Model Driven Aspect Weaving

Prof. Jean-Marc Jézéquel
(Univ. Rennes 1 & INRIA)
Triskell Team @ IRISA
Campus de Beaulieu
F-35042 Rennes Cedex
Tel : +33 299 847 192 Fax : +33 299 847 171
e-mail : jezequel@irisa.fr
http://www.irisa.fr/prive/jezequel

Modeling in Engineering

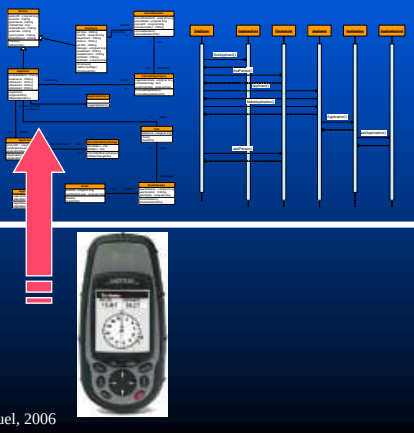
- A Model is a *simplified* representation of an *aspect* of the World for a specific *purpose*
  - Consider modeling both the machine & its environment (M. Jackson)
- UML paved the way from OOP to Model Based SE

Specificity of Engineering: Model something not yet existing(in order to build it)

M_1 (modeling space)

Is represented by

M_0 (the world)



Model and Reality in Software

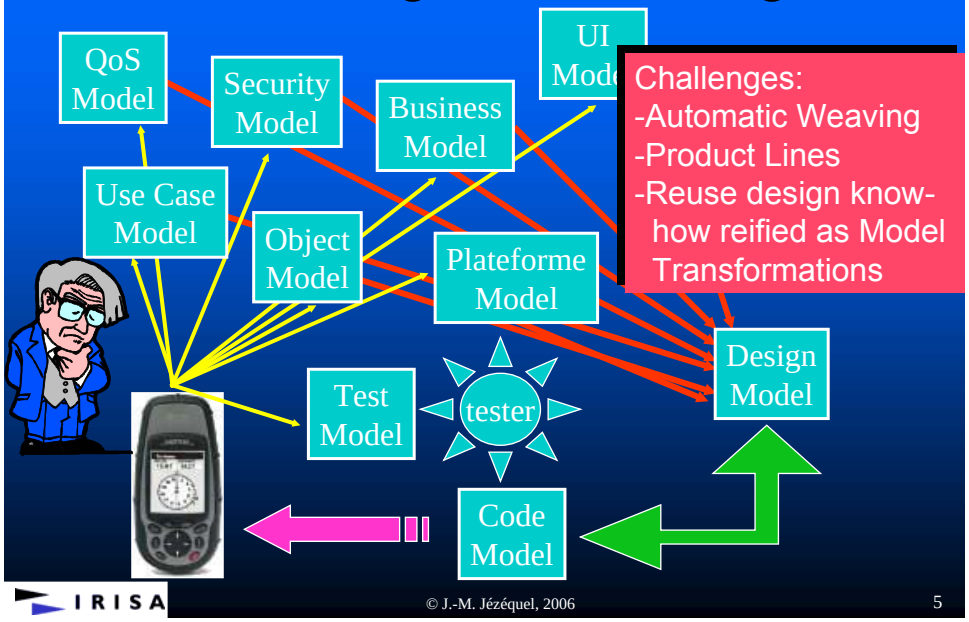
- Sun Tse: *Do not take the map for the reality*
- Magritte

- Software Models: from contemplative to productive

Principles of Model-Driven Engineering

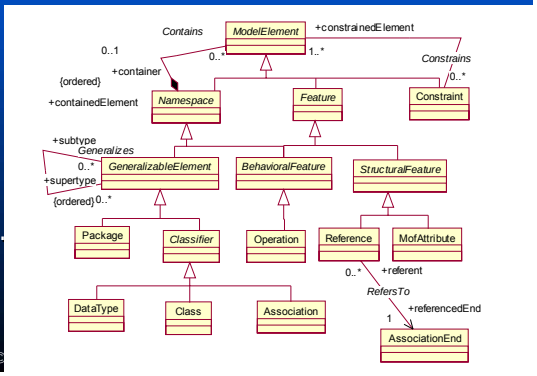
- A kind of (software) development approach
- Models as first class entities
- Everything is a model
- A model conforms to another model (meta-model)
- A model transformation takes models and produces models
- A model transformation is a model

Modeling and Weaving

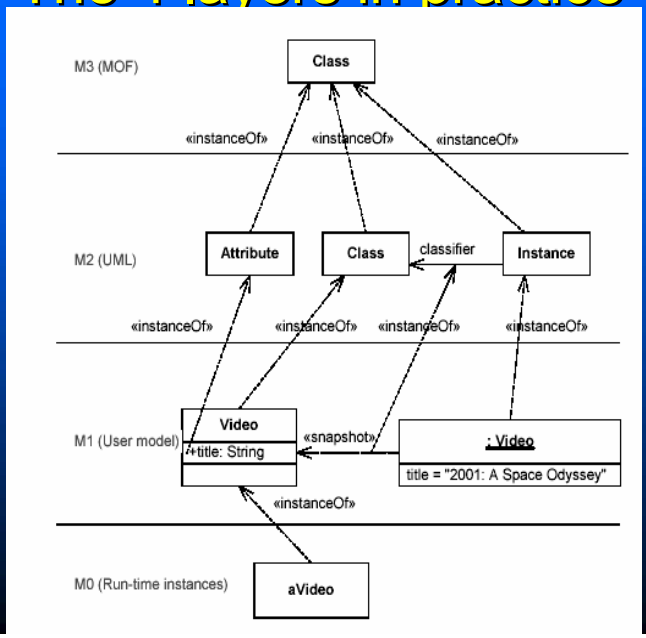


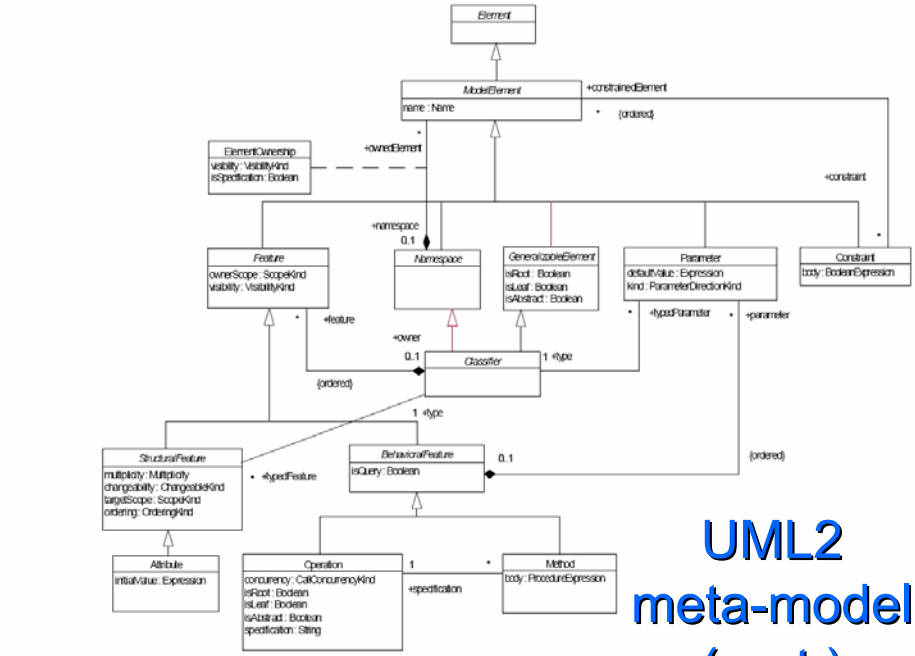
Assigning Meaning to Models

- If a UML model *is no longer* just
  - fancy pictures to decorate your room
  - a graphical syntax for C++/Java/C#/Eiffel...
- Then tools must be able to manipulate models
  - Let's make a model of what a model is!
  - => *meta-modeling*
    - » & meta-meta-modeling...

