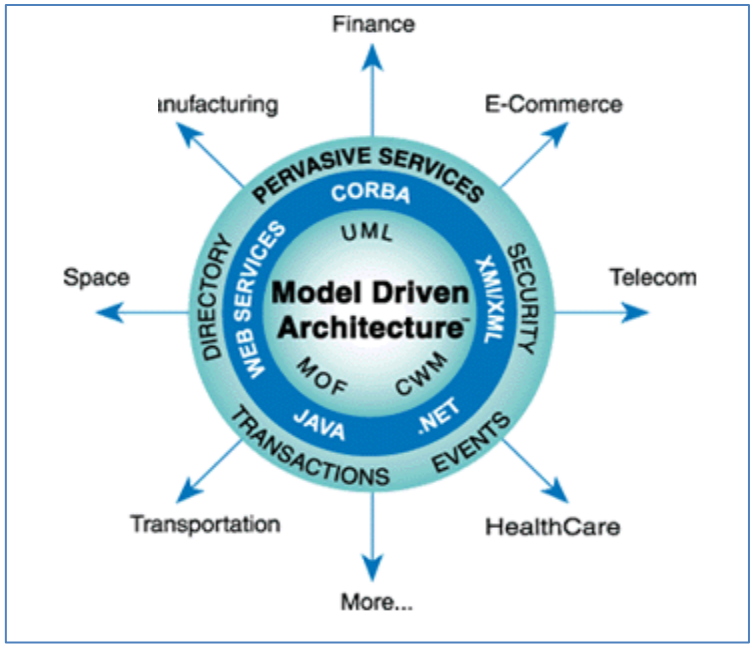
Aspect-Orientation in Modelling: Lessons Learned

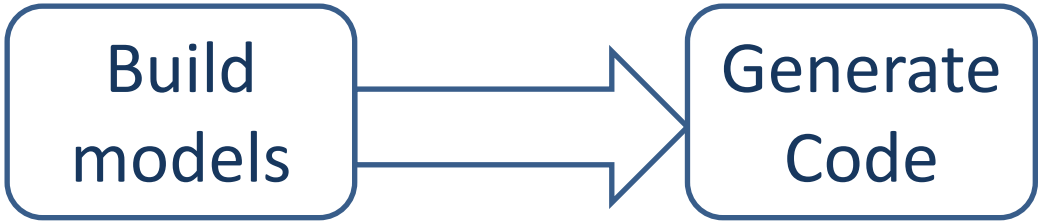
Ella Roubtsova

Open University of the Netherlands and

Munich University of Applied Sciences, Germany

Some things can be understood only retrospectively



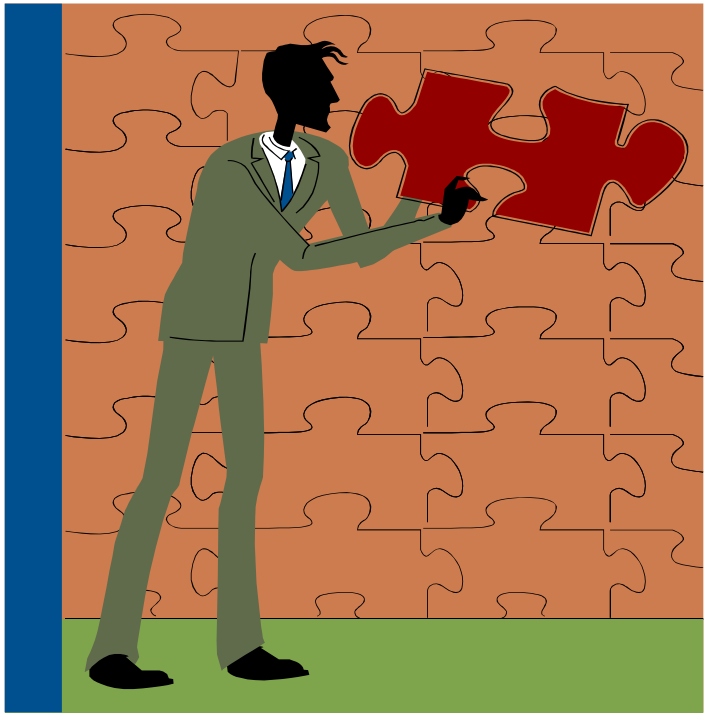


Problem of maintainability moved from the programs into the models

1. Impossible to separate particular concerns in models
2. The higher level of abstraction of models does not prevent models from complexity
3. The models are changed with requirements and the introducing changes, spread through the model, causes modeling mistakes

Let's use the ideas of Aspect-oriented programming in models

- **Optimistic**: Revise and Extend the existing modelling notations and semantics in order to enable separation of concerns
- **Skeptical:** Understand the possibilities and limitations of modelling semantics in keeping intellectual control over evolving models



Achievements of AOP: Limited ability to compose features in Object–Oriented Programming

The cause of the difficulties with separation of concerns on OOP is the single inheritance and subtype polymorphism.

