

# Plug-in Development for the Open Source AADL Tool Environment

## Part 4: OSATE Infrastructure & Language Extensions

Peter Feiler / Aaron Greenhouse/ Lutz Wrage  
Software Engineering Institute  
(phf / aarong/ lwrage)@sei.cmu.edu  
412-268- (7790 / 6464 / 7771 )

### 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

- Persistence
- Modal system models
- Sublanguage extensions

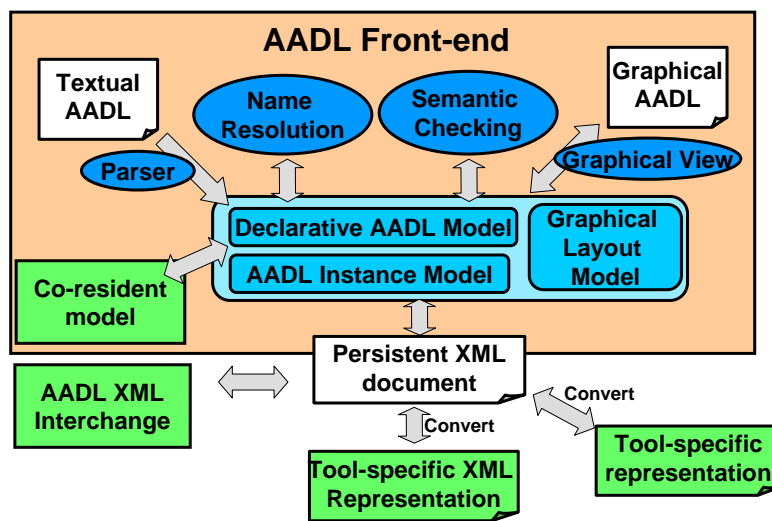


# Outline

- ➔ Persistent AADL models
  - Multi-file Support
  - Modal system models
  - Sublanguage extensions



# Tool Interoperability





## Persistence in OSATE

- Eclipse Files & IResources
- Eclipse Modeling Framework XML, resources, resource sets
- OSATE resource manager



## Persistence in Eclipse

- Files & IResources
- IResource-based navigation
- IResources with temporary and persistent markers & attributes
- Synchronization between files and IResources
- IResource delta & change propagation



# Eclipse Resource Navigator

**Eclipse Navigator**

**File system**

Address: SYSTEM (C:)

My Computer

3½ Floppy (A:)

SYSTEM (C:)

