INRIA

UNIVERSITE DE RENNES 1

Models and 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

IRISA

Modeling in Science & Engineering

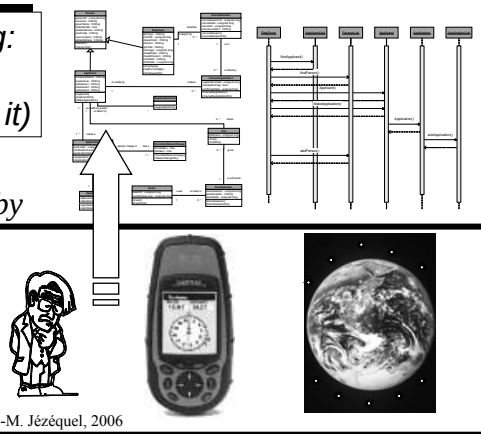
- A Model is a *simplified* representation of an *aspect* of the World for a specific *purpose*

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

M_1 (modeling space)

Is represented by

M_0 (the world)



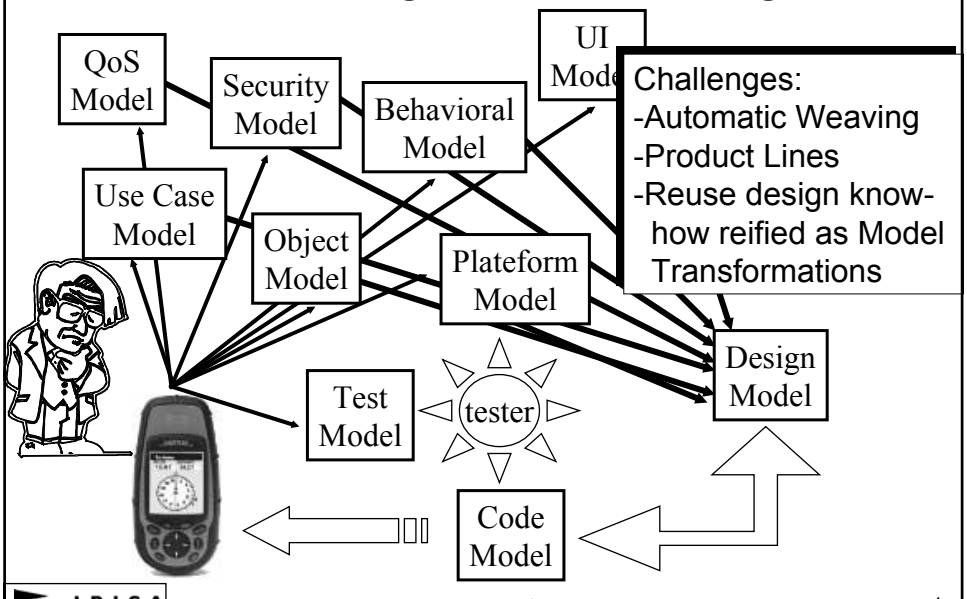
IRISA

Model and Reality in Software

- Usualy in Engineering, Models & Systems have different natures (bridge drawings and concrete bridge)
  - Sun Tse: *Do not take the map for the reality*
  - Magritte

- Whereas a program *is a* model
- Software Models: from contemplative to productive

Modeling and Weaving



Développement Logiciel Par Aspect (AOSD)

- Eviter la "*tyrannie de la décomposition dominante*"[Tar99]
  - qui empêche de modulariser certaines *préoccupations*
- Les concepts de l'AOSD:
  - ces préoccupations sont encapsulées dans des *aspects*
  - un aspect défini un ensemble de *points de jonction* spécifié par une *expression de coupe*
  - la recomposition des aspects est appelée *tissage*
- A pour origine l'AOP popula
  par AspectJ
  - Kizcalez et al., ECOOP 97

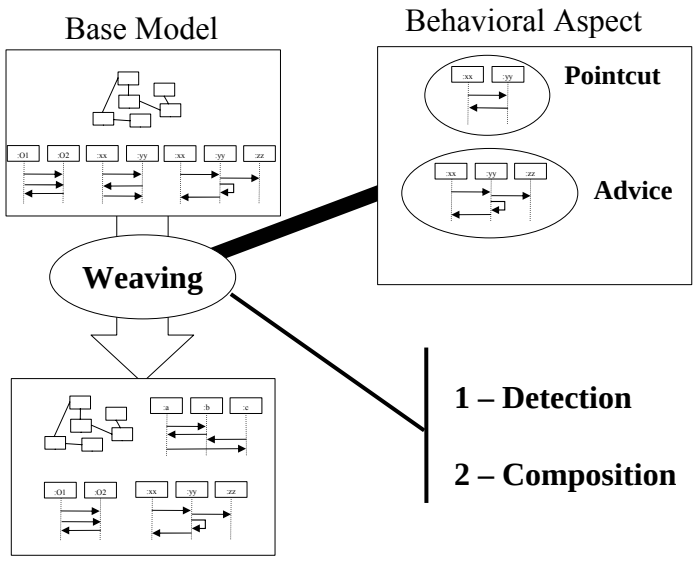
  >> identifié par le MIT comme une des 10 techn

Bénéfices attendus:
→ meilleur modularité,
+ maintenable
+ évolutif
+ réutilisable

Base Models and Aspects

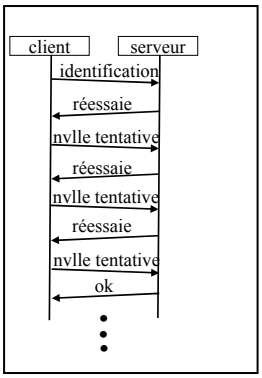
- Ideally, all aspects are equally important
  - Symmetrical AOSD
- In practice, a base model is useful to provide a backbone (canvas) on which aspects are woven
- An aspect is then described as
  - A pointcut
    - » pattern describing relevant points in the execution
  - An advice
    - » New behavior to replace (or complement) the matched ones

Overview

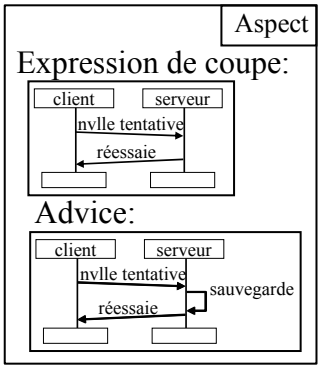


Tissage Statique d'Aspects Comportementaux

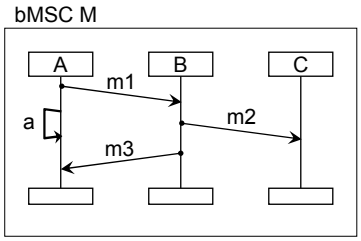
Un scénario de base



Un aspect comportemental

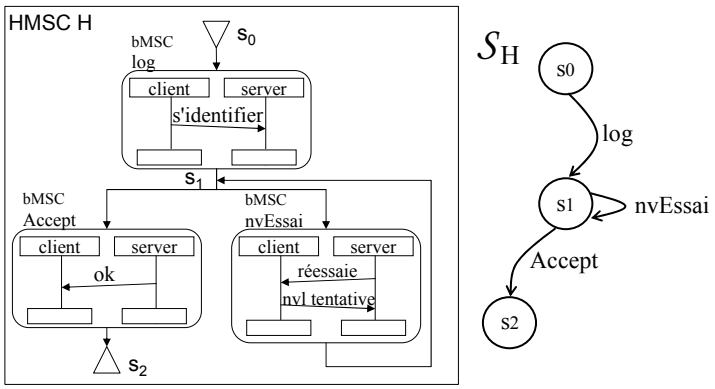


Message Sequence Charts (MSCs) (ou SD) bas niveau: bMSC



- bMSC définit un ensemble d'événements et une relation de précédence sur ces événements