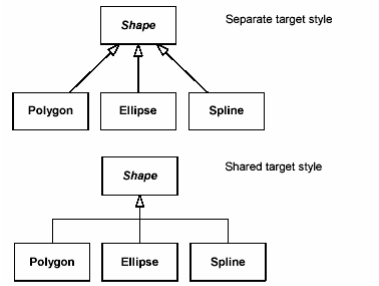


The 4 layers in practice



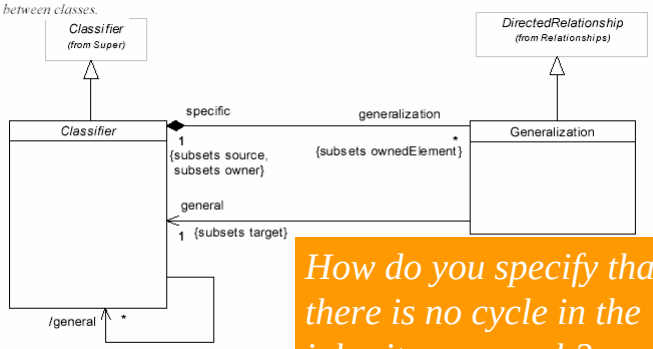


UML2 meta-model (part.)



Example: Generalizations in UML

Figure 3-33. Examples of generalizations between classes.



How do you specify that there is no cycle in the inheritance graph?

Figure 3-32. The elements defined in the Generalizations package.

IRISA

Static Semantics with OCL

- Complementing a meta-model with Well-Formedness Rules, aka *Contracts* e.g.;
  - ModelElement has a unique name in a Namespace
  - no cycle in a UML2 inheritance graph…
- Expressed with the OCL (Object Constraint Language)
  - The OCL is a language of typed expressions.
  - A constraint is a valid OCL expression of type Boolean.
  - A constraint is a restriction on one or more values of (part of) an object-oriented model or system.

IRISA

The core idea of MDA: PIMs & PSMs

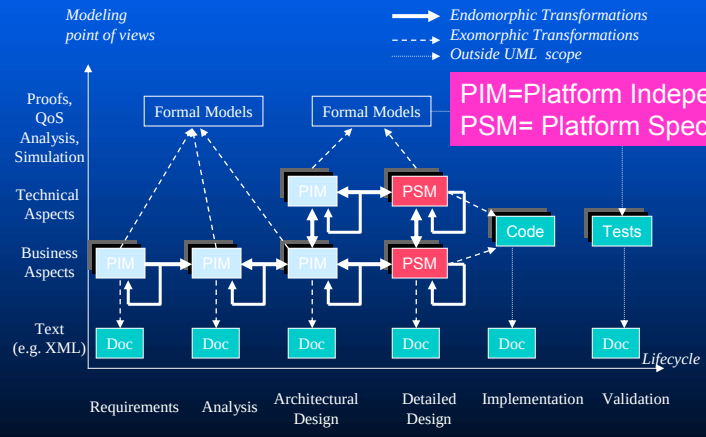
- MDA models
  - PIM: Platform Independent Model
    - » Business Model of a system abstracting away the deployement details of a system
    - » Example: the UML model of the GPS system
  - PSM: Platform Specific Model
    - » Operational model including platform specific aspects
    - » Example: the UML model of the GPS system on .NET
      - Possibly expressed with a UML profile (.NET profile for UML)
  - Not so clear about platform models
    - » Reusable model at various levels of abstraction
      - CCM, C#, EJB, EDOC, ...

How to go From PIM to PSM?

- "just" weave the platform aspect !
- How can I do that?
  - Through Model transformations
  - Now hot topic at OMG with RFP Q/V/T
    - » Query/View/Transformation

Weaving aspects into UML Models?

- It's what Model Driven Engineering is about!

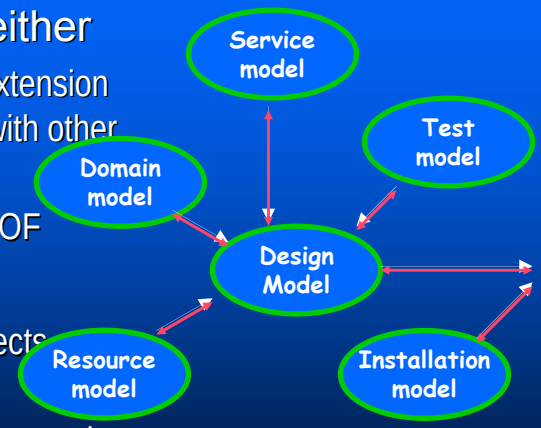


But many more dimensions in modeling!

- Beyond Design Model
  - where UML is arguably good…
- Business model
- GUI model
- Development process model
- Performance & Resource model
- Deployment model
- Test model
- Etc.

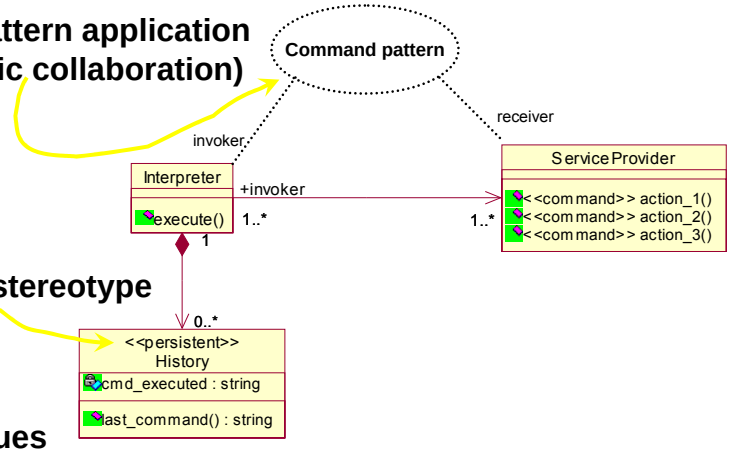
How to take these dimensions into account?

