Applications Of Model Weaving Techniques

Hugo Bruneliere, Jendrik Johannes

INRIA, TUD

Context of this work

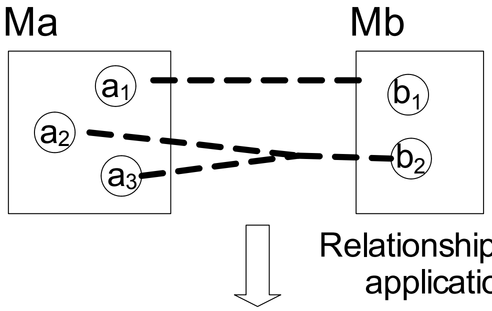
- The present courseware has been elaborated in the context of the MODELPLEX European IST FP6 project (http://www.modelplex.org/).
- Co-funded by the European Commission, the MODELPLEX project involves 21 partners from 8 different countries.
- MODELPLEX aims at defining and developing a coherent infrastructure specifically for the application of MDE to the development and subsequent management of complex systems within a variety of industrial domains.
- To achieve the goal of large-scale adoption of MDE, MODELPLEX promotes the idea of a collaborative development of courseware dedicated to this domain.
- The MDE courseware provided here with the status of open-source software is produced under the EPL 1.0 license.

Outline

- Introduction
- Model Weaving with AMW
  - Principles
  - The AMW project
  - Concrete applications (use cases)
- Weaving in Aspect-Oriented Modeling with Reusware
  - Principles
  - The Reusware Composition Framework
  - Concrete applications (use cases)
- Conclusion

Introduction

- It is often necessary to establish relationships between elements of different models, for several reasons



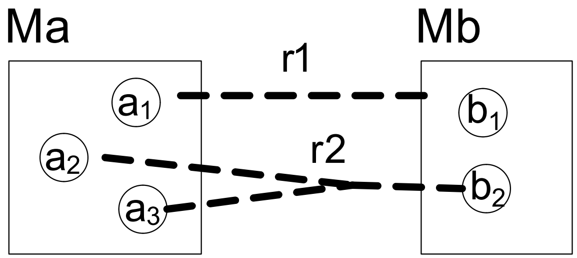
Relationships used in many application scenarios

Traceability, transformation production, merging, aspect oriented modeling (AOM), etc.

- Current MDE solutions focus on the support of model transformations
  - Designed to execute automatic operations
  - Transformations are typically general purpose languages
- Depending on the application scenario, the relationships have a different interpretation
- The relationships should be stored or modified

Introduction

- What is the form and semantics of the relationships?
  - Cardinality
  - Traceability, merging, equality, annotation, etc.
  - The relationships should not modify the related models



- What r1 means?
  a = b? a is derived from b ?, etc.
- How r1 is created?
- What do we do with r1?

- How these relationships will be created?
  - Manually, automatic, with graphical or textual interfaces

- How to use these relationships?
  - To trace, to merge, to interoperate, to annotate

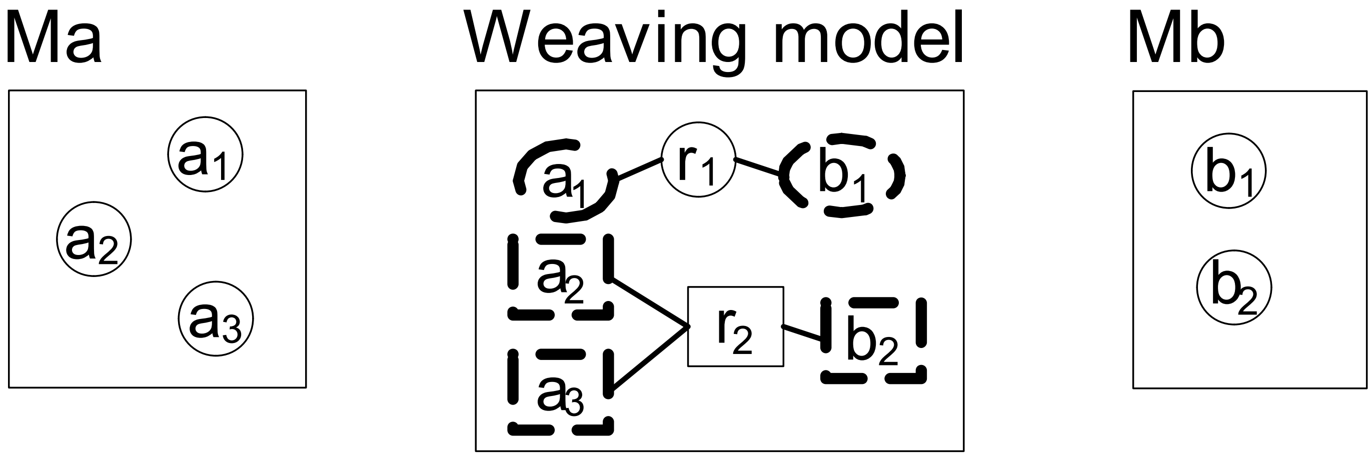
→A complete workbench must be customizable according to different application requirements

- Adaptive mechanisms should be plugged as needed

Model Weaving with AMW

AMW Principles: model weaving

- AtlanMod Team (INRIA) solution to capture relationships between model elements
- Relationships are reified in a **weaving model**
  - The model elements represent the relationships and the related elements
  - As any kind of model, the weaving model can be saved, stored, transformed, modified

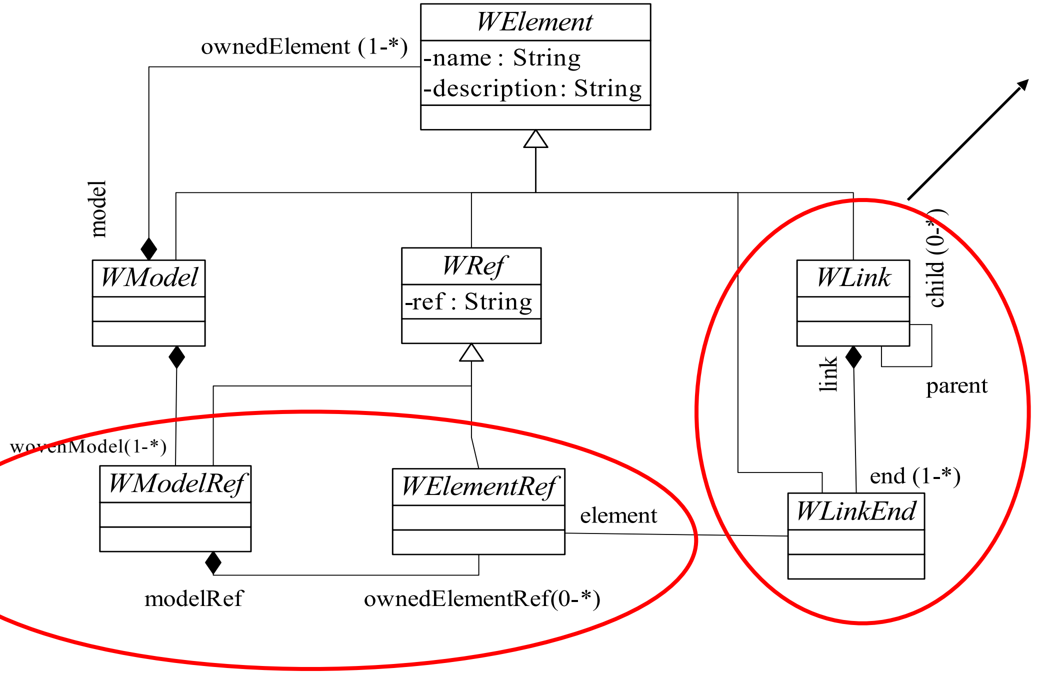


AMW Principles: weaving metamodel

- A weaving model conforms to a weaving metamodel
  - Defines the nature of the relationships that can be created
    - Cardinality (1-*, *-*, *-1)
    - Semantics of the relationships
- However, it is not practical to define a weaving metamodel that supports all kinds of relationships
  - For instance, a traceability relationship is not useful in a merging scenario
- The weaving metamodel is extensible
  - We define an core weaving metamodel

AMW Principles: core weaving metamodel

- Supports basic link management



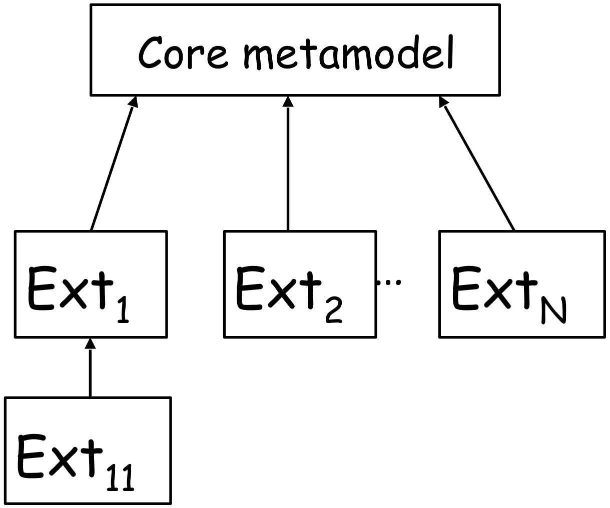
Linking semantics

(one link has N endpoints)

Namespace semantics

AMW Principles: weaving metamodel extension

- Core is extended with different kinds of relationships
  - Merging
    (merge, override, inherits)
  - Interoperability
    (translate, concatenate)
  - Traceability
    (A generates B)
  - AOM
    (A executesAfter B)



AMW Principles: creation of model weaving using matching

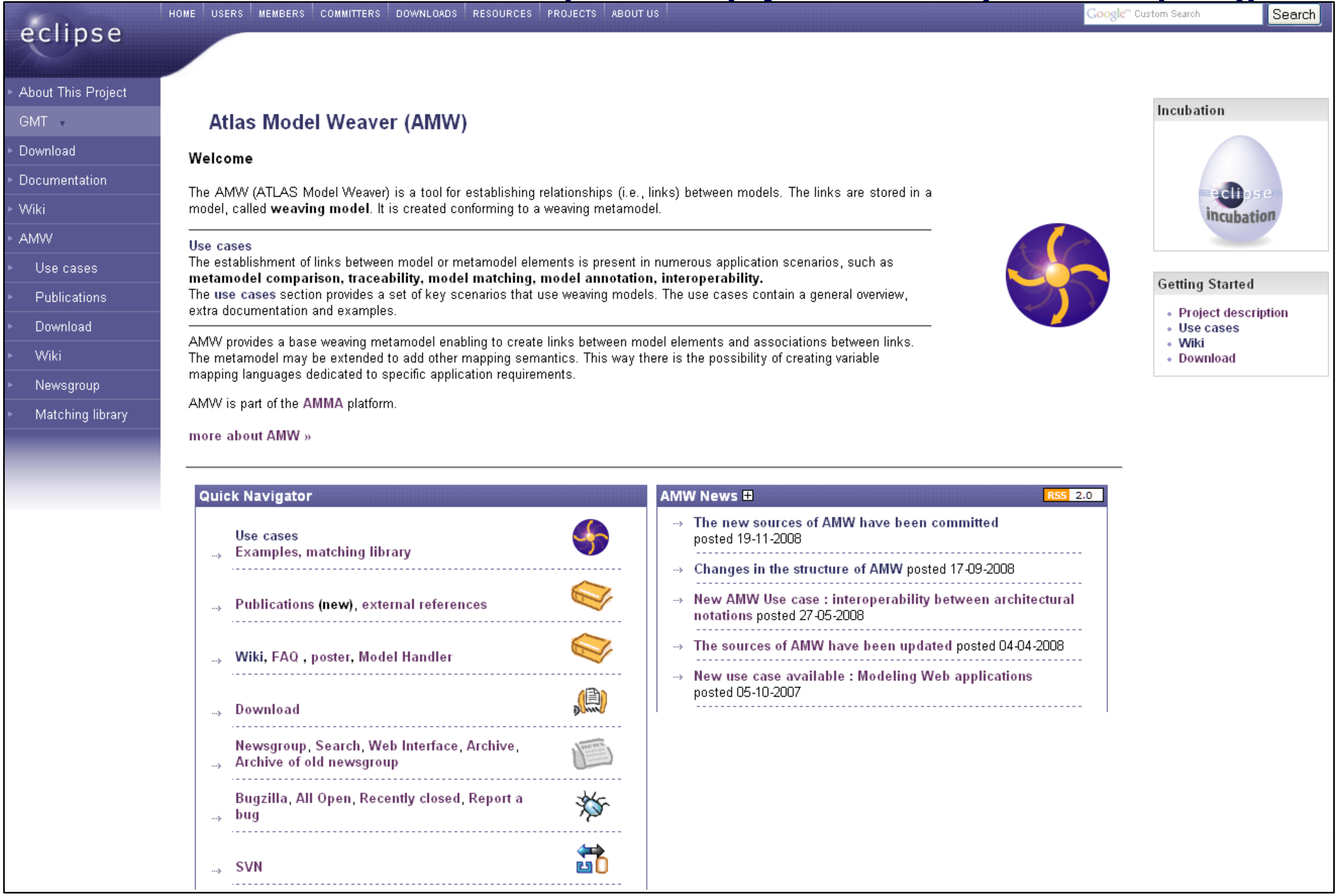
- Automatic
  - Execution of matching heuristics
    - Exploit the properties of the model elements to calculate a similarity measure between them
      - Typically dependent of the weaving metamodel extensions
      - Generic heuristics can be implemented based on the elements of the core metamodel
      - Customized heuristics are used to obtain better results
  - Execution trace
    - A weaving model is created to save the execution trace of a transformation
- Manual
  - User interface

AMW Principles: various uses

- Traceability
  - Weaving models keep track of the source elements used to generated a set of target elements
- Transformation production
  - The model elements are used as specification to produce general purpose transformations
    - Higher-order transformations translate weaving models into transformation models
- Annotations
  - The weaving model contains extra information used to annotate model elements
- Merging
  - The weaving model is used as input to merge algorithms
- Metamodel comparison
  - The weaving model is used to compare different models

The AMW Project: website homepage

- Available under Eclipse.org (Modeling/GMT project)



Atlas Model Weaver (AMW)

Welcome

The AMW (ATLAS Model Weaver) is a tool for establishing relationships (i.e., links) between models. The links are stored in a model, called **weaving model**. It is created conforming to a weaving metamodel.

Use cases
The establishment of links between model or metamodel elements is present in numerous application scenarios, such as **metamodel comparison, traceability, model matching, model annotation, interoperability.**
The use cases section provides a set of key scenarios that use weaving models. The use cases contain a general overview, extra documentation and examples.

AMW provides a base weaving metamodel enabling to create links between model elements and associations between links. The metamodel may be extended to add other mapping semantics. This way there is the possibility of creating variable mapping languages dedicated to specific application requirements.

AMW is part of the AMMA platform.

more about AMW »

Quick Navigator

- Use cases
  Examples, matching library
- Publications (new), external references
- Wiki, FAQ , poster, Model Handler
- Download
- Newsgroup, Search, Web Interface, Archive, Archive of old newsgroup
- Bugzilla, All Open, Recently closed, Report a bug
- SVN

AMW News

- The new sources of AMW have been committed posted 19-11-2008
- Changes in the structure of AMW posted 17-09-2008
- New AMW Use case : interoperability between architectural notations posted 27-05-2008
- The sources of AMW have been updated posted 04-04-2008
- New use case available : Modeling Web applications posted 05-10-2007

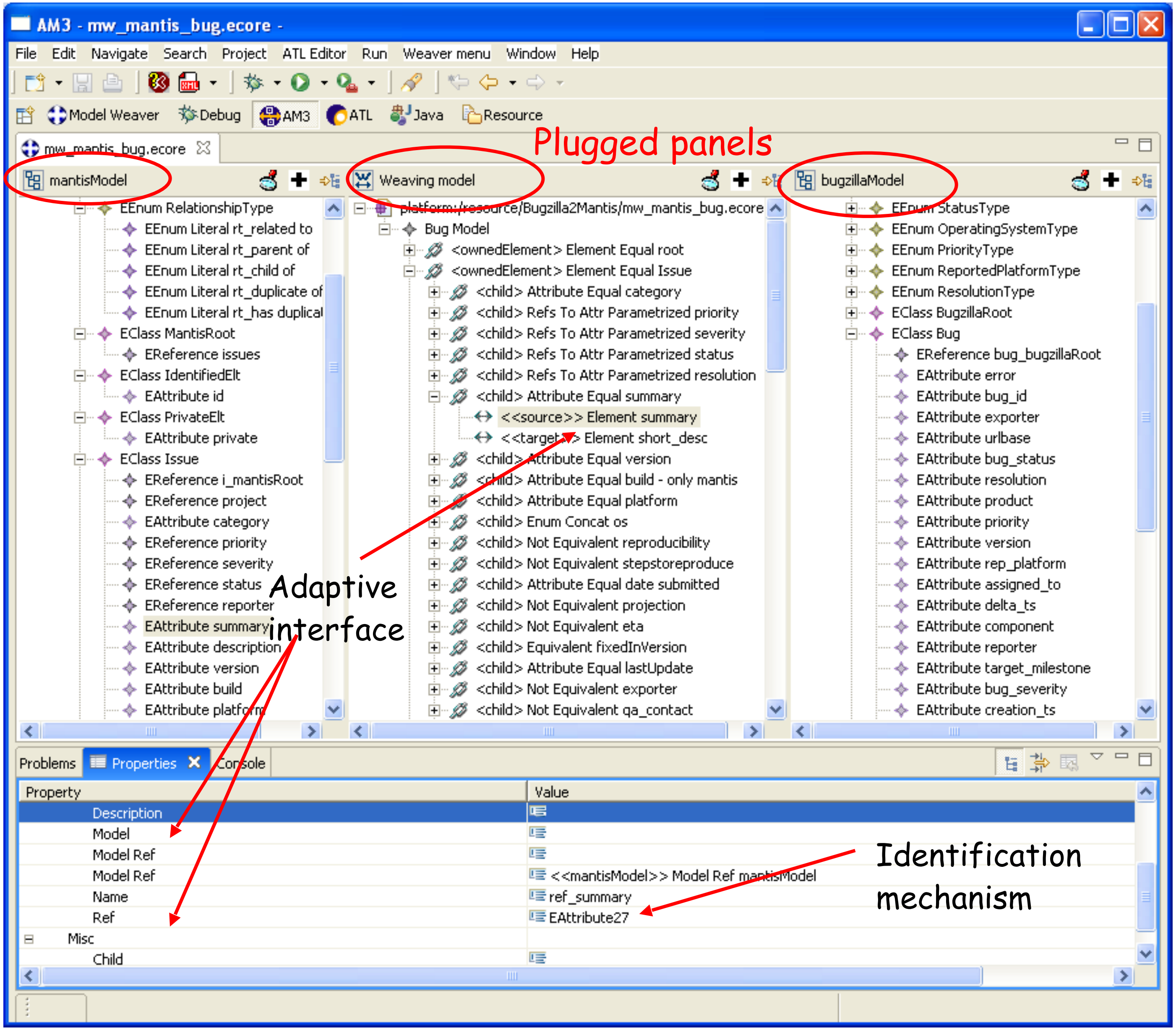
The AMW Project: plugins

- Supports the three main issues presented:
  - Which relationships to create
    - Weaving metamodel extensions
  - How to create them
    - Creation of weaving models using matching transformations integrated in the user interface
  - For what (utilization)
    - Visualization, storage, modification, execution of transformations