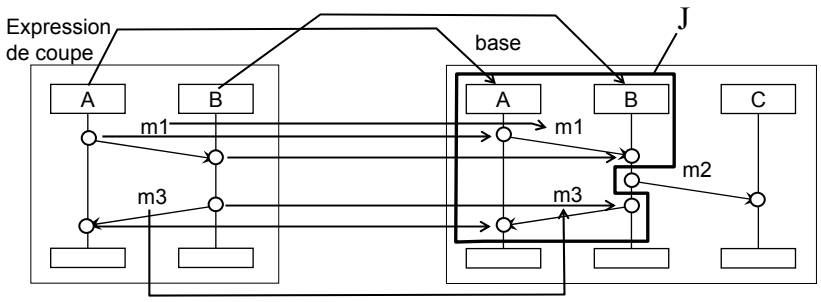
Message Sequence Charts haut niveau: HMSC



Points de jonction et langage d'expression de coupe

- Lié à un langage d'expression de coupe qui permet de spécifier où un aspect doit être composé avec le modèle de base
- Un point de jonction représente une "zone" où un aspect est entremêlé avec une autre préoccupation
- Le langage d'expression de coupe est le mécanisme qui permet de séparer une préoccupation transversale (un aspect) du modèle de base.

Définition de Points de Jonction



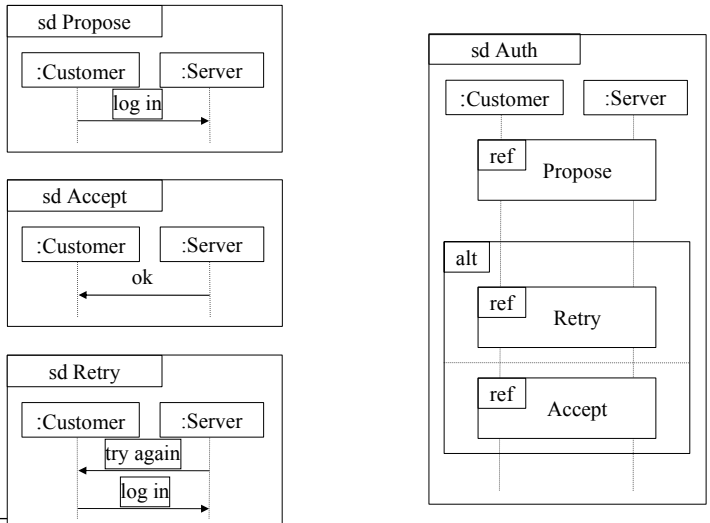
- Une partie J d'un scénario de base est un point de jonction s'il existe un isomorphisme de bMSCs entre l'expression de coupe et J.

Composability & Aspect Weaving

- Unfortunately, traditional aspect weaving (e.g. in AspectJ) has very bad composability properties.
- After weaving aspect A1 into B, maybe A1oB does no longer match A2 pointcut, while B alone did.
  - And conversely…
- Let's explore these issues with scenario languages, as in UML/HMSC etc.
  - And show how it can be automated with Kermeta

Example: Base Model

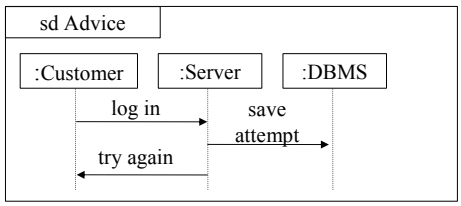
(based on J. Klein & F. Fleurey works)



