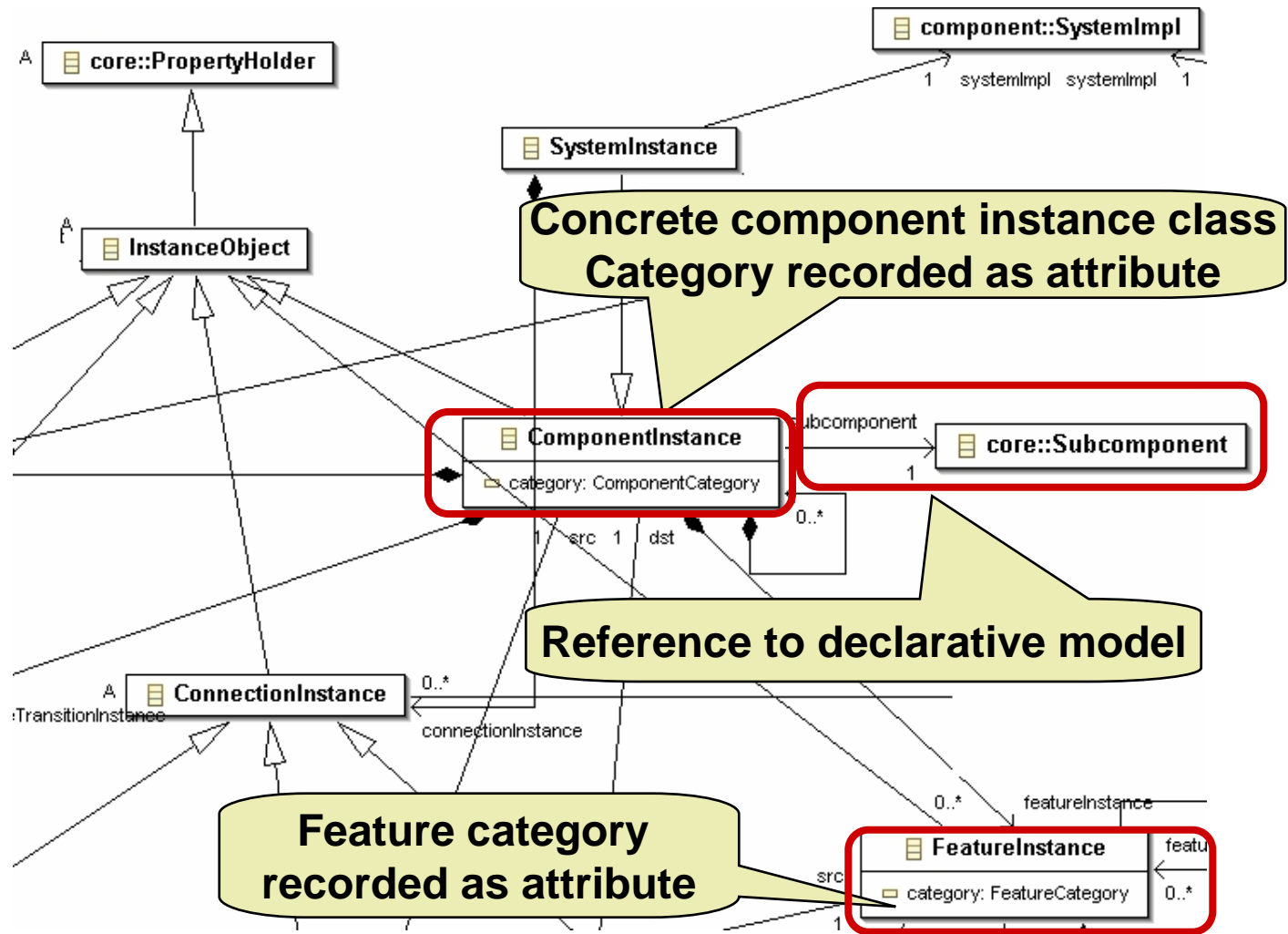


AADL Instance Model



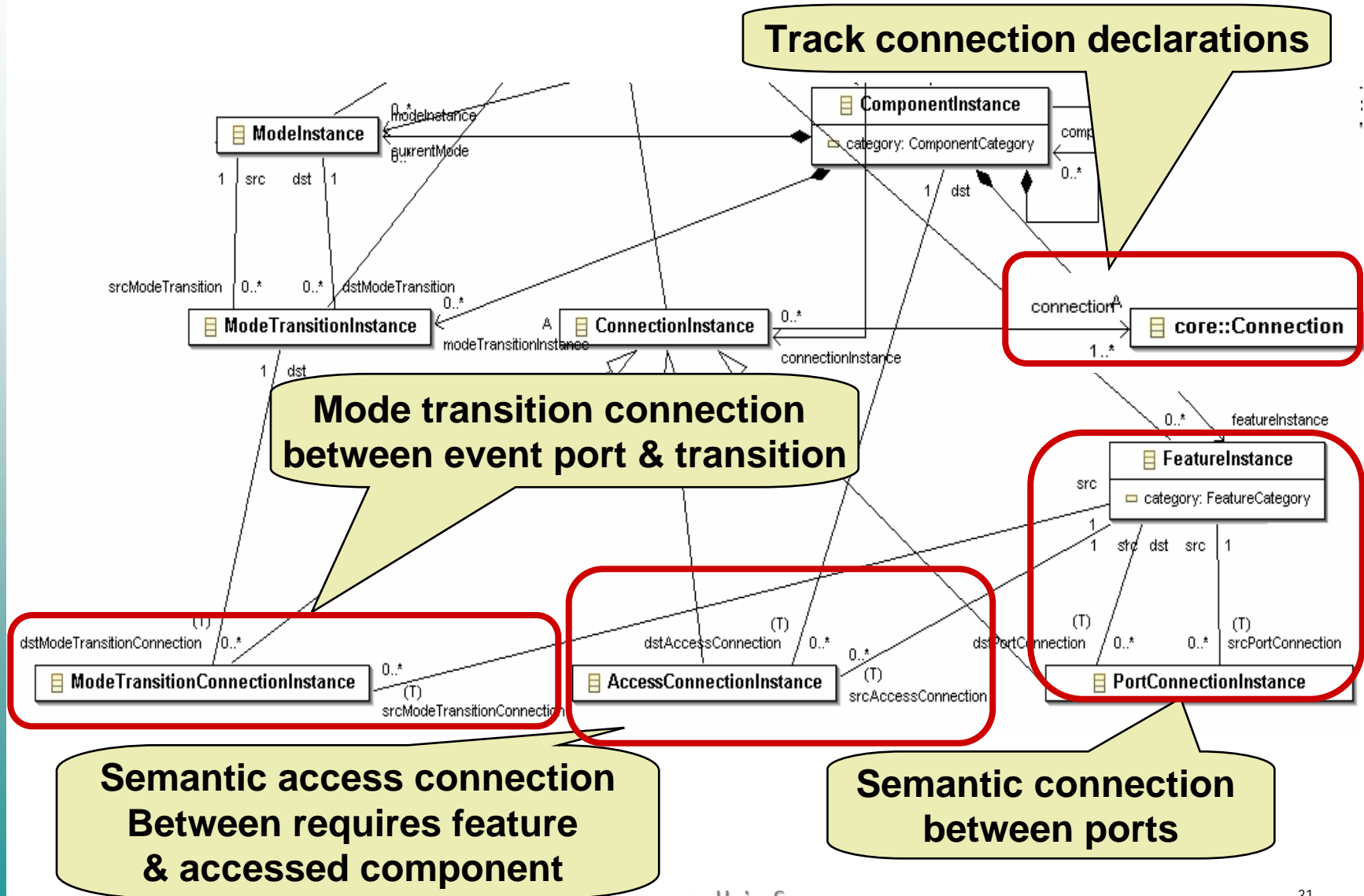


Instance Meta Model





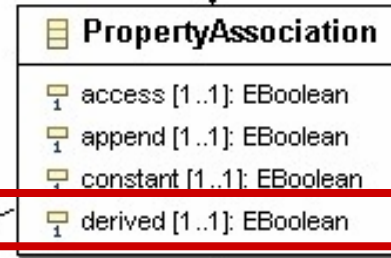
Semantic Connections



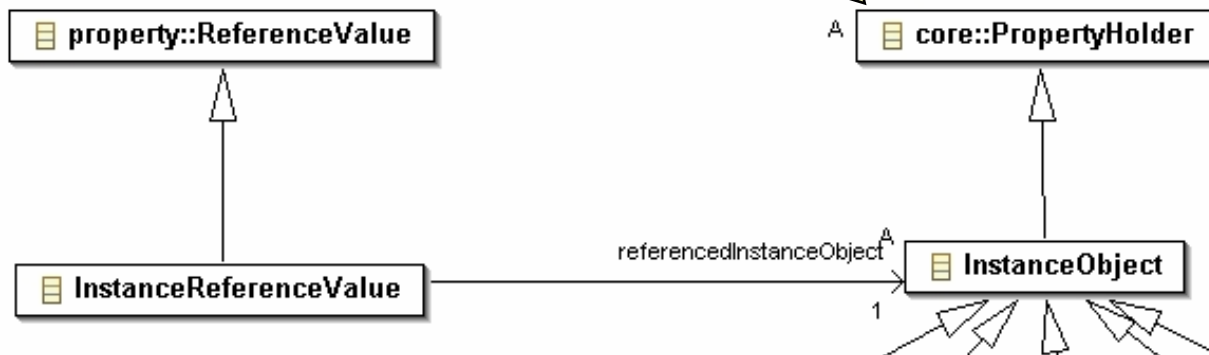


Instance Property Values

Cached property association indicator

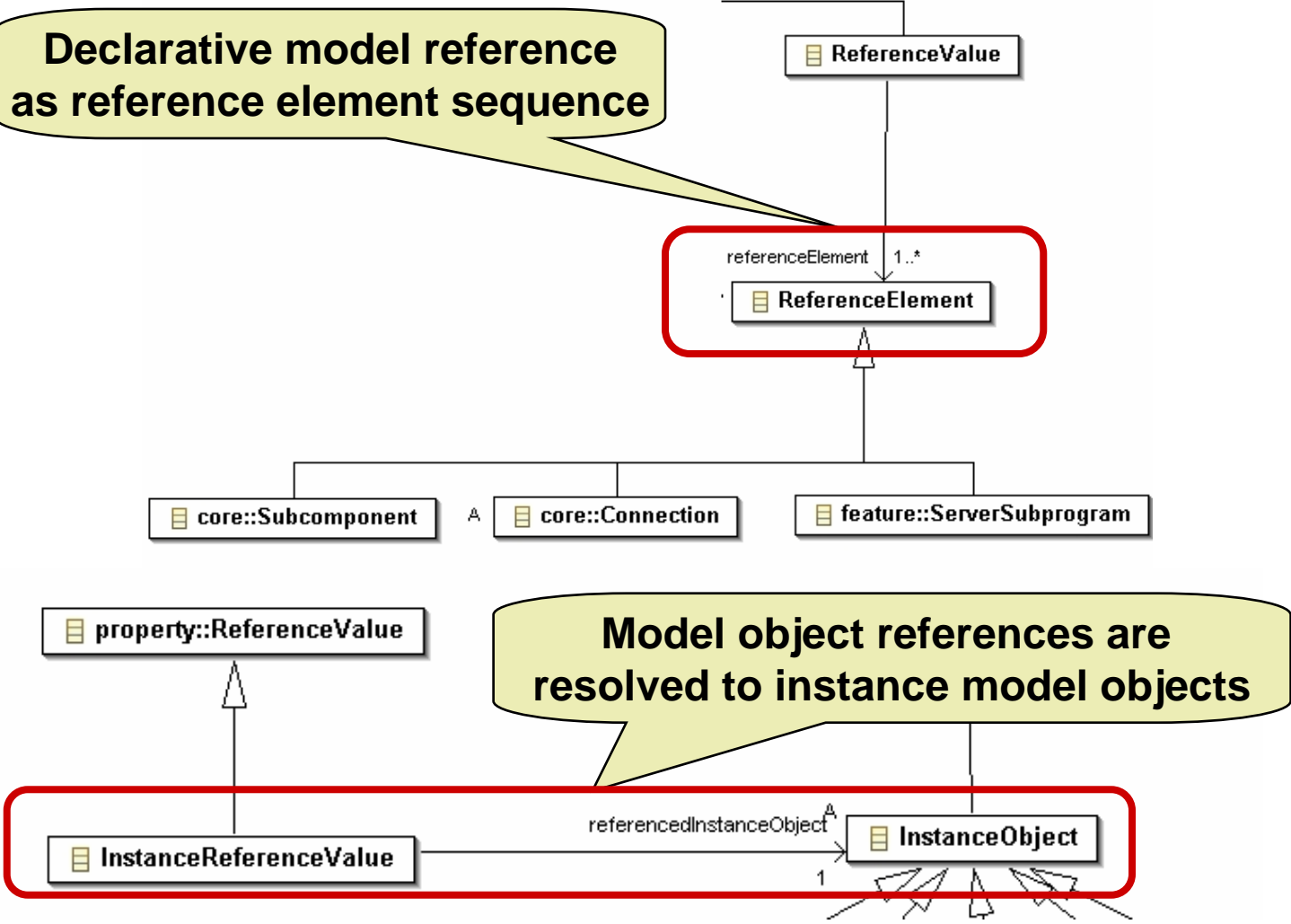


All instance model objects can have property associations