The AMW Project: plugins

- Adapts to any weaving metamodel extension
  - The user interface is automatic generated according to the metamodel extensions
    - Reflective API of EMF (Eclipse Modeling Framework)
- A set of extension points is defined to enable to customize the standard user interface
  - Extension points to the panels, to the model elements, and to execute model transformations in ATL (AtlanMod Transformation Language)
  - New interfaces can be easily developed

The AMW Project: user interface



mantisModel:

EEnum RelationshipType
EEnum Literal rt_related to
EEnum Literal rt_parent of
EEnum Literal rt_child of
EEnum Literal rt_duplicate of
EEnum Literal rt_has duplical
EClass MantisRoot
EReference issues
EClass IdentifiedElt
EAttribute id
EClass PrivateElt
EAttribute private
EClass Issue
EReference i_mantisRoot
EReference project
EAttribute category
EReference priority
EReference severity
EReference status
EReference reporter
EAttribute summary
EAttribute description
EAttribute version
EAttribute build
EAttribute platform

Weaving model:

platform:/resource/Bugzilla2Mantis/mw_mantis_bug.ecore
Bug Model
<ownedElement> Element Equal root
<ownedElement> Element Equal Issue
<child> Attribute Equal category
<child> Refs To Attr Parametrized priority
<child> Refs To Attr Parametrized severity
<child> Refs To Attr Parametrized status
<child> Refs To Attr Parametrized resolution
<child> Attribute Equal summary
<<source>> Element summary
<<target>> Element short_desc
<child> Attribute Equal version
<child> Attribute Equal build - only mantis
<child> Attribute Equal platform
<child> Enum Concat os
<child> Not Equivalent reproducibility
<child> Not Equivalent stepstoreproduce
<child> Attribute Equal date submitted
<child> Not Equivalent projection
<child> Not Equivalent eta
<child> Equivalent fixedInVersion
<child> Attribute Equal lastUpdate
<child> Not Equivalent exporter
<child> Not Equivalent qa_contact

bugzillaModel:

EEnum StatusType
EEnum OperatingSystemType
EEnum PriorityType
EEnum ReportedPlatformType
EEnum ResolutionType
EClass BugzillaRoot
EClass Bug
EReference bug_bugzillaRoot
EAttribute error
EAttribute bug_id
EAttribute exporter
EAttribute urlbase
EAttribute bug_status
EAttribute resolution
EAttribute product
EAttribute priority
EAttribute version
EAttribute rep_platform
EAttribute assigned_to
EAttribute delta_ts
EAttribute component
EAttribute reporter
EAttribute target_milestone
EAttribute bug_severity
EAttribute creation_ts

Property	Value
Description	
Model	
Model Ref	
Model Ref	<<mantisModel>> Model Ref mantisModel
Name	ref_summary
Ref	EAttribute27
⊟ Misc	
Child	

Plugged panels

Adaptive interface

Identification mechanism

The AMW Project: matching

- An extension point enables to plug different matching heuristics
  - Implemented with ATL transformations
  - The transformations have a fixed signature
    - Input: weaving model, left model, right model
    - Output: weaving model
  - The menus are automatically generated

The AMW Project: provided set of matching transformations

- Typed Cartesian product
- Data types and conformance
- Cardinality
- Name similarity
- Name equality
- Similarity flooding
  - Handles inheritance, containment and reference trees
- Model information
- Selection methods
  - Best values, based on a threshold

→ These transformations can be executed sequentially, or combined

The AMW Project: use cases available (project's website)

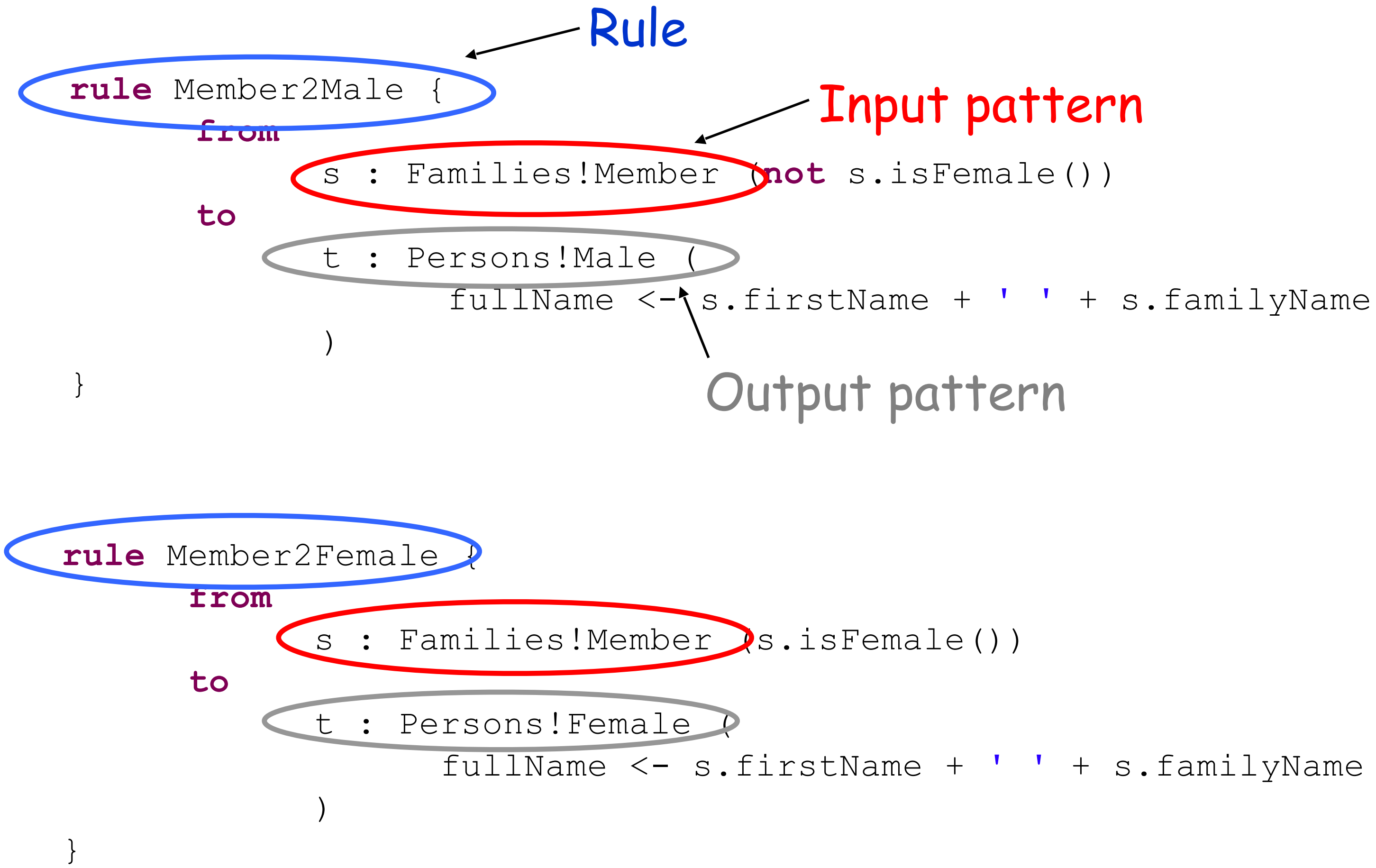
- Traceability
- Transformation production
  - Tool interoperability, data integration
- Matching
- Model alignment
- Metamodel comparison
- Aspect oriented modeling

AMW Use Case: abstract representation of ATL

- ATL Transformation
  - Textual syntax
    - Defines a complete transformation from a set of source into a set of target models
  - Issues
    - How to have a global view of the transformation?
    - How to easily develop ATL transformations without much previous knowledge on the language?

AMW Use Case: abstract representation of ATL

- Simple “Families-to-Persons” example



Rule

Input pattern

Output pattern

```
rule Member2Male {
   from
      s : Families!Member (not s.isFemale())
   to
      t : Persons!Male (
         fullName <- s.firstName + ' ' + s.familyName
      )
}


rule Member2Female {
   from
      s : Families!Member (s.isFemale())
   to
      t : Persons!Female (
         fullName <- s.firstName + ' ' + s.familyName
      )
}
```

AMW Use Case: abstract representation of ATL

- Deal with simple ATL
  - Weaving metamodel extension that enables to graphically develop ATL transformations
  - Extension close to the ATL metamodel
    - But in a higher abstraction level

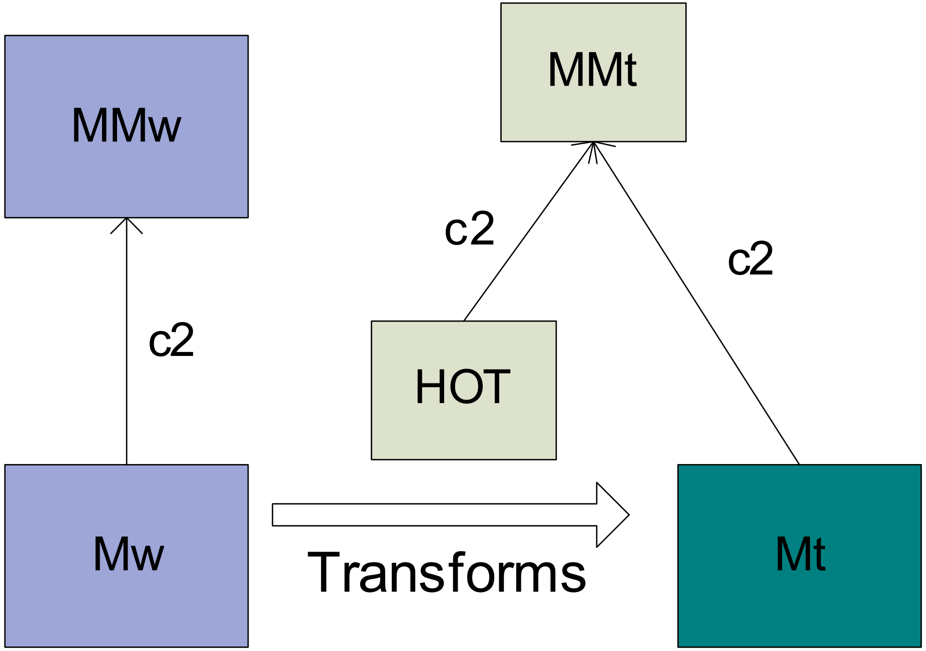
→ The skeleton of the transformation can be then automatically generated

AMW Use Case: abstract representation of ATL

- Weaving metamodel extension for ATL:
- Module
  - Input and output models
  - Rules
    - Input element
      - (some condition)
    - Output element
      - Some bindings
        - Simple bindings
        - Complex bindings
- A weaving model captures these different kinds of relationships in an abstract representation

AMW Use Case: abstract representation of ATL

- A HOT transforms the weaving model (i.e. the abstract specification of the transformation) into an ATL model
  - The transformation Mt can be refined, or directly executed (if correct)



AMW Use Case: abstract representation of ATL

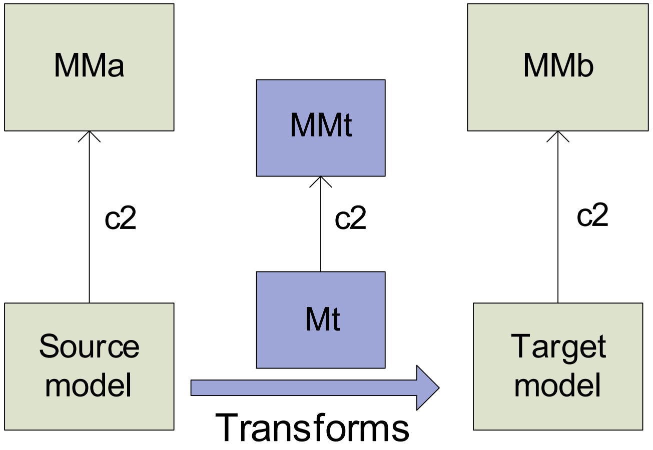
- Process to follow step by step
  - Define a metamodel extension
  - Create a new weaving model using the AMW wizard
    - Select the correct extension
    - Select the correct parameters
  - Create the weaving links
  - Produce a transformation model

AMW Use Case: traceability in model transformations

- A weaving model stores the execution traces of an ATL transformation
  - The behavior of the initial transformation is not modified
  - The weaving model is generated automatically when the transformation is executed
    - Produced as an additional output of the tranformation
- Details available from: http://www.eclipse.org/gmt/amw/usecases/traceability/

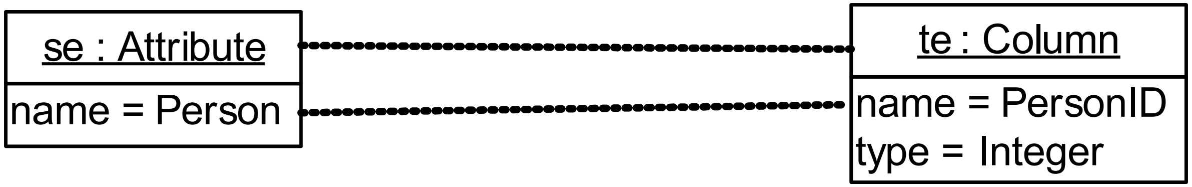
AMW Use Case: traceability in model transformations

- Class-to-Relational



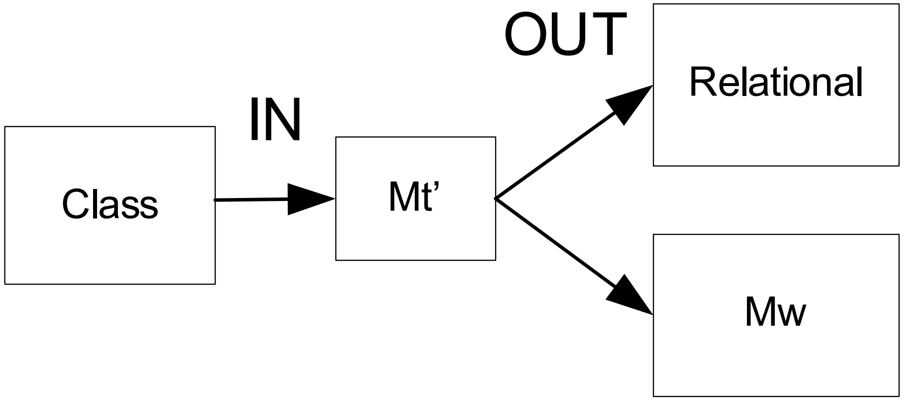
```
rule ClassAttribute 2Column {
  from
    a : Class!Attribute (
       a.type.oclIsKindOf (Class!Class)
       and not a.multiValued
    )
  to
    foreignKey : Relational!Column (
      name <- a.name + 'Id',
      type <- thisModule.objectIdType
   )
}
```

- Traceability



AMW Use Case: traceability in model transformations

- Traceability weaving model
  - Mt (Class to Relational) is used to produce Mt’
    - Generated automatically: ATL2WTracer.atl



  - Mt’ generates a weaving model and the relational model
    - The weaving model is opened in the AMW tool

AMW Use Case: traceability in model transformations

- Weaving metamodel extension

```
class TraceLink  extends WLink{
    attribute ruleName : String;
    reference sourceElements[*] ordered container : WLinkEnd;
    reference targetElements[*] ordered container : WLinkEnd;
}
class TraceLinkEnd extends WLinkEnd {
}
class ElementRef extends WElementRef {
}
```

AMW Use Case: traceability in model transformations

- Process to follow step by step
  - Define a metamodel extension
  - Produce a modified Class2Relational transformation
  - Execute this transformation
    - This transformation produces a weaving model and a relational model
  - Open it in the AMW tool

AMW Use Case: generic matching

- Creation of weaving models
  - Automatic
    - Execution of matching heuristics
      - Exploit the properties of the model elements to calculate a similarity measure between them
        - Typically dependent of the weaving metamodel extensions
        - Generic heuristics can be implemented based on the elements of the core metamodel
        - Customized heuristics are used to obtain better results
  - Manual
    - User interface
  - Used in different use cases:

    http://www.eclipse.org/gmt/amw/usecases/matching/

AMW Use Case: generic matching

- Different kinds of links

- Direct links

String similarity

Date ↔ BirthDate

Descr ↔ Description

Dictionaries of Synonyms

Car ↔ Automobile

Professor ↔ Teacher

Structural features

Class
└ name ↔

Table
└ name

Since *Class* and *Table* has a sub element *name*,
they are considered similar.