- Expression with either
  - UML, using built-in extension mechanisms to link with other semantic domains
  - DS(M)L, based on MOF
- Exploitation
  - Weave all these aspects into a design model
  - Define *mappings* between these aspects (as in e.g. *federations*)



Embedding implicit semantics into a model

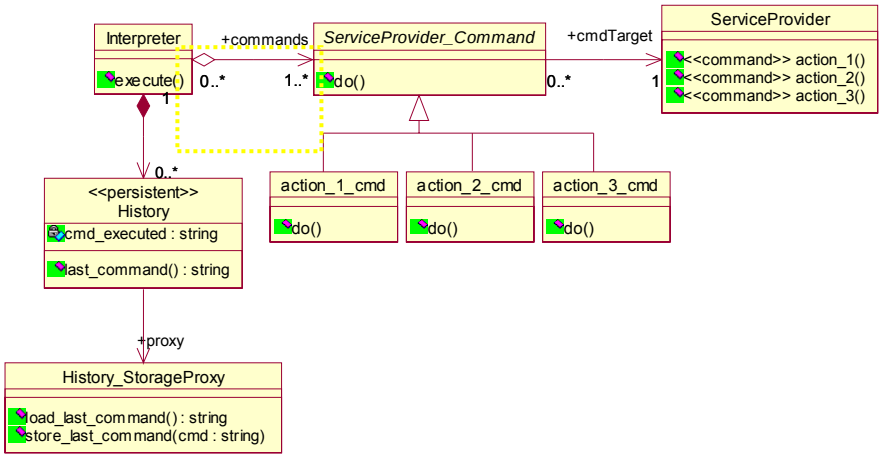
Design pattern application (parametric collaboration)



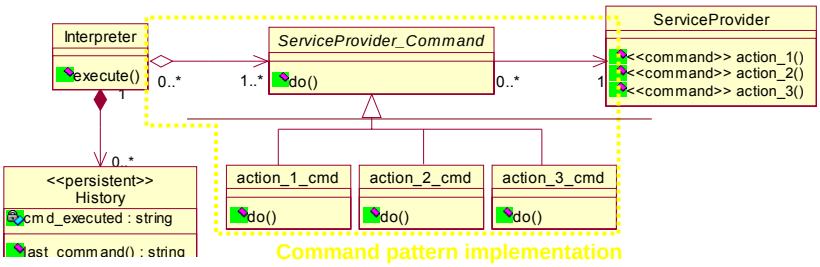
Element stereotype

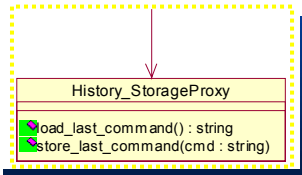
…and also Tagged values & Contracts

…and the result we want...



How To: Automatic Model Transformations



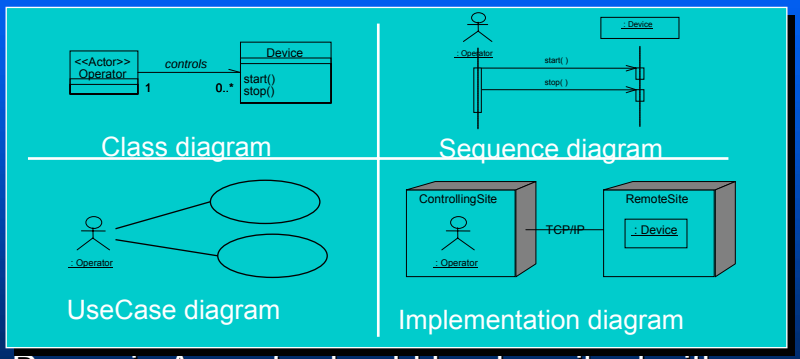


Persitence implementation

In some domains (e.g.; RT systems) transformations can get more complex than initial model!
=> must be managed with sound SE principles

Weaving Dynamic Aspects

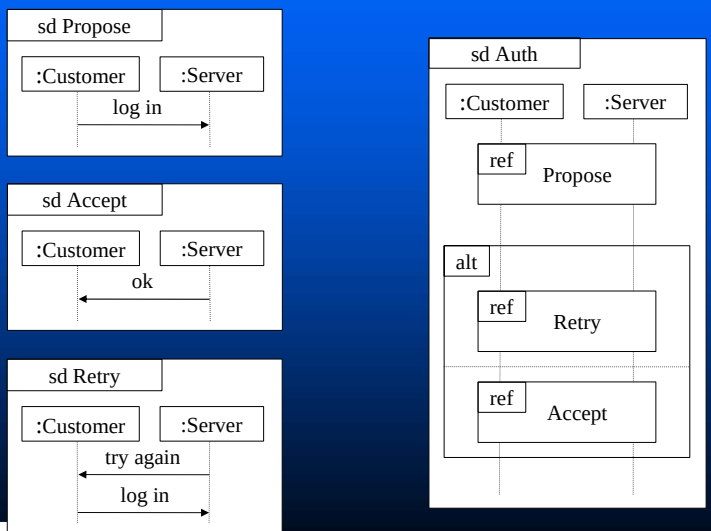
- UML is more than "just class diagrams"



- Dynamic Aspects should be described with Dynamic Diagrams
  - Not with weird stereotypes on static diagrams!

Example: Base Model

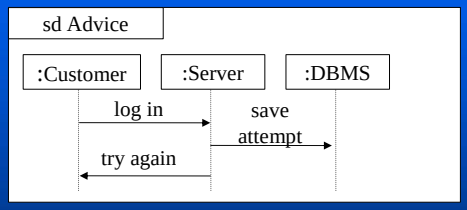
(based on J. Klein & F. Fleurey works)