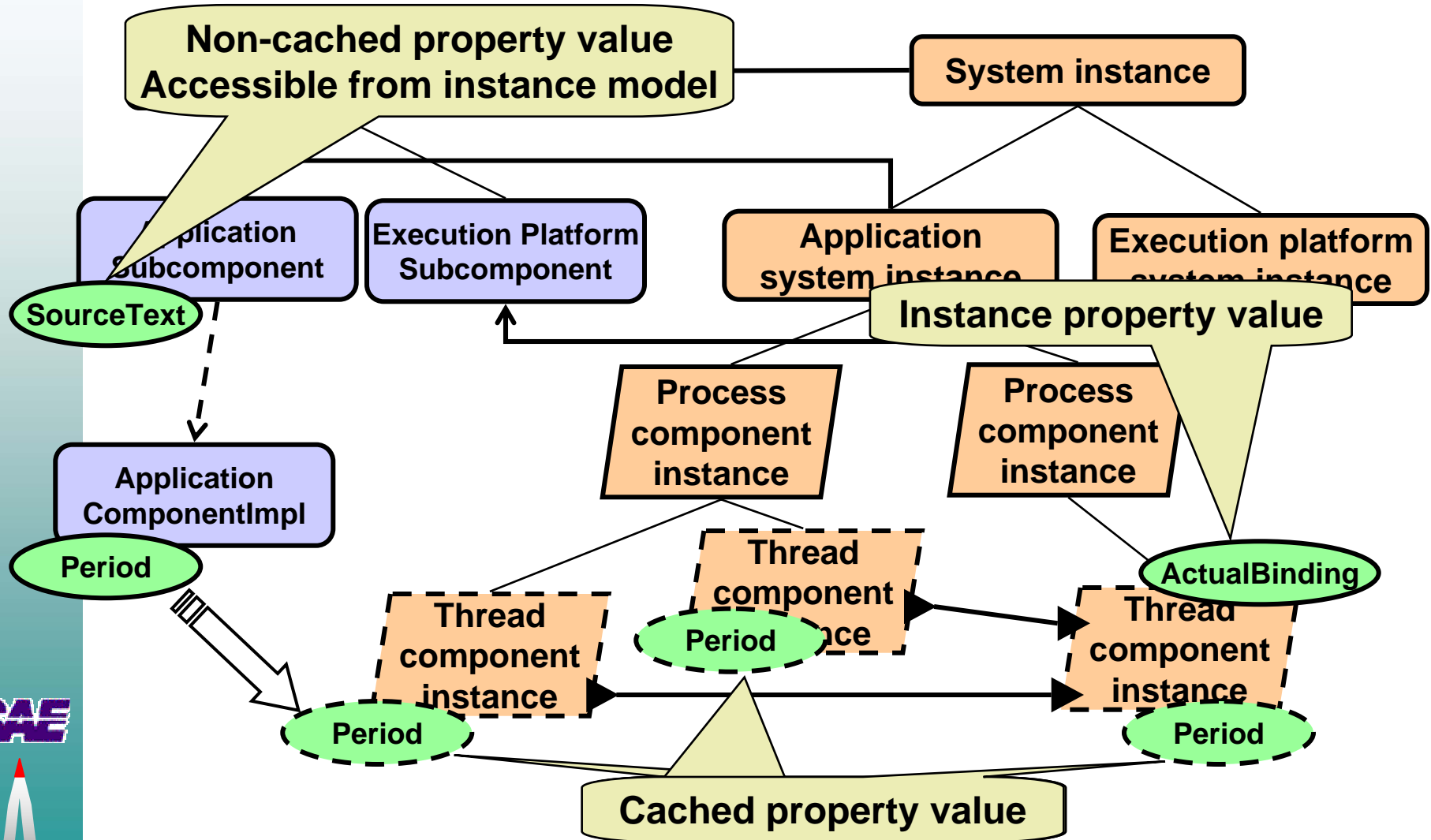


Model Object Reference Values





AADL Properties & Instance Model





Caching of Property Values

- Recording of contained property values
 - Cannot be evaluated in declarative model
 - Always recorded as cached property values
- Caching of declared property values
 - Profile of property definitions
 - Default implementation

```
EList propertyDefinitionList =
```

```
AadlUtil.getAllUsedPropertyDefinition(as);
```

```
InstantiateModel instantiateModel = new  
InstantiateModel(getMarkerReporter(),  
propertyDefinitionList);
```

Property caching profile





Instance Model Property Values

- Use for results of instance model analysis
- Associated with instance model only
- Translated into textual representation as contained property associations
- System Operation Mode specific property value sets