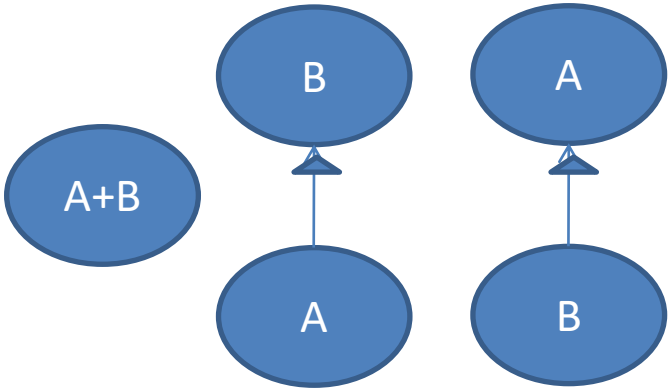
For example, there are three possible ways to organize two features A and B. into classes:

(1) put them in the same class,

(2) make A a subclass of B,

(3) make B a subclass of A.

The first two choices force B to be included whenever A is included and B is scattered. As the number of features grows, this problem becomes more severe.



Achievements of AOP: Invention of new types of Abstraction called Static Aspect

A Static Aspect exists only for code management not for execution.

It contains two parts:

- **a join point designator**

  specifies groups of join points being well defined places in the structure of a program

- **an advice**,

  the code that should be woven into the program

  as it indicated by the join point designator

Achievements of AOP: Invention of a new type of Abstraction called Dynamic Aspect

A Dynamic Aspect is a new abstraction.

In EJB model it is called an interceptor.

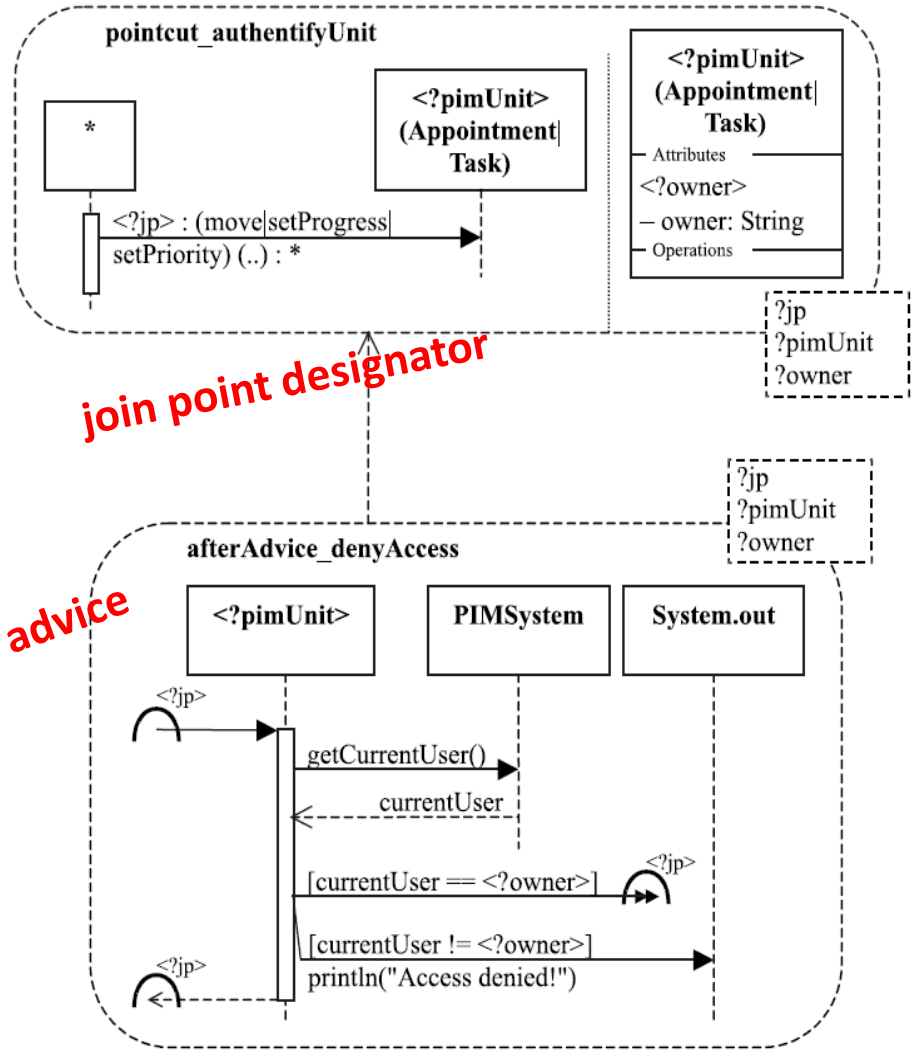
It recognizes the method calls and state before and after method calls in objects .

- contains specification of join points (methods calls) and states before and after
- can react the same way on each method call in a group thanks to the weaving algorithm implemented in middleware