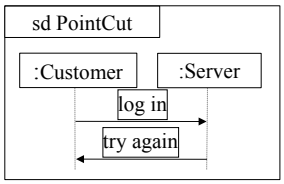
Example: Behavioral Aspect

Behavioral Aspect = Advice + Pointcut + Morphism

Advice



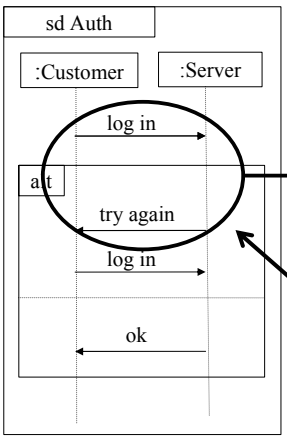
Pointcut



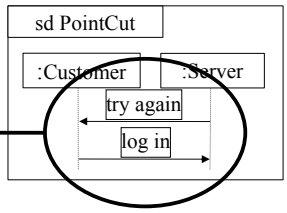
Matching based on sub-SD inclusion

BaseModel = sd1 **seq** *pointcut* **seq** *sd2*

Base Model



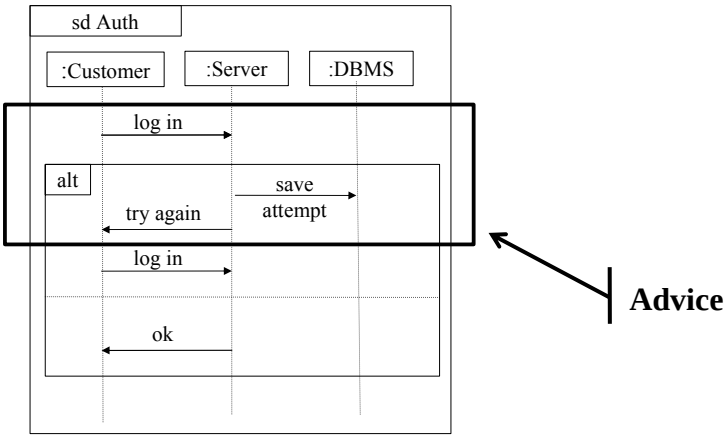
Pointcut



1 Joinpoint

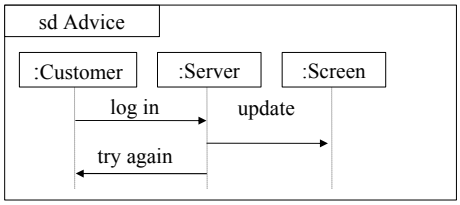
Example: Composition

Result Model

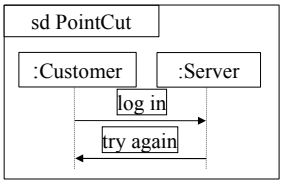


2nd aspect: update screen

Advice

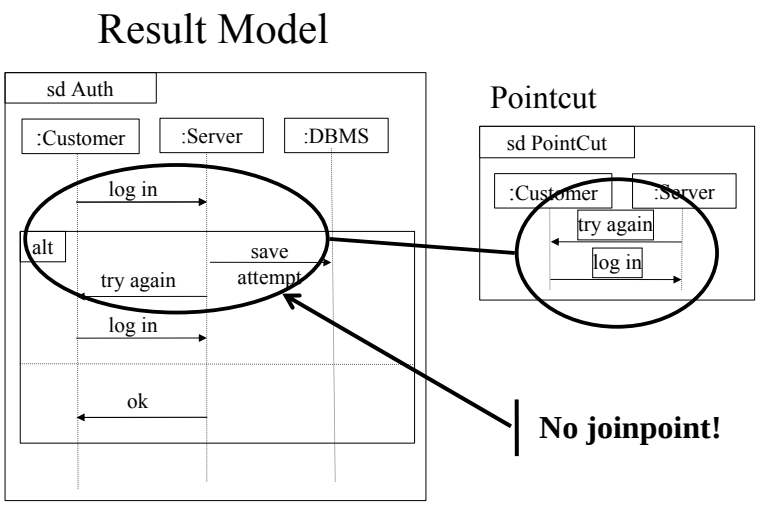


Pointcut



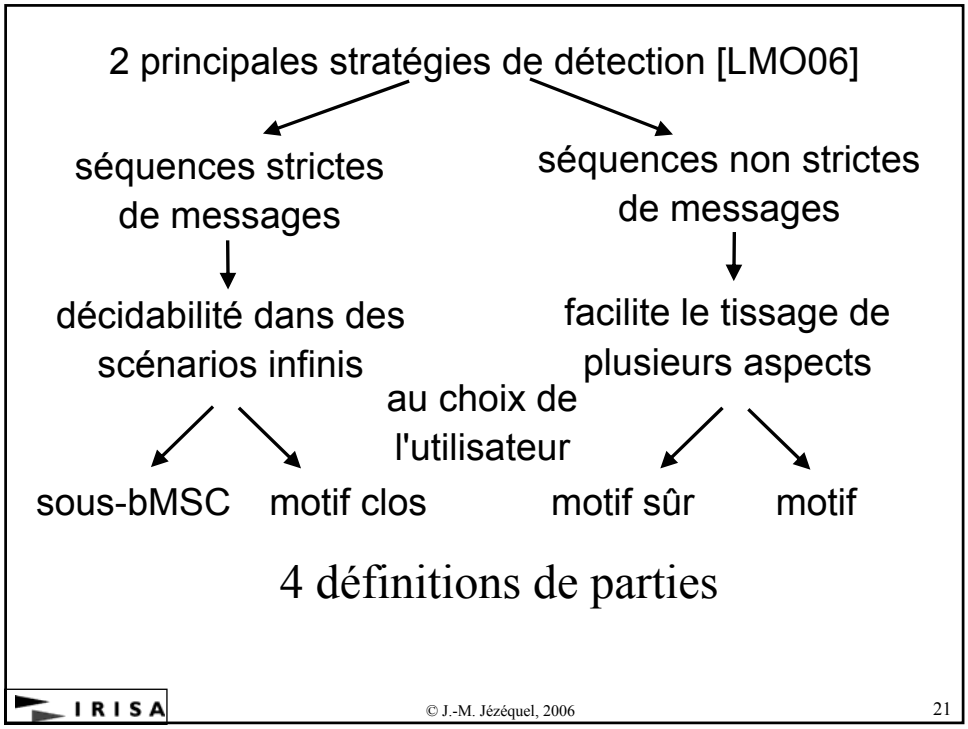
Matches base scenario, but no longer woven one!

Impossible Composition



More advanced detection needed

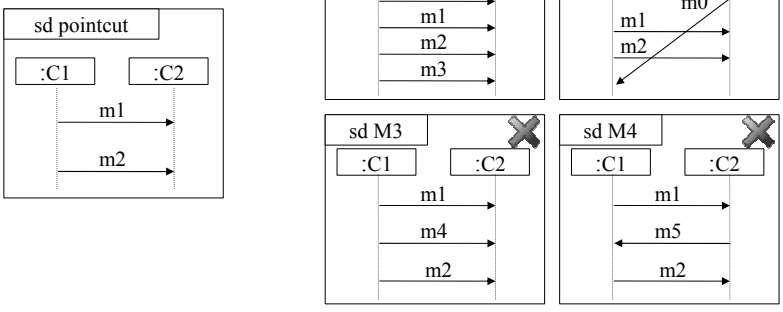
- Two possible strategies beyond sub-sequence diagram
  - Closed part
  - Pattern
- Static Analysis to find Joinpoints over HMSCs (loop unrolling!).



Detection Strategies (1)

detection

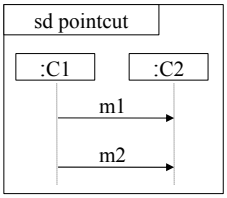
- Sequence Sub-diagram:
  BaseSD = *sd1* **seq** *pointcut* **seq** *sd2*

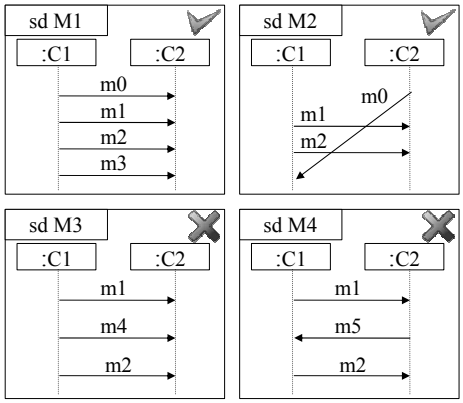


Detection Strategies (2)

detection

- Closed Part

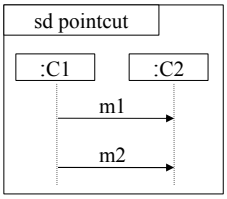


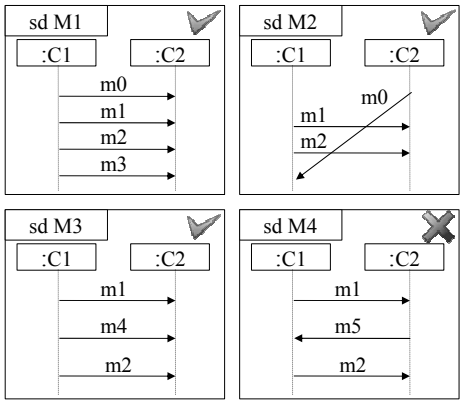


Detection Strategies (3)