amaranth

Documents and Settings

downloads

Eclipse3

eclipse213

EclipseProjects

bug

graphics

Models

osate

runtime-workspace

test

FILES

gme

Java

java1.4.2.docs

jdk1.3.1\_08

MATLAB6p5

nosee

otherlocation

Program Files

svm

© 2004 by Carnegie Mellon University

www.aadl.info

7

# Resource Markers

**Marker decorators**

**Marker locations**

**Filtered marker viewing**

```

1 * <copyright>
2 * </copyright>
3 *
4 *
5 * $Id: ErrorAnnexResourceFactoryImpl.java,v 1.1 2004/12/02 05:25:04 xjared
6 */
7 package edu.cmu.sei.errorAnnex.util;
8
9 import org.eclipse.emf.common.util.URI;
10
11
12
13
14
15
16 * <!-- begin-user-doc -->
17 * The <b>Resource Factory</b> associated with the package.
18 *
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

Description	Resource	In Folder	Location
The import org.eclipse.emf.ecore.xml cannot be resolved	ErrorAnnexResourceFa...	edu.cmu.sei.aadl.errorAnnex[org.eclipse.cmu]se...	line 13
XMLResourceFactoryImpl cannot be resolved or is not a valid superclass	ErrorAnnexResourceFa...	edu.cmu.sei.aadl.errorAnnex[org.eclipse.cmu]se...	line 22
Type mismatch: cannot convert from ErrorAnnexResourceImpl to Resource	ErrorAnnexResourceFa...	edu.cmu.sei.aadl.errorAnnex[org.eclipse.cmu]se...	line 40
The import org.eclipse.emf.ecore.xml cannot be resolved	ErrorAnnexResourceFa...	edu.cmu.sei.aadl.errorAnnex[org.eclipse.cmu]se...	line 11
ResourceImpl cannot be resolved or is not a valid superclass	ErrorAnnexResourceFa...	edu.cmu.sei.aadl.errorAnnex[org.eclipse.cmu]se...	line 20
urn:rhact:01 is undefined	ErrorAnnexResourceFa...	edu.cmu.sei.aadl.errorAnnex[org.eclipse.cmu]se...	line 79

© 2004 by Carnegie Mellon University

www.aadl.info

8



## Persistence in Eclipse

- Files & IResources
- IResource-based navigation
- IResources with temporary and persistent markers & attributes
- Synchronization between files and IResources
  - On startup
  - Automatic or manual for active Eclipse
- IResource delta & change propagation
  - Set of changed IResources
  - Propagation through builder



## Persistence in EMF

- Resources & XML documents
  - Meta model-based content hierarchy
  - Multiple content structures per Resource are possible
- Meta model based XML processors
  - Generated methods
  - Meta model interpretation
- Tailorable generated resource factories
- Parameterization of XML processors
- Resource sets & XML cross reference resolution



# Resource Factory Generation

The screenshot shows an IDE with a Package Explorer on the left and a Properties window on the right. The Package Explorer shows a project structure with packages like 'edu.cmu.sei.aadl.model.component' and 'edu.cmu.sei.aadl.model.core'. The Properties window is open for 'CoreResourceImpl.java' and shows the 'Resource Type' property set to 'Basic'. Other properties include 'Base Package' (edu.cmu.sei.aadl.model), 'Prefix' (Core), and 'Package' (core). A callout bubble points to the 'Resource Type' property.

Ecore based, XML based, XMI based