Achievements of AOP: Aspect Categories

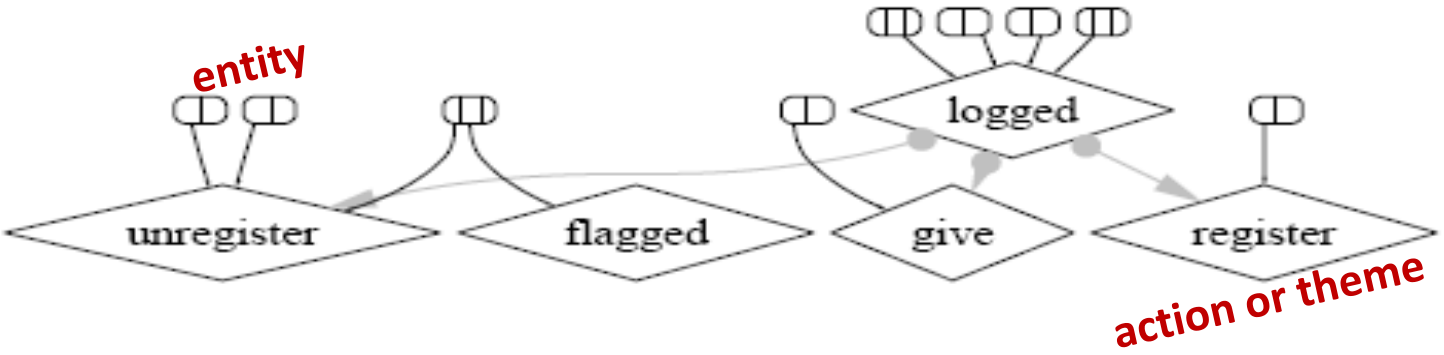
- **Spectative** aspect can change the values of variables local to the aspect, but can't change the value of any variable of other aspects and the flow of method calls.
- **Regulative** aspects affect the flow of control by restricting operations, delaying, or preventing the continuation of a computation.
- **Invasive** aspects change the values of variables in the underlying program. In case of invasive aspects no guaranty can be given about preserving of dynamic properties of the underlying program.

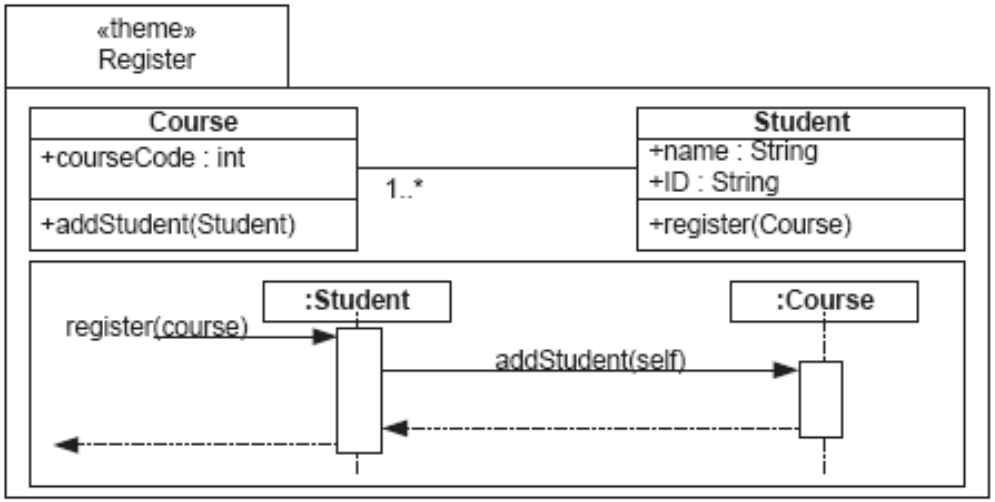
Revision of Sequence Diagrams: Join Point Designation Diagrams **JPDDs**

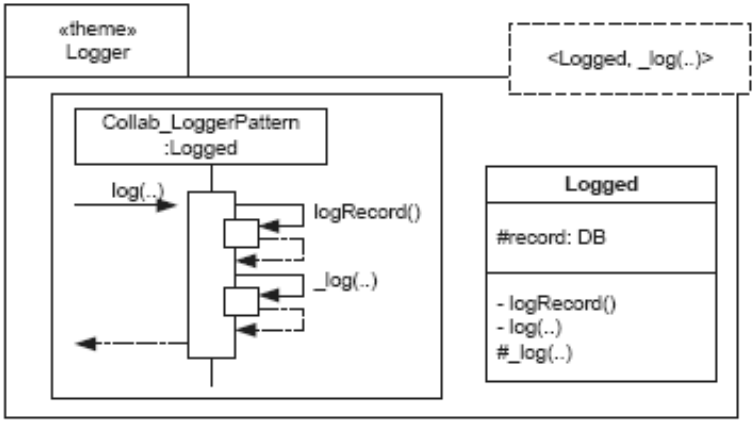



```
aspect Authorization(){
    pointcut restrictAccess(PIMUnit
        pimUnit):
     ( call(* Appointment.move(..)) ||
       call(* Task.setProgress(..)) ||
       call(* Task.setPriority(..)))
            target(pimUnit) &&
    if(!pimUnit.getOwner().equals(PI
        MSystem.getCurrentUser())
    || (PIMSystem.getCurrentUser()==
        null));
void around(PIMUnit pimUnit) :
    restrictAccess(pimUnit) {
    System.out.println("Access
    Denied!") ; }
```