detection

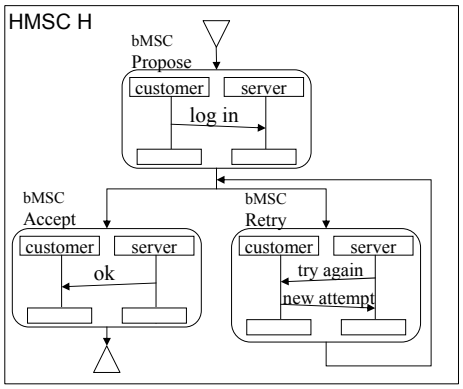
- Pattern



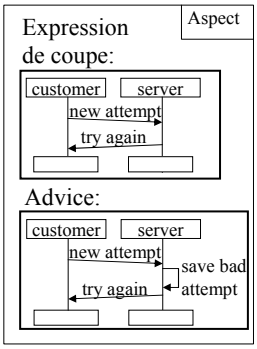


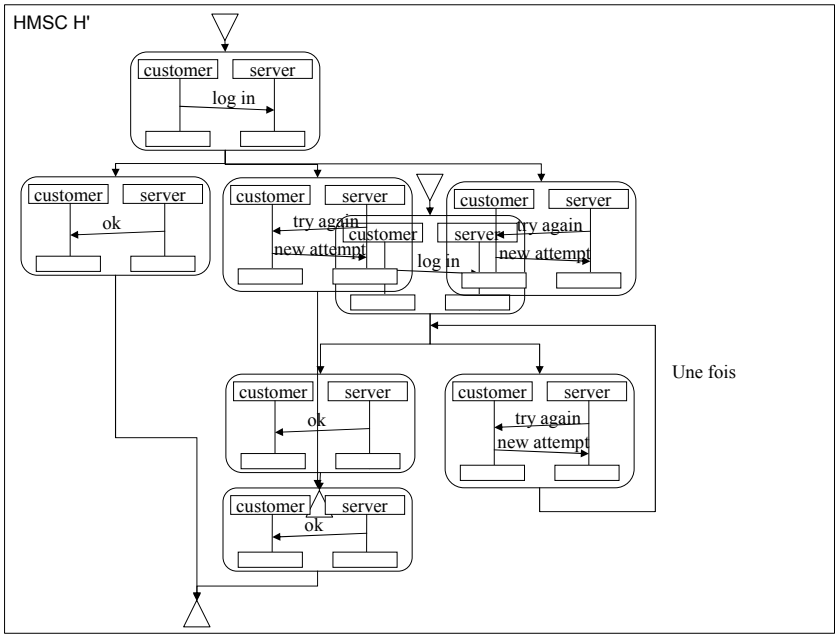
Detection in infinite scenarios

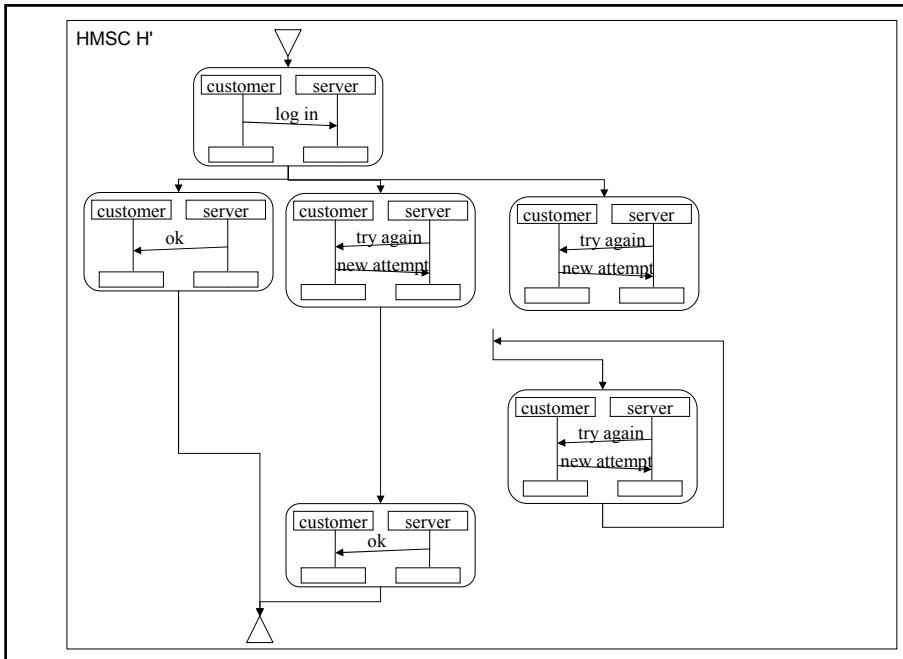
Base scenario

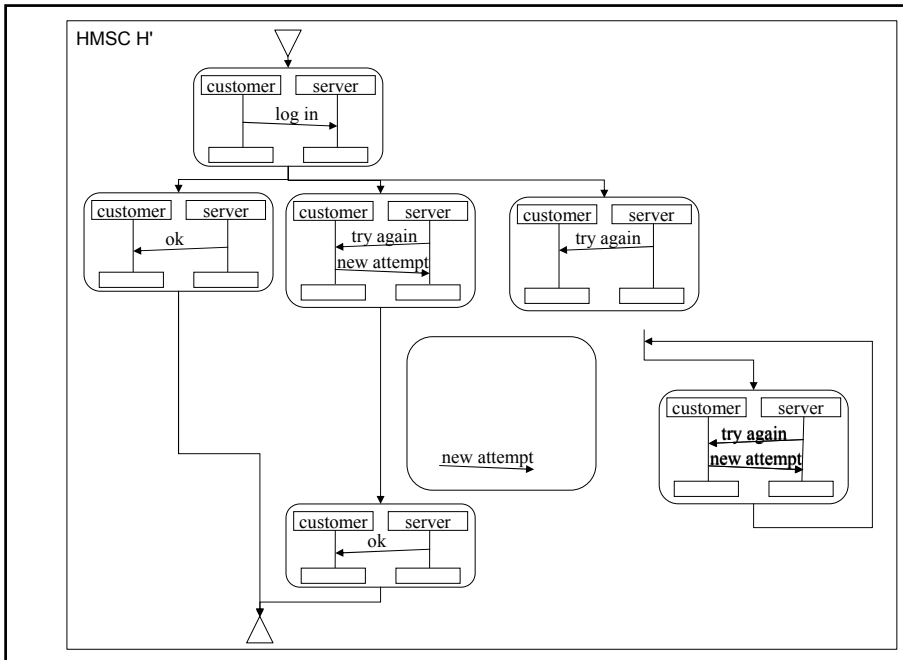


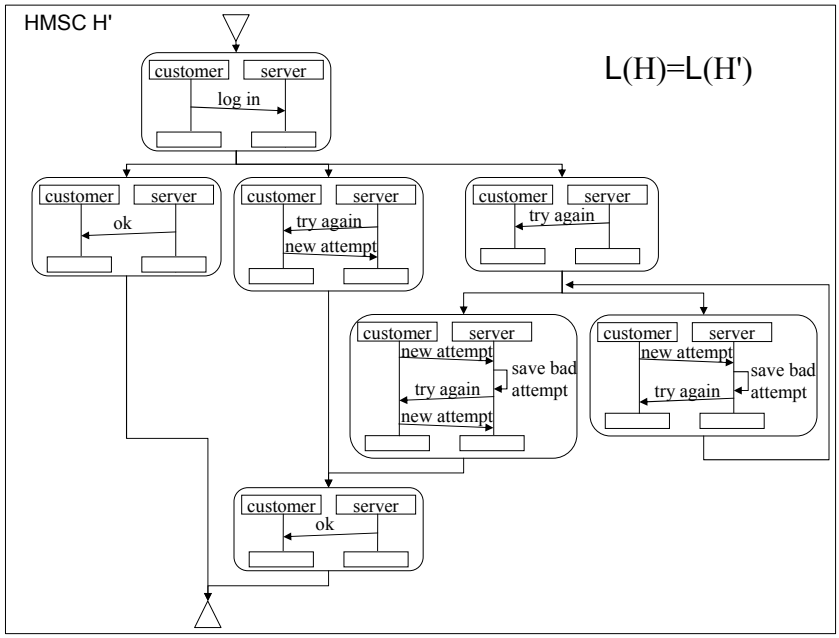
Behavioral aspect





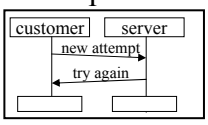

HMSC H'
customer
server
log in
customer
server
ok
customer
server
try again
new attempt
customer
server
try again
new attempt
customer
server
try again
new attempt
customer
server
ok


HMSC H'
customer
server
log in
customer
server
ok
customer
server
try again
new attempt
customer
server
try again
new attempt
customer
server
try again
new attempt
customer
server
ok