# Generated Resource Factory

The screenshot shows the source code of 'CoreResourceFactoryImpl.java'. The code includes a class declaration that extends 'ResourceFactoryImpl' and a constructor 'public CoreResourceFactoryImpl()' that calls 'super()'. A callout bubble points to the constructor method. The code also includes a 'createResource' method that returns a new instance of 'CoreResourceImpl'.

Default implementation





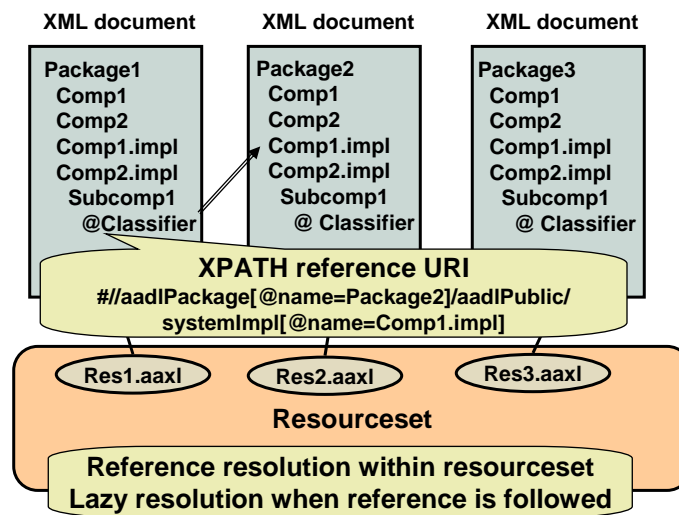
## Parameterization of XML Processor

- Reference representation
  - XPath based URI
  - Method redefinition in AObject
    - String eURIFragmentSegment
    - EObject eObjectForURIFragmentSegment
- Generator parameterization
  - Change of tag names

For more details see EMF book



## Cross Document References



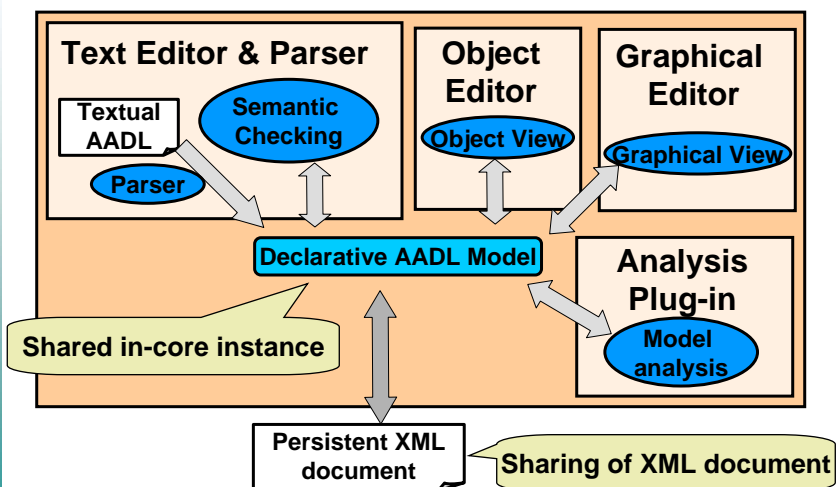


## OSATE Resource Manager

- Assures single resource set per project
  - across plug-ins
  - Text editor, parser, AADL object editor, graphical editor
- Assures single resource per XML document
  - getResource, getEmptyResource methods
  - Hides idiosyncrasies of EMF resource API
- IResource <-> Resource conversion
  - Selection of IResource in Eclipse navigator
  - Markers on IResources
  - IResource & IPath
  - Resources & URIs
  - Aadl model objects & Resources



## AADL Models as Resources





## Synchronization of Models

- Batch processing
  - Change notification in-core and resource-based
  - Reprocess whole model
  - Coordination of processing activities
- Incremental update
  - In-core listener notification on individual AADL model objects
  - Cross-reference methods in EcoreUtil
  - User-defined reference-based propagation



## References & Proxies

- Saving of Aadl models
  - Root AADL model object as Resource content
  - Save method on Resource
  - Cross references stored as URI
- Loading of Aadl models
  - getResource method for lookup by name
  - Res.eContent().get(0) to get root Aadl model object
  - Implicit loading of XML document with lazy reference evaluation
- References & proxies
  - Proxy representation of references
  - Automatic proxy resolution
  - Unresolved references remain proxies
  - Res.unload() results in conversion to reference proxies






## OSATE Resource Manager API

- IPath getOsatePath(String)
- IResource createFolderIResource(String)
- Resource getResource(IResource)
- Resource getResource(URI)
- Resource getEmptyResource(URI)
- IResource convertToIResource(Resource)
- ResourceSet getResourceSet()
  
- EObject (EObject) methods
  - elsProxy()
  - eResource()

SAE



## Outline

- Persistent AADL models
-  Multi-file Support
- Modal system models
- Sublanguage extensions

Initial implementation in test

SAE





## Multi-file Support

- OSATE 0.4
  - Single textual AADL
  - Single declarative AADL XML & separate instance model XML resources
- Upcoming OSATE release
  - AADL text multi-file support
  - AADL XML multi-file support
  - Search paths

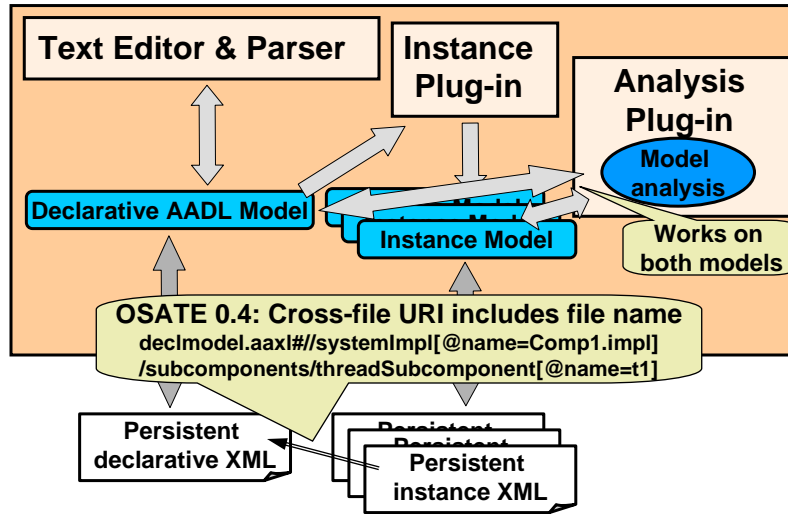


## OSATE 0.4 File Support

- Textual AADL
  - Single file contains AADL specification with all packages & property sets