SAE



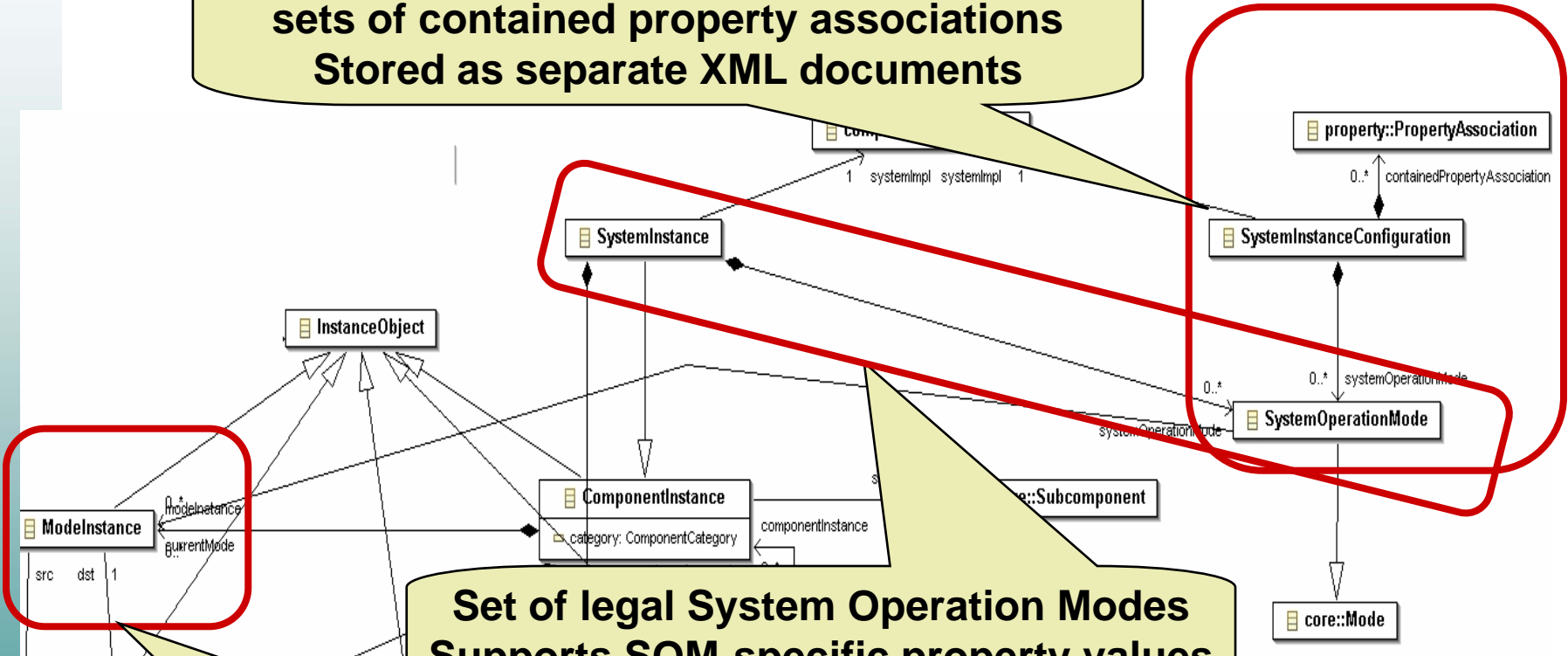


Modal & Configurable System Instances

System instance configurations in form of sets of contained property associations
Stored as separate XML documents

Set of legal System Operation Modes
Supports SOM-specific property values

Mode instances per modal component instance





Outline

- Generation of Textual Models
- Generation of XML-based Models
- AADL Instance Meta Model
- ➔ AADL Instance Model Generation
- Instance Model Analysis: Priority Inversion
- In-core External Model Generation
- Resource Binding & Scheduling Analysis

SAE





Generation of Persistent Instance Model

- Derive resource URI from declarative model resource
- Add instance model as resource content
- Save resource

```
Resource res = si.eResource();  
URI modeluri = res.getURI();  
String last = modeluri.lastSegment();  
URI instanceURI =  
    modeluri.trimSegments(1).appendSegment(si.getTypeName()+ "_Instance.aaxl");
```

```
Resource aadlResource =  
    OsateResourceManager.getEmptyResource(instanceURI);
```

```
SystemInstance root =  
    instantiateModel.createSystemInstance(si, aadlResource);
```

```
OsateResourceManager.save(aadlResource);
```



Generation of Semantic Connections

- Traceability to connection declarations
- Fan-out of connection declarations
 - Results in separate semantic connections
- Packing and unpacking through port groups
 - Selection of correct port group element
- Port groups as semantic connection end-points
 - Unfolding into individual semantic connections
 - Asymmetric packing & unpacking
- Aggregate data port
 - Single semantic connection

SAE





Semantic Checking of Instance Model

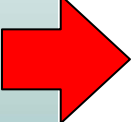
- Required components
 - Thread, processor, memory
- Required thread properties
- Semantic connection consistency
 - Endpoint consistency
 - Delayed connection end-point characteristics
 - Property value consistency
- Required initial mode
- Threads containing server subprograms

SAE





Outline

- Generation of Textual Models
- Generation of XML-based Models
- AADL Instance Meta Model
- AADL Instance Model Generation
-  Instance Model Analysis: Priority Inversion
- In-core External Model Generation
- Resource Binding & Scheduling Analysis

SAE





Priority Inversion Plug-in

- Checking of threads with explicitly assigned priority
 - Execution related property values
 - Priority as new property
- Requires processing of instance model
 - Application threads & processors
 - Runtime configuration with threads bound to processors
- Filtered instance model object processing
- Sorted list processing

SAE





Priority Inversion Processing

- For all processors do
- Get all threads bound to processor
- Sort threads by period
- Compare assigned priority for monotonicity across rate groups

SAE





ForAll Processor Processing

- Alternative 1: invoke checking as part of visitation

Redefine the processing call

```
public void  
    checkSystemPriorityInversion(SystemInstance si){  
    ForAllAObject mal = new ForAllAObject (){  
        public void process(AObject obj){  
            checkPriorityInversion((ComponentInstance)obj);  
        }  
    };  
    mal.processPreOrderComponentInstance(si,  
        ComponentCategory.PROCESSOR_LITERAL);  
}
```

Visit all processor instances



ForAll Processor Processing - 2

- Alternative 2: generate a processor list

Generate processor list using default implementation of ForAllAObject

```
public void  
  checkSystemPriorityInversion(SystemInstance si){  
    EList proclist = new  
    ForAllAObject().processPreOrderComponentInstance(  
    si,  
      ComponentCategory.PROCESSOR_LITERAL);  
    for (Iterator it = proclist.iterator();  
        it.hasNext());){  
      checkPriorityInversion(  
        (ComponentInstance)it.next());  
    }  
  }  
}
```

Iterate over processors in list





Thread List Per Processor - 1

```
public void checkPriorityInversion(final  
    ComponentInstance curProcessor){  
    SystemInstance root =  
        currentProcessor.getSystemInstance();  
    List boundThreads = new ForAllAObject(){  
        protected boolean suchThat(AObject obj){  
            ComponentInstance boundProcessor =  
                TimingUtil.getActualProcessorBinding((ComponentIn  
                stance)obj);  
            return (boundProcessor == currentProcessor);  
        }  
    }.processPreOrderComponentInstance(root,  
        ComponentCategory.THREAD_LITERAL);
```

Get instance model root



Thread List Per Processor - 2

```
public void checkPriorityInversion( final  
    ComponentInstance currentProcessor){  
    SystemInstance root = currentProcessor.getSystemInstance();  
    EList boundThreads = new ForAllAObject(){  
        protected boolean suchThat(AObject obj){  
            ComponentInstance boundProcessor =  
                TimingUtil.getActualProcessorBinding((ComponentInst  
                    ance)obj);  
            return (boundProcessor == currentProcessor);  
        }  
    }.processOrderComponentInstance(root,  
        ComponentInstance.PROCESSOR_ORDER);
```