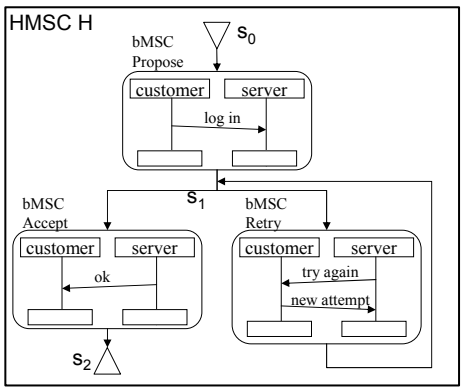


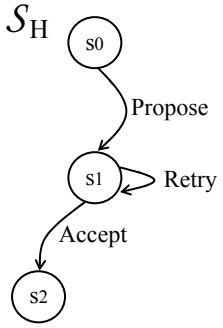
Transformation de HMSC

Expression de coupe



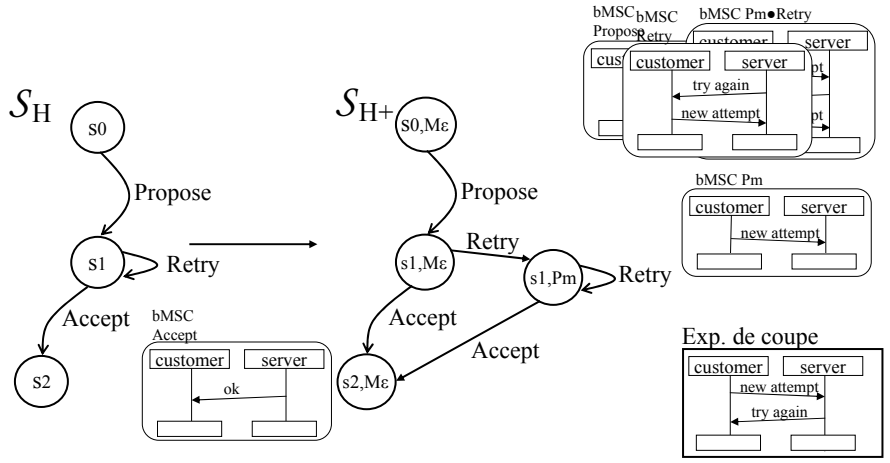
Un scénario de base





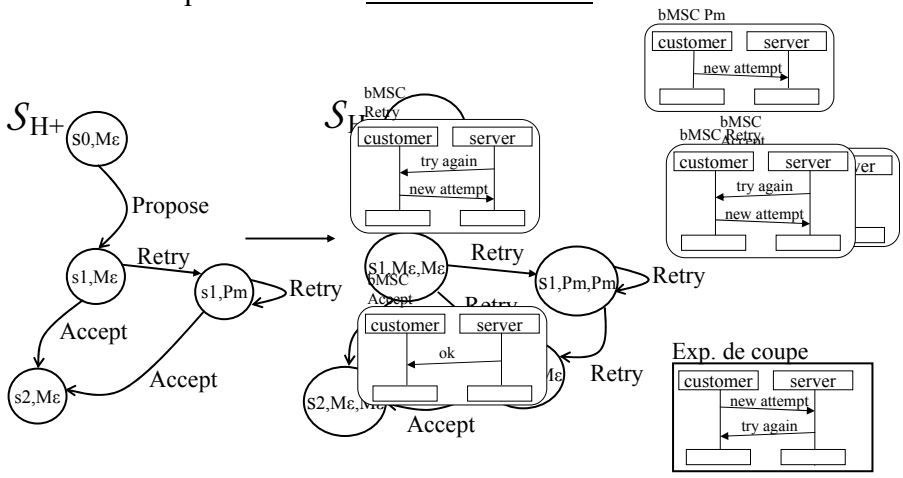
Transformation de HMSC

- Première étape: calcul des <u>détections potentielles</u>



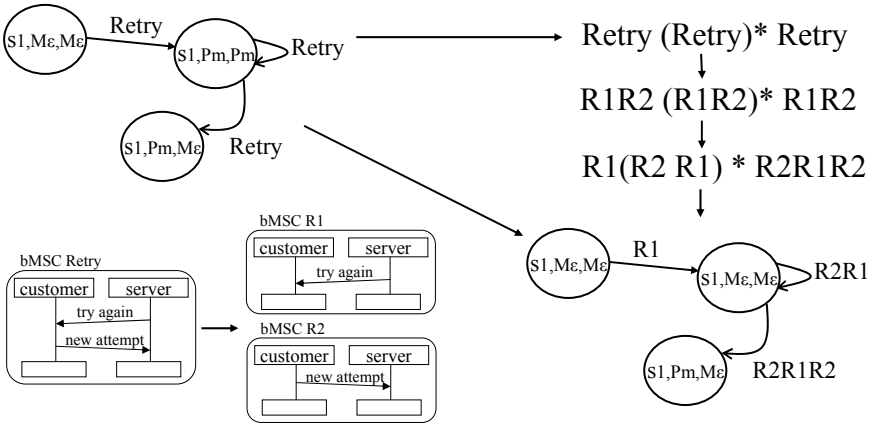
Transformation de HMSC

- Deuxième étape: calcul des <u>détections futures</u>



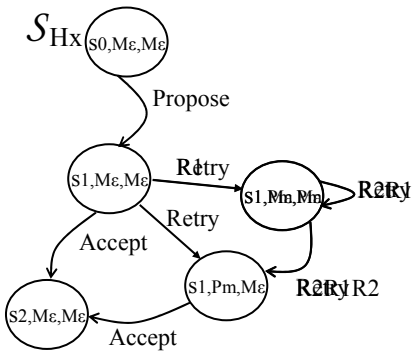
Transformation de HMSC

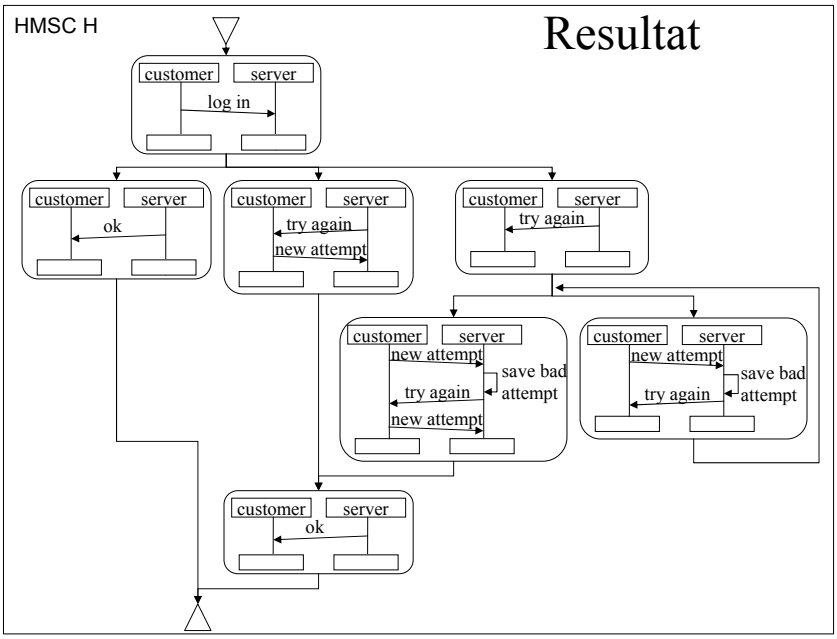
- Troisième étape: utilisation d'expressions régulières, découpe de bMSCs en atomes et permute ces atomes



Transformation de HMSC

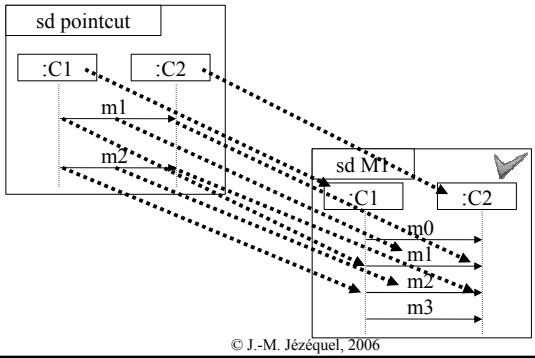
Resultat





Result of the Detection Step

- Create a morphism for each Joinpoint
  - Morphism of instances
  - Morphism of events
  - Morphism of messages (implied)