  - Predeclared property sets are auto-inserted by parser
  - Inclusion of each predeclared property set in AADL text file overrides auto-insert of property set
- AADL XML
  - Parsing into single file declarative AADL model
  - Includes predeclared property sets
  - Each AADL instance model as separate XML file
    - File name derived from instantiated system implementation



## Declarative & Instance Models

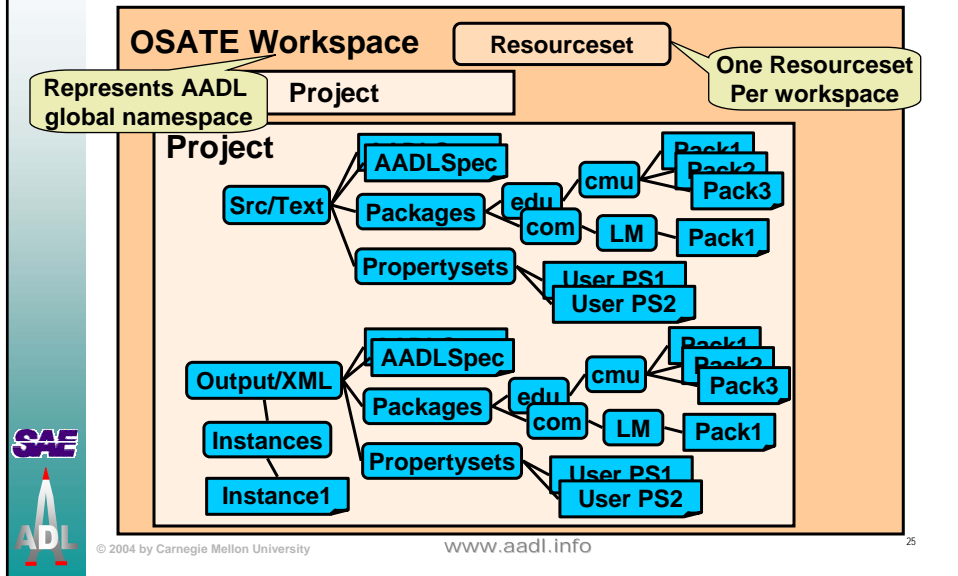


## OSATE Multi-File Support

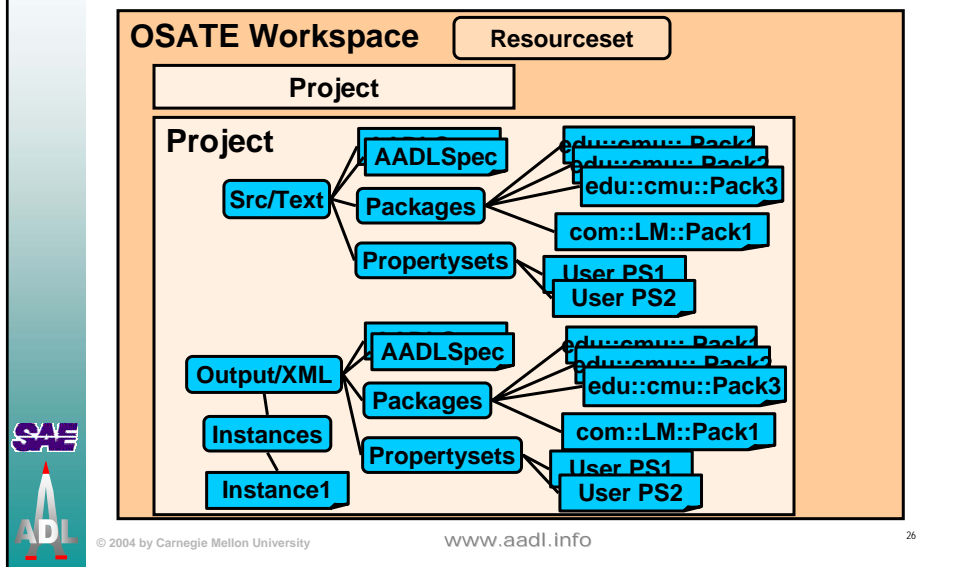
- Textual AADL in OSATE
  - Separate text file per package
  - Separate text file per property set
  - Separate text file for AADL specification non-package/non-propertyset declarations
  - Import of text files with multiple packages/propertysets
- AADL XML in OSATE
  - Separate XML file per package
  - Separate XML file per property set
  - Separate XML file for AADL specification non-package/non-propertyset declarations
  - Separate XML file for each instance model



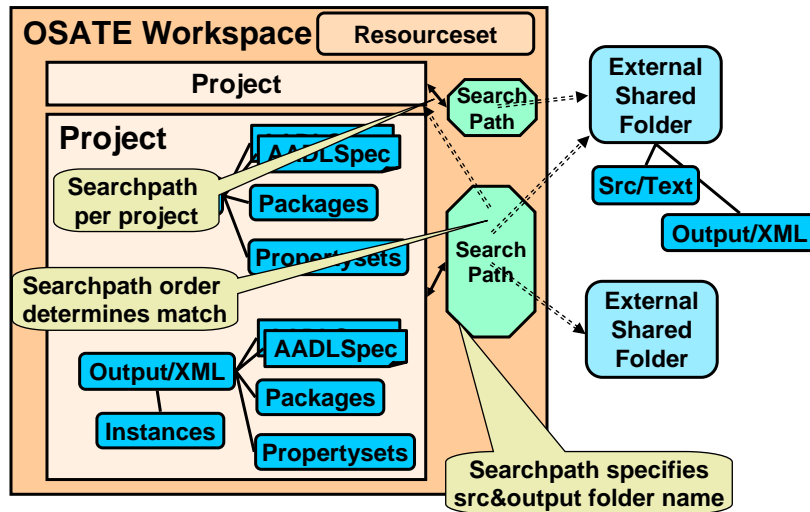
## OSATE Resource Organization



## Alternative OSATE Package View



## OSATE Searchpath



## Future OSATE Project Refinement

- Library project
  - Shared read-only collection of packages/propertysets
  - Update notification from repository
- Source project
  - Modifiable collection of packages/propertysets
  - Modification coordination via version control
- Local project
  - AADL specification with anonymous namespace only
  - Not accessible by packages
- Instance project
  - Work with system instance
  - Instance model with variations
    - Analysis results
    - Derived/transformed instance models





## Outline

- Persistent AADL models
- Multi-file Support
- Modal system models
- Sublanguage extensions



Initial implementation available

SAE



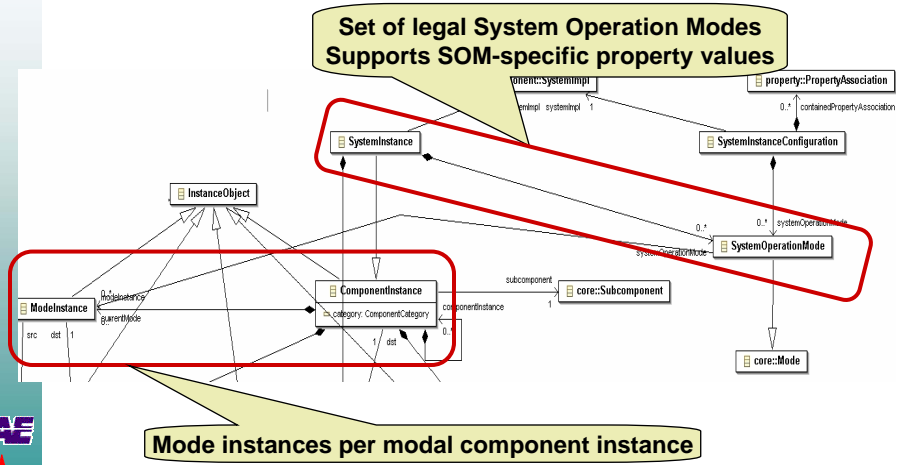
## Instance Model Implementation Status

- Instantiate from system implementation
- Semantic connections
  - Port connections implemented
  - Port group unfolding to be done
  - Access connections implemented
  - Mode transition connections to be done
- Property values
  - Contained property associations in instance model implemented
  - Cached property associations implemented
  - Modal contained property values in test
  - Modal instance property value lookup in test

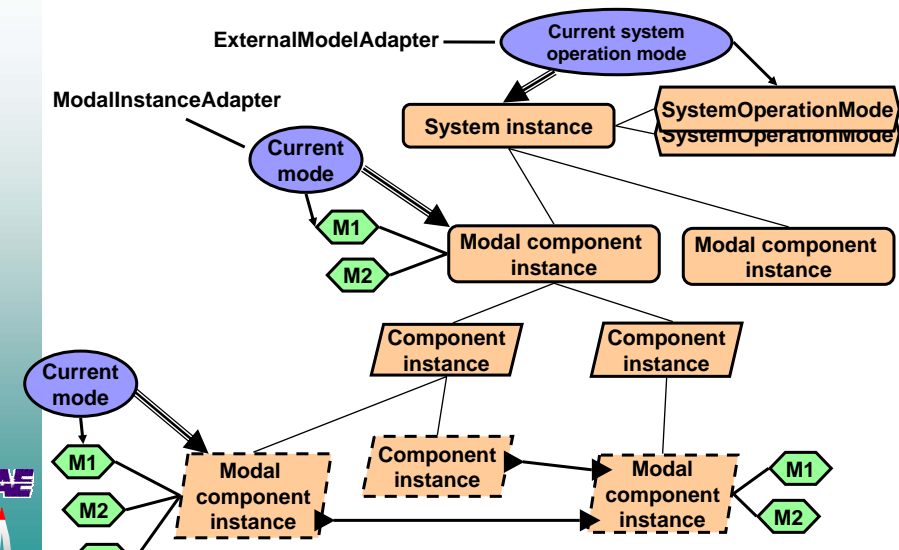
SAE



# Modal & Configurable System Instances



# Modal Instance Model





## System Operation Modes

- Class SystemOperationMode
  - List of mode instances
- Class SystemInstance
  - List of SystemOperationMode
  - Current mode via adapter
- Class SOMIterator
  - SOMIterator(SystemInstance root)
    - Iterate of the system operation modes in root
  - nextSOM()
    - Returns the next system operation mode
    - Sets the modal adapters to the current system operation mode
      - [See next 2 slides]
  - hasNext()



## Modal Instance Adapter

- EMF provides a generated adapter factory
- OSATE provides an adapter for modal instances
- Factory associates one adapter per object being adapted & adaptation key