- Complex links

Format compatibility

Date = Day / Month / Year

Concatenation

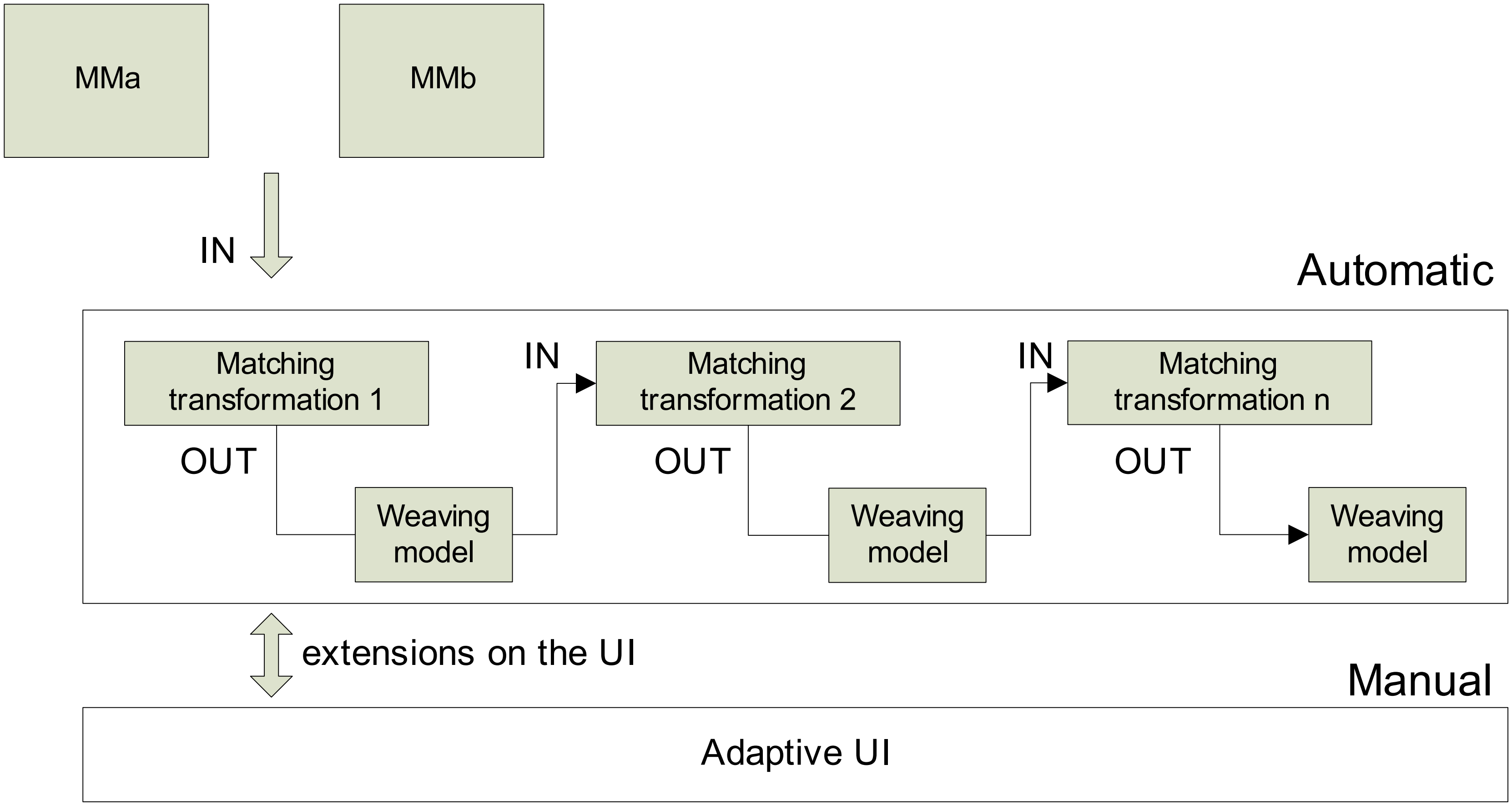
Name = FirstName + LastName

Data conversions

Dollar = Euro x ConvertionRate

AMW Use Case: generic matching

- General overview



AMW Use Case: generic matching

- AMW supports matching
  - An extension point enables to plug different matching heuristics
    - Implemented with ATL transformations
    - The transformations have a fixed signature
      - Input: weaving model, left model, right model
      - Output: weaving model
    - The menus are automatically generated

AMW Use Case: generic matching

- Use the provided set of matching transformations
  - Typed Cartesian product
  - Data types and conformance
  - Cardinality
  - Name similarity
  - Name equality
  - Similarity flooding
    - Handles inheritance, containment and reference trees
  - Model information
  - Selection methods
    - Best values, based on a threshold

→ These transformations can be executed sequentially, or combined

→ The transformations interpret specific metamodel extensions

→ This procedure is used in several use cases (see the next "metamodel comparison" use case)

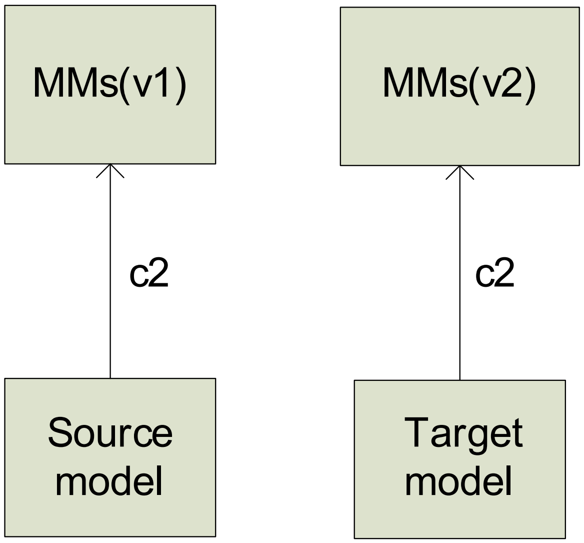
AMW Use Case: metamodel comparison & model migration

- Metamodels need to be compared for several reasons
  - One important reason is to discover the equivalent elements between two versions of a metamodel
  - Different utilisations
    - Migrate one model to another
    - Apply metamodel difference

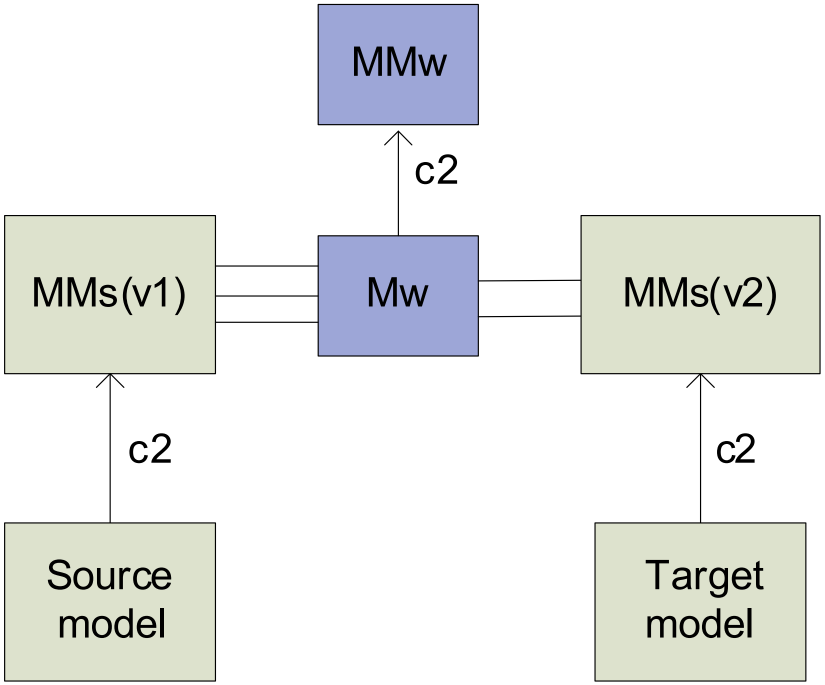
AMW Use Case: metamodel comparison & model migration

- General overview

1) Two versions of similar metamodels



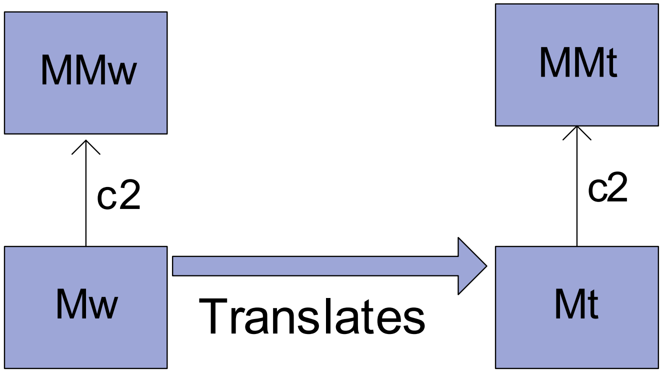
2) A set of transformations produces a weaving model between the metamodels



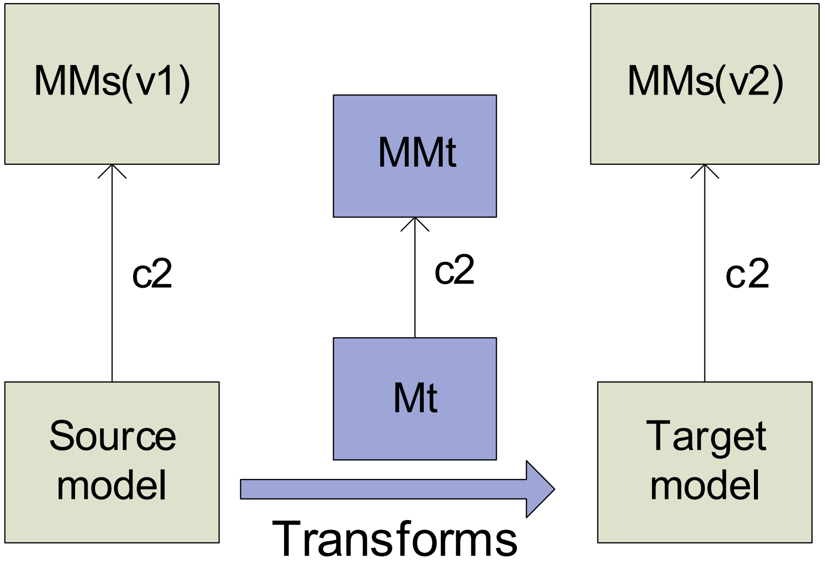
AMW Use Case: metamodel comparison & model migration

- General overview (comparison and model migration)

3) The weaving model is translated in a model transformation



4) The source model is transformed into the target model .



AMW Use Case: metamodel comparison & model migration

- Weaving metamodel extension

```
abstract class Equivalent extends WLink {
   attribute similarity : Double;
   reference left container : ReferredElement;
   reference right container : ReferredElement;
}
abstract class Equal extends Equivalent {
}
class ElementEqual extends Equal {
}
class AttributeEqual extends Equal {
}
class ReferenceEqual extends Equal {
}
class NotEquivalent extends WLink {
   reference left container : ReferredElement;
   reference right container : ReferredElement;
}
class ReferredElement extends WLinkEnd {
}
```

AMW Use Case: metamodel comparison & model migration

- Process to follow step by step
  - Definition of the metamodel extension
  - Creation of a weaving model (comparison model)
  - Parameterization and execution of the matching transformations to refine the comparison model
  - Generation of an ATL transformation
  - Application of the generated transformation for model migration

The AMW Technology: conclusions

- Model weaving
  - Weaving models reify relationships between model elements from different models
  - Several application scenarios
    - An extensible metamodel is essential
  - Weaving models are stored, transformed, modified

- AMW is a flexible model weaving plug-in
  - Weaving metamodels extensions in KM3
    - Adaptive user interface
    - No need to develop a new tool for each application scenario
  - ATL transformations are plugged as needed
    - Matching transformations
    - Higher-order transformations
  - Several examples available

Weaving in Aspect-Oriented Modeling with Reuseware

Reuseware Principles

- Reuseware is...
  - A language independent modularisation approach [1]
  - A framework: Reuseware Composition Framework [2]
- Common concepts for different composition systems for arbitrary languages
  - Easy specification of new composition techniques and porting of techniques from one language to another
  - Reuse of composition tooling
  - Tailor tooling for composition system and language
- Support features of aspect-oriented systems
  - Support homogeneous cross-cuts (and quantification)
  - Support heterogeneous cross-cuts

[1] On Language-Independent Model Modularisation,Transactions on Aspect-Oriented Development, 2008

[2] http://reuseware.org

Reuseware Principles - Core Concepts

- Model Fragments
  - (Partial) models that may contain variation points
  - Offer a **Composition Interface**
  - **Composition Interface** consists of **Ports**
  - **Ports** point at elements of the model fragment that can be accessed for composition
  - One **Port** can point at several elements at arbitrary places in the model fragment (heterogeneous crosscut)
  - Similar Ports can be joined to one **HomogeneousPort** (homogeneous crosscut)
- Composition Programs
  - Define **composition links** between Ports
  - Are executed to produce a composed model where model fragments are woven at the elements pointed out by the linked Ports

Reuseware Principles - Core Concepts

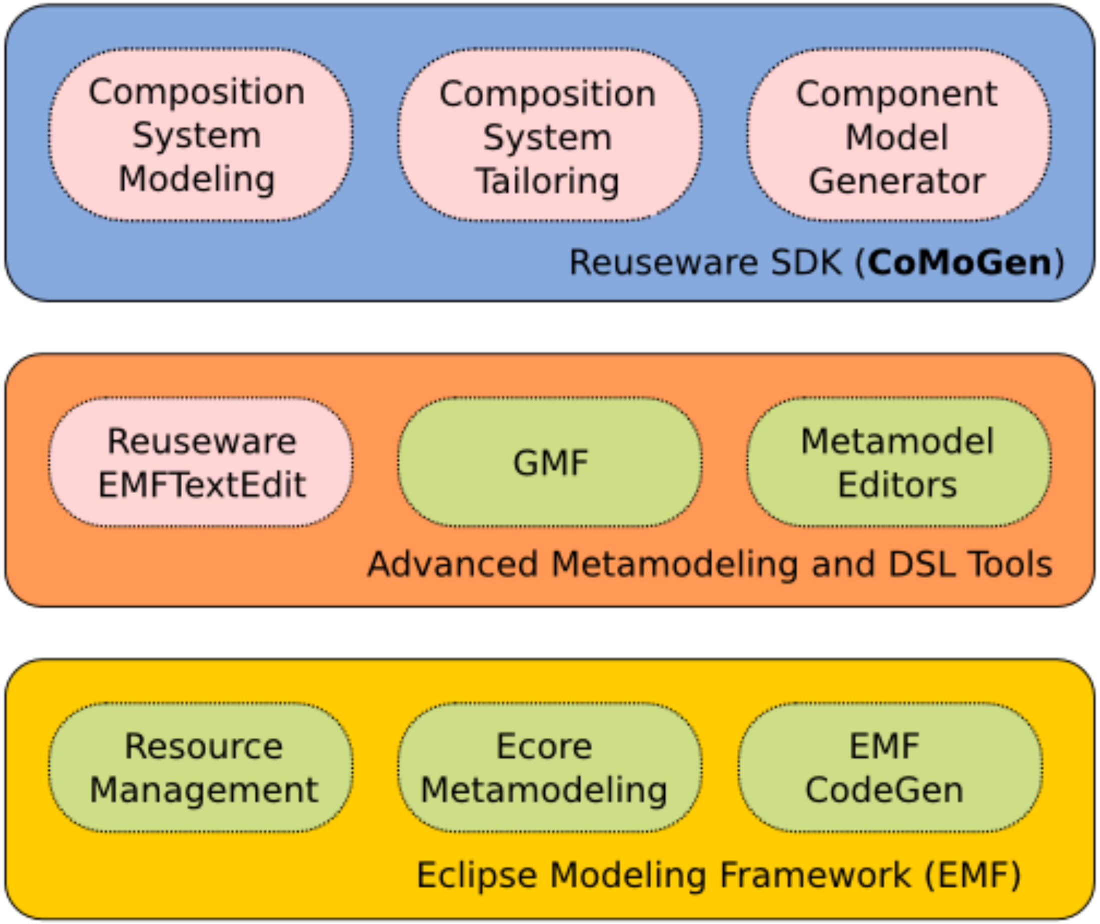
- Composition Systems
  - Define modularisation concepts (e.g., Modules, Packages, Aspects)
  - Define relations between modularisation concepts (e.g, an aspect relates to a core)
- Reuse extensions (for a particular language)
  - Define how modularization concepts defined in a composition system are realized in a concrete language
  - Define which ports are related to which model elements of a model fragment