Final makes currentProcessor accessible to redefined suchThat method

Define condition for filtering thread instances

Retrieve instance property value



Thread List Per Processor - 3

```
public void checkPriorityInversion(final
```

```
ComponentInstance root, Processor  
SystemInstance systemInstance, Processor  
currentProcessor, SystemInstance());
```

```
EList boundThreads = new ForAllAObject(){  
    protected boolean suchThat(AObject obj){  
        ComponentInstance boundProcessor =
```

```
TimingUtil.getActualProcessorInstance(obj);  
        return (boundProcessor == currentProcessor);  
    }  
};
```

```
}.processPreOrderComponentInstance(root,  
    ComponentCategory.THREAD_LITERAL);
```

Default action returns thread instances bound to processor

Filter visits only thread instances



Sort the Thread List

```
QuickSort periodSort = new QuickSort(){  
    protected int compare(Object obj1, Object obj2){  
        double a = TimingUtil.getPeriodInUS(  
            (ComponentInstance) obj1 );  
        double b = TimingUtil.getPeriodInUS(  
            (ComponentInstance) obj2 );  
        if ( a > b ) return 1;  
        if ( a == b ) return 0;  
        return -1;  
    }  
}
```

Reusable sort method

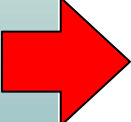
Define Period as sorting criteria

Sort the thread list by period

```
periodSort.quickSort( boundThreads );  
checkIncreasingMonotonicity( boundThreads );  
}
```



Outline

- Generation of Textual Models
- Generation of XML-based Models
- AADL Instance Meta Model
- AADL Instance Model Generation
- Instance Model Analysis: Priority Inversion
-  In-core External Model Generation
- Resource Binding & Scheduling Analysis

SAE





Traceability Between Models

- Temporary traceability to in-core non-AADL model
- From AADL model to external model
 - Propagate AADL model changes to external model
 - Use of ExternalModelAdapter to record external model object references with AADL model objects
- From external model to AADL model
 - Map results back to AADL model for persistent record as AADL properties
 - Use of UserObject reference in external model object OR adapter technique

SAE





External Model Object Adapter

- EMF provides a generated adapter factory
- OSATE provides an adapter for generic references to any Java object
- Factory associates one adapter per object being adapted & adaptation key

Class used as adaptation key

```
ExternalModelAdapterFactory.INSTANCE.adapt  
(aobject, ExternalModelAdapter.class);
```

```
adapter.getExternalModelObject()
```

```
adapter.setExternalModelObject(Object obj)
```





Persistent Traceability Between Models

- Both models must be persistent
- Persistent reference representation
 - URI for AADL models
 - URI or other XML references for external model XML documents
 - Model-specific textual reference representation
- Recording of persistent reference
 - As property value in AADL model

SAE





URI

- Standard XML reference notation
- URL part for XML document reference
 - File system path makes reference location sensitive
 - Web URL permits resolution to file system location
- Fragment part for path within XML document
 - Symbol name path (XML XPATH)
 - Requires unique names
 - Sensitive to renaming
 - Unique ID (XML/XMI)
 - Globally unique IDs required

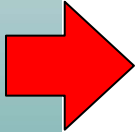
**URI fragments in XPATH format used
As cross document references in AADL models**





Outline

- Generation of Textual Models
- Generation of XML-based Models
- AADL Instance Meta Model
- AADL Instance Model Generation
- Instance Model Analysis: Priority Inversion
- In-core External Model Generation
- Resource Binding & Scheduling Analysis



SAE





The Processor Binding Plug-In

Given *system instance*, binds threads to processors

- Sets the `Actual_Processor_Binding` property

Respects thread binding constraints

- Considers `Not_Collocated`
 - Must not be on the same processor as the named threads
- Considers `Allowed_Processor_Binding`
 - Names allowed processor instances
 - Names system instances whose processors may be allowed
- Considers `Allowed_Processor_Binding_Class`
 - Names processor *classifiers* that are allowed
 - Names system *classifiers* whose processors may be allowed

Respects processor scheduling constraints

- Considers `Scheduling_Protocol`
 - Influences capacity of processor; e.g., EDF vs. RMS





Not_Collocated Example

...

```
process implementation P.i
  subcomponents
    t1: thread T.i;
    t2: thread T.i;
  end P.i;

system implementation S.packable
  subcomponents
    p1: process P.i;
    p2: process P.i;

    proc1: processor Proc.i;
    proc2: processor Proc.i;

  properties
    NOT_COLLOCATED => (reference p1.t1) applies to p2.t1;
  end S.packable;
```

First thread of process **p1** must not be bound to the same processor as the first thread of process **p2**

NOT_COLLOCATED => (reference p1.t1) applies to p2.t1;



Allowed_Processor_Binding Example

...

```
process implementation P.i
  subcomponents
    t1: thread T.i;
    t2: thread T.i;
end P.i;
```

```
system implementation Singleton.basic
  subcomponents
    proc: processor Proc.i;
end Singleton.basic;
```

```
system implementation S.Impl
  subcomponents
    p1: process P.i;
    p2: process P.i;

    s1: system Singleton.basic;
    s2: system Singleton.basic;
```

properties

```
Allowed_Processor_Binding => reference s1 applies to p1;
Allowed_Processor_Binding => reference s2 applies to p2;
end S.packable_allowed_system;
```

Threads in **p1** can only be bound
to processors in system **s1**

Threads in **p2** can only be bound
to processors in system **s2**



Allowed_Processor_Binding_Class Example

...

```
process implementation P.i
  subcomponents
    t1: thread T.i;
    t2: thread T.i;
end P.i;
```

```
system implementation hardware.mixed
  subcomponents
    proc1: processor Proc.i;
    proc2: processor Proc.x;
end hardware.mixed;
```

```
system implementation S.Impl
  subcomponents
    p1: process P.i;
    p2: process P.i;

    hw: system hardware.mixed;
```

properties

```
Allowed_Processor_Binding_Class => processor Proc.i applies to p1.t1;
Allowed_Processor_Binding_Class => processor Proc.i applies to p2.t1;
Allowed_Processor_Binding_Class => processor Proc.x applies to p1.t2;
Allowed_Processor_Binding_Class => processor Proc.x applies to p2.t2;
end S.packable_class_test;
```

The 1st thread in each process can only be bound to processors descended from `Proc.i`, i.e., `hw.proc1`.

The 2nd thread in each process can only be bound to processors descended from `Proc.x`, i.e., `hw.proc2`.