Composition

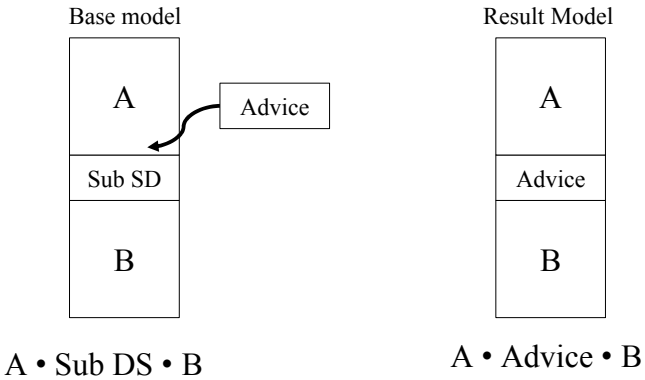
- Compose the advice into the base model
- Depend on the detection strategy

Composition

Composition

- Sequence sub-diagram

⇨ Weak Sequential Composition



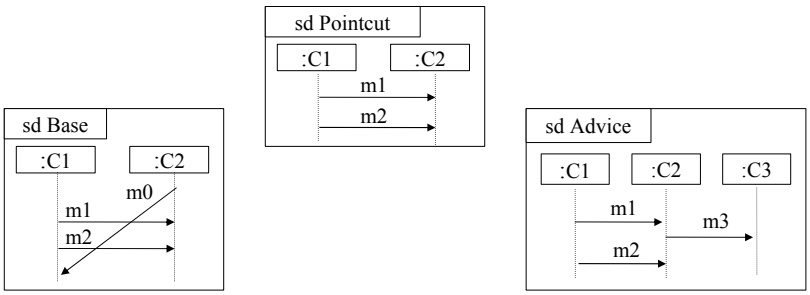
A • Sub DS • B

A • Advice • B

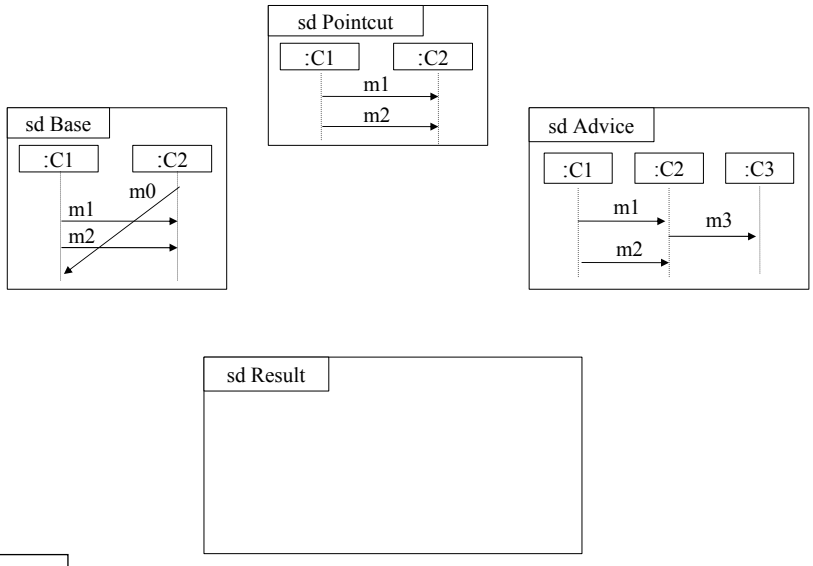
Amalgamed Sum

- For matching on Closed Part & Patterns

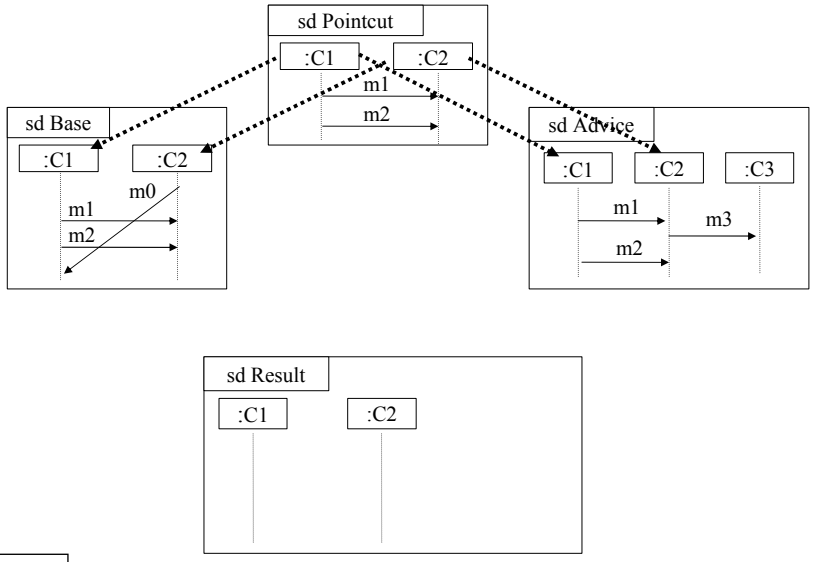
  ⇨ Simple composition not possible



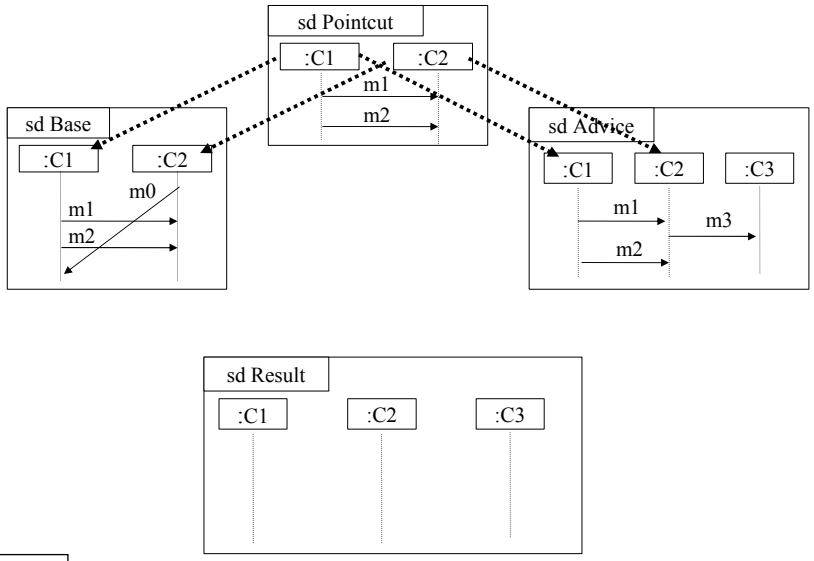
Amalgamed Sum



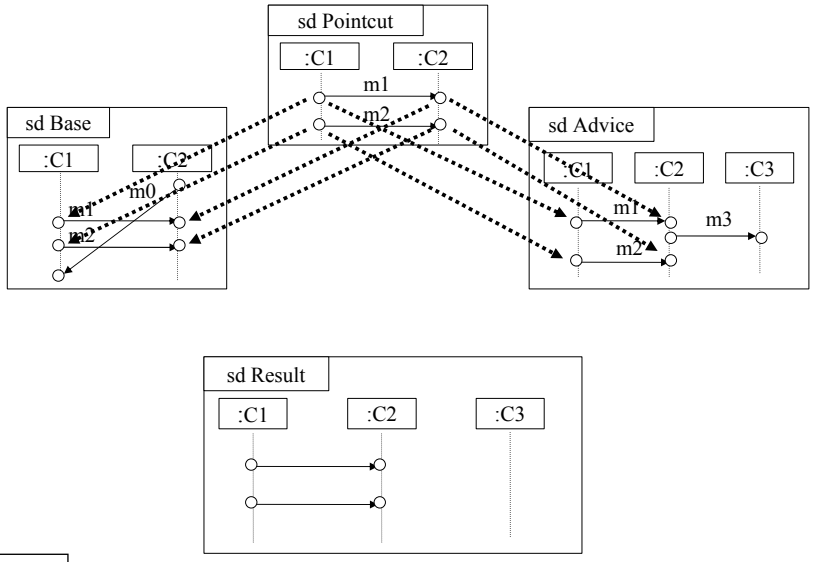
Amalgamed Sum



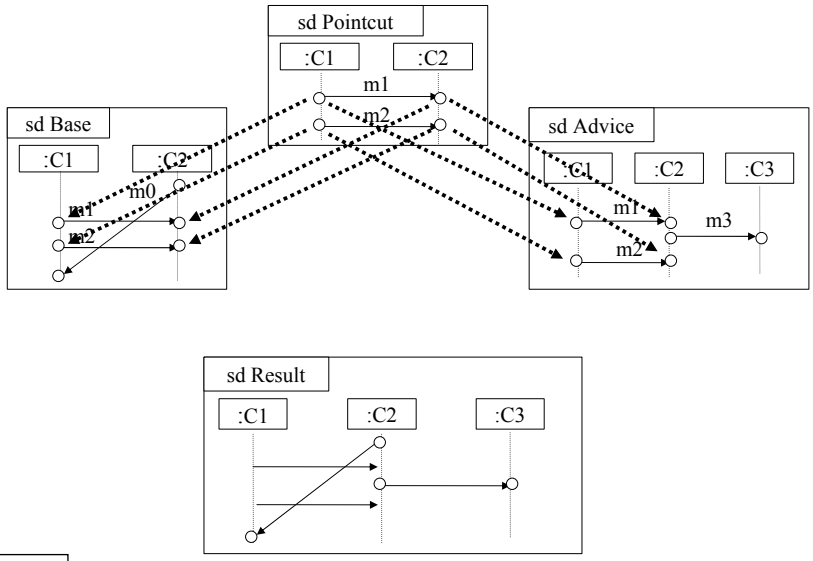
Amalgamed Sum



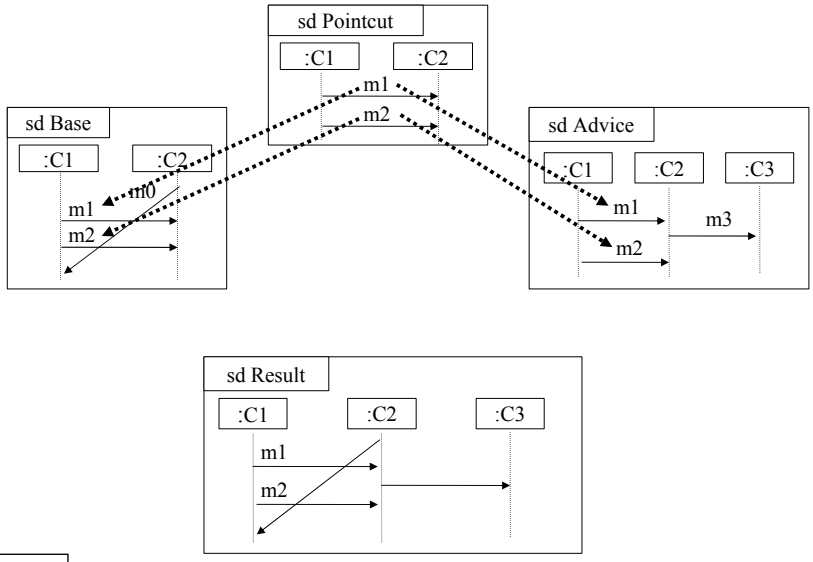
Amalgamed Sum



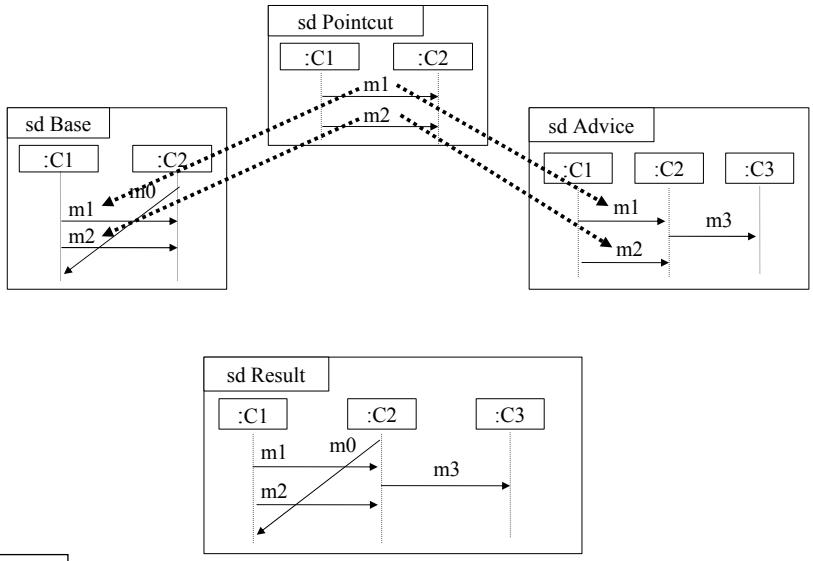
Amalgamed Sum



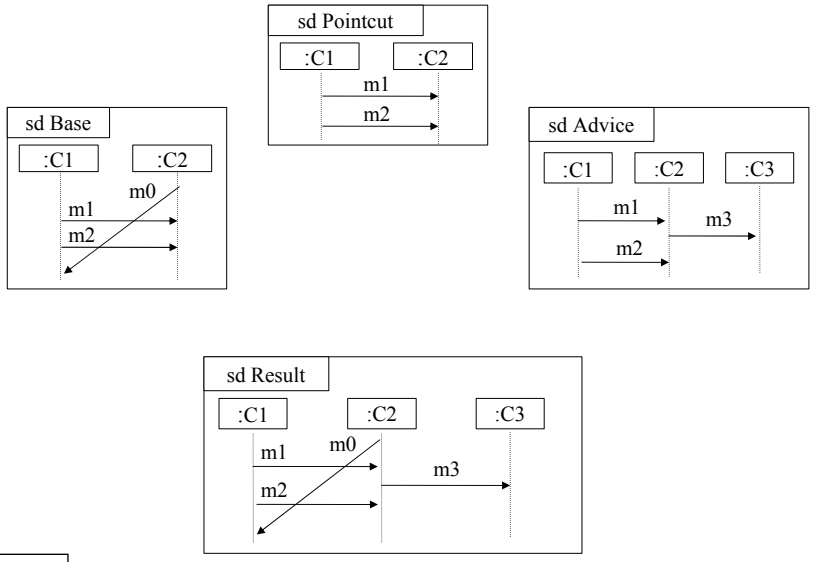
Amalgamed Sum



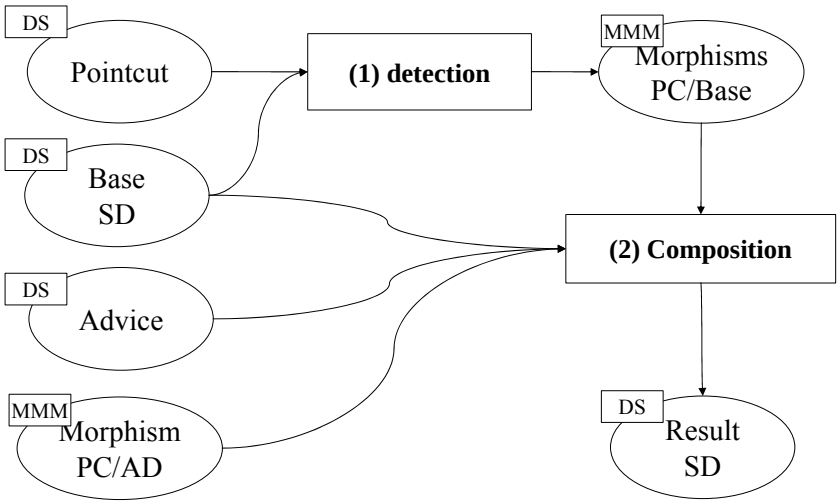
Amalgamed Sum



Amalgamed Sum



Implementation



KerMeta in a NutShell

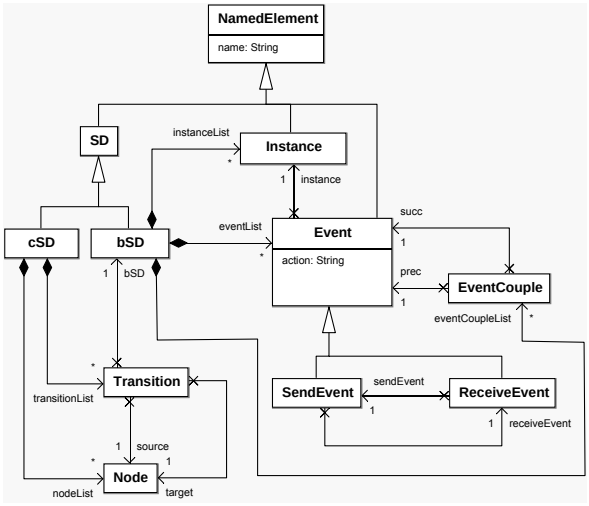
- EMOF superset