Reuseware Composition Framework

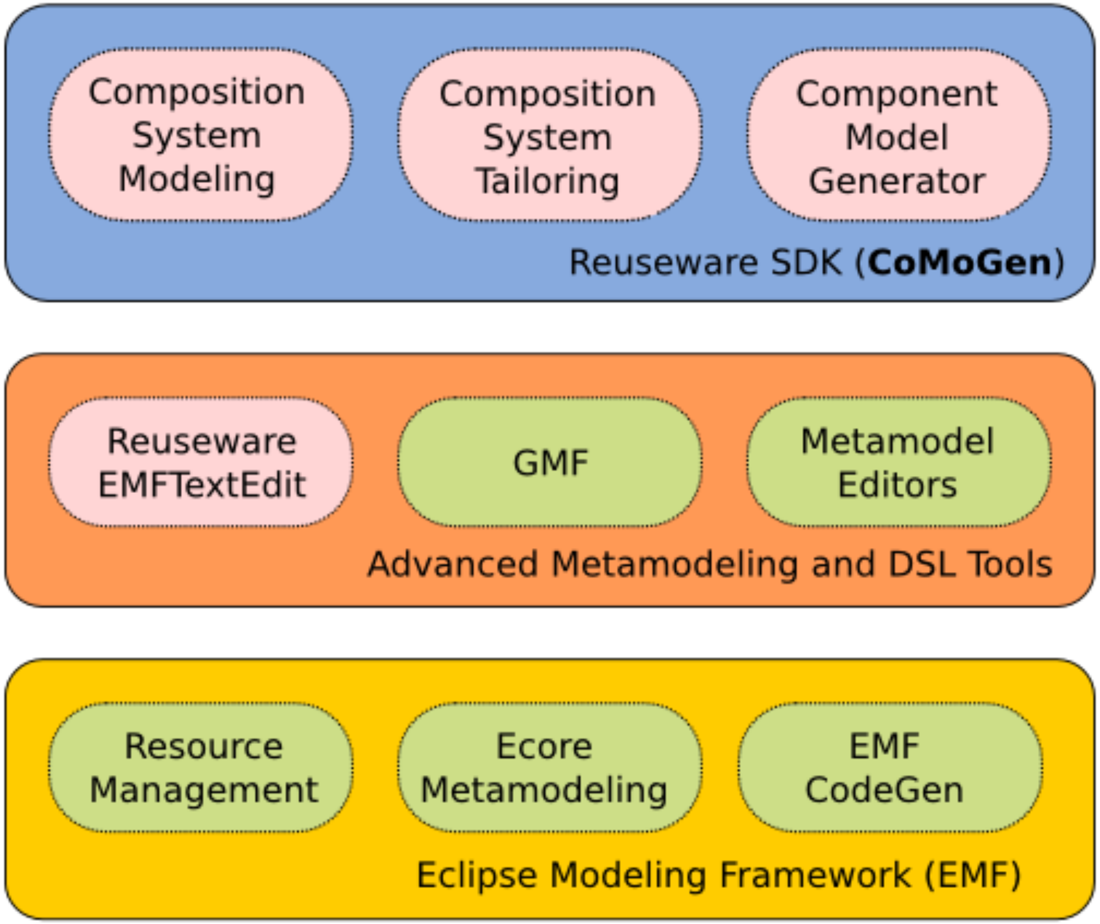
reuseware
composition framework

- CoMoGen (Reuseware SDK)
  - Enables developers to define new composition systems
  - Addition to other language engineering (metamodelling /DSL) tools to define modularisation aspect of a language
- CoCoNut (Reuseware Runtime)
  - Provides language independent composition engine
  - Provides language independent component repository
  - Provides language independent composition program editor
  - Composition systems defined with CoMoGen plug into CoCoNut and tailor the above functionality

CoMoGen (Reuseware SDK)

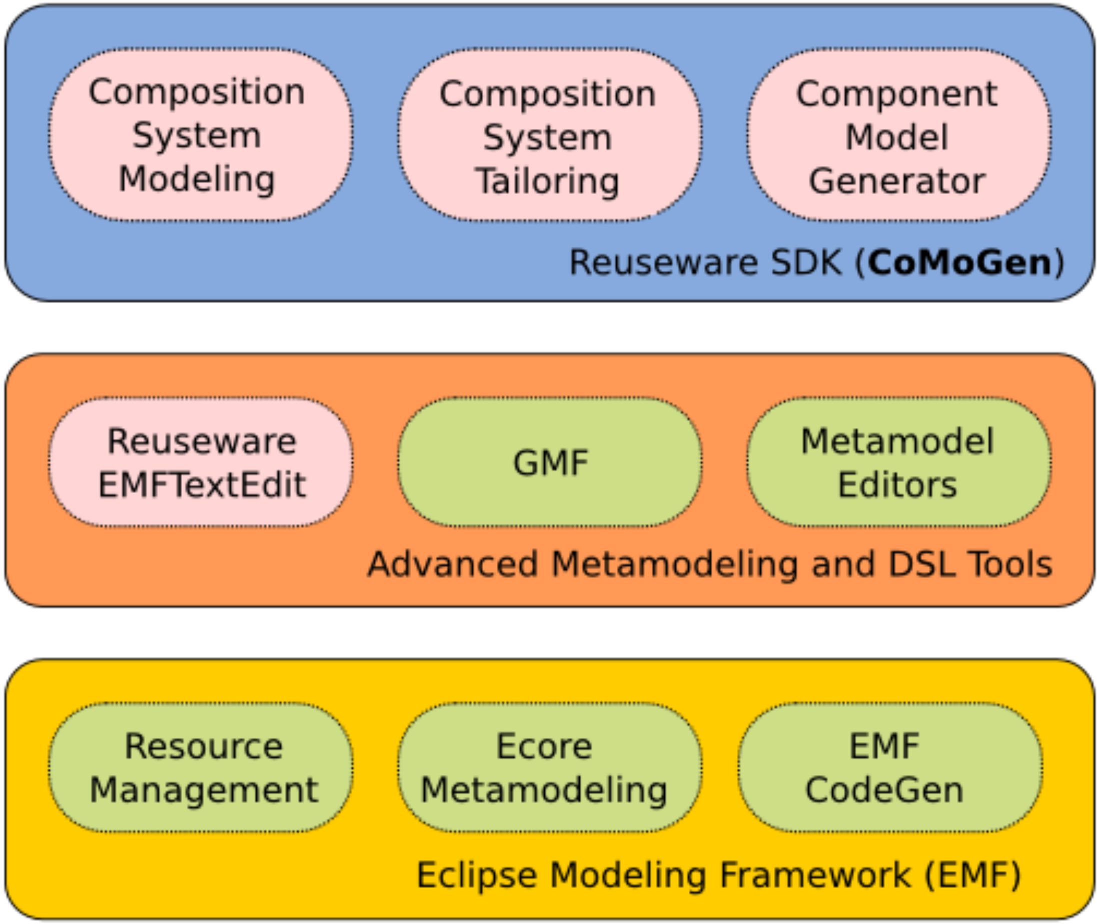

Composition System Modeling
Composition System Tailoring
Component Model Generator
Reuseware SDK (**CoMoGen**)
Reuseware EMFTextEdit
GMF
Metamodel Editors
Advanced Metamodeling and DSL Tools
Resource Management
Ecore Metamodeling
EMF CodeGen
Eclipse Modeling Framework (EMF)

CoMoGen (Reuseware SDK)


Composition System Modeling
Composition System Tailoring
Component Model Generator
Reuseware SDK (**CoMoGen**)
Reuseware EMFTextEdit
GMF
Metamodel Editors
Advanced Metamodeling and DSL Tools
Resource Management
Ecore Metamodeling
EMF CodeGen
Eclipse Modeling Framework (EMF)

Reuseware builds on EMF and works together with other EMF-based tools

CoMoGen (Reuseware SDK)


Composition System Modeling
Composition System Tailoring
Component Model Generator
Reuseware SDK (**CoMoGen**)
Reuseware EMFTextEdit
GMF
Metamodel Editors
Advanced Metamodeling and DSL Tools
Resource Management
Ecore Metamodeling
EMF CodeGen
Eclipse Modeling Framework (EMF)

Metamodeling tools can be used to define a language and tools for the language (examples are shown here)

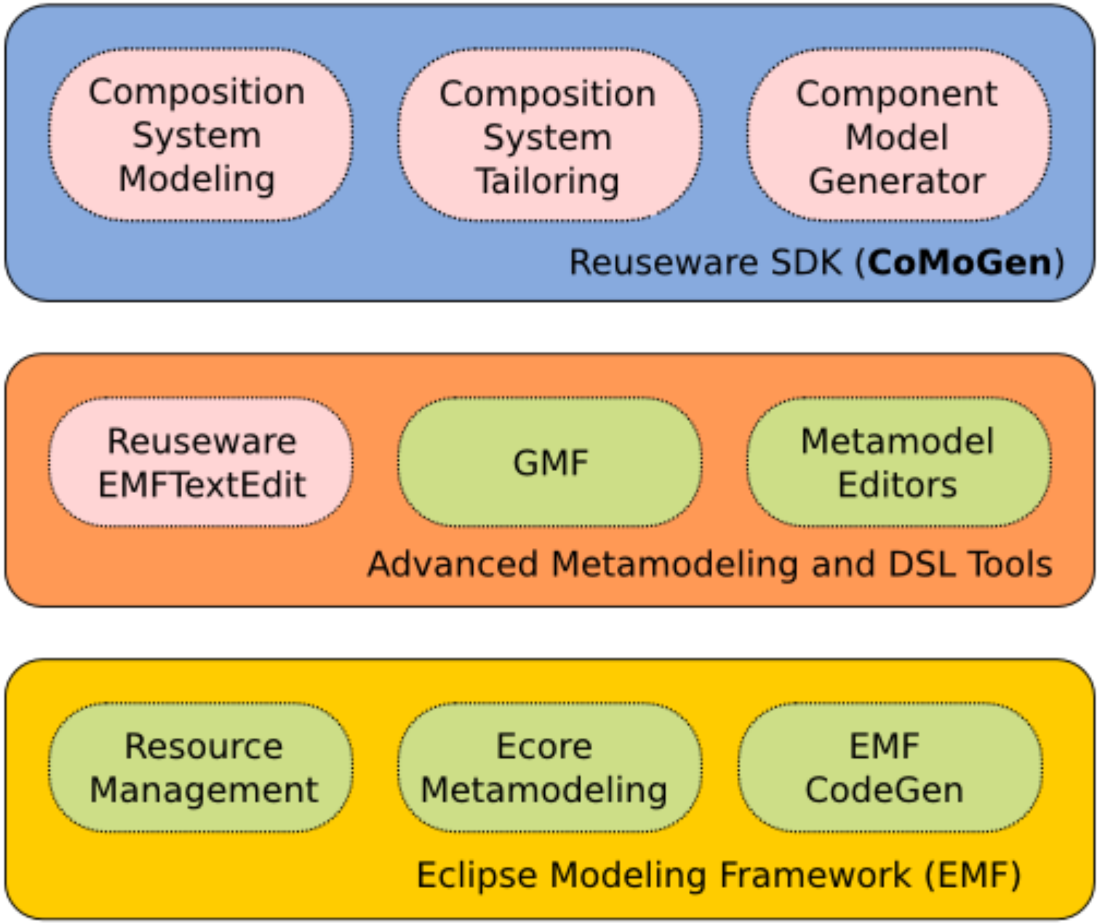
Reuseware builds on EMF and works together with other EMF-based tools

CoMoGen (Reuseware SDK)

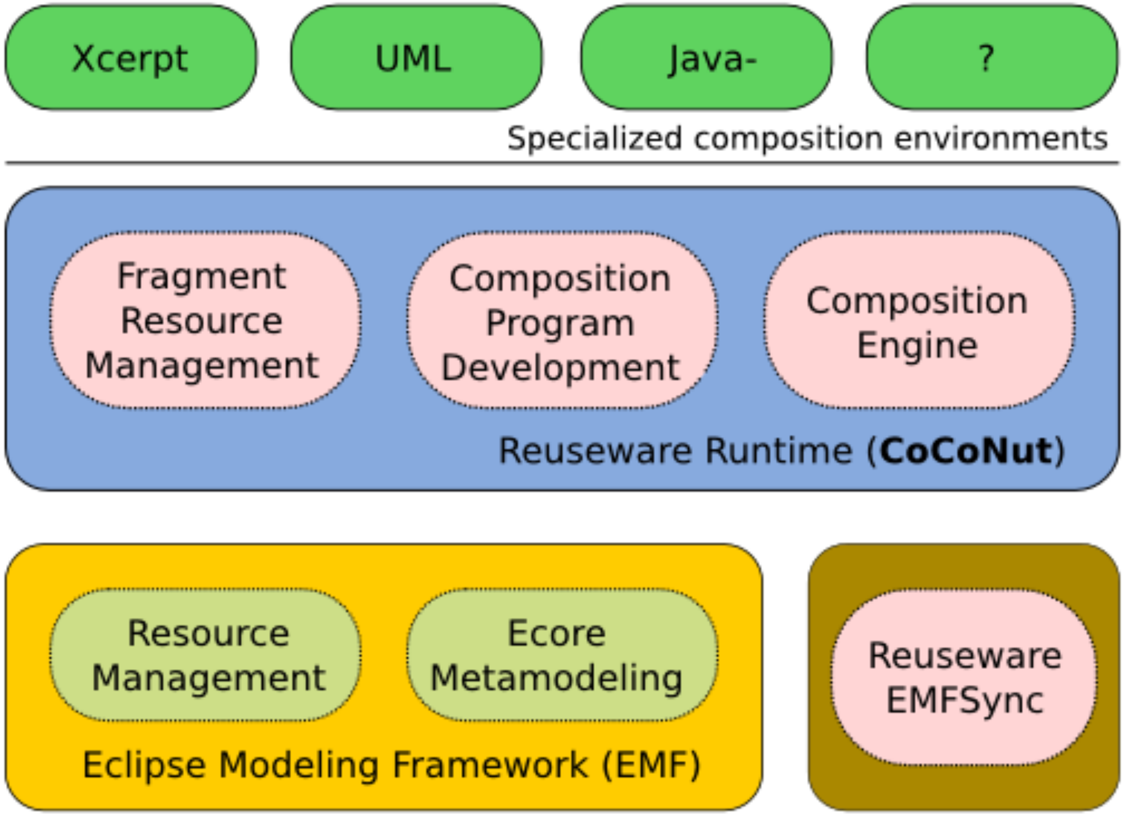
With CoMoGen, model composition systems can be modelled based on a prior defined metamodel

Metamodeling tools can be used to define a language and tools for the language (examples are shown here)

Reuseware builds on EMF and works together with other EMF-based tools


Composition System Modeling
Composition System Tailoring
Component Model Generator
Reuseware SDK (CoMoGen)
Reuseware EMFTextEdit
GMF
Metamodel Editors
Advanced Metamodeling and DSL Tools
Resource Management
Ecore Metamodeling
EMF CodeGen
Eclipse Modeling Framework (EMF)

CoCoNut (Reuseware Runtime)

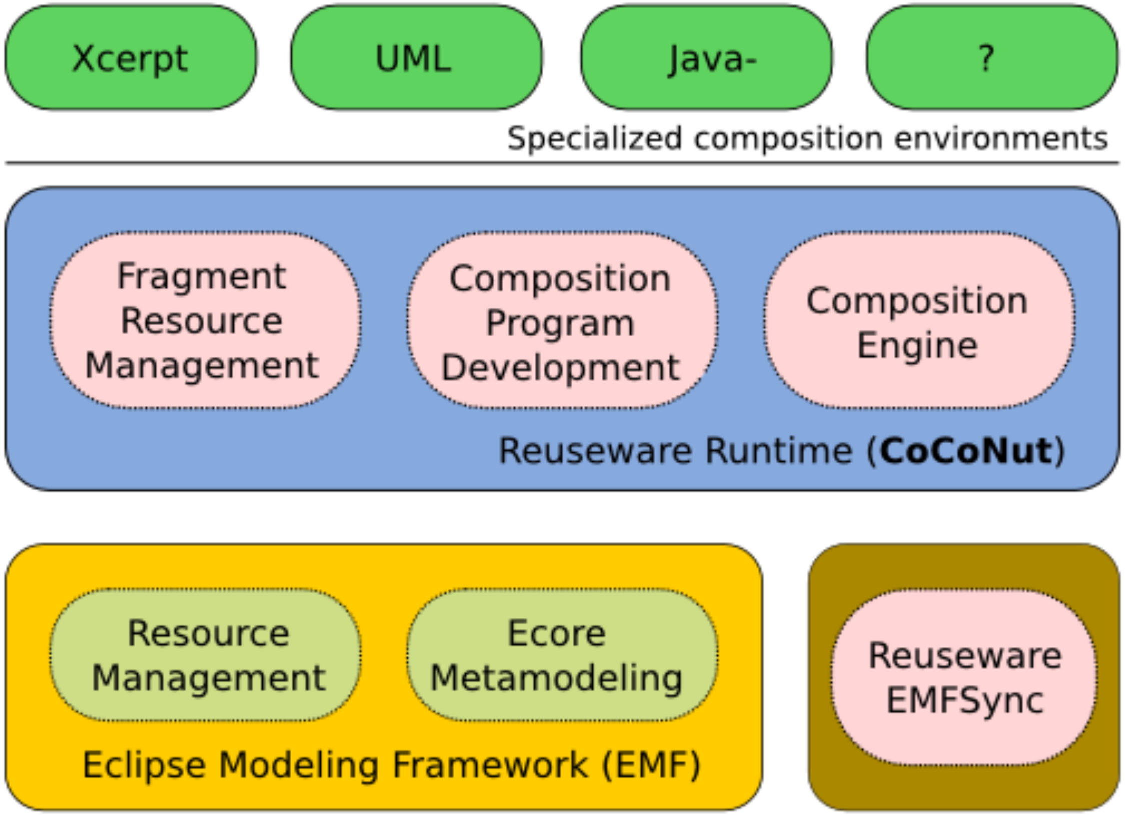

Xcerpt
UML
Java-
?
Specialized composition environments
Fragment Resource Management
Composition Program Development
Composition Engine
Reuseware Runtime (**CoCoNut**)
Resource Management
Ecore Metamodeling
Eclipse Modeling Framework (EMF)
Reuseware EMFSync

Reuseware builds on EMF and works together with other EMF-based tools

CoCoNut (Reuseware Runtime)

The three language-independent features of CoCoNut can be used with every composition system