Binding via Bin-Packing

We use a bin-packing library from TimeWeaver [de Niz] to map threads to processors

- Uses `Processor` objects to model processors
 - Parameterized by scheduler, cycles per second
 - Already has link to “semantic object”

- Uses `softwareNode` objects to model threads
 - Parameterized by cycles, period, deadline
 - Already has link to “semantic object”

- Uses `Constraint` objects to describe binding constraints
 - In terms of `Processor` and `softwareNode` objects

SAE





Interfacing Strategy (1)

Create AADL-specific subclasses

- AADLProcessor extends Processor
 - Derive cycles per second from `clock_Period`
 - Create scheduler from `scheduling_Protocol`
 - **Aside:** “Borrowed” rate-monotonic scheduler code [Shelton]

- AADLThread extends softwareNode
 - Get period from `Period`
 - Get deadline from `Deadline`
 - Derive cycles from `Compute_Execution_Time`
 - **More on this coming up...**

SAE





Interfacing Strategy (2)

Maintain maps

- From processor `ComponentInstances` to `AADLProcessors`
 - `AADLProcessor` already points back to `ComponentInstance`
- From thread `ComponentInstances` to `AADLThreads`
 - `AADLThread` already points back to `ComponentInstance`

SAE





Interfacing Strategy (3)

Generate constraint objects

- From `Allowed_Processor_Binding`
- From `Allowed_Processor_Binding_Class`
- From `Not_Collocated`

SAE





Deriving Thread Cycles

Problem:

- Bin-Packing needs thread execution time in ***processor cycles***
- Standard AADL properties provide execution time in ***seconds***
 - The upper bound of `Compute_Execution_Time`

Solution:

- Property `SEI::Reference_Proc`
 - Points to processor used to generate `Compute_Execution_Time`
 - Use processor's `Clock_Period`
- Property `SEI::Reference_Clock_Period`
 - Explicit clock period for thread's timing
- (Assumes alternative processors have the same architecture)



Example: Creating AADLProcessors

```
private AssignmentResult binPackSystem(SystemInstance root) { . . .
    final Map procToHardware = new HashMap();

    // Add procs
    final ForAllAObject addProcessors = new ForAllAObject() {
        public void process(EObject obj) {
            ComponentInstance ci = (ComponentInstance) obj;
            final Processor proc = AADLProcessor.createInstance(ci);
            siteArchitecture.addSiteGuest(proc, ci.site);
            problem.hardwareGraph.add(proc);
            // add reverse mapping
            procToHardware.put(ci, proc);
        }
    };
    addProcessors.setReporter(getMarkerReporter());
    addProcessors.processPreOrderComponentInstance(root,
        ComponentCategory.PROCESSOR_LITERAL);

    . . .
}
```

Create AADLProcessor
from ComponentInstance



Example: Creating AADLProcessors

```
private AssignmentResult binPackSystem(SystemInstance root) { . . .  
    final Map procToHardware = new HashMap();  
  
    // Add procs  
    final ForAllAObject addProcessors =  
        public void process(EObject obj) {  
            ComponentInstance ci = (ComponentInstance) obj;  
            final Processor proc = AADLProcessor.createInstance(ci);  
            siteArchitecture.addSiteGuest(proc, theSite);  
            problem.hardwareGraph.add(proc);  
            // add reverse mapping  
            procToHardware.put(ci, proc);  
        }  
    };  
    addProcessors.setReporter(getMarkerReporter());  
    addProcessors.processPreOrder(ComponentInstance(root,  
        ComponentCategory.PROCESS_LITERAL);  
  
    . . .  
}
```

Add the AADLProcessor to
the bin-packing problem

Add to ComponentInstance
to AADLProcessor map



Example: Creating AADLProcessors

```
private AssignmentResult binPackSystem(SystemInstance root) { . . .
    final Map procToHardware = new HashMap();

    // Add procs
    final ForAllAObject addProcessors = new ForAllAObject() {
        public void process(EObject obj) {
            ComponentInstance ci = (ComponentInstance) obj;
            final Processor proc = AADLProcessor.createInstance(ci);
            siteArchitecture.addSiteGuest(proc, theSite);
            problem.hardwareGraph.add(proc);
            // add reverse mapping
            procToHardware.put(ci, proc);
        }
    };
    addProcessors.setReporter(getMarkerReporter());
    addProcessors.processPreOrderComponentInstance(root,
        ComponentCategory.PROCESSOR_LITERAL);
    . . .
}
```

Only process those components
whose category is Processor



Example: Creating AADLThreads

```
private AssignmentResult binPackSystem(SystemInstance root) { . . .
    final Map threadToSoftwareNode = new HashMap();

    final ForAllAObject addThreads = new ForAllAObject() {
        public void process(EObject obj) {
            ComponentInstance ci = (ComponentInstance) obj;
            final SoftwareNode thread = AADLThread.createInstance(ci);
            problem.softwareGraph.add(thread);
            threadToSoftwareNode.put(ci, thread);

            final List disjunctFrom =
                ci.getPropertyValueList(PredeclaredPr NOT_COLLOCATED);
            if (disjunctFrom != null) {
                final Set disjunctSet = new HashSet();
                for (final Instance
                    Component
                    (ComponentInstance) iv.getReferencedInstanceObject();
                    if (refCI.getCategory() == ComponentCategory.THREAD_LITERAL) {
                        disjunctSet.add(refCI);
                    }
                }
                if (!disjunctSet.isEmpty()) notCollocated.put(ci, disjunctSet);
            }
        }
    };
    addThreads.setReporter(getMarkerReporter());
    addThreads.processPreOrderComponentInstance(root,
        ComponentCategory.THREAD_LITERAL);
    . . .
}
```

**Create AADLThread from ComponentInstance,
add to problem, and update map.**



Example: Creating AADLThreads

```
private AssignmentResult binPackSystem(SystemInstance root) { . . .  
    final Map threadToSoftwareNode = new HashMap();
```

```
final  
pub
```

**Get the collocation constraints for the thread.
Set disjunctSet contains ComponentInstances.**

```
    final SoftwareNode node = AADLThread.createInstance(ci);  
    problem.softwareNode.add(node, thread);  
    threadToSoftwareNode.put(i, thread);
```

```
    final List disjunctFrom =  
        ci.getPropertyValueList(PredeclaredProperties.NOT_COLLOCATED);  
    if (disjunctFrom != null) {  
        final Set disjunctSet = new HashSet();  
        for (final Iterator i = disjunctFrom.iterator(); i.hasNext();) {  
            InstanceReferenceValue rv = (InstanceReferenceValue) i.next();  
            ComponentInstance refCI =  
                (ComponentInstance) rv.getReferencedInstanceObject();  
            if (refCI.getCategory() == ComponentCategory.THREAD_LITERAL) {  
                disjunctSet.add(refCI);  
            }  
        }  
        if (!disjunctSet.isEmpty()) notCollocated.put(ci, disjunctSet);  
    }  
}
```

```
};  
addThreads.setReporter(getMarkerReporter());  
addThreads.processPreOrderComponentInstance(root,  
    ComponentCategory.THREAD_LITERAL);  
. . .  
}
```

SAE





Example: Collocation Constraints

```
private AssignmentResult binPackSystem(SystemInstance root) {
```

...