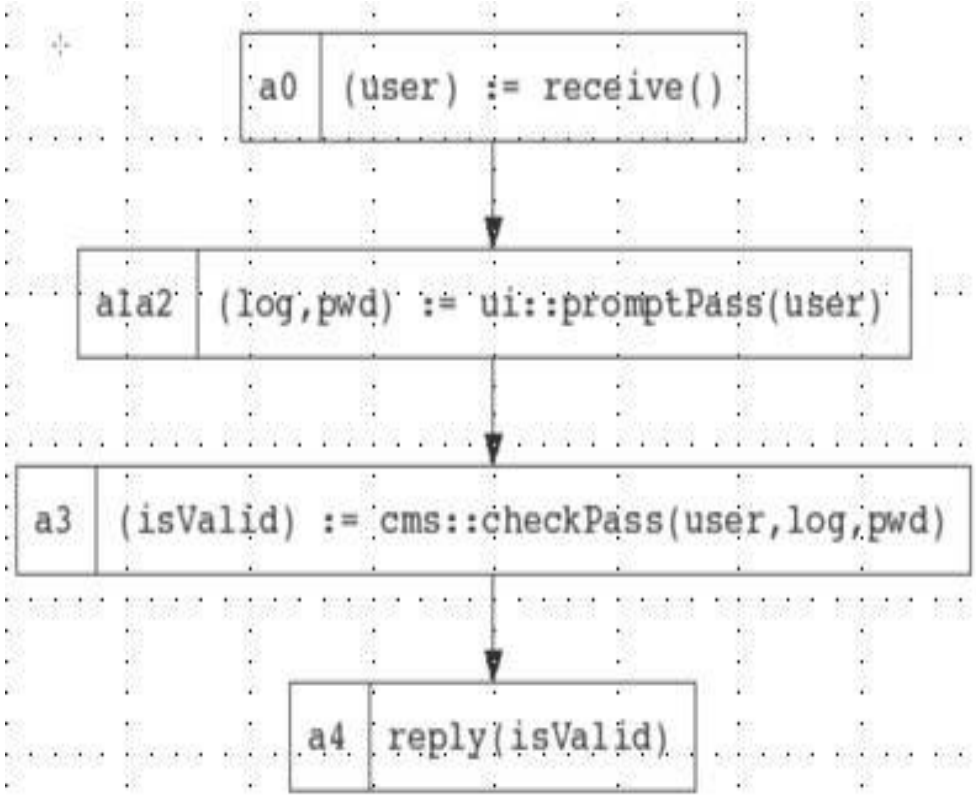
AOM Revision of Workflows: Theme







AOM revision of Activity diagrams: Activity moDel supOrting oRchestration Evolution ADORE