Reuseware builds on EMF and works together with other EMF-based tools

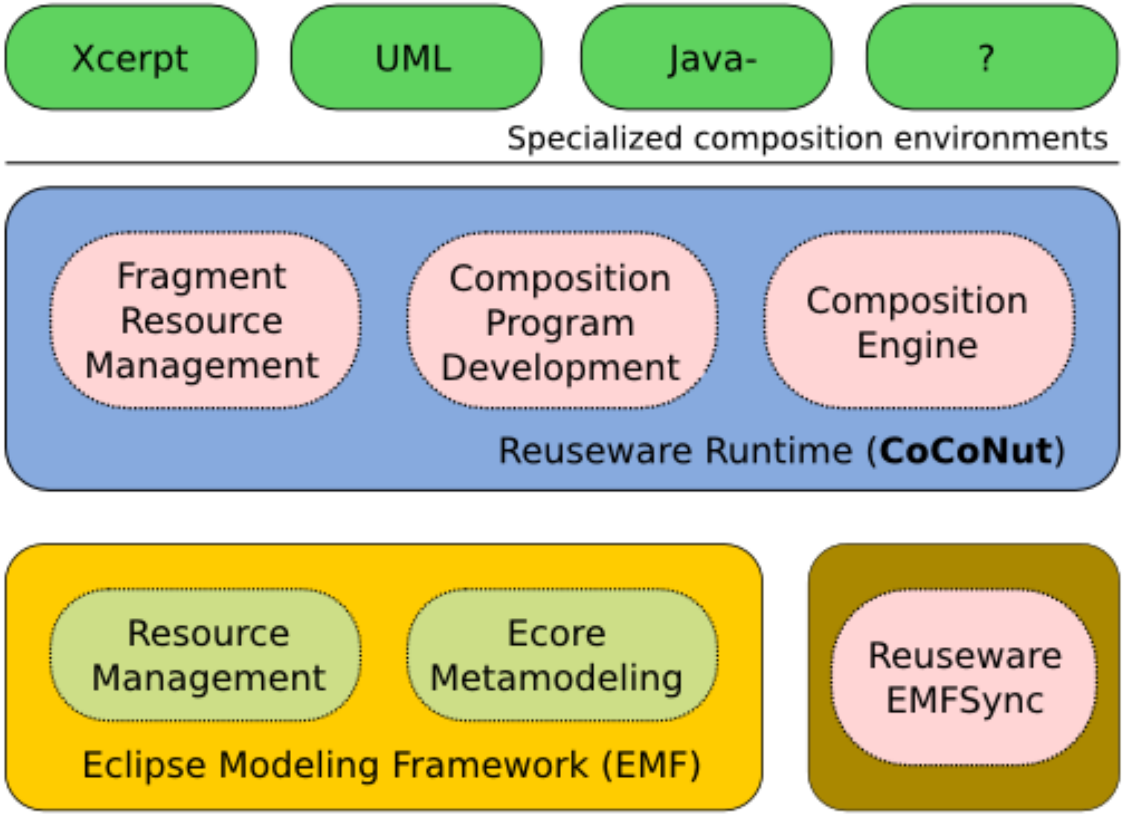

Xcerpt
UML
Java-
?
Specialized composition environments
Fragment Resource Management
Composition Program Development
Composition Engine
Reuseware Runtime (CoCoNut)
Resource Management
Ecore Metamodeling
Eclipse Modeling Framework (EMF)
Reuseware EMFSync

CoCoNut (Reuseware Runtime)

Specific composition systems defined with CoMoGen plug into CoCoNut

The three language-independent features of CoCoNut can be used with every composition system

Reuseware builds on EMF and works together with other EMF-based tools

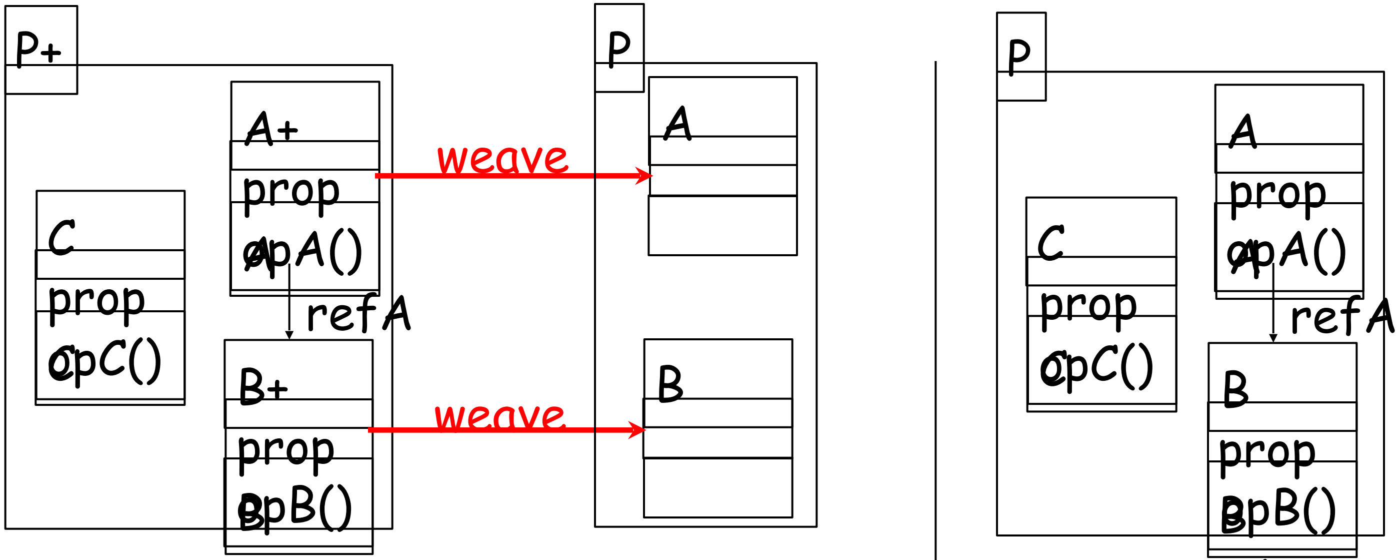

Xcerpt
UML
Java-
?
Specialized composition environments
Fragment Resource Management
Composition Program Development
Composition Engine
Reuseware Runtime (**CoCoNut**)
Resource Management
Ecore Metamodeling
Eclipse Modeling Framework (EMF)
Reuseware EMFSync

Reuseware Use Case: Class Weaving

- Demonstrates CoMoGen Features
  - How to define a concrete composition system
  - How to use the composition system with different language
- Demonstrates CoCoNut Features
  - Using the composition program editor
  - Using the model fragment repository
  - Using the composition engine

Reuseware Use Case: Class Weaving

- Based on Class Concept found in many languages
  - A Class usually has operations, properties and references to other classes and is contained in a package
- Weaving Class A+ (Advice) into Class A means:
  - All operations, properties and references of A+ are added to A
  - if any of the new elements of A points at a class B+ which is woven into another class B, the pointers are redirected to B
  - Classes that are not source of a weaving should be copied completely



Model fragments + weaving description Woven (composed) Model

Reuseware Use Case: ClassWeaving Composition System

- A composition system defines
  - Fragment roles
    - Role a model fragment plays in the modularisation (e.g., aspect or core)
    - Fragment roles collaborate through associations between ports
  - Static ports
    - Defined for one fragment role
    - Each fragment playing the role has to offer the port
  - Dynamic ports
    - Defined for one fragment role
    - Each fragment playing the role can offer several of these ports
  - Contribution Associations
    - Defines that two ports are related
    - Executing a composition link between the two ports will trigger the copying of model elements
  - Configuration Associations
    - Defines that two ports are related
    - Executing a composition link between the two ports will NOT trigger the copying of model elements

Reuseware Use Case: ClassWeaving Composition System

A Core acts as container for additional content (Container); it contains Classes which should be individually accessible for extension (therefore the number of ports is dynamic - it depends on the number of existing classes)

```
compositionsystem ClassWeaving {

  fragment role Core {
    static port Container;
    dynamic port Classes;
  }

  fragment role Aspect {
    static port Content;
    dynamic port Advices;
  }


  contribution Aspect.Content --> Core.Container;

  contribution Aspect.Advice --> Core.Class;
}
```

Reuseware Use Case: ClassWeaving Composition System

```
compositionsystem ClassWeaving {

  fragment role Core {
    static port Container;
    dynamic port Classes;
  }

  fragment role Aspect {
    static port Content;
    dynamic port Advices;
  }



  contribution Aspect.Content --> Core.Container;

  contribution Aspect.Advice --> Core.Class;
}
```

A Core acts as container for additional content (Container); it contains Classes which should be individually accessible for extension (therefore the number of ports is dynamic - it depends on the number of existing classes)

An Aspect offers additional Content; it contains Advices as extensions for classes which should be individually accessible

Reuseware Use Case: ClassWeaving Composition System

```
compositionsystem ClassWeaving {

  fragment role Core {
    static port Container;
    dynamic port Classes;
  }

  fragment role Aspect {
    static port Content;
    dynamic port Advices;
  }


  contribution Aspect.Content --> Core.Container;

  contribution Aspect.Advice --> Core.Class;
}
```

A Core acts as container for additional content (Container); it contains Classes which should be individually accessible for extension (therefore the number of ports is dynamic - it depends on the number of existing classes)

An Aspect offers additional Content; it contains Advices as extensions for classes which should be individually accessible

A Content contributes to a Container

Reuseware Use Case: ClassWeaving Composition System

```
compositionsystem ClassWeaving {

  fragment role Core {
    static port Container;
    dynamic port Classes;
  }

  fragment role Aspect {
    static port Content;
    dynamic port Advices;
  }


  contribution Aspect.Content --> Core.Container;

  contribution Aspect.Advice --> Core.Class;
}
```

A Core acts as container for additional content (Container); it contains Classes which should be individually accessible for extension (therefore the number of ports is dynamic - it depends on the number of existing classes)

An Aspect offers additional Content; it contains Advices as extensions for classes which should be individually accessible

A Content contributes to a Container

An Advice contributes to a Class

Reuseware Use Case: ClassWeaving Reuse Extensions

- A Reuse Extension defines
  - How a composition interface define by a fragment role (which is defined in a composition system) is linked to the content of a model fragment
  - Each port links to a set of model elements treated as:
    - **Prototype**: Element that can be copied with its contained elements
    - **Anchor**: Element that can be referenced by other elements
    - **Hook**: Variation point where Prototypes can be put
    - **Slot**: Variation point where Anchors can be put
- For ClassWeaving we define
  - A reuse extension for Ecore
  - A reuse extension for UML

Reuseware Use Case: ClassWeaving for Ecore

The ClassWeaving composition system is implemented for the Ecore language (using the URI of the Ecore metamodel)

```
reuseextension ClassWeavingEcore
implements ClassWeaving
epackages <http://www.eclipse.org/emf/2002/Ecore>
rootclass EPackage {
 fragment role Core if $not name.startsWith('advice')$ {
    port Container {
      EPackage.eClassifiers is hook {}
    }
    port Class {
      EClass.eOperations is hook {
        port expr = $name$
      }
      EClass.eStructuralFeatures is hook {
        port expr = $name$
      }
      EClass is anchor {
        port expr = $name$
      }
    }
  }

 fragment role Aspect if $name.startsWith('advice')$ {
    port Content {
      EPackage.eClassifiers is prototype {}
    }
    port Advice {
      EClass.eOperations is prototype {
        port expr = $name$
      }
      EClass.eStructuralFeatures is prototype {
        port expr = $name$
      }
      EClass is slot {
        port expr = $name$
```

Reuseware Use Case: ClassWeaving for Ecore

```
reuseextension ClassWeavingEcore
implements ClassWeaving
epackages <http://www.eclipse.org/emf/2002/Ecore>
rootclass EPackage {
 fragment role Core if $not name.startsWith('advice')$ {
    port Container {
      EPackage.eClassifiers is hook {}
    }
    port Class {
      EClass.eOperations is hook {
        port expr = $name$
      }
      EClass.eStructuralFeatures is hook {
        port expr = $name$
      }
      EClass is anchor {
        port expr = $name$
      }
    }
  }

 fragment role Aspect if $name.startsWith('advice')$ {
    port Content {
      EPackage.eClassifiers is prototype {}
    }
    port Advice {
      EClass.eOperations is prototype {
        port expr = $name$
      }
      EClass.eStructuralFeatures is prototype {
        port expr = $name$
      }
      EClass is slot {
        port expr = $name$
```

The ClassWeaving composition system is implemented for the Ecore language (using the URI of the Ecore metamodel)

A core can be extended with new classes by extending the eClassifiers reference of the EPackage metaclass

Reuseware Use Case: ClassWeaving for Ecore

```
reuseextension ClassWeavingEcore
implements ClassWeaving
epackages <http://www.eclipse.org/emf/2002/Ecore>
rootclass EPackage {
 fragment role Core if $not name.startsWith('advice')$ {
    port Container {
      EPackage.eClassifiers is hook {}
    }
    port Class {
      EClass.eOperations is hook {
        port expr = $name$
      }
      EClass.eStructuralFeatures is hook {
        port expr = $name$
      }
      EClass is anchor {
        port expr = $name$
      }
    }
  }

 fragment role Aspect if $name.startsWith('advice')$ {
    port Content {
      EPackage.eClassifiers is prototype {}
    }
    port Advice {
      EClass.eOperations is prototype {
        port expr = $name$
      }
      EClass.eStructuralFeatures is prototype {
        port expr = $name$
      }
      EClass is slot {
        port expr = $name$
```

The ClassWeaving composition system is implemented for the Ecore language (using the URI of the Ecore metamodel)

A core can be extended with new classes by extending the eClassifiers reference of the EPackage metaclass

Extending a class means extending the eOperations and eStructuralFeatrues references of an EClass; An EClass itself will be referenced (anchor) as replacement for advices; Each EClass and its references are accessible through a port identified by the class name

Reuseware Use Case: ClassWeaving for Ecore

```
reuseextension ClassWeavingEcore
implements ClassWeaving
epackages <http://www.eclipse.org/emf/2002/Ecore>
rootclass EPackage {
 fragment role Core if $not name.startsWith('advice')$ {
    port Container {
      EPackage.eClassifiers is hook {}
    }
    port Class {
      EClass.eOperations is hook {
        port expr = $name$
      }
      EClass.eStructuralFeatures is hook {
        port expr = $name$
      }
      EClass is anchor {
        port expr = $name$
      }
    }
  }

 fragment role Aspect if $name.startsWith('advice')$ {
    port Content {
      EPackage.eClassifiers is prototype {}
    }
    port Advice {
      EClass.eOperations is prototype {
        port expr = $name$
      }
      EClass.eStructuralFeatures is prototype {
        port expr = $name$
      }
      EClass is slot {
        port expr = $name$
```

The ClassWeaving composition system is implemented for the Ecore language (using the URI of the Ecore metamodel)

A core can be extended with new classes by extending the eClassifiers reference of the EPackage metaclass

Extending a class means extending the eOperations and eStructuralFeatrues references of an EClass; An EClass itself will be referenced (anchor) as replacement for advices; Each EClass and its references are accessible through a port identified by the class name

The eClassifiers reference of the EPackage metaclass defines the content of an aspect

Reuseware Use Case: ClassWeaving for Ecore

```
reuseextension ClassWeavingEcore
implements ClassWeaving
epackages <http://www.eclipse.org/emf/2002/Ecore>
rootclass EPackage {
 fragment role Core if $not name.startsWith('advice')$ {
    port Container {
      EPackage.eClassifiers is hook {}
    }
    port Class {
      EClass.eOperations is hook {
        port expr = $name$
      }
      EClass.eStructuralFeatures is hook {
        port expr = $name$
      }
      EClass is anchor {
        port expr = $name$
      }
    }
  }

 fragment role Aspect if $name.startsWith('advice')$ {
    port Content {
      EPackage.eClassifiers is prototype {}
    }
    port Advice {
      EClass.eOperations is prototype {
        port expr = $name$
      }
      EClass.eStructuralFeatures is prototype {
        port expr = $name$
      }
      EClass is slot {
        port expr = $name$
```

The ClassWeaving composition system is implemented for the Ecore language (using the URI of the Ecore metamodel)

A core can be extended with new classes by extending the eClassifiers reference of the EPackage metaclass

Extending a class means extending the eOperations and eStructuralFeatrues references of an EClass; An EClass itself will be referenced (anchor) as replacement for advices; Each EClass and its references are accessible through a port identified by the class name

The eClassifiers reference of the EPackage metaclass defines the content of an aspect

An advice is modelled as an instance of EClass; The eOperations and eStructuralFeatrues references are exported; The EClass itself is to be replaced (slot); Each advice EClass and its references are accessible through a port identified by the class name

Reuseware Use Case: ClassWeaving for UML

```
reuseextension ClassWeavingUML
implements ClassWeaving
for <http://www.eclipse.org/uml2/2.1.0/UML>
rootclass Model {
 fragment role Core if $not name.startsWith('advice')$ {
    port Container {
      Package.packagedElement is hook {}
    }
    port Class {
      Class.ownedOperation is hook {
        port expr = $name$
      }
      Class.ownedAttribute is hook {
        port expr = $name$
      }
      Class is anchor {
        port expr = $name$
      }
    }
  }

 fragment role Aspect if $name.startsWith('advice')$ {
    port Content {
      Package.packagedElement is prototype {}
    }
    port Advice {
      Class.ownedOperation is prototype {
        port expr = $name$
      }
      Class.ownedAttribute is prototype {
        port expr = $name$
      }
      Class is slot {
        port expr = $name$
```

The ClassWeaving composition system is implemented for the UML language (using the URI of the UML metamodel)

Reuseware Use Case: ClassWeaving for UML

```
reuseextension ClassWeavingUML
implements ClassWeaving
for <http://www.eclipse.org/uml2/2.1.0/UML>
rootclass Model {
 fragment role Core if $not name.startsWith('advice')$ {
    port Container {
      Package.packagedElement is hook {}
    }
    port Class {
      Class.ownedOperation is hook {
        port expr = $name$
      }
      Class.ownedAttribute is hook {
        port expr = $name$
      }
      Class is anchor {
        port expr = $name$
      }
    }
  }

 fragment role Aspect if $name.startsWith('advice')$ {
    port Content {
      Package.packagedElement is prototype {}
    }
    port Advice {
      Class.ownedOperation is prototype {
        port expr = $name$
      }
      Class.ownedAttribute is prototype {
        port expr = $name$
      }
      Class is slot {
        port expr = $name$
```

The ClassWeaving composition system is implemented for the UML language (using the URI of the UML metamodel)

In UML, the packagedElement reference contains Classes and Associations...