  - Any EMOF MetaModel is a valid KerMeta program, and conversely
- Object-Oriented
  - Multiple inheritance / behavior selection
  - Operation overiding / late binding
  - Full reflection (read-only at this time)
- Statically Typed
  - Generics
  - Function types to allow OCL's *forall/exist/iterate*

"Programming style" Issues

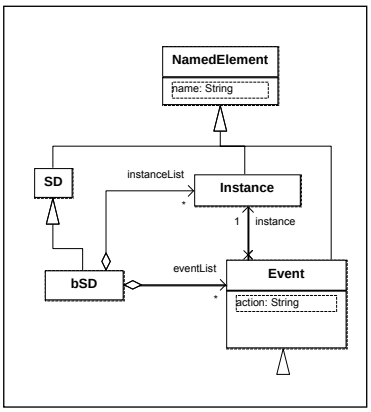
- The transformation is simply the model of an object-oriented program that manipulates model elements
  - Navigation through model is first class though (like in OCL)
- OO techniques
  - Customizability through inheritance/dyn. binding
  - Pervasive use of GoF like Design Patterns

Defining the metamodels

input and output metamodels are the same

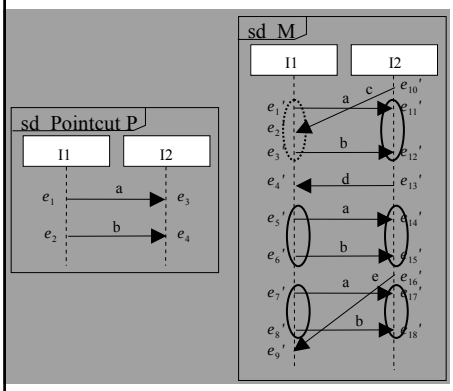


Visual/Textual



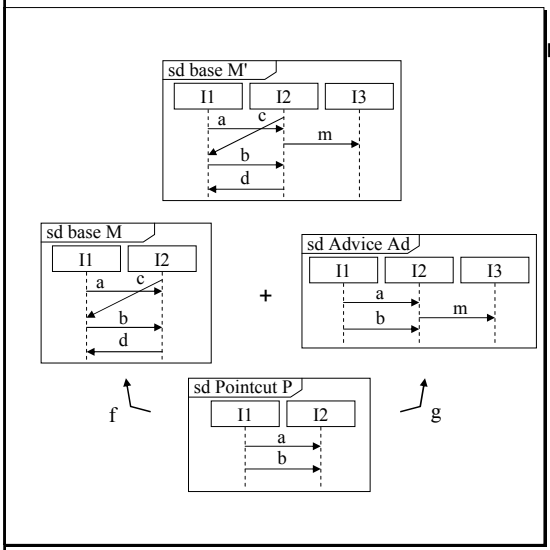

```
package bigSd;
using kermeta::standard
using kermeta::persistence
abstract class NamedElement
{
    attribute name : String[1..1]
}
abstract class SD inherits NamedElement
{}
class BSD inherits SD