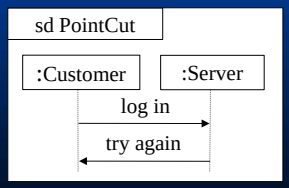
Example: Behavioral Aspect

Behavioral Aspect = Advice + Pointcut + Morphism

Advice

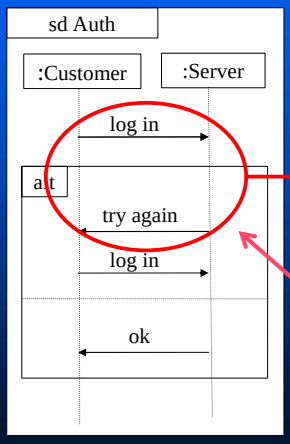


Pointcut

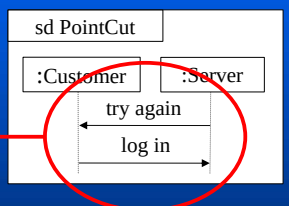


Example: Detection

Base Model



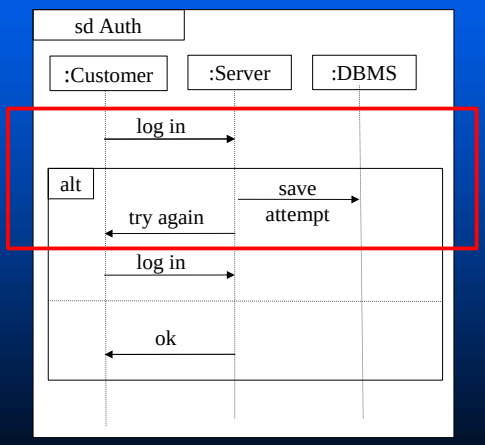
Pointcut



1 Joinpoint

Example: Composition

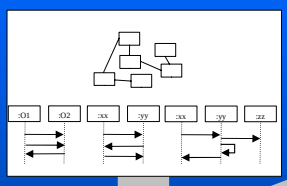
Result Model



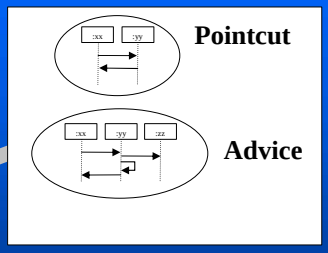
Advice

Overview

Base Model



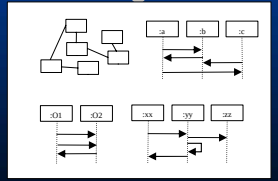
Behavioral Aspect



Pointcut

Advice

Weaving



1 – Detection

2 – Composition

Modèle tissé

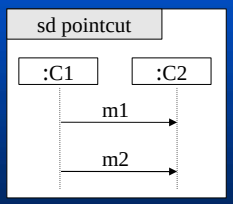
Detection

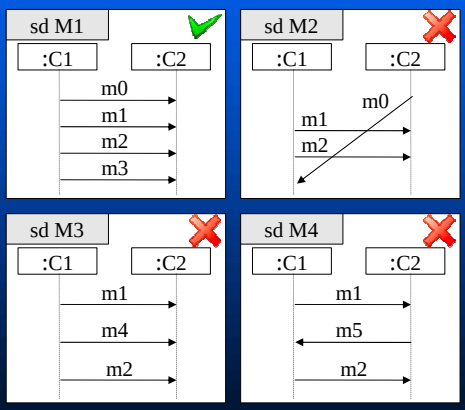
- Pointcut : Sequence diagram
- Static Analysis to find Joinpoints
- Three possible strategies
  - Sub sequence diagram
  - Closed part
  - Pattern

Detection Strategies (1)

detection

- Sequence Sub-diagram

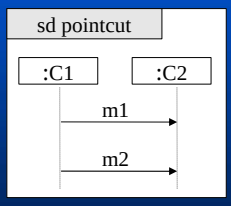


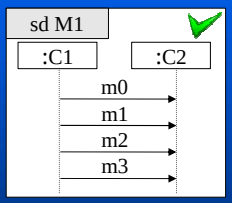


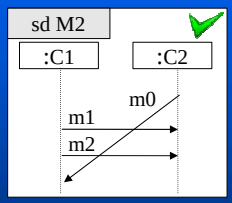
Detection Strategies (2)

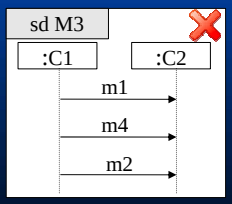
detection

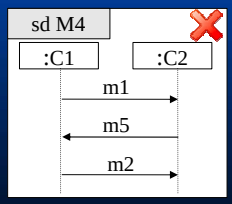
- Closed Part









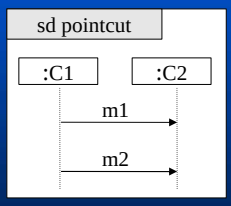


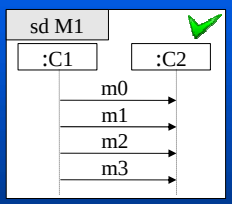
IRISA

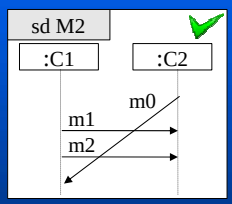
Detection Strategies (3)

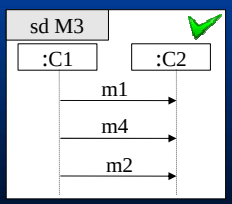
detection

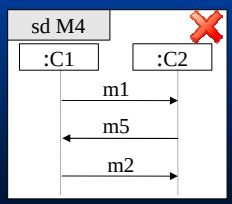
- Pattern







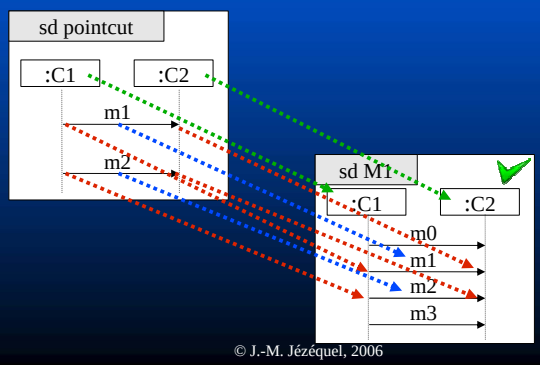




IRISA

Detection

- A morphism for each Joinpoint
  - Morphism of instances
  - Morphism of events
  - Morphism of messages