Reuseware Use Case: ClassWeaving for UML

```
reuseextension ClassWeavingUML
implements ClassWeaving
for <http://www.eclipse.org/uml2/2.1.0/UML>
rootclass Model {
 fragment role Core if $not name.startsWith('advice')$ {
    port Container {
      Package.packagedElement is hook {}
    }
    port Class {
      Class.ownedOperation is hook {
        port expr = $name$
      }
      Class.ownedAttribute is hook {
        port expr = $name$
      }
      Class is anchor {
        port expr = $name$
      }
    }
  }

 fragment role Aspect if $name.startsWith('advice')$ {
    port Content {
      Package.packagedElement is prototype {}
    }
    port Advice {
      Class.ownedOperation is prototype {
        port expr = $name$
      }
      Class.ownedAttribute is prototype {
        port expr = $name$
      }
      Class is slot {
        port expr = $name$
```

The ClassWeaving composition system is implemented for the UML language (using the URI of the UML metamodel)

In UML, the packagedElement reference contains Classes and Associations...

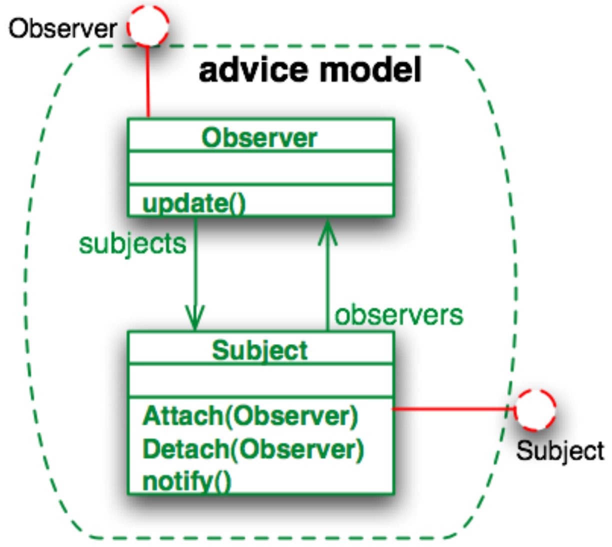
...the class contains only ownedAttributes (and no references or associaitons)

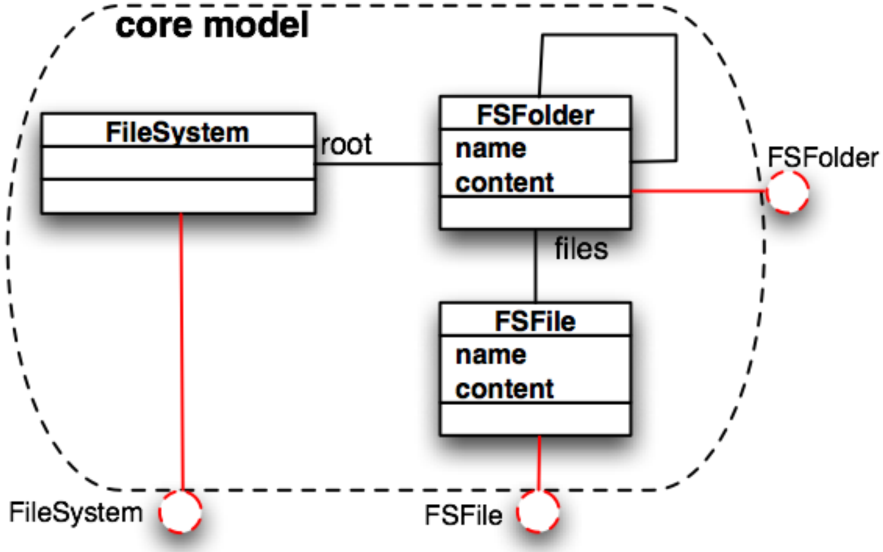
Reuseware Use Case: Using Ecore ClassWeaving

- Activate in CoConut:
  - ClassWeaving composition system
  - ClassWeaving for Ecore reuse extendsion
- Activation through
  - Eclipse plugin extension point
  - Dynamically at runtime
- Enables use of
  - Fragment repository
  - Composition program editor
  - Composition engine

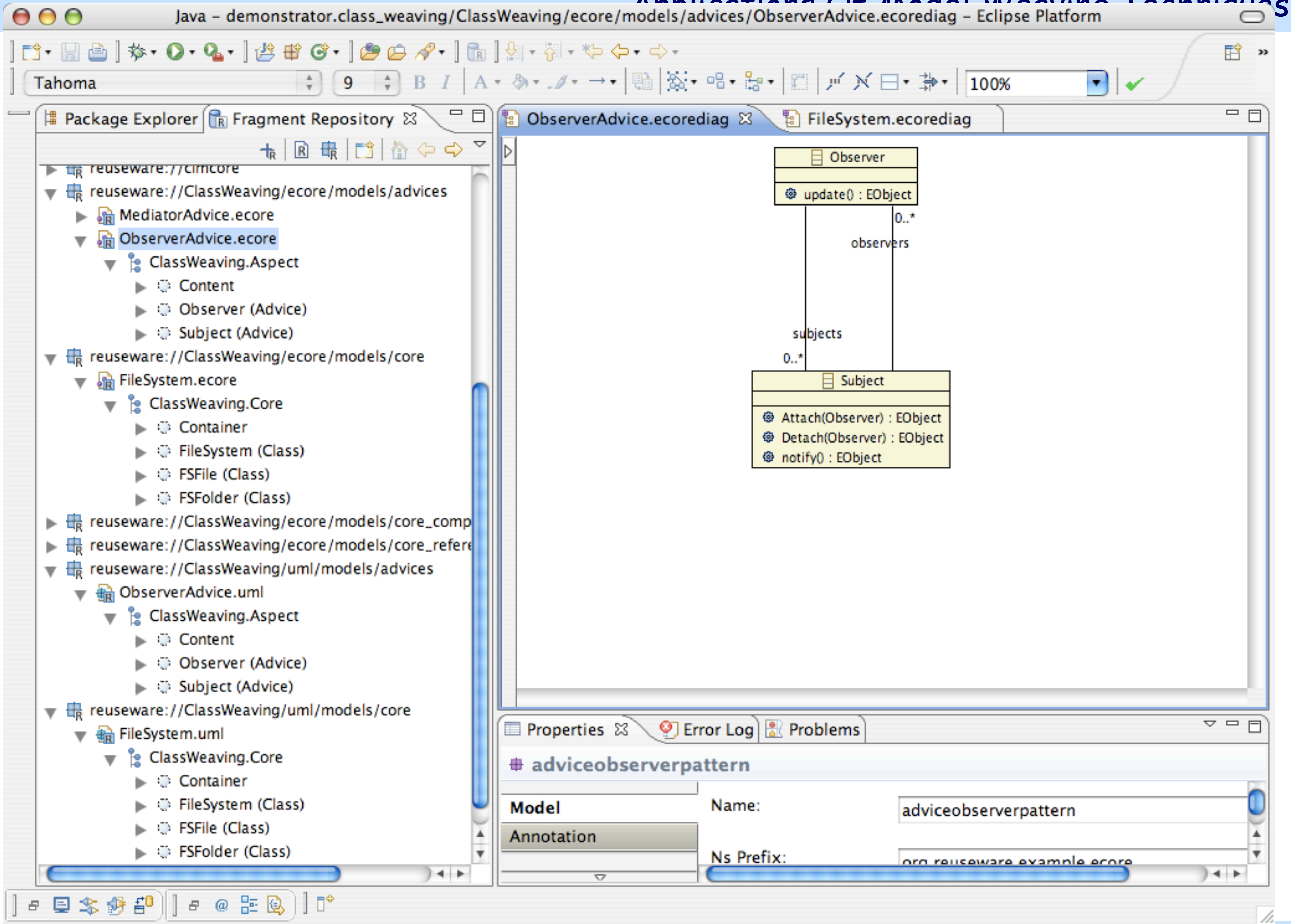
Reuseware Use Case: Example Observer

- Weaving observer functionality into a model of a file system





Applications Of Model Weaving Techniques



Java – demonstrator.class_weaving/ClassWeaving/ecore/models/advices/ObserverAdvice.ecorediag – Eclipse Platform

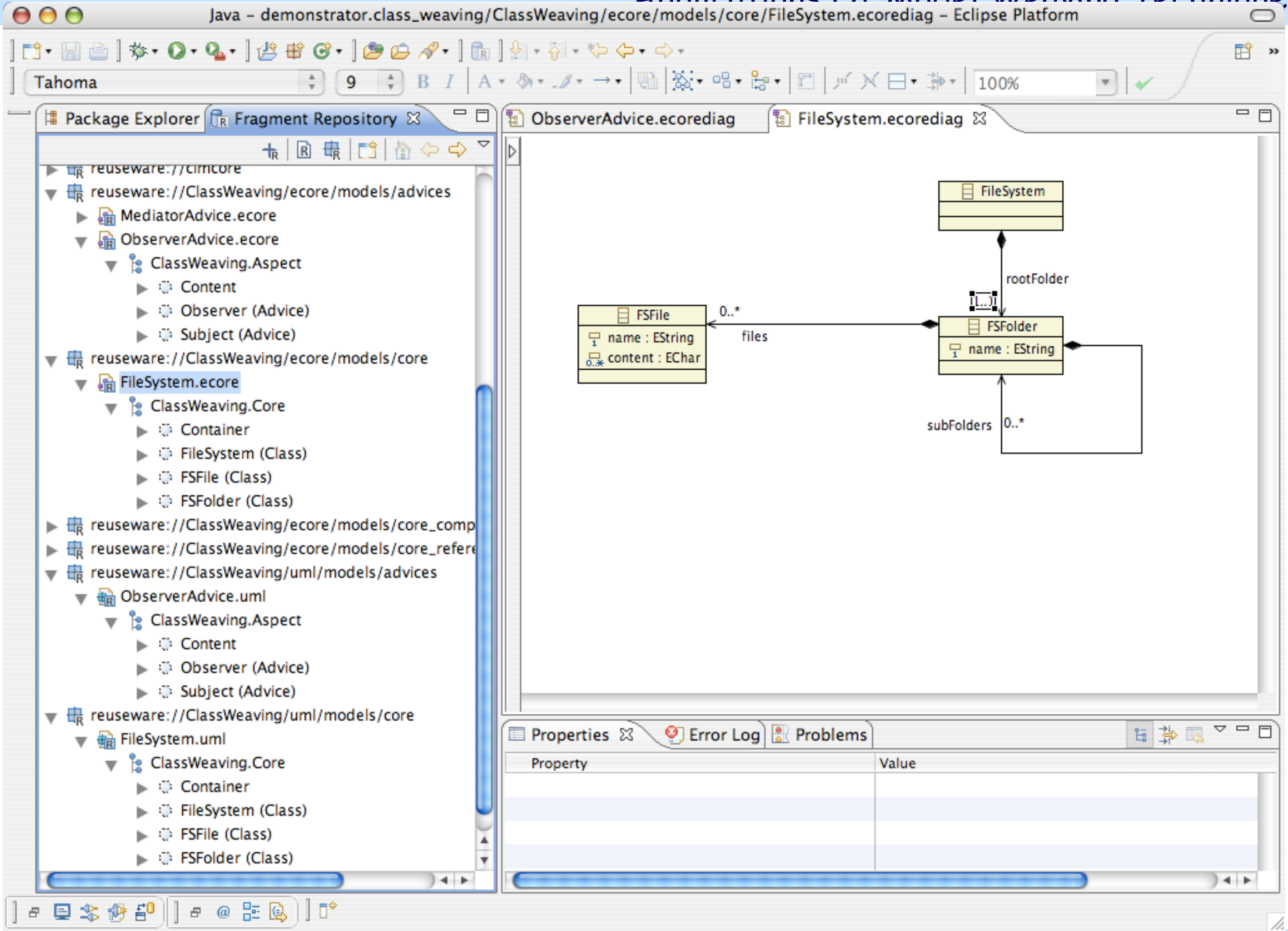
```
▶ reuseware://cimcore
▼ reuseware://ClassWeaving/ecore/models/advices
    ▶ MediatorAdvice.ecore
    ▼ ObserverAdvice.ecore
        ▼ ClassWeaving.Aspect
            ▶ Content
            ▶ Observer (Advice)
            ▶ Subject (Advice)
▼ reuseware://ClassWeaving/ecore/models/core
    ▼ FileSystem.ecore
        ▼ ClassWeaving.Core
            ▶ Container
            ▶ FileSystem (Class)
            ▶ FSFile (Class)
            ▶ FSFolder (Class)
▶ reuseware://ClassWeaving/ecore/models/core_comp
▶ reuseware://ClassWeaving/ecore/models/core_refere
▼ reuseware://ClassWeaving/uml/models/advices
    ▼ ObserverAdvice.uml
        ▼ ClassWeaving.Aspect
            ▶ Content
            ▶ Observer (Advice)
            ▶ Subject (Advice)
▼ reuseware://ClassWeaving/uml/models/core
    ▼ FileSystem.uml
        ▼ ClassWeaving.Core
            ▶ Container
            ▶ FileSystem (Class)
            ▶ FSFile (Class)
            ▶ FSFolder (Class)
```

adviceobserverpattern

Name: adviceobserverpattern

Ns Prefix: org.reuseware.example.ecore

Applications Of Model Weaving Techniques



Java – demonstrator.class_weaving/ClassWeaving/ecore/models/core/FileSystem.ecorediag – Eclipse Platform

Package Explorer | Fragment Repository

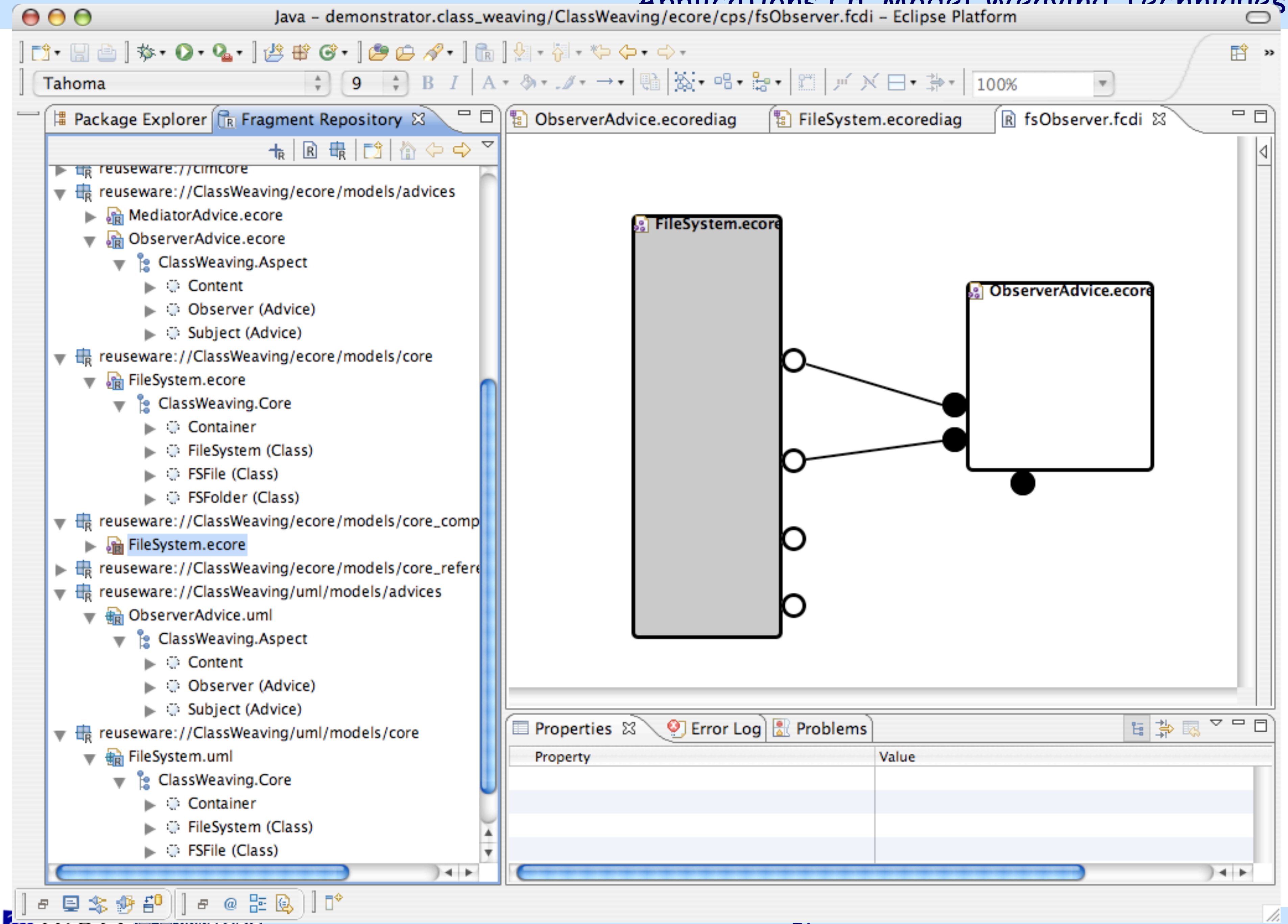
- reuseware://cimcore
- reuseware://ClassWeaving/ecore/models/advices
  - MediatorAdvice.ecore
  - ObserverAdvice.ecore
    - ClassWeaving.Aspect
      - Content
      - Observer (Advice)
      - Subject (Advice)
- reuseware://ClassWeaving/ecore/models/core
  - FileSystem.ecore
    - ClassWeaving.Core
      - Container
      - FileSystem (Class)
      - FSFile (Class)
      - FSFolder (Class)
- reuseware://ClassWeaving/ecore/models/core_comp
- reuseware://ClassWeaving/ecore/models/core_refere
- reuseware://ClassWeaving/uml/models/advices
  - ObserverAdvice.uml
    - ClassWeaving.Aspect
      - Content
      - Observer (Advice)
      - Subject (Advice)
- reuseware://ClassWeaving/uml/models/core
  - FileSystem.uml
    - ClassWeaving.Core
      - Container
      - FileSystem (Class)
      - FSFile (Class)
      - FSFolder (Class)