For each thread that has a `Not_Collocated` property

```
    for (Iterator constrained = notCollocated.keySet().iterator();
         constrained.hasNext();) {
        final ComponentInstance ci = (ComponentInstance) constrained.next();
        final SoftwareNode sn = (SoftwareNode) threadToSoftwareNode.get(ci);
        final Set disjunctFrom = (Set) notCollocated.get(ci);
        for(Iterator dfIter = disjunctFrom.iterator(); dfIter.hasNext();) {
            final ComponentInstance ci2 = (ComponentInstance) dfIter.next();
            final SoftwareNode[] disjunction =
                new SoftwareNode[] { sn,
                                     (SoftwareNode) threadToSoftwareNode.get(ci2) };
            problem.addConstraint(new Disjoint(disjunction));
        }
    }
    ...
}
```

Lookup AADLThread



Example: Retrieving the Bindings

Build a map from thread ComponentInstance to processor ComponentInstance objects

Given set of AADLProcessors—each has set of SoftwareNodes

```
private Map getThreadBindings(final Set hardware) {
    final Map threadsToProc = new HashMap();
    for (Iterator iter = hardware.iterator(); iter.hasNext();) {
        HardwareNode n = (HardwareNode) iter.next();
        for (Iterator taskSet = n.getTaskSet().iterator();
            taskSet.hasNext();) {
            SoftwareNode m = (SoftwareNode) taskSet.next();
            if (m instanceof CompositeSoftNode) {
                final Set set = ((CompositeSoftNode) m).getBasicComponents();
                for (Iterator software = set.iterator(); software.hasNext(); ) {
                    final SoftwareNode sn = (SoftwareNode) software.next();
                    threadsToProc.put(
                        sn.getSemanticObject(), n.getSemanticObject());
                }
            } else {
                threadsToProc.put(m.getSemanticObject(), n.getSemanticObject());
            }
        }
    }
    return threadsToProc;
}
```

Use “semantic object” link to get back the ComponentInstance objects



Binding Plug-in Output

Bin Packing Results

Bin packing successful.

Thread bindings

Thread	Processor
p1.t2	proc2
p2.t1	proc2
p1.t1	proc1
p2.t2	proc2

Processor Capacities

Processor	% Available
proc2	55%
proc1	85%

Paste into "S.packable"

```
properties
Actual_Processor_Binding => reference proc2 applies to p1.t2;
Actual_Processor_Binding => reference proc2 applies to p2.t1;
Actual_Processor_Binding => reference proc1 applies to p1.t1;
Actual_Processor_Binding => reference proc2 applies to p2.t2;
```

Buttons: Okay (Do nothing), **Set Instance Properties**, Set Declarative Properties

Callout box text: Can set the **Actual_Processor_Binding** properties on either the instance or declarative model





Binding Plug-In Status

- A work in progress
 - Not yet included in OSATE distribution
- To do:
 - Consider data rates on connections
 - Bin-pack across buses

SAE





Summary

- Traversal vs. content-based model processing
- Leverage EMF for XML-based model generation
- Compact modal instance models
- Instance model based analysis
- Interfacing with existing Java-based analysis methods

SAE





Short Demo of OSATE SDK

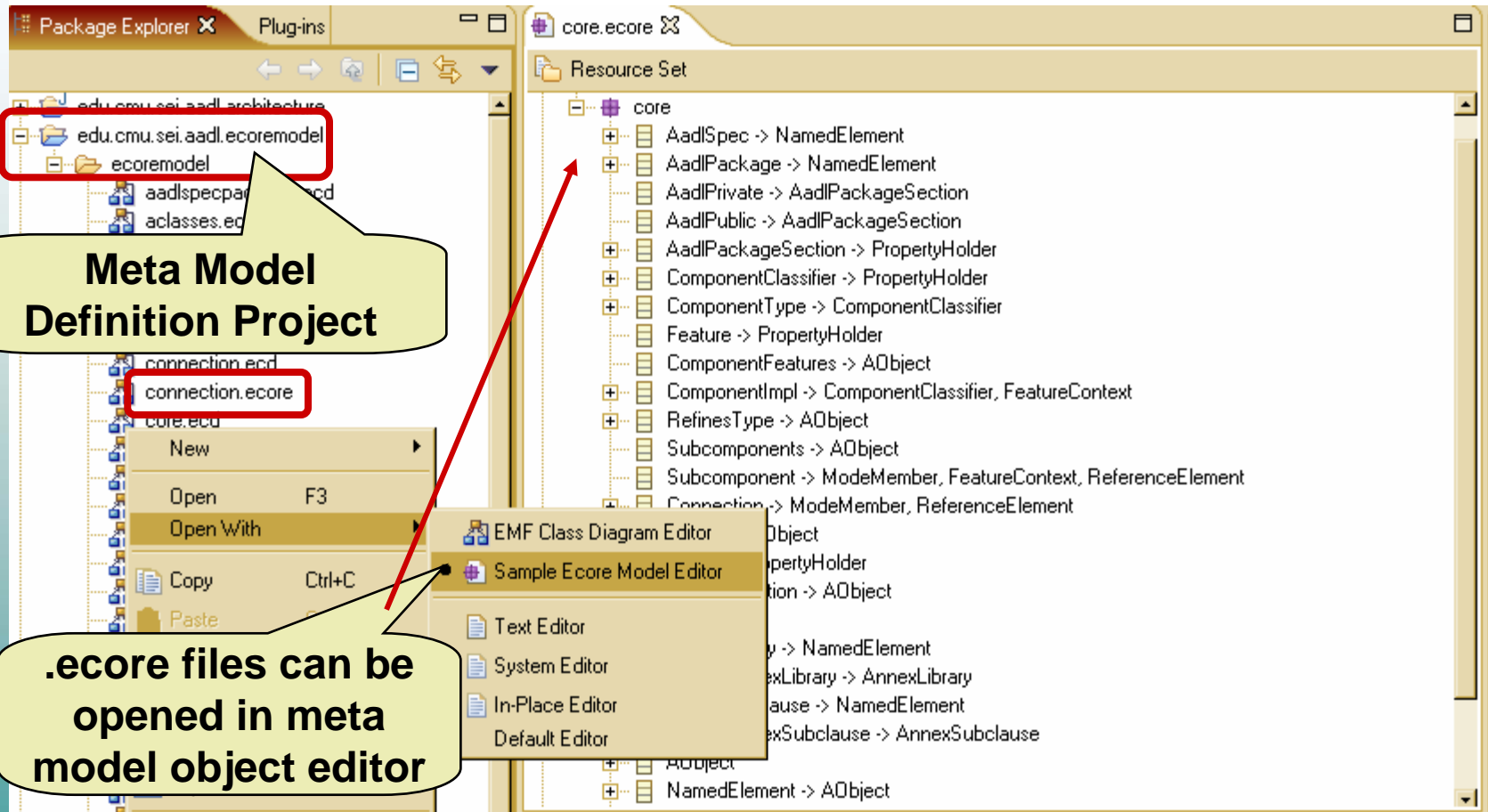
- Use of the OSATE SDK
 - Available at www.aadl.info
- Download the OSATE SDK with Omondo for graphical viewing of the Meta model
 - Unzip into an Eclipse/OSATE installation folder
- Download SDKProject
 - This contains several plug-ins and the AADL meta model as Eclipse projects
 - Unzip into a folder for Eclipse workspaces
- Start OSATE/Eclipse (found in the OSATE folder)
 - Select to SDKProject folder as workspace
- For further details see the OSATE User manual