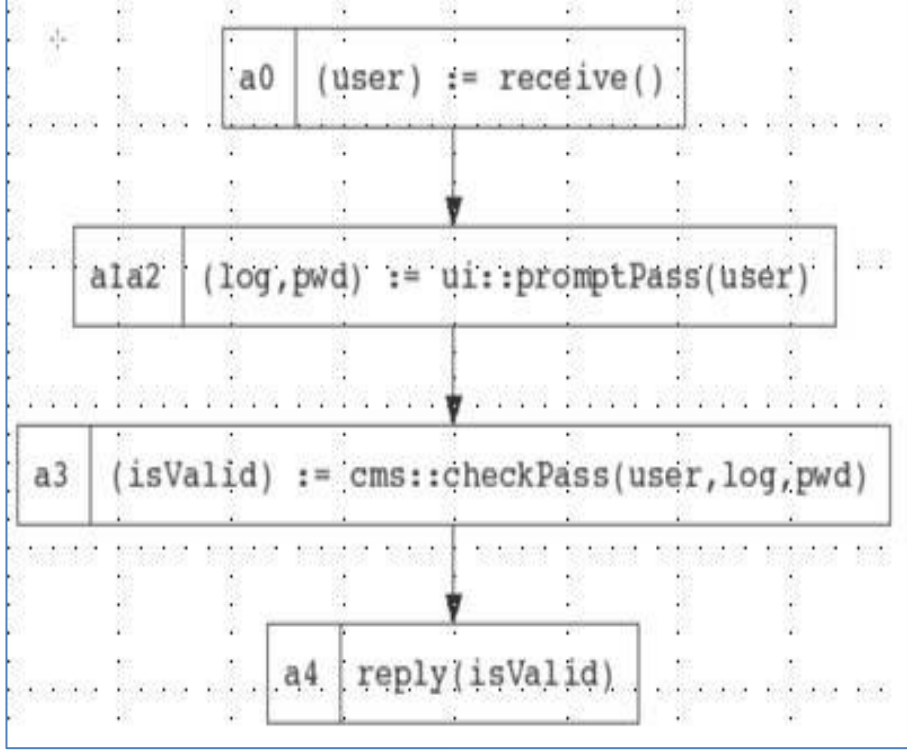


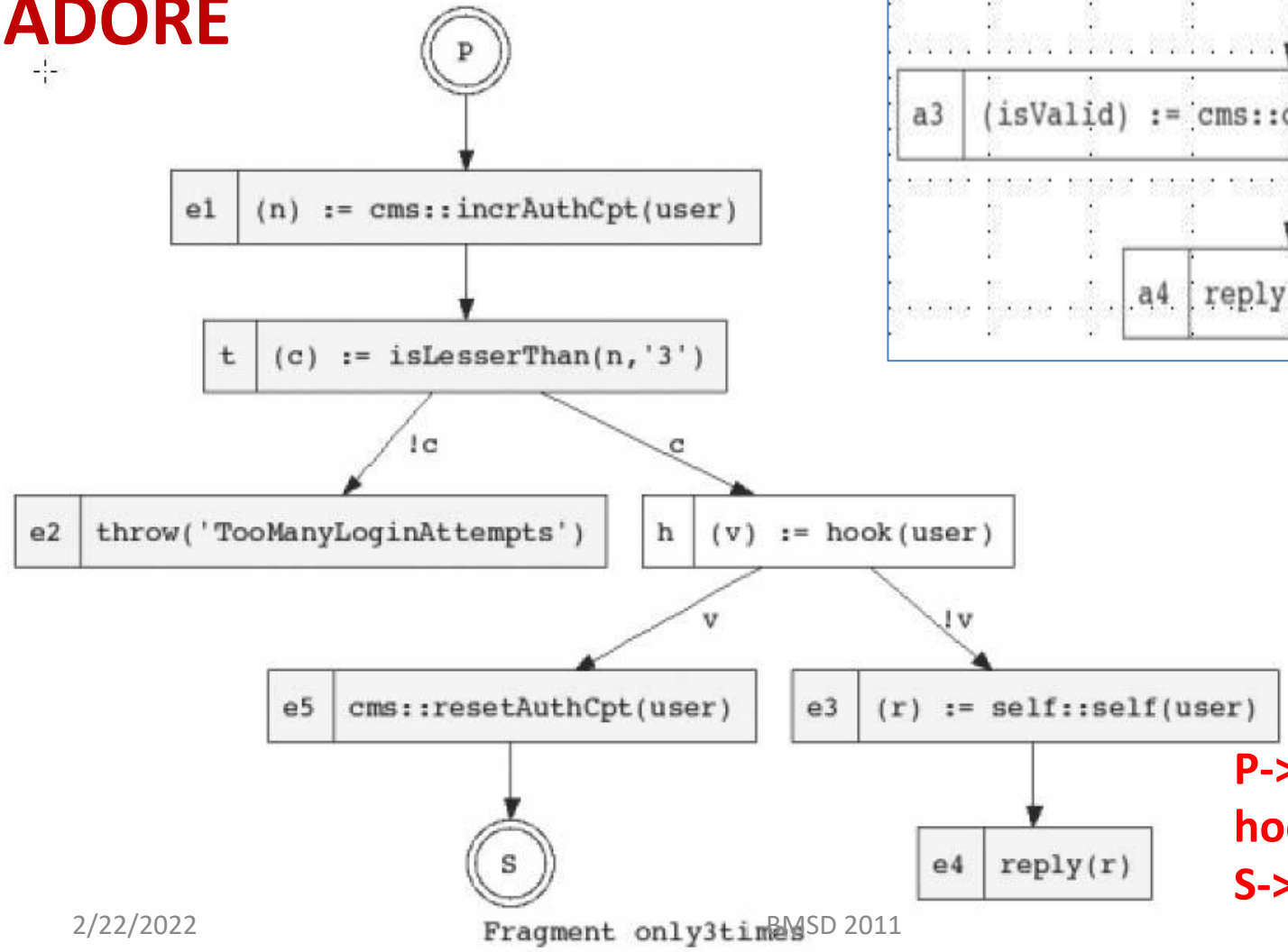
In ADORE, an orchestration of services defined as a partially ordered set of activities.

AOM revision of workflows:

Activity moDel supOrting oRchestration Evolution

ADORE



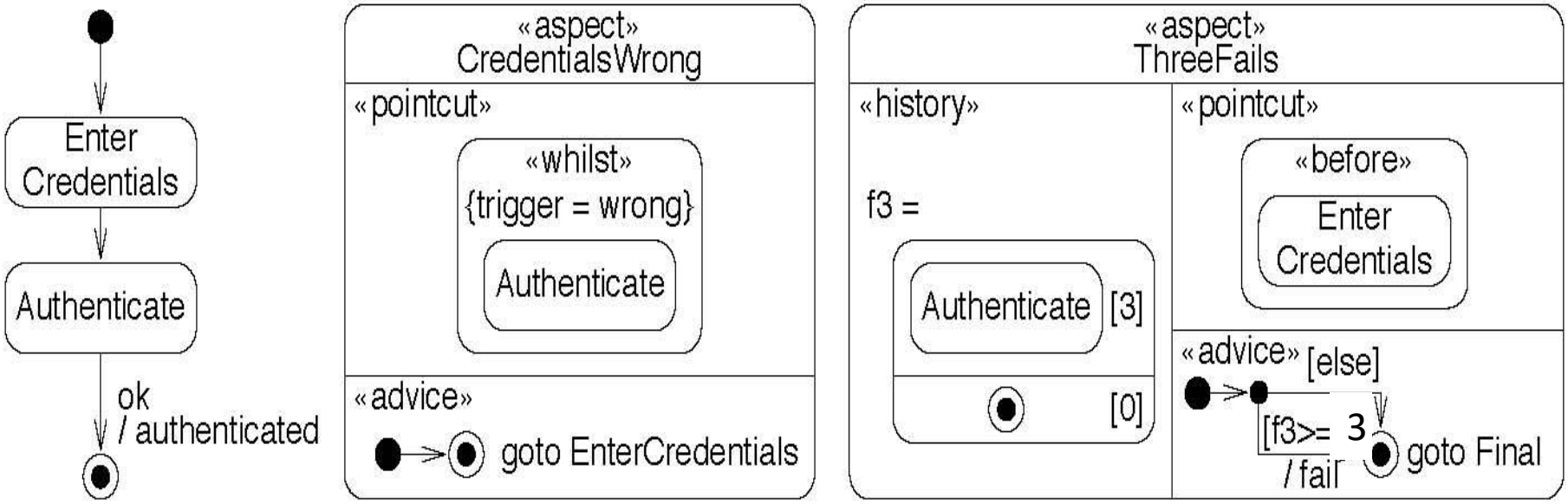


P->a0

hook -> {a1, a2,a3}

S->a4

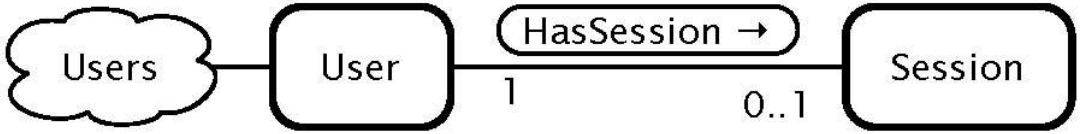
AOM revision of Behavioural State Machines: High-Level Aspects (HiLA)