ObserverAdvice.ecorediag | FileSystem.ecorediag

Properties | Error Log | Problems

Property	Value

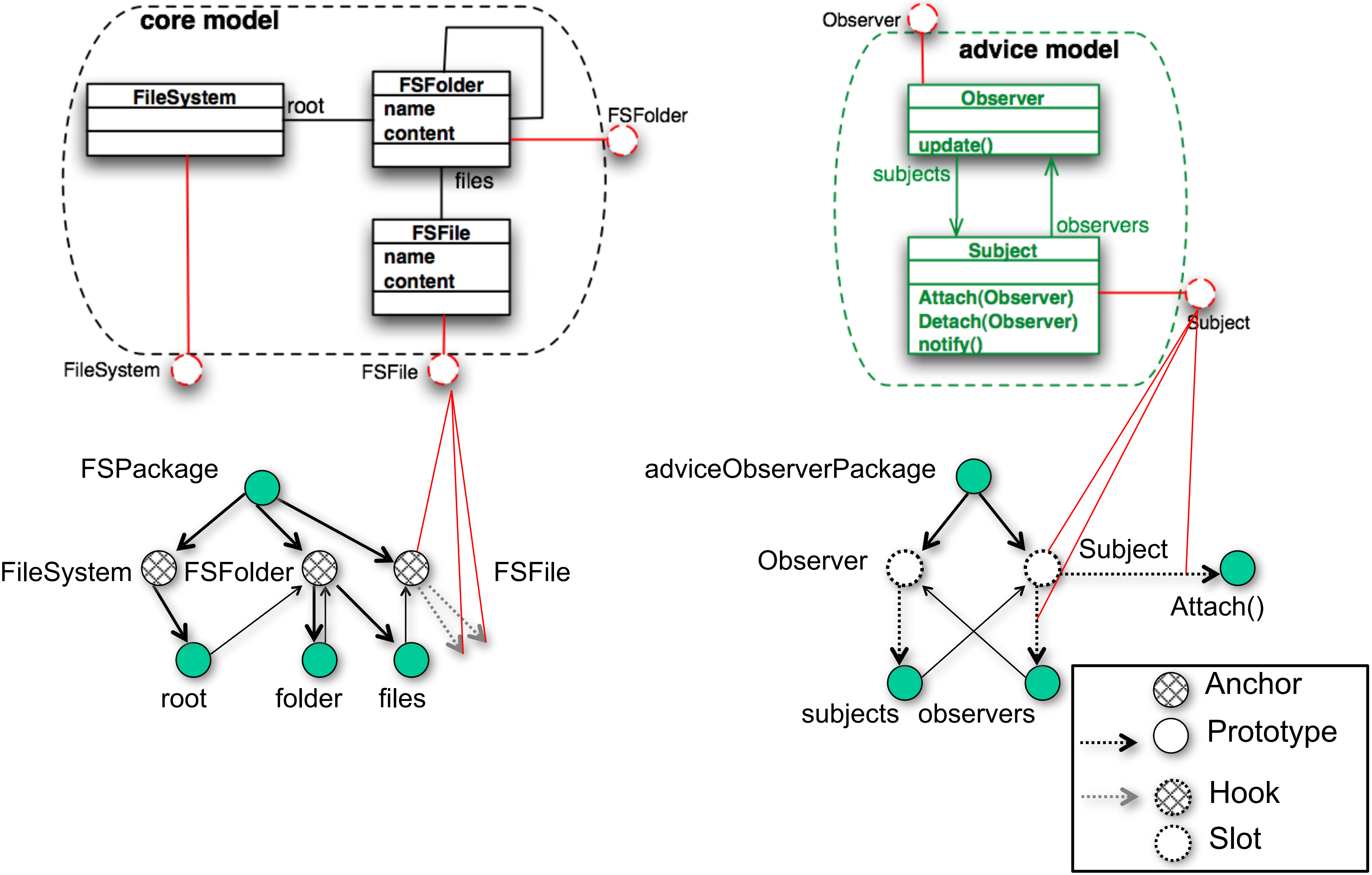
Applications Of Model Weaving Techniques



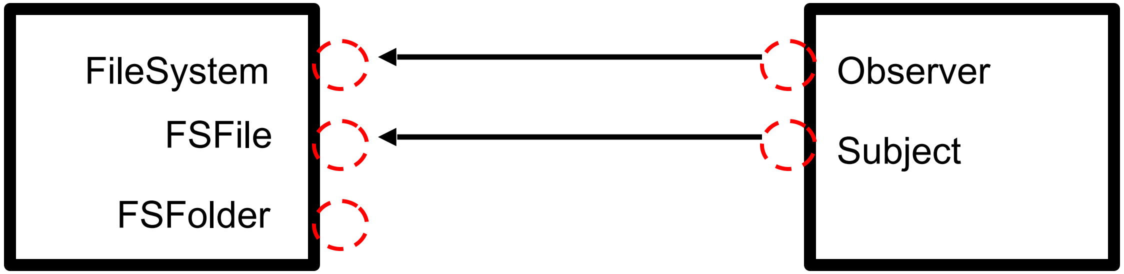
Java – demonstrator.class_weaving/ClassWeaving/ecore/cps/fsObserver.fcdi – Eclipse Platform

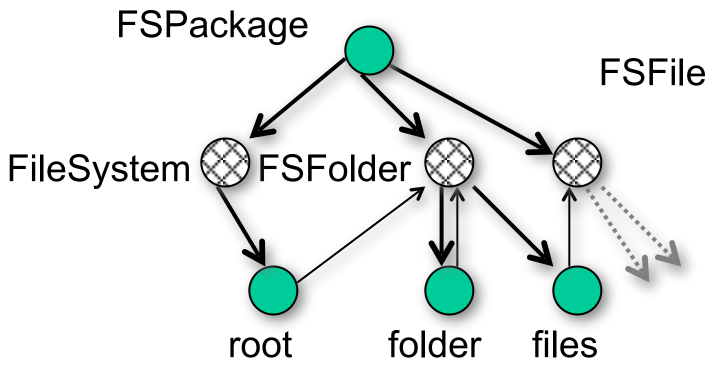
- reuseware://cimcore
- reuseware://ClassWeaving/ecore/models/advices
  - MediatorAdvice.ecore
  - ObserverAdvice.ecore
    - ClassWeaving.Aspect
      - Content
      - Observer (Advice)
      - Subject (Advice)
- reuseware://ClassWeaving/ecore/models/core
  - FileSystem.ecore
    - ClassWeaving.Core
      - Container
      - FileSystem (Class)
      - FSFile (Class)
      - FSFolder (Class)
- reuseware://ClassWeaving/ecore/models/core_comp
  - FileSystem.ecore
- reuseware://ClassWeaving/ecore/models/core_refere
- reuseware://ClassWeaving/uml/models/advices
  - ObserverAdvice.uml
    - ClassWeaving.Aspect
      - Content
      - Observer (Advice)
      - Subject (Advice)
- reuseware://ClassWeaving/uml/models/core
  - FileSystem.uml
    - ClassWeaving.Core
      - Container
      - FileSystem (Class)
      - FSFile (Class)