SAE





OSATE SDK: Editing Meta-Model (1)



Meta Model Definition Project

.ecore files can be opened in meta model object editor





OSATE SDK: Editing Meta-Model (2)

.ecd files open in graphical editor (Omondo Plug-In)

The graphical editor displays the following UML diagram structure:

- ComponentClassifier** (Abstract Class):
 - Generalized by **ComponentType** (Solid arrow)
 - Generalized by **AnnexSubclause** (Solid arrow)
 - Generalized by **RefinesType** (Dashed arrow)
 - Associated with **ComponentClassifier** (Self-association, solid arrow)
- AnnexSubclause** (Class):
 - Generalized by **DefaultAnnexSubclause** (Solid arrow)
 - Generalized by **Subcomponents** (Solid arrow)
 - Associated with **ComponentClassifier** (Solid arrow, multiplicity: *)
- ComponentType** (Class):
 - Implements **ComponentClassifier** (Dashed arrow)
 - Associated with **flow::FlowSpecs** (Solid arrow, multiplicity: 0..1)
 - Associated with **ComponentFeatures** (Solid arrow, multiplicity: A)
- flow::FlowSpecs** (Class):
 - Associated with **ComponentType** (Solid arrow, multiplicity: 0..1)
- ComponentFeatures** (Class):
 - Associated with **ComponentType** (Solid arrow, multiplicity: A)
- Subcomponents** (Class):
 - Associated with **AnnexSubclause** (Solid arrow)
 - Refines **RefinesType** (Dashed arrow, multiplicity: 0..1)
- RefinesType** (Class):
 - Refines **Subcomponents** (Dashed arrow, multiplicity: 0..1)

Abstract contains time constraints realized in component categories to enforce contain time constraints.





OSATE SDK: OSATE Plug-In Projects

Security Level Plug-in from session 2

```
/*
 *
 * <copyright>
 * Copyright 2004 by Carnegie Mellon University, all rights reserved.
 *
 * OSATE Tool Environment (OSATE) is a registered trademark of Carnegie Mellon University.
 * See http://www.cmu.edu/legal/cpl-v10.html.
 *
 * ANY INFORMATION, MATERIALS, SERVICES, INTELLECTUAL PROPERTY OR OTHER RIGHTS
 * CARNEGIE MELLON UNIVERSITY PURSUANT TO THIS LICENSE (HEREINAFTER THE "LICENSE")
 * CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, INCLUDING BUT NOT
 * LIMITED TO, WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR WARRANTY OF
 * NONINFRINGEMENT, OR ERROR-FREE OPERATION. CARNEGIE MELLON UNIVERSITY MAKES NO
 * CONSEQUENTIAL DAMAGES, SUCH AS LOSS OF PROFITS OR INABILITY TO USE SOFTWARE,
 * REGARDLESS OF WHETHER SUCH PARTY WAS AWARE OF THE POSSIBILITY OF SUCH DAMAGES.
 * MAKE ANY WARRANTY ON BEHALF OF CARNEGIE MELLON UNIVERSITY. THE USER WILL BE
 * APPLICATION OF OR THE RESULTS TO BE OBTAINED WITH THE DELIVERY OF THESE
 *
 * Licensee hereby agrees to defend, indemnify, and hold harmless Carnegie Mellon
 * employees, and agents from all claims or demands made against Carnegie Mellon
 * attorney's fees) arising out of, or relating to Licensee's use of or
 * misuse of or negligent conduct or willful misconduct regarding OSATE.
```





OSATE SDK: Accessing OSATE Plug-Ins

The screenshot shows the Eclipse IDE interface. On the left, the Package Explorer displays a tree of plug-ins under the 'edu.cmu.sei.aadl' package. A red box highlights the 'Plug-in Dependencies' folder, which lists various jar files. A callout bubble points to this folder with the text: "Plug-in Dependencies" links to the jar files of the needed plug-ins. Another callout bubble points to 'aadlmodel.jar' in the list with the text: E.g., aadlmodel.jar is the AADL meta model plug-in. On the right, the Editor window shows the content of 'ForAllADObject.class', which is a license agreement. The bottom of the IDE shows the Properties view for the selected file, displaying metadata such as 'derived', 'editable', 'last modified', and 'linked'.

Property	Value
Info	
derived	false
editable	true
last modified	12/15/04 10:45 PM
linked	false





OSATE SDK: OSATE Source Code

The screenshot shows the Eclipse IDE interface. On the left, the Package Explorer displays a tree view of project packages. A red box highlights the 'edu.cmu.sei.aadl.model.core.impl' package, with 'AadlSpecImpl.class' selected. A yellow callout bubble points to this selection with the text 'Can view the plug-in contents'. On the right, the editor window shows the source code for 'AadlSpecImpl.class'. A red arrow points from a yellow callout bubble at the bottom to the code, with the text 'Opening the "class" goes to the source because we have packaged the source code with the OSATE Plug-ins'. The code includes a 'getThreadType()' method and an 'addThreadType()' method, both with Javadoc comments.

```
public EList getThreadType() {  
    return ((FeatureMap) getContents()).list(CorePackage.e  
}  
  
/**  
 * <!-- begin-user-doc -->  
 * phf: add element to multiplicity EList  
 * <!-- end-user-doc -->  
 * @generated  
 */  
public void addThreadType(ThreadType newThreadType) {  
    if (newThreadType != null) {  
        this.getThreadType().add(newThreadType);  
    }  
}  
  
/**  
 * <!-- begin-user-doc -->  
 * <!-- end-user-doc -->  
 * @generated  
 */
```

Can view the
plug-in contents

Opening the "class" goes to the
source because we have packaged
the source code with the OSATE
Plug-ins