```
ModalInstanceAdapterFactory.INSTANCE.adapt  
(aobject, aobject);
```

```
adapter.getCurrentMode()  
adapter.setCurrentMode(ModeInstance mi)
```





## System Instance Adapter

- We use a second adapter to associate the current system operation mode with a system instance.
- Retrieve adapter:

```
ExternalModelAdapter adapter =  
    (ExternalModelAdapter) ExternalModelAdapterFactory.INSTANCE.adapt(  
        systemInstanceObj, SystemInstance.CURRENT_SOM_ADAPTER_KEY);
```

- Get the current system operation mode:

```
SystemOperationMode som =  
    (SystemOperationMode) adapter.getExternalModelObject();
```



## Traversal Methods Revisited

- Class AObject
  - getChildren(): eContent()
- Class ComponentInstance
  - getChildren(): subset of children active in current mode
  - Uses ModalInstanceAdapter
- Class SystemInstance
  - setSystemOperationMode(SystemOperationMode)
  - clearSystemOperationMode()
- Class ForAllAObject
  - Traversal methods call on getChildren







# Modal Property Values

A component's property value may depend on the modes of the components in the model.

- In a declarative model
  - The modes of the component itself.
  - As a subcomponent, the modes of the containing component.
- In an instance model
  - The modes of any arbitrary containing ancestor.
    - Due to contained property associations.



# Modal Property Values: Examples

```

system implementation Inner.Impl
modes
  InnerModel: initial mode;
  InnerMode2: mode;
properties
  PS::x => 1 in modes (InnerModel);
  PS::x => 2 in modes (InnerMode2);
end Inner.Impl;

```

**Declarative model:** Property values of **component** affected by modes of component itself.

```

system implementation Outer.Impl
subcomponents
  sub: system Inner.Impl {
    PS::y => 3 in modes (OuterModel1);
    PS::y => 4 in modes (OuterMode2); };
modes
  OuterModel1: initial mode;
  OuterMode2: mode;
end Outer.Impl;

```

**Declarative model:** Property values of **subcomponent** affected by modes of containing component.