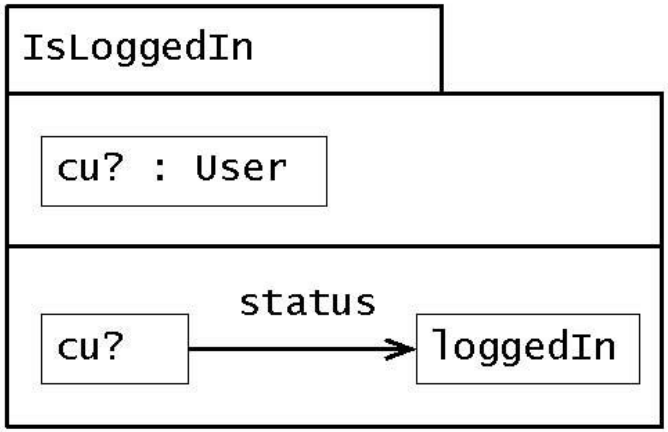


AOM revision of design by contract: Visual Contract Language (VCL)

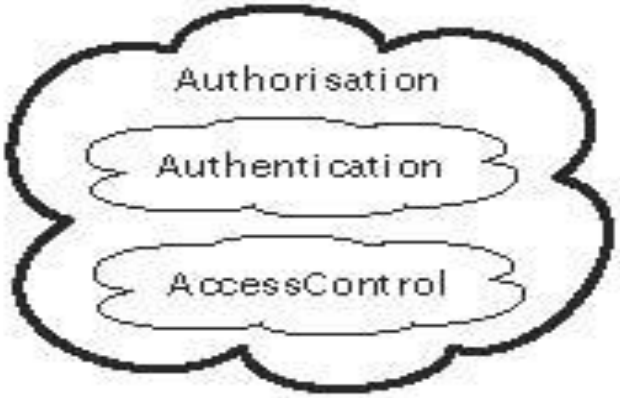
- DbC extends the ordinary definition of abstract data types with pre-conditions, post-conditions and invariants.
- If a precondition is violated, the effect of the section of code becomes **undefined** and thus may or may not carry out its intended work.

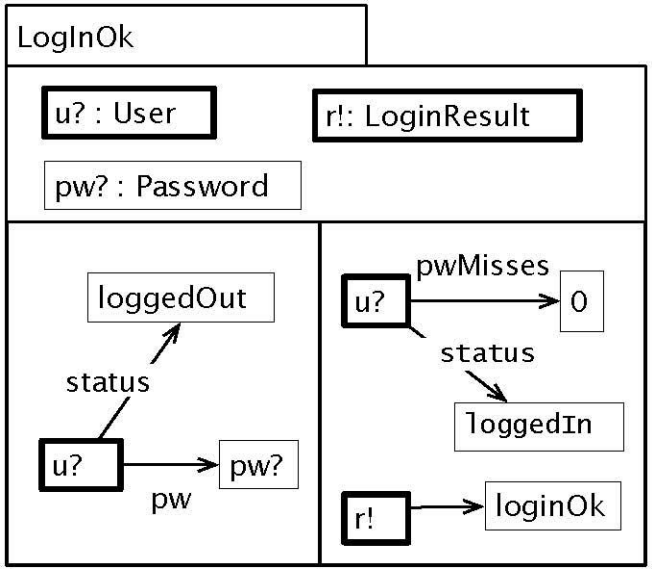
AOM revision of design by contract: Visual Contract Language (VCL)