{
    attribute events : Event[0..*]
    attribute couples : EventCouple[0..*]
    reference instances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

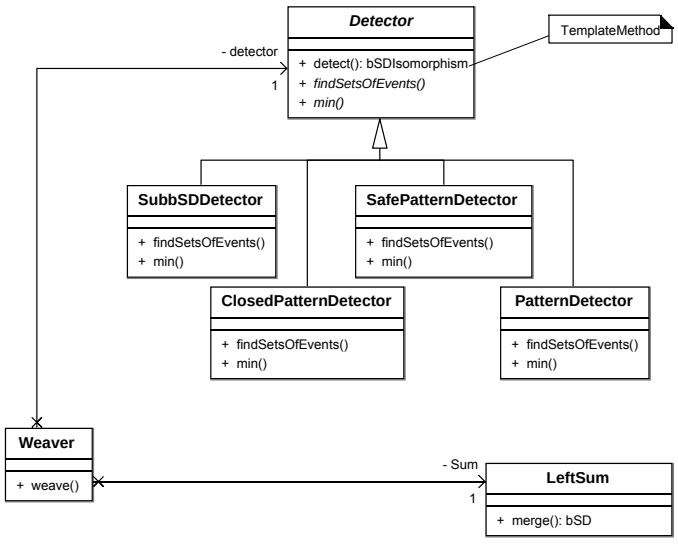


- Composition step: amalgamated sum
  - use P and two morphisms f and g (f being computed by the detection step)
  - keep the common parts between M and Ad
  - add new object I3
  - add the events of M by respecting the order specified on M
  - add the events of Ad by respecting the order specified on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

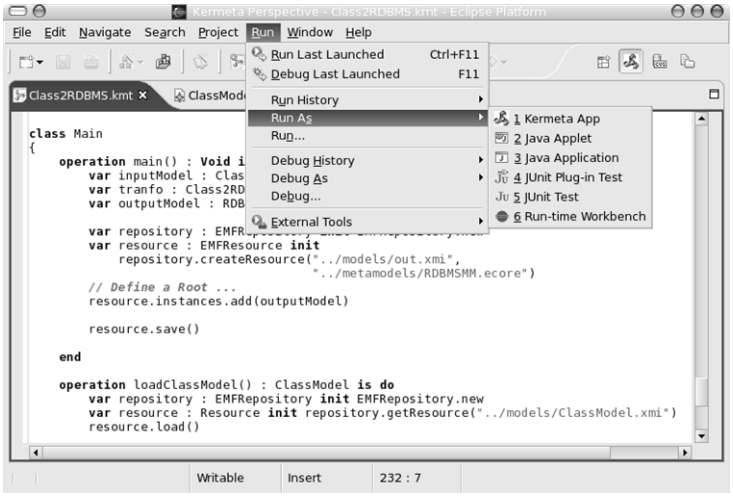
```
require kermeta   require "../models/bigSd.kmt"
using kermeta::standard    using bigSd
class LeftSum {
 operation merge(base : BSD, pointcut : BSD, advice : BSD, f : BSDMorphism, g : BSDMorphism) : BSD is do
        result := BSD.new
        var map: MyHashtable init MyHashtable.new

        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



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/