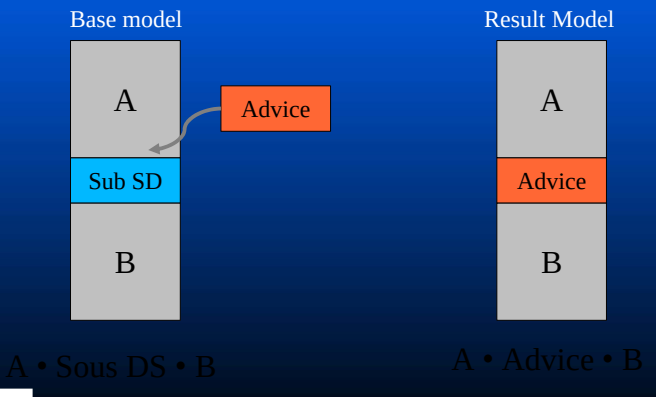
Composition

- Compose the advice into the base model
- Depend on the detection strategy

Composition

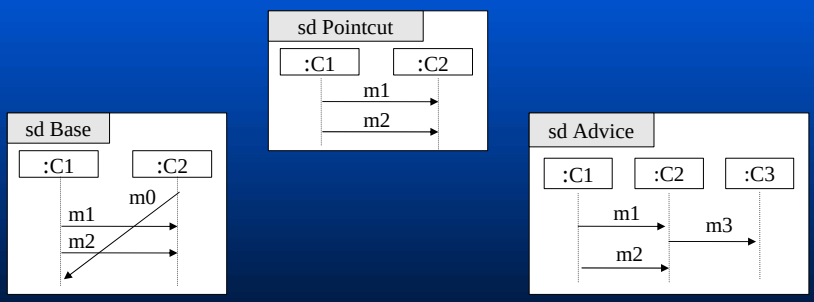
Composition

- Sequence sub-diagram
  - ➡ Weak Sequential Composition

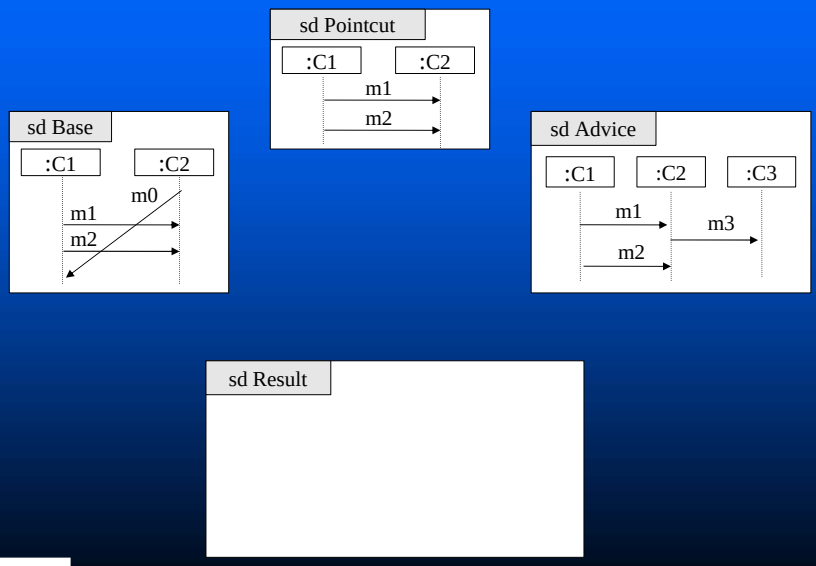


Amalgamed Sum

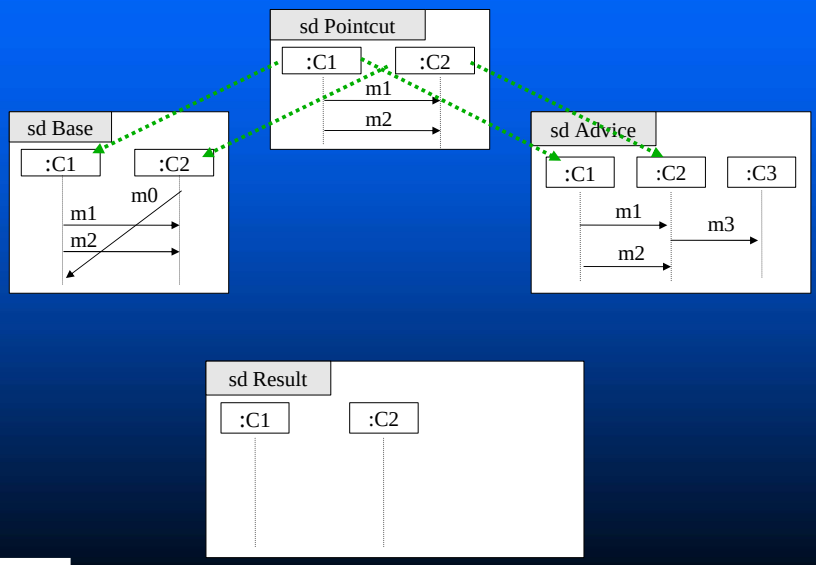
- For matching on Closed Part & Patterns
  - ➡ Simple composition not possible