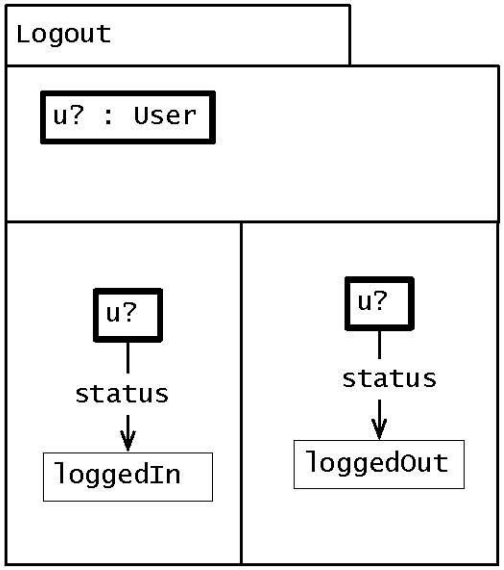




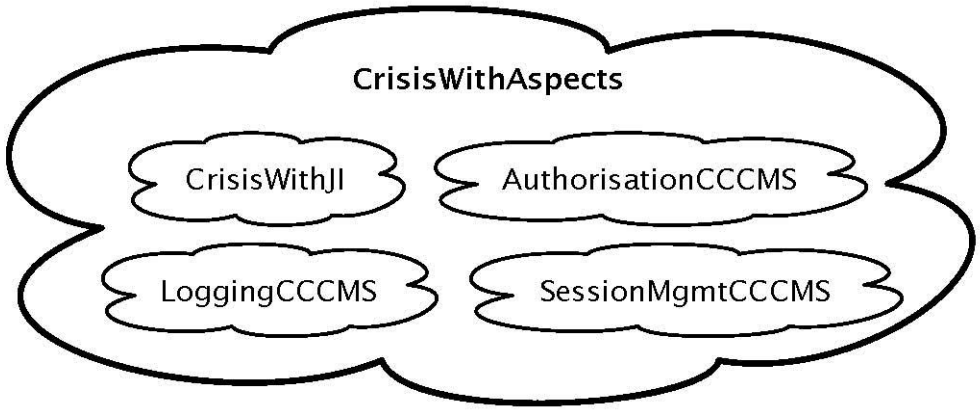
VCL: Package Authentication

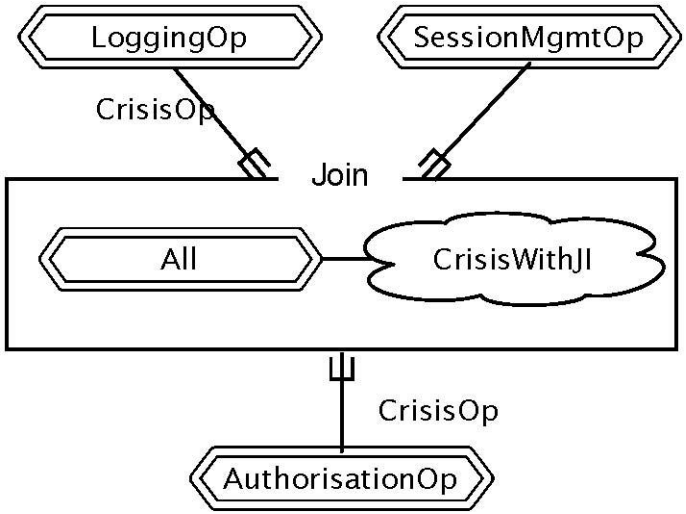


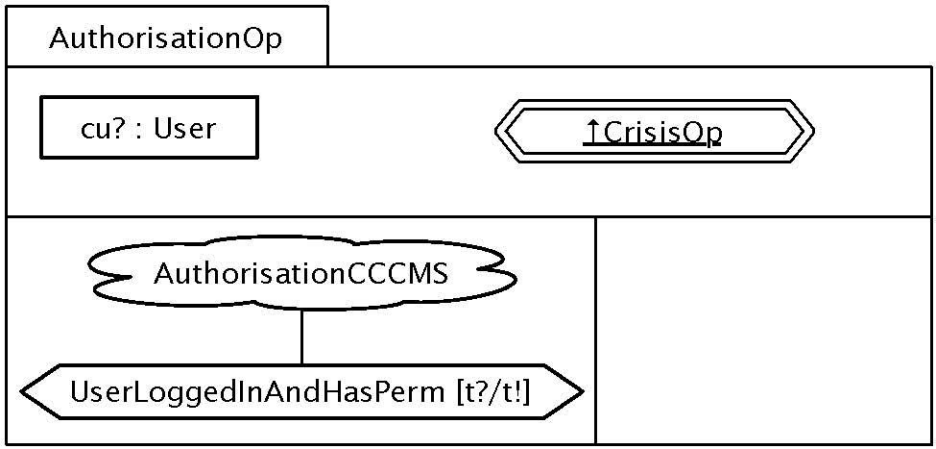




VCL: Crisis with Aspects







In all modelling semantics presented so far

- PROS:
  - compact presentation of aspects
  - models of join points become reusable artefacts that eases evolution of diagrams.
- CONS:
  - conventional composition semantics do not preserve the behaviour of unchanged parts
  - the models demand **verification** after every modification

AOM Revision of Mixins

Events contain data

Machines contain local storage

Events:

Register

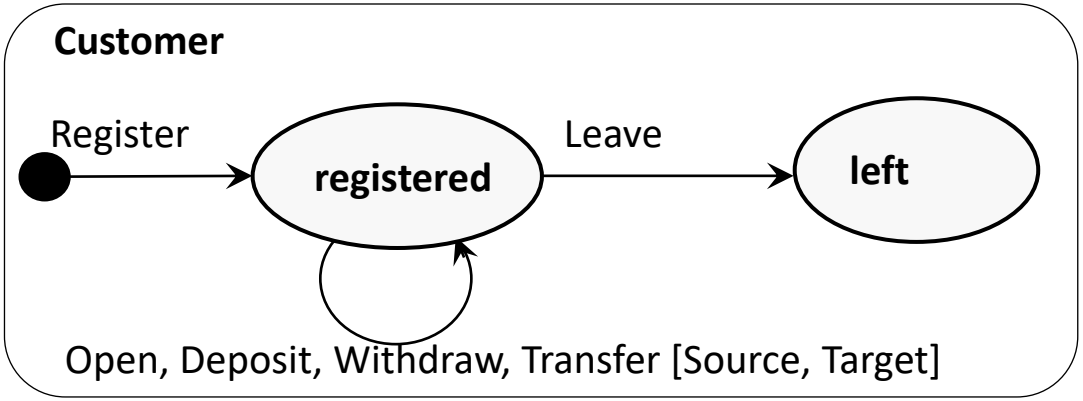
Date:Date
Customer OID: Customer
Full Name: String
Address: String