```

system Main.Impl
subcomponents
  sub: system S.I;
  ...
modes:
  M1: initial mode;
  M2: mode;
properties
  PS::z => 5 applies to sub.a.b.c.d in modes (M1);
end Main.Impl;

```

**Instance model:** Property values of **component instance** affected by modes of arbitrary ancestor component.





# Modal Property Value Complexities

A property value may also be modal because

- A component may exist in certain modes only.
  - Thus, a property value may **not exist** in some modes.
    - This is distinct from a property value being **not present**, or unset.
- It references another property whose value is modal.



# Modal Subcomponents Example

```
system implementation ModalSubcomponents.Impl
subcomponents
```

```
sub1: system S1.Impl {
  PS::x => 0;
} in modes (M1, M2);
```

```
sub2: system S2.Impl {
  PS::x => 1 in modes (M1);
} in modes (M1, M3);
```

```
sub3: system S3.Impl {
  PS::x => 2 in modes (M2);
  PS::x => 3 in modes (M3);
} in modes (M2, M3);
```

```
modes
M1: initial mode;
M2: mode;
M3: mode;
end ModalSubcomponents.Impl;
```

Assuming PS::x doesn't have a default value.

Mode	Value of x
M1	0
M2	0
M3	Nonexistent

Mode	Value of x
M1	1
M2	Nonexistent
M3	Not Present

Mode	Value of x
M1	Nonexistent
M2	2
M3	3



## Modal Property Reference Example

```
property set PS is
  bool1: aadlboolean => true applies to (system);
  bool2: aadlboolean => false applies to (system);
  bool3: aadlboolean applies to (system);
end PS;
```

```
system Example
properties
  PS::bool3 => value(PS::bool1)
              and value(PS::bool2);
end Example;
```

Value of `bool3` for component type `Example` is non-modal: it's always `false`

```
system implementation Example.Impl
modes
  M1: initial mode;
  M2: mode;
properties
  PS::bool1 => false in modes (M1);
  PS::bool2 => true in modes (M2);
end Example.Impl;
```

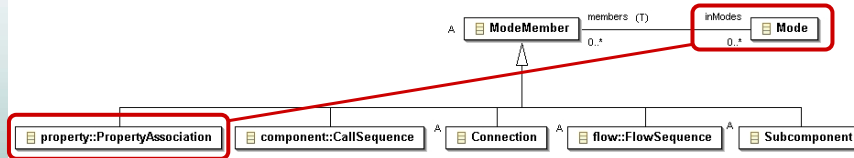
Value of `bool3` for component implementation `Example.Impl` is now modal:

Mode	bool1	bool2	bool3
M1	False	False	False
M2	True	True	True

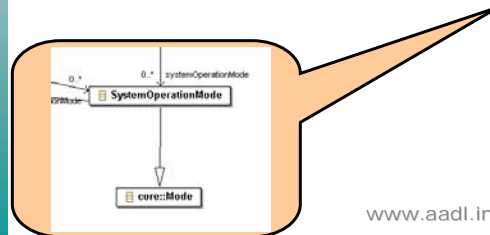


## Modal Property Associations in Meta Model

PropertyAssociation objects refer to modes via the `inModes` association.



- In declarative model: List of `Mode` objects
- In instance model: List of `SystemOperationMode` objects





## Review: 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 follows...*

SAE



## Interface ModalPropertyValue (1)

- Encapsulates mode-dependent property values
  - (In package `edu.cmu.sei.aadl.model.properties`.)
- Methods
  - `boolean isModal()`
    - Whether the value is modal.
  - `AadlPropertyValue getValue()`
    - Gets the value when it is non-modal.
    - Throws `ModeNotSpecifiedException` if a mode is required.
  - `Set getModeContexts()`
    - Get the `ModeContext` objects whose modes affect the value.
      - The modes visible in the context are returned by `ModeContext.getModes()`.
  - `ModeContext[] getModeContextsAsArray()`
    - Same as above, but as an array.

SAE





## Interface ModalPropertyValue (2)

- Methods (continued)
  - `AadlPropertyValue getValue(Map modes)`
    - Get the value given specific mode bindings.
    - Parameter `modes` maps `ModeContexts` to `Modes`.
      - The `ModeContext` should be from `getModeContexts()`.
      - The `Mode` should be from the `ModeContext`'s `modes`.
    - Throws `ModeNotSpecifiedException` if a needed `ModeContext` is not bound to a `Mode`.
  - Collection `getAllModeBindings()`
    - Get all the `Maps` that could be used with `getValue()`.
  - Collection `getAllValues()`
    - Get all the values the property could have, as `ReflectiveAadlPropertyValue` objects.



## Interface ModeContext

- Represents objects that contain mode declarations.
  - (In package `edu.cmu.sei.aadl.model.properties`.)
- Two implementations:
  - `DeclarativeModeContext` for declarative models.
    - Refers to `Modes` from a `ComponentImpl` object.
  - `InstanceModeContext` for instance modes.
    - Refers to `SystemOperationModes` from a `SystemInstance` object.
- Methods
  - `String getName()`
    - Get the name of the underlying model object
  - `List getModes()`
    - Get the modes available in the context.



## Interface Aa1PropertyValue (1)

- Encapsulates the property value.
  - In particular, distinguishes *nonexistence* from *not present*.
  - (In package `edu.cmu.sei.aadl.model.properties`.)
- Methods
  - `boolean exists()`
    - Whether the property value exists.
    - If `false`, the other methods are irrelevant.
  - `boolean isNotPresent()`
    - Whether the property value is not present.
  - `boolean isList()`
    - Whether the property value is a list.
    - `False if !exists() || isNotPresent()`.



## Interface Aa1PropertyValue (2)

- Methods (continued)
  - `PropertyValue getScalarValue()`
    - The property value if `!isList()`.
    - Throws `UnsupportedOperationException` if `isList()`.
  - `List getValue()`
    - The property value as a `List` of `PropertyValue` objects.
      - List of length 1 if `!isList()`.





## Interface ReflectiveAadlPropertyValue

- Extends AadlPropertyValue
  - (In package edu.cmu.sei.aadl.model.properties.)
- Adds method
  - Map getModeBinding()
    - Returns the mode binding this value is for.
    - Same as for ModalPropertyValue.getValue(Map).



## Property Lookup Example