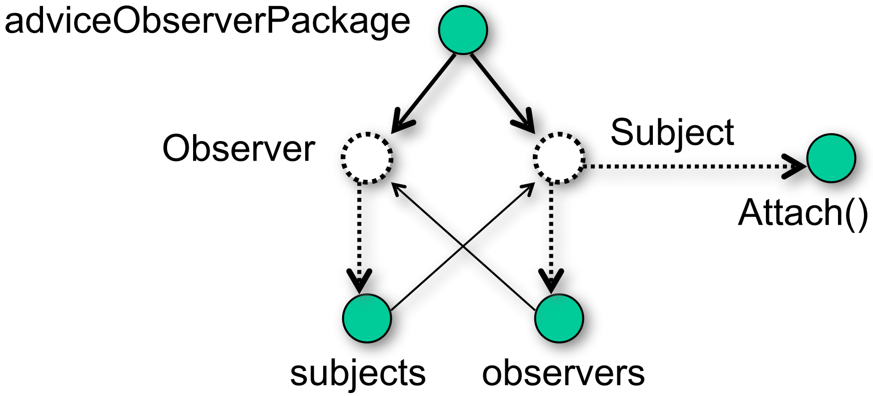
Reuseware Use Case: Example - Observer

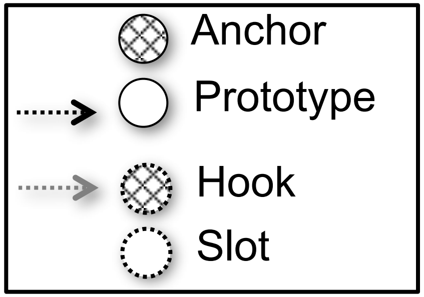


Reuseware Use Case: Example - Observer

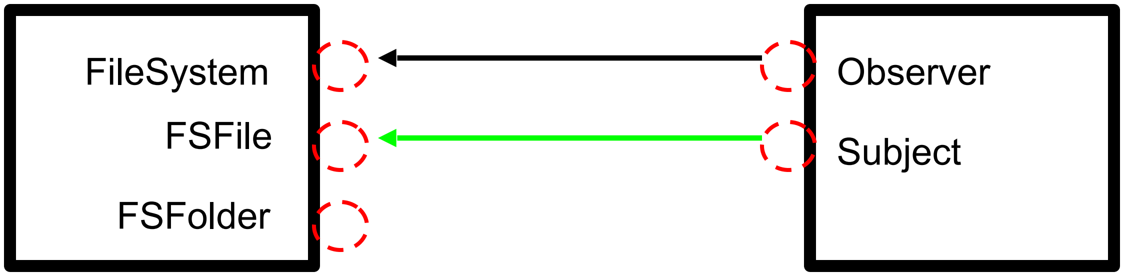


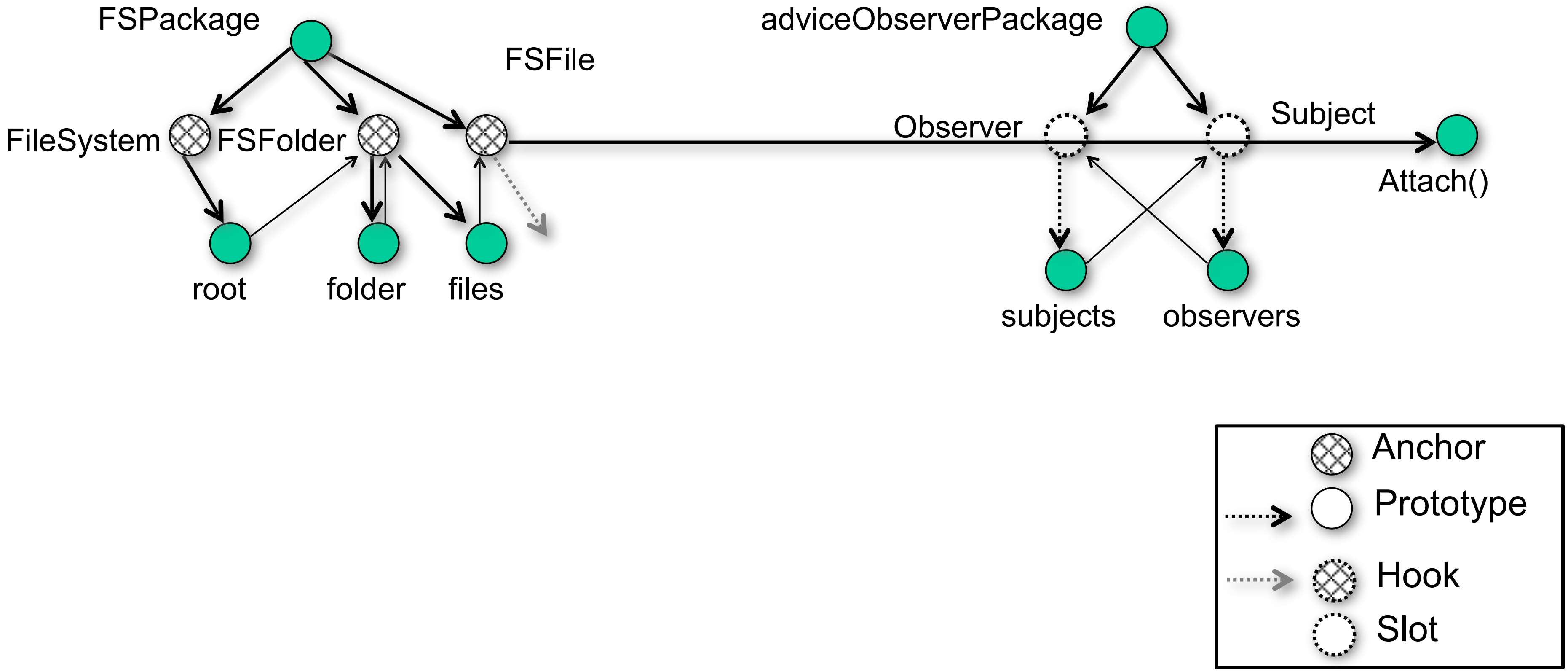




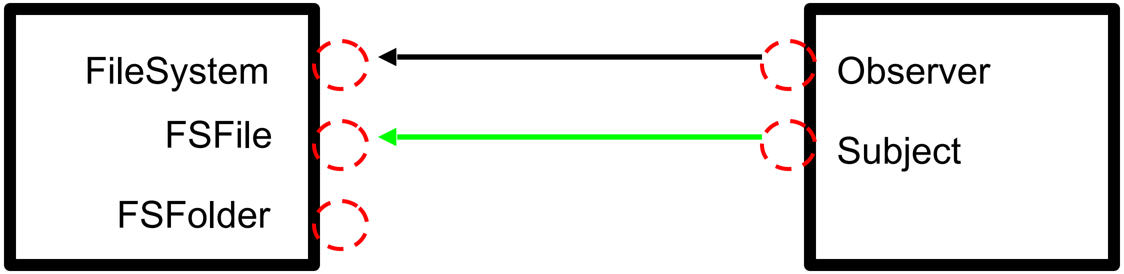


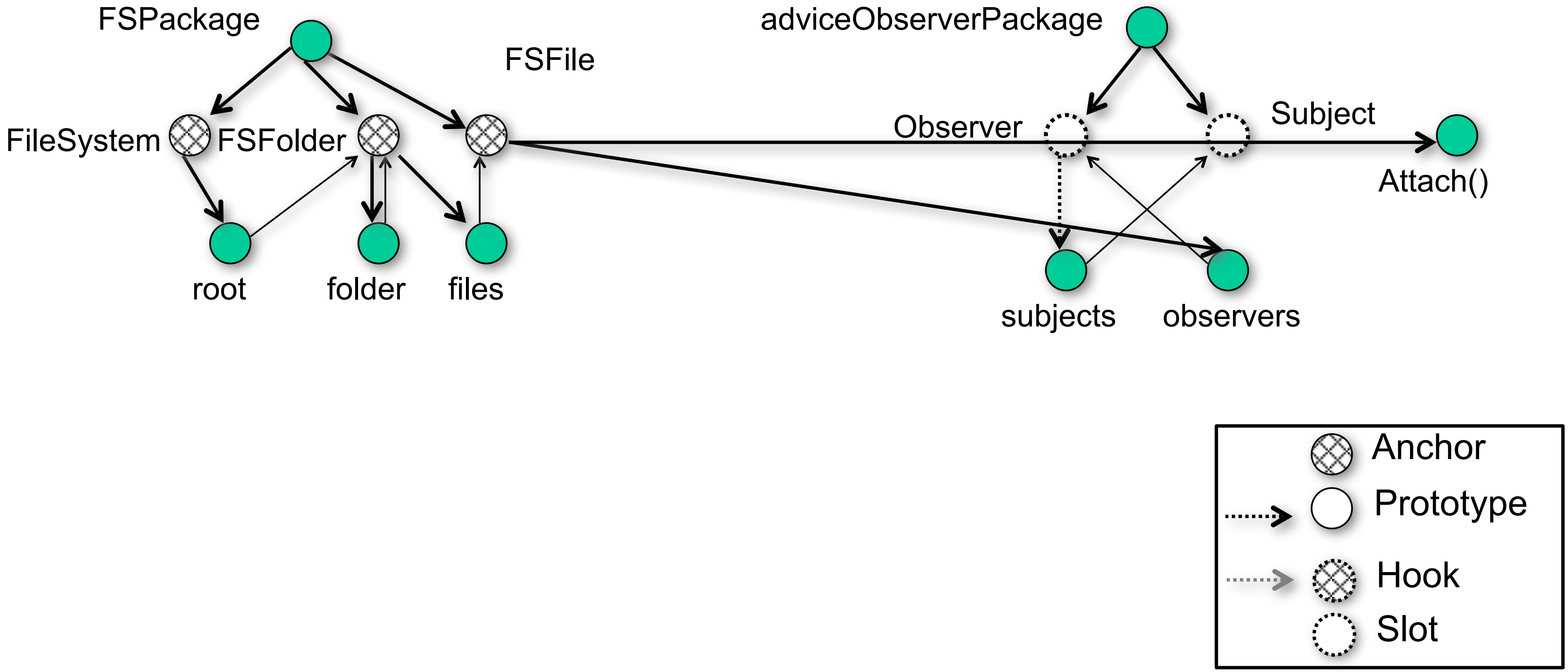
Reuseware Use Case: Example - Observer



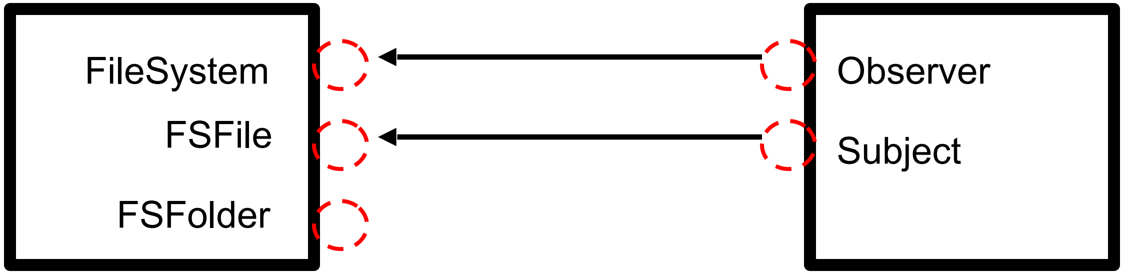


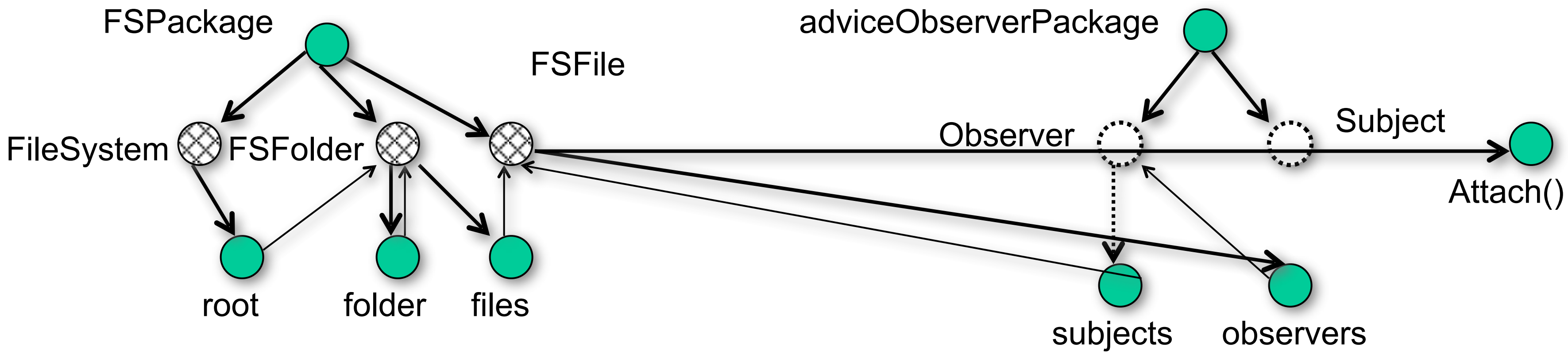
Reuseware Use Case: Example - Observer

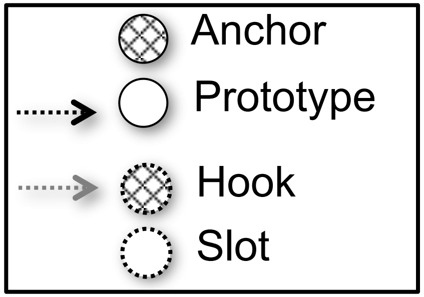




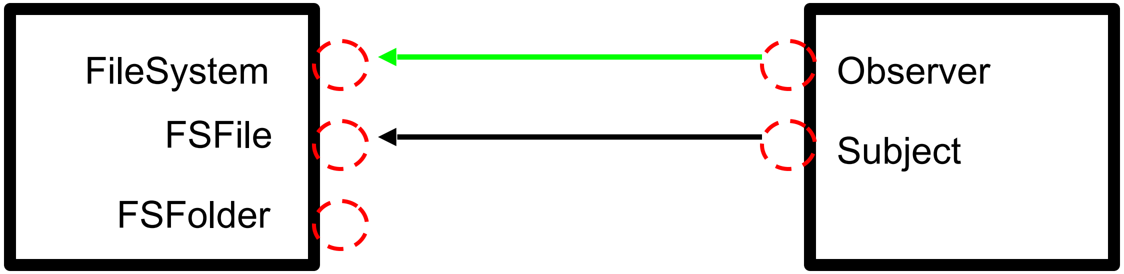
Reuseware Use Case: Example - Observer

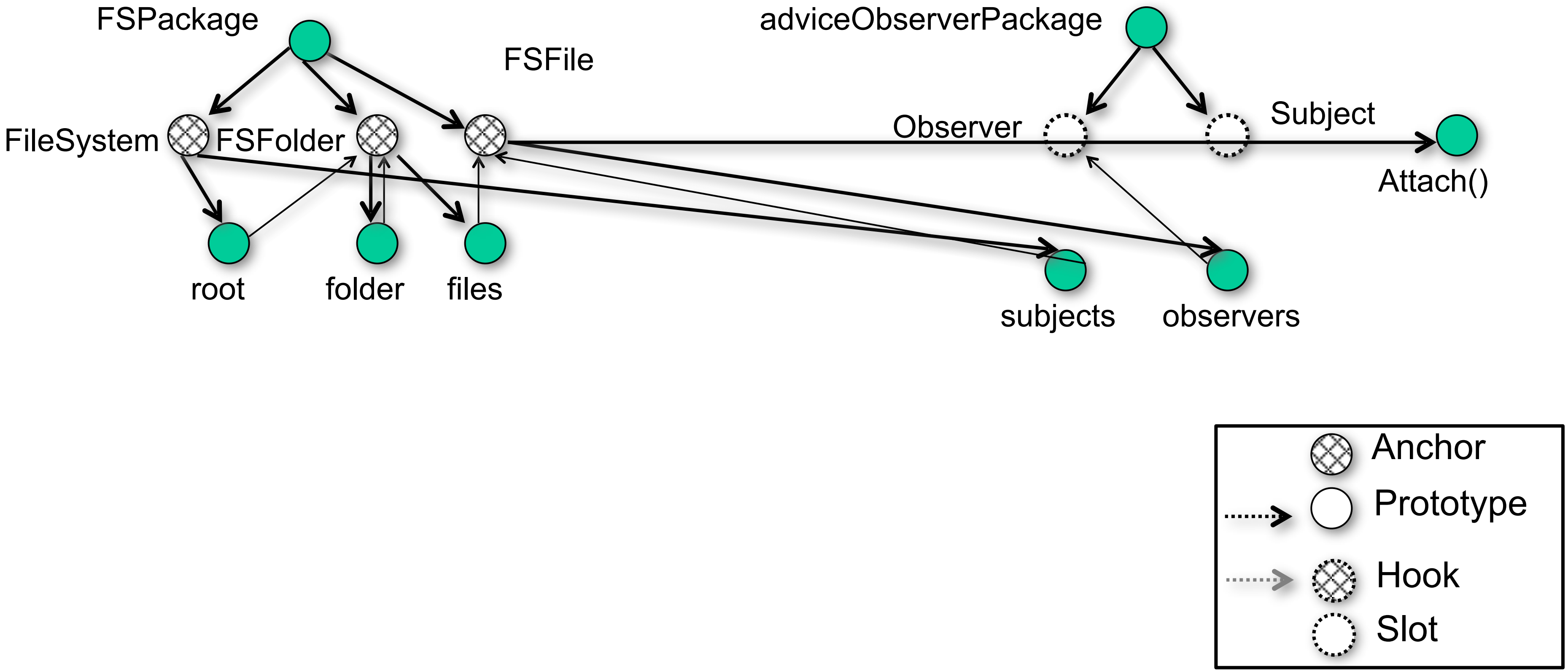




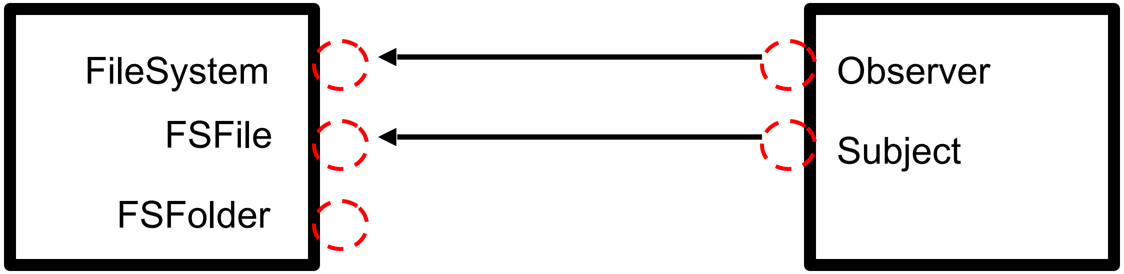


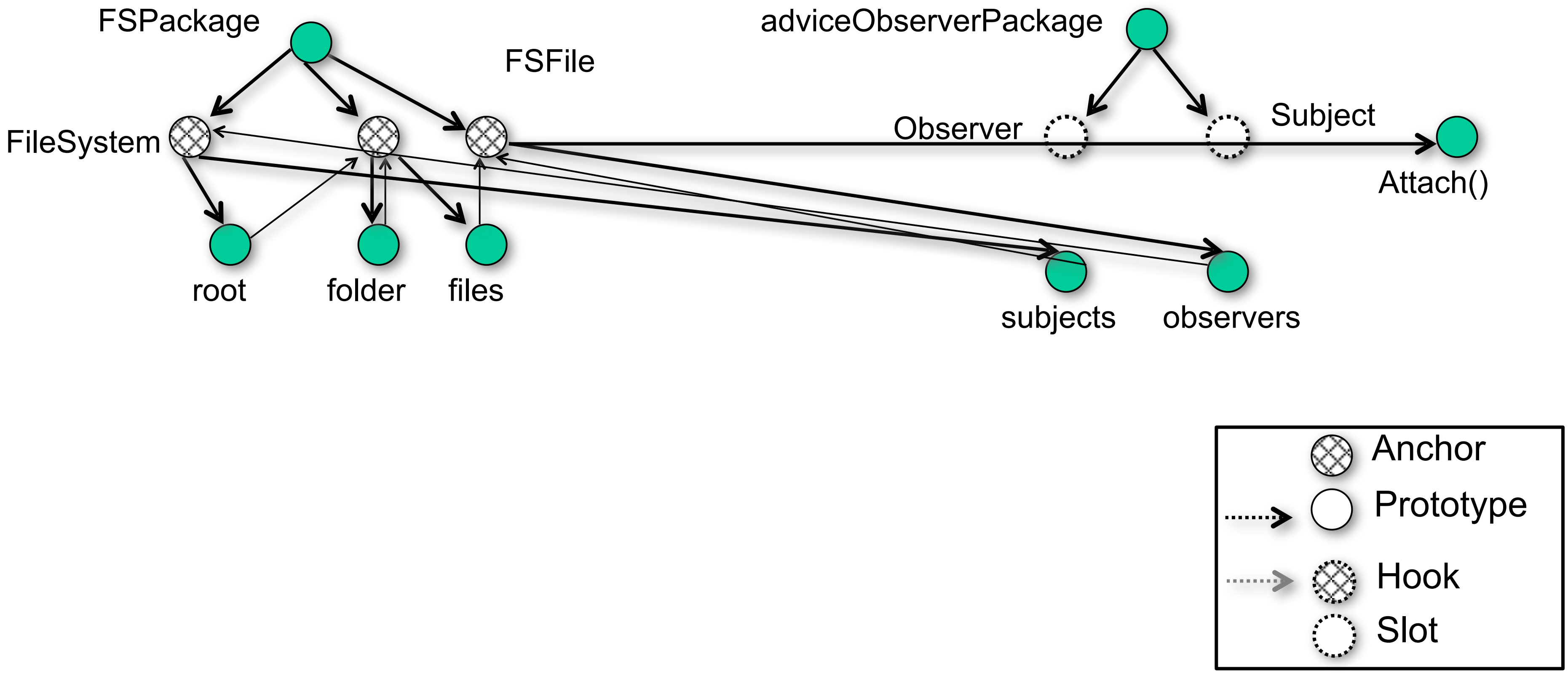
Reuseware Use Case: Example - Observer

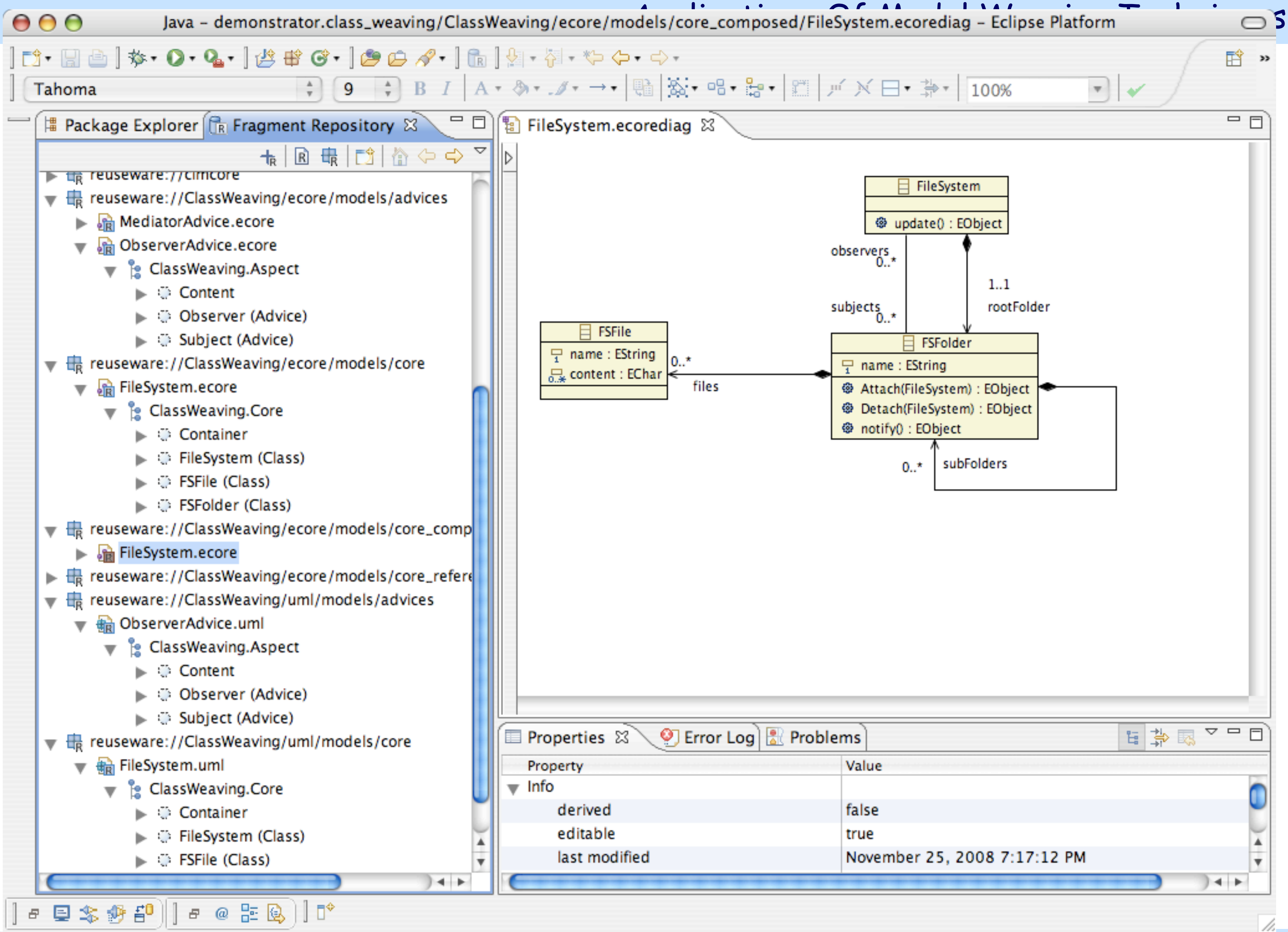




Reuseware Use Case: Example - Observer







Java – demonstrator.class_weaving/ClassWeaving/ecore/models/core_composed/FileSystem.ecorediag – Eclipse Platform

- reuseware://cimcore
- reuseware://ClassWeaving/ecore/models/advices
  - MediatorAdvice.ecore
  - ObserverAdvice.ecore
    - ClassWeaving.Aspect
      - Content
      - Observer (Advice)
      - Subject (Advice)
- reuseware://ClassWeaving/ecore/models/core
  - FileSystem.ecore
    - ClassWeaving.Core
      - Container
      - FileSystem (Class)
      - FSFile (Class)
      - FSFolder (Class)
- reuseware://ClassWeaving/ecore/models/core_comp
  - FileSystem.ecore
- reuseware://ClassWeaving/ecore/models/core_refere
- reuseware://ClassWeaving/uml/models/advices
  - ObserverAdvice.uml
    - ClassWeaving.Aspect
      - Content
      - Observer (Advice)
      - Subject (Advice)
- reuseware://ClassWeaving/uml/models/core
  - FileSystem.uml
    - ClassWeaving.Core
      - Container
      - FileSystem (Class)
      - FSFile (Class)

Property	Value
Info	
derived	false
editable	true
last modified	November 25, 2008 7:17:12 PM

Conclusion

- Model weaving is a core technique within MDE
  - Creation, representation, storage and use of the relationships between elements of different models
- Model weaving has a lot of different application domains
  - Traceability, tool interoperability, data integration transformation production, matching, composition, metamodel comparison, model alignment, etc
- The Eclipse-GMT AMW project:

  http://www.eclipse.org/gmt/amw/
- The Reuseware Composition Framework:

  reuseware.org