Leave

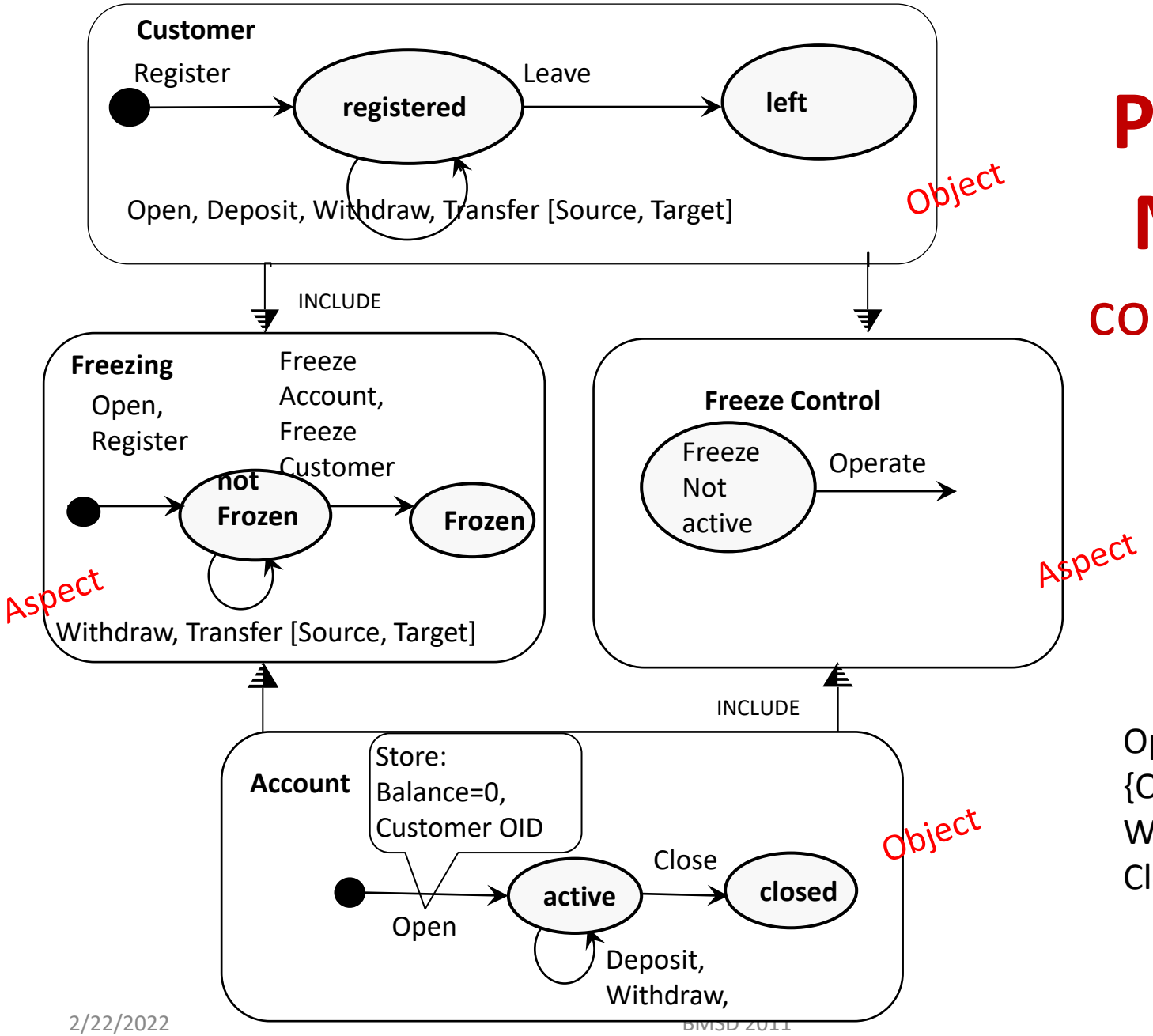
Date:Date
Customer OID: Customer
Reason: String

Open

Date:Date
Owner OID: Customer
Account: String



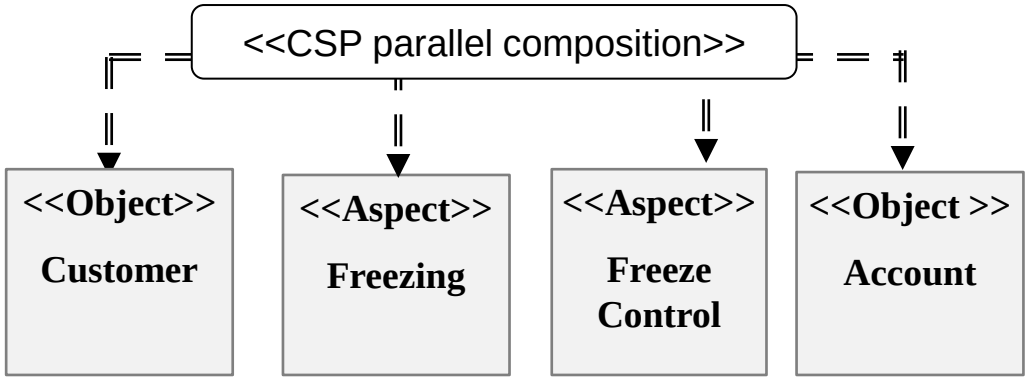
A protocol machine can ignore, accept and refuse events depending of its state



Protocol Model: composition rules

Operate=
{Open, Deposit,
Wthdraw, Transfer,
Close, Leave, Register}

CSP parallel composition of protocol machines



- Localization of concerns
- Synchronous handling of an event by different protocol machines
- Deterministic behaviour
- **Invasive aspects are not possible**
- CSP composition preserves the order of sequences of events and makes possible **localization of reasoning**

Conclusions