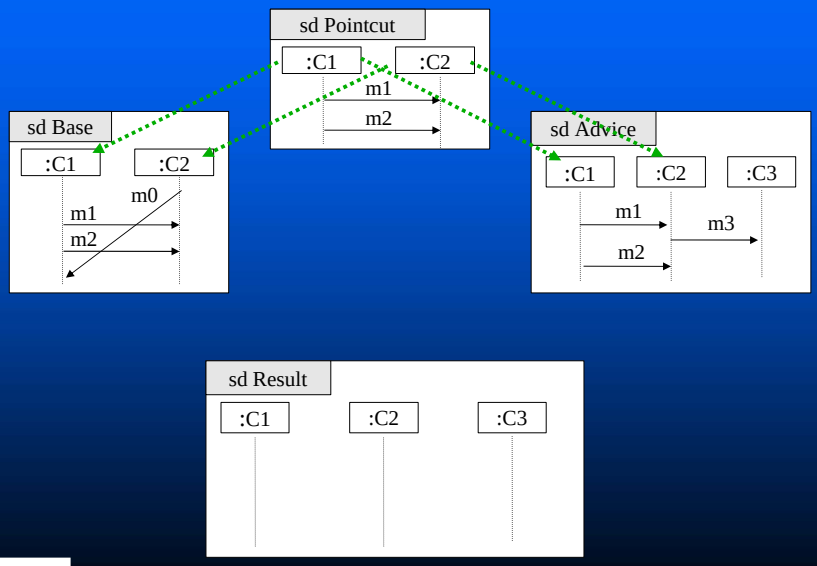
Amalgamed Sum



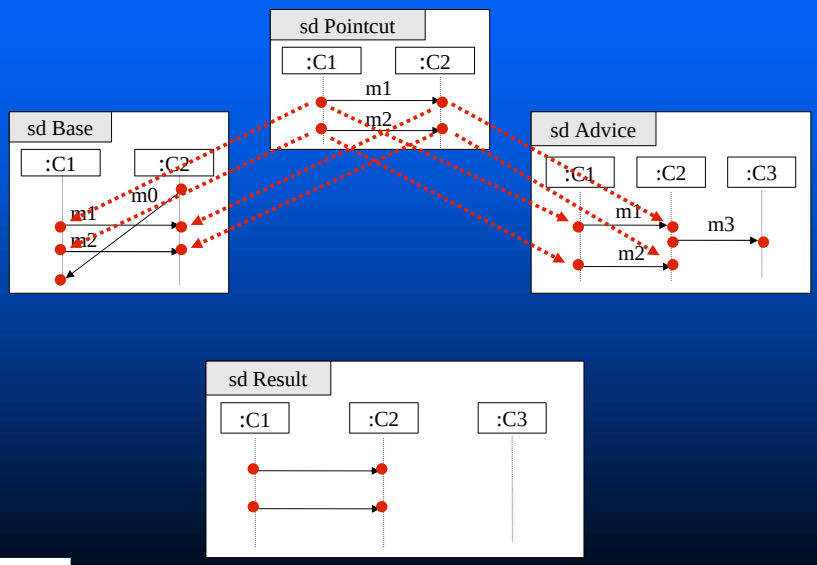
Amalgamed Sum



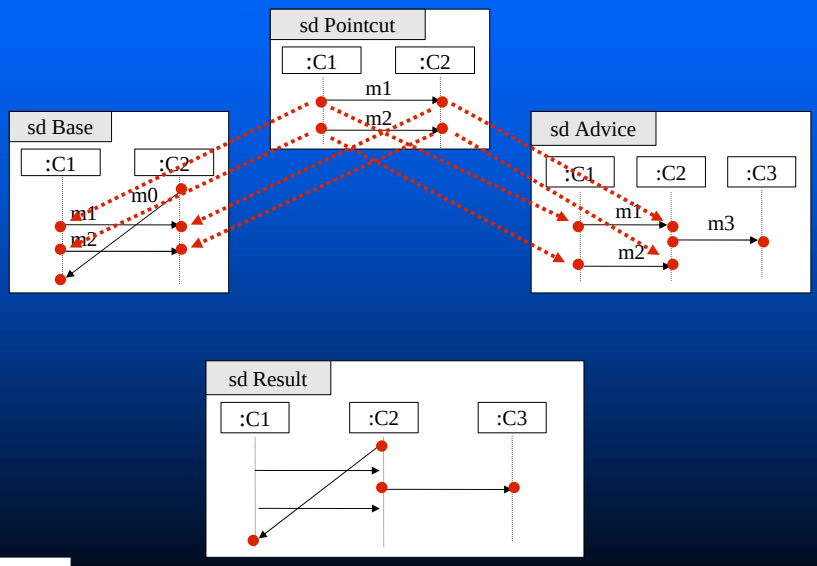
Amalgamed Sum



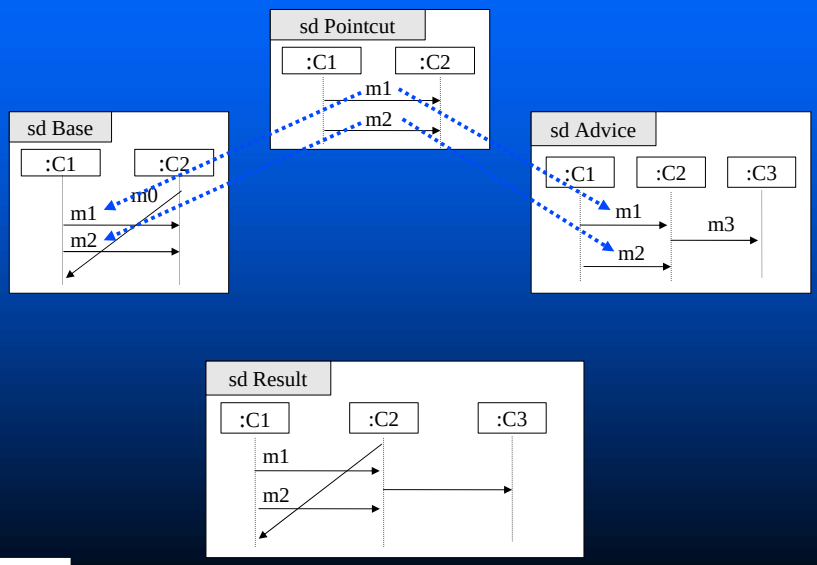
Amalgamed Sum



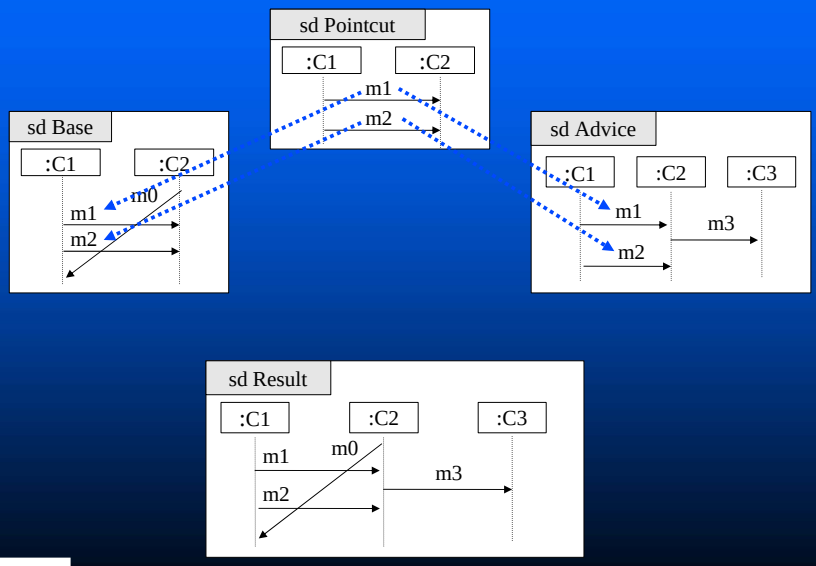
Amalgamed Sum



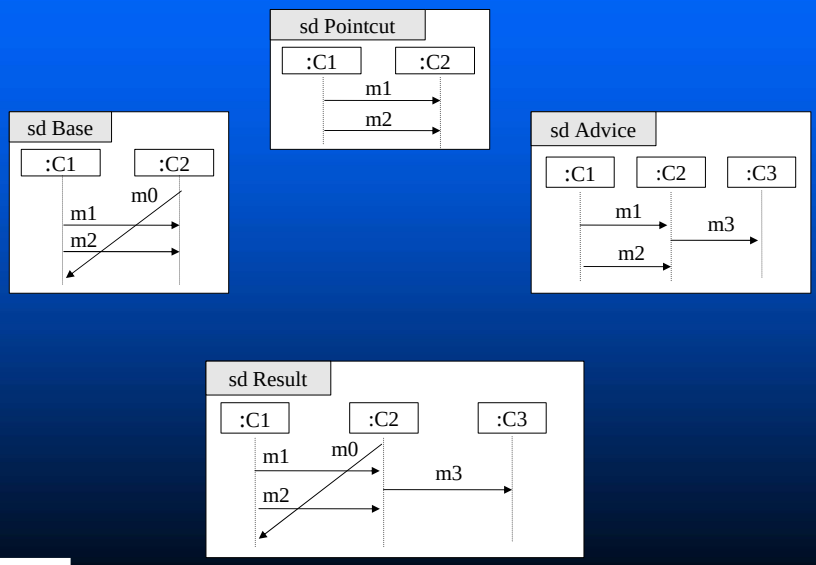
Amalgamed Sum



Amalgamed Sum



Amalgamed Sum



Tooling issues

- Aspect Weaving at Model Level
  - Rely on sophisticated model transformations
  - Sometime algorithmically complex
    - » Specifically for the static weaving of dynamic aspects

Model-to-Text vs. Model-to-Model

- Model-to-Text Transformations
  - For generating: code, xml, html, doc.
  - Should be limited to syntactic level transcoding
- Model-to-Model Transformations
  - PIM to PSM a la OMG MDA
  - Refining models
  - Reverse engineering (code to models)
  - Generating new views
  - Applying design patterns
  - Refactoring models
  - Deriving products in a product line
  - … any model engineering activity that can be automated…

Model-to-Text Approaches

- For generating: code, xml, html, doc.
  - Visitor-Based Approaches:
    - » Some visitor mechanisms to traverse the internal representation of a model and write code to a text stream
    - » Iterators, Write ()
  - Template-Based Approaches
    - » A template consists of the target text containing slices of meta-code to access information from the source and to perform text selection and iterative expansion
    - » The structure of a template resembles closely the text to be generated
    - » Textual templates are independent of the target language and simplify the generation of any textual artefacts

Classification of Model Transformation Techniques

1. General purpose programming languages
   - Java/C#...
2. Generic transformation tools
   - Graph transformations, XSLT…
3. CASE tools scripting languages
   - Objecteering, Rose…
4. Dedicated model transformation tools
   - OMG QVT style
5. Meta-modeling tools
   - Metacase, Xactium, Kermeta…

General purpose language approach:

- No overhead to learn a new language
- Tool support to write the transformations

=> Monsieur Jourdain's approach

- But wait:
  - We resort to modeling (before programming) for mastering complexity, right?
  - Does it mean that transformations never get complex?
    - » Or that the problem only appears for non toy applications…

Generic transformation tools: Conclusion

- Awk-like (inc. *sed, perl*…) SE Limit: ~100 LOC
- XSLT SE Limit: ~1000 LOC
  - Good for syntactic transcoding, not for semantic manipulations
- Graph Transformation tools SE Limit: PhD needed!
  - Powerfull, but complex because of the non-determinism in scheduling and application strategy
  - Require careful consideration of termination of the transformation process and the rule application ordering

CASE tools scripting languages

- Pro
  - Good level of maturity
  - Excellent integration with their CASE tool
- Drawbacks
  - Proprietary languages and/or tight coupling with the CASE
  - Often developed as a second thought, not central
  - Many limitations when model transformation get complex
    - » Structuration, modularity, reuse problem
    - » Configuration management issue when they need to be evolved and maintained for long periods

4. Dedicated Transformation Language

- OMG QVT style
  - Kind of DSL for transformation
- Simplify development and maintenance of model-transformations
- Higher expression power
- Enhanced structuration
  - Composition of rules
  - Interoperability

M2M: Relational Approaches

- Declarative, based on mathematical relations
  - Good balance between flexibility and declarative expression
- Implementable with logic programming
  - Mercury, F-Logic programming languages
  - Predicate to describe the relations
  - Unification based-matching, search and backtracking

M2M : Structure-Driven Approaches

- Pragmatic approaches developed in the context of EJB and Databases schema generation from UML models
- Strong support for 1-to-1 and 1-to-n correspondence between source and target
- Unclear how well these approaches can support other kinds of applications