```
final ModalPropertyValue dispatchMPV =  
    ph.getPropertyValue(PredeclaredProperties.DISPATCH_PROTOCOL);  
  
for (Iterator values = dispatchMPV.getAllValues().iterator();  
     values.hasNext();) {  
  
    final ReflectiveAadlPropertyValue dispatchPV =  
        (ReflectiveAadlPropertyValue) values.next();  
  
    boolean bad = !dispatchPV.exists() || dispatchPV.isNotPresent();  
    if (!bad) {  
        final EnumValue dispatch = (EnumValue) dispatchPV.getScalarValue();  
        bad = !PredeclaredProperties.APERIODIC.equals(  
            dispatch.getEnumLiteral())  
            && !PredeclaredProperties.SPORADIC.equals(  
                dispatch.getEnumLiteral());  
    }  
    if (bad) {  
        reportError(ph,  
            "\"Dispatch_Protocol\" must be \"Aperiodic\" or "  
            "\"Sporadic\" when thread contains a server subprogram" +  
            convertModesToString(dispatchPV.getModeBinding());  
    }  
}
```

Code fragment checking that all values of Dispatch\_Protocol  
are Aperiodic or Sporadic





## Caveat: Property Lookup and Instance Models

- “Modeless” property lookup methods understand the modal adapters on instance models.
  - `getSimplePropertyValue(PropertyDeclaration pd)`
  - `getSimplePropertyValue(String ps, String pn)`
  - `getPropertyValueList(PropertyDeclaration pd)`
  - `getPropertyValueList(String ps, String pn)`
- Property value looked up based on the current system operation mode.

SAE



- “Need an example of modal property lookup in an instance model with the current mode set on the model. This should allow us to hide the map stuff.”

SAE





## Setting Property Values

`PropertyHolder` declares property setting methods.

- `setPropertyValue(`  
    `PropertyDefinition pd, PropertyValue pv)`
  - Set a non-list property value.
  - Throws exception if property is not applicable to the property holder, or if the property value is inappropriate.
- `setPropertyValue(PropertyDefinition pd, List pvl)`
  - Set a list property value.
  - Throws exception if property is not applicable to the property holder, or if the property value is inappropriate.
- `removePropertyAssociations(PropertyDefinition pd)`
  - Remove **all** property associations for the given property from the property holder.



## Setting Modal Property Values

- “Modeless” property setting methods understand the modal adapters on instance models.
  - `setPropertyValue(`  
    `PropertyDefinition pd, PropertyValue pv)`
  - `setPropertyValue(PropertyDefinition pd, List pvl)`
  - Property value set for current system operation mode only.
- Modal variants of property setting methods:
  - `setPropertyValue(PropertyDefinition pd,`  
    `PropertyValue pv, List modes)`
  - `setPropertyValue(PropertyDefinition pd,`  
    `List pvl, List Modes)`
  - `removePropertyAssociations(PropertyDefinition pd,`  
    `List modes)`
    - Removes associations for the given property in the given modes.





## Setting Contained Property Associations

PropertyHolder declares the methods

- `setContainedPropertyValue(PropertyDefinition pd, List appliesToPath, PropertyValue pv)`
- `setContainedPropertyValue(PropertyDefinition pd, List appliesToPath, PropertyValue pv, List modes)`
- `setContainedPropertyValue(PropertyDefinition pd, List appliesToPath, List pvl)`
- `setContainedPropertyValue(PropertyDefinition pd, List appliesToPath, List pvl, List modes)`
- `removeContainedPropertyAssociations(PropertyDefinition pd, List appliesToPath)`
- `removeContainedPropertyAssociations(PropertyDefinition pd, List appliesToPath, List modes)`
- Parameter `appliesToPath` is a list of declarative components
  - This path is not checked to see that it makes sense with respect to the features/subcomponents of current PropertyHolder.

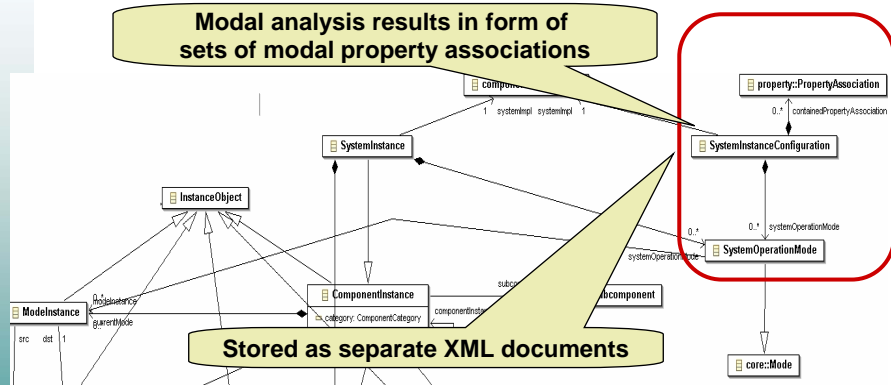


## Modal Analysis Results

- Stored as modal instance property values
- Multiple sets in same instance model
- Option: keep (modal) result property value sets in separate XML document



# Modal Analysis Result Sets



# Outline

- Persistent AADL models
- Multi-file Support
- Modal system models
- ➔ Sublanguage extensions

Not yet prototyped





## AADL Language Extensions

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



## An Error Model Extension

- Supports reliability, availability, maintainability, safety, and related specification and analysis activities
- Useful in system safety certification and qualification
- Optional set of declarations and associated semantics
- Facilitates a variety of analyses
  - top-down hazard analysis
  - a bottom-up failure modes and effects analysis
  - fault tree and stochastic process analyses
  - safety, reliability, availability, maintainability
  - integrated analyses
- Provides a representation of
  - system hazards
  - component failure modes





## Error Model

- A component has **error-free** and **error states**
- **Fault event** occurs local to a component with a specified probability.
- The error state may persist for some interval of time (the error latency) before it becomes externally **observable**
- Observable error states are **propagated** to dependent components
- Propagations can be **masked** by the observer
- (Subcomponent error model states can be mapped into a composite error model state)