Dedicated model transformation tools: Conclusion

- How many developers are familiar with the prolog-like style of rules writing?
- Where is the advantage of a dedicated explicit language vs. a general purpose language?
- Hybrid Languages or transformation libraries for general purpose languages…

5. Meta-modeling tools

- **Build (OO) Models of Transformations**
- **Use MDE to run them**
- **Commercial tools:**
  - MetaEdit+ from MetaCase
  - XMF-Mosaic from Xactium
- **Open-Source**
  - KerMeta from INRIA
  - www.kermeta.org

KerMeta in a NutShell

- EMOF superset
  - Any EMOF MetaModel is a valid KerMeta program, and conversely
- Object-Oriented
  - Multiple inheritance / behavior selection
  - Operation overiding / late binding
  - Full reflection (read-only at this time)
- Statically Typed
  - Generics
  - Function types to allow OCL's *forall/exist/iterate*

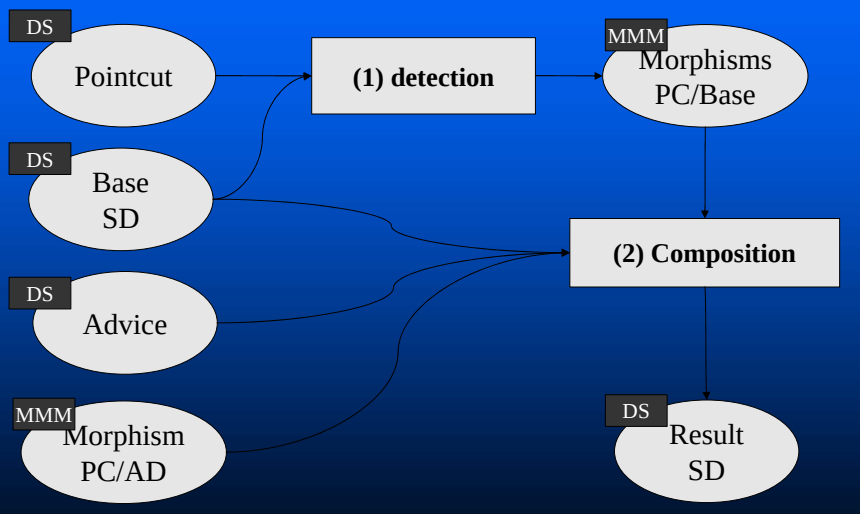
Current Status

- Latest version (0.2.2)
  - Parser, type checker, interpreter, debugger
  - Eclipse plug-in: Textual Editor, Browser, Launcher
  - EMF Ecore metamodel Import / Export
  - EMF model Import / Export
  - Documentation and Examples
  - OCL Constraints Checking
- Under development / test
  - Graphical Editor (generated with Topcased)
  - Seamless import of Java classes in Kermeta
  - Model type

"Programming style" Issues

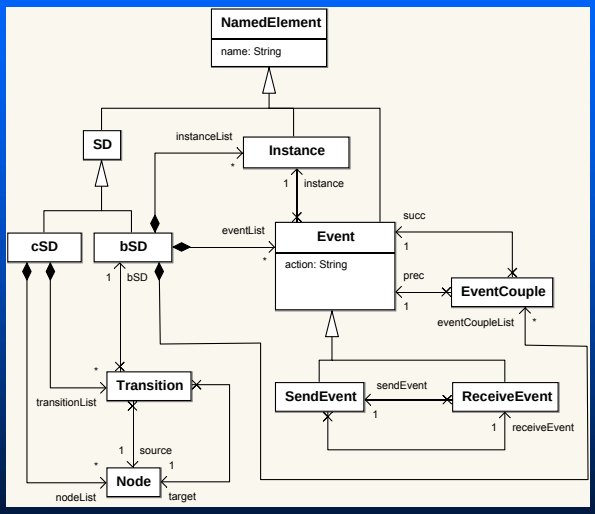
- The transformation is simply the model of an object-oriented program that manipulates model elements
  - Navigation through model is first class though (like in OCL)
- OO techniques
  - Customizability through inheritance/dyn. binding
  - Pervasive use of GoF like Design Patterns

Implementation

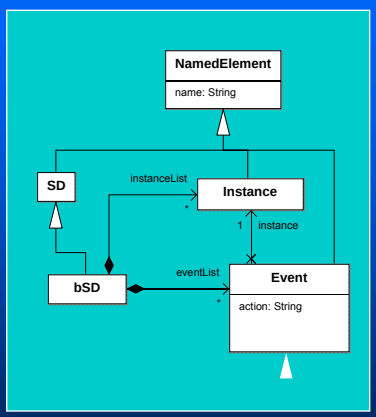


Defining the metamodels

input and output metamodels are the same

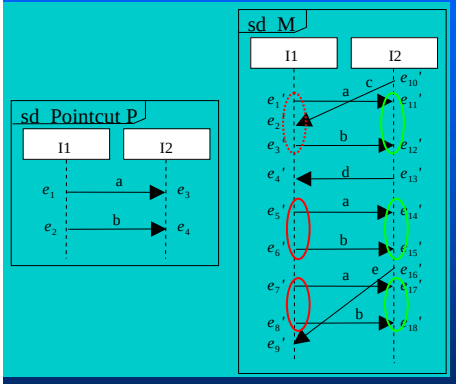


Visual/Textual



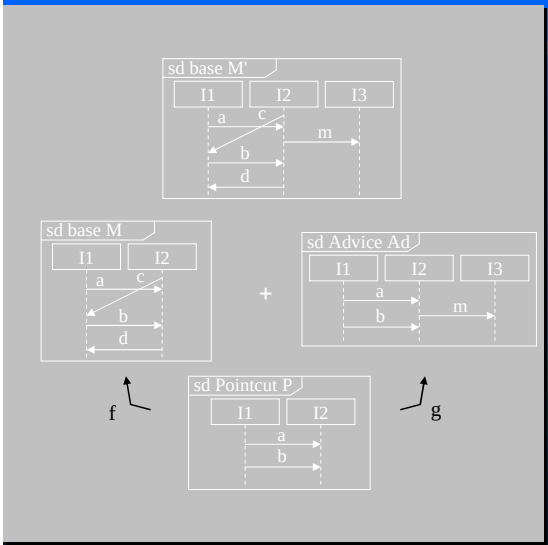

```
package bigSd;
using kermeta::standard
using kermeta::persistence
abstract class NamedElement
{
    attribute name : String[1..1]
}
abstract class SD inherits NamedElement
{}
class BSD inherits SD
{
    attribute events : Event[0..*]
    attribute couples : EventCouple[0..*]
    reference instances : Instance[0..*]
                                …
}
abstract class Event inherits NamedElement
{
    attribute action : String[1..1]
    reference onInstance : Instance[1..1]
    …
}
```

Sequence Diagrams Weaving



- Choice of the join point policy
- Detection step:
  - for each object, we compute the sets of (successive or not) events which have the same label as the events of P
  - compute the minimum set of events, called Jm, which satisfies the properties related to the join point policy
  - build isomorphism μ from P to Jm
  - repeat on M-Jm while Jm is not empty

Sequence Diagrams Weaving

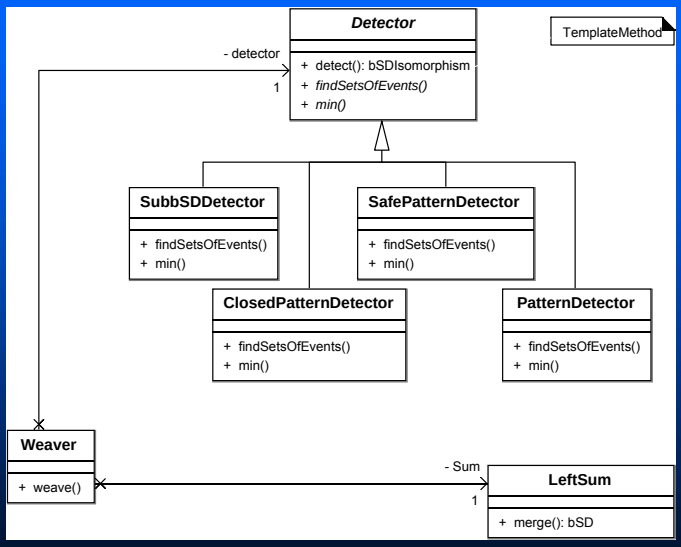


- Composition step: amalgamated sum
  - use P and two morphisms f and g (f being computed by the detection step)
  - keep the common parts between M and Ad
  - add new object I3
  - add the events of M by respecting the order specified on M
  - add the events of Ad by respecting the order specified on the advice

Object-orientation

- Classes and relations, multiple inheritance, late binding, static typing, class genericity, exception, typed function objects
- OO techniques such as patterns, may be applied to model transformations
  - Template method to encapsulate basic *detect* algorithm
    - » Substeps redefined in subclasses

SD Weaver Architecture



Writing the transformation: Weaver

```
require kermeta   require "../models/bigSd.kmt" require "../detectionAlgorithm/Detection.kmt"
require "../amalgamatedSum/LeftSum.kmt"
using kermeta::standard    using bigSd

class Weaver {
 operation weave(base : BSD, pointcut : BSD, advice : BSD, g : BSDMorphism) : BSD is do
        result := BSD.new                                                          Initialization
        //Choice of join point policy
        var detection: Detection init ClosedPatternDetection.new
        var sum: LeftSum init LeftSum.new
        var f: BSDMorphism init BSDMorphism.new
        var setOfMorphism : Set< BSDMorphism > init Set< BSDMorphism >.new

        //Detection Step                                                           Detection Step
        f:= detection.detect(pointcut, base)
        while (f != null)
              setOfMorphism.add(f)
              f:= detection.detect(pointcut, minus(base,f))
        end

        //Composition Step                                                       Composition Step
        setOfMorphism.each{f | result := sum.merge(result, pointcut, advice, f, g)
end
```


Writing the transformation: Detection

```
require kermeta     require "../models/bigSd.kmt"
using kermeta::standard     using bigSd
abstract class Detector{                                                  abstract method
  operation findSetOfEvent(evtsOfP: Set<Event>, evts: Set<Event>): Set<Set<Event>>is abstract
  operation min(setOfEvent: Set<Set<Event>>) : Set<Event> is abstract
  operation detect (pointcut : BSD, base : BSD) : BSDMorphisms is do
        result := BSDMorphisms.new                                              initialization
        var evts : Set<Event> init Set<Event>.new  var evtsOfP : Set<Event> init Set<Event>.new
        var V : Set<Set<Event>> init Set<Set<Event>>.new
        var setOfEvent: Set<Set<Set<Event>>> init Set<Set<Set<Event>>>.new
        pointcut.instances.each{ instance |
              //projection on an instance
              evts := base.events.select{e|e.onInstance== instance}
              evtsOfP := pointcut.events.select{e|e.onInstance== instance}
              //sets of events which have the same action name as the events of P on instance
              // findSetsOfEvent depends of the join point definition
              V := findSetsOfEvent(evtsOfP, evts)
              setOfEvent.add(V)
        }
      // take the first set of events satisfying the properties     // min depends of the join point definition
      Epart=min(setOfEvent)
      // build the isomorphism from pointcut to Epart
     result := buildIsomorphism(pointcut, Epart)
    end
```

Writing the transformation: amalgamated Sum

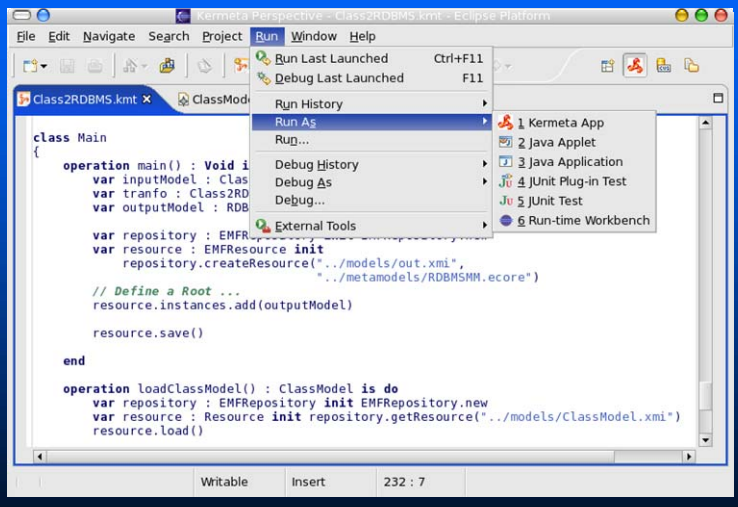
```
require kermeta   require "../models/bigSd.kmt"
using kermeta::standard    using bigSd
class LeftSum {
 operation merge(base : BSD, pointcut : BSD, advice : BSD, f : BSDMorphism, g : BSDMorphism) : BSD is do
        result := BSD.new
        var map: MyHashtable init MyHashtable.new

        result.name:= "woven-"+  base.name                                    new instances set
        result.copyInstances(base.instances)
        result.copyInstances(self.complementaryUnion(advice.instances,g.rinstancesMappings))

                                                                              new events set
        result.copyEvents2(self.complementaryUnion(base.events,f.reventsMappings),void,void)
        result.copyEvents2(self.complementaryUnion(advice.events,g.reventsMappings),g.rinstancesMapping
                                    s,f.instancesMappings)
        result.copyEvents2(self.twoTimesMapped(base.events,f.reventsMappings,g.eventsMappings),void,void)

                                                                              new events order
        result.events.each{ event |
              if SendEvent.isInstance(event) then
                    var e: SendEvent
                    e?= event
                    result.addCouple(e,e.receiveEvent)
              end
        …
end
```

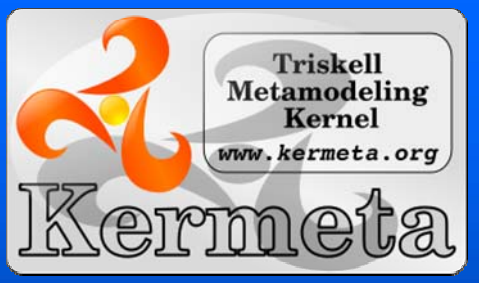

Executing the transformation



Software Engineering Concerns

- Modularity in the small and the large
  - classes & packages
- Reliability
  - static typing, typed function objects and exception handling
- Extensibility and reuse
  - inheritance, late binding and genericity
- V & V
  - test cases

Smoothly interoperates with Eclipse/EMF
Open Source
▸Download it *now!*



A statically typed object-oriented executable meta-language

- Home page
  - http://www.kermeta.org
- Development page
  - http://kermeta.gforge.inria.fr/