## Error Modeling Approach

Error model annex clauses declare

- Error states and transitions
- Fault events & occurrence rates
- Error propagation events & occurrence rates
- Masking propagation events

Architecture model provides

- Dependency information
- Basis for isolation analysis





## Language Extension Implementation

- Property sets handled by OSATE
- Sublanguage extension
  - AADL meta model provides abstract Annex classes
  - Sublanguage meta model as additional EMF meta model package
  - AADL parser callout to sublanguage lexer/parser generated from ANTLR ([www.antlr.org](http://www.antlr.org))



## AADL Metamodel as Basis

- Metamodel Elements (in core package)
  - AnnexLibrary (aadlspecpackage.ecd)  
in AadlSpec and AadlPackageSection (public / private)
  - AnnexSubclause (core.ecd)  
in ComponentClassifier (types / implementations)

Both are named elements with no further internal structure.

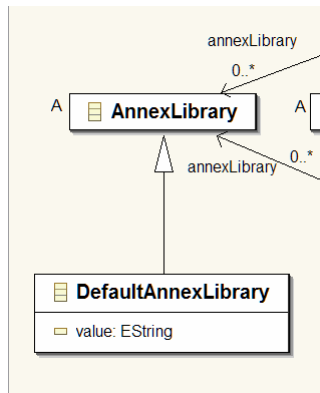
Default implementations:

DefaultAnnexLibrary and DefaultAnnexSubclause

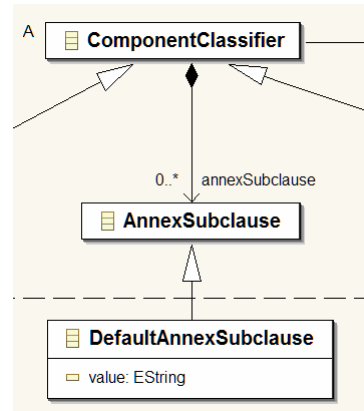
Return content of annex as a string with no processing.



## AADL Metamodel – 2



aadlspecpackage.ecd



core.ecd

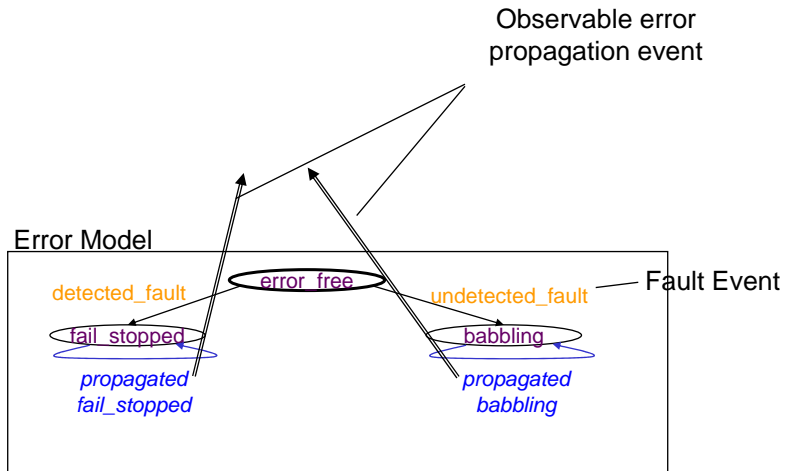


## Example Sublanguage

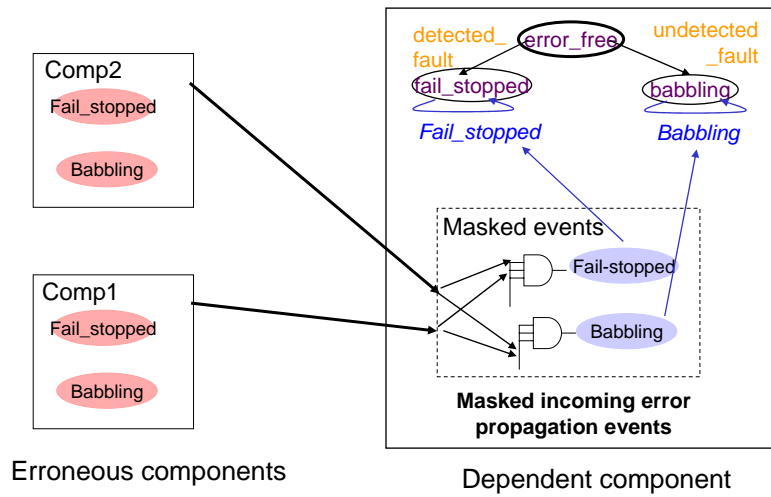
- Simplified error model annex
  - Error models
    - State machine associated with component type or implementation
    - Error model states and transitions
  - Error model events
    - Internal faults trigger transitions
    - Propagation events (outgoing)
  - Combining error models
    - Masks (in) – filter incoming propagation events
  - Properties on error model elements



# Observable Events



# Error Propagation



## Component Dependencies

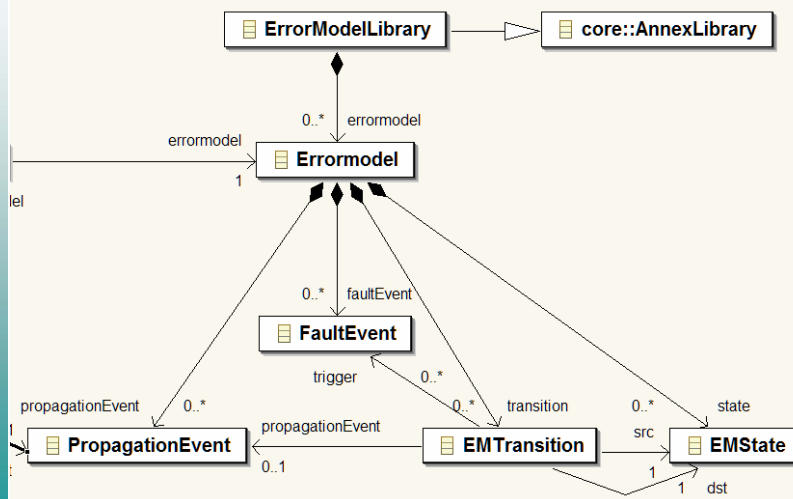
Component dependencies are determined from the architecture. Errors may propagate:

- through a connection **Port**
- through a shared subcomponent **Data access**
- from an hardware component to all dependent software components
- from a software component to other software components that share a common hardware resource, according to partitioning rules **Resource binding**
- Bus access**



## Extending the Metamodel – 1

New package: errormodel – ErrorModelLibrary





## Extension Rules

- Do
  - Extend AADL metamodel classes
    - ErrorModelLibrary → AnnexLibrary
    - Errormodel → NamedElement
    - etc.Ultimately, all new classes should have AObject as a superclass to support XMI / XML (de)serialization.
  - Reference AADL metamodel classes
  - Reuse existing constructs where possible
- Don't change the basic AADL model by
  - Introducing new superclass for AADL metamodel element
    - Indirect change to basis AADL Model → complete regeneration
  - Referencing new class from AADL class



## Error Model Properties

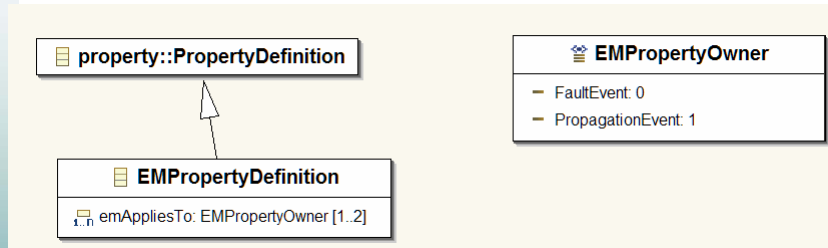
- Example: Occurrence probability of fault event
- AADL
  - Probability: `aadlinteger` applies to `FaultEvent`
- Issue
  - `PropertyDefinition` can only apply to `PropertyOwnerCategory`, which is an enumeration
- Solution / Workaround
  - Subclass `PropertyDefinition`

How to model this?



## Extending the Metamodel – 2

- Error model properties

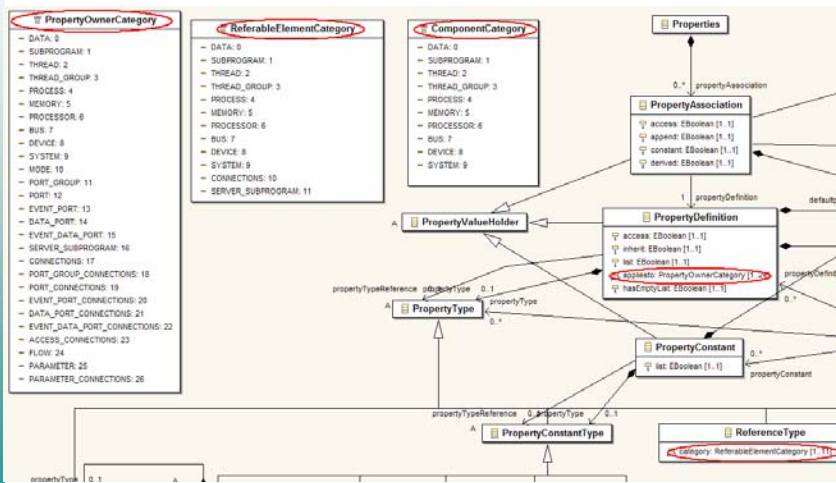


- Extensibility of property metamodel is currently limited
  - Applies to and property references are based on enumerations
  - EMF enumerations cannot be extended



## Properties

- Current metamodel



# Properties

Possible solution approaches

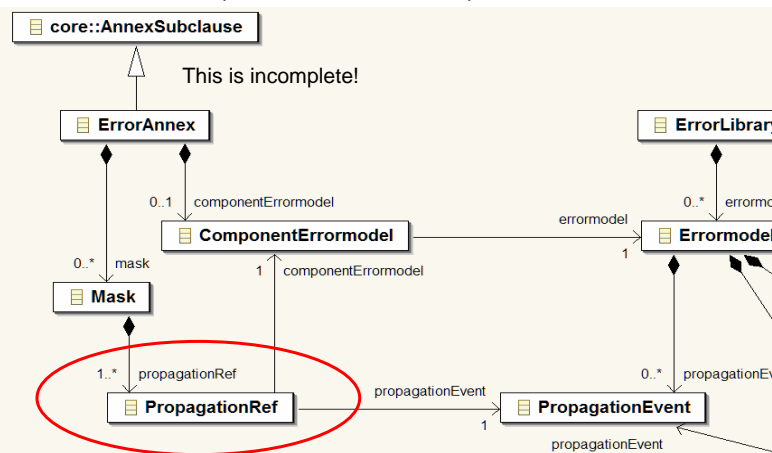
- Programmatic extensibility
  - Manually extend the enumerations based on a plug-in mechanism
  - Dynamic registration without numeric constants
- Re-design the metamodel for extensibility
  - Details TBD

Future OSATE release



# Extending the Metamodel – 2

- ErrorAnnex (AnnexSubclause) inside classifier





## Masks

- A mask filters incoming propagation events from
  - Sibling component
  - Subcomponent
  - Accessed data
  - Etc.
- In textual AADL:  
Specify the path to an outgoing propagation event in another component error model
  - Name resolution must be extended to resolve the reference
  - Similar to a property referencing a subcomponent



## Parsing an Annex – 1

Textual annex must be parsed to create the model:  
2 possible approaches

1. Two-pass approach
2. Parser extension





## Parsing an Annex – 2

### Two-pass approach

- Similar to analysis plug-in
  - Don't change standard parser
  - Traverse the model and parse value strings from default annex implementation
  - Replace the DefaultAnnex object with the new ones from the extended metamodel
  - Currently the way to do it

SAE



## Parsing an Annex – 3

### Extend the parser

- Create a parser plug-in
  - Each annex provides a plug-in to the parser to handle annex specific syntax
  - Annex parser must register with the AADL parser
  - ANTLR provides some basic support to switch to another parser

Planned to be supported in a  
future OSATE release

SAE





## Summary

- Leveraged Eclipse & EMF for persistent XML document support
- Multi-file AADL models in test
- Modal system models
  - Easy to use
  - Hard to implement
- Presented AADL sublanguage extension approach
  - Being prototyped

SAE

