# Hierarchical Object Modeling with ADORA

Martin Glinz

Requirements Engineering Research Group
University of Zurich
http://www.ifi.unizh.ch/~glinz

Universität Zürich
Institut für Informatik

# Overview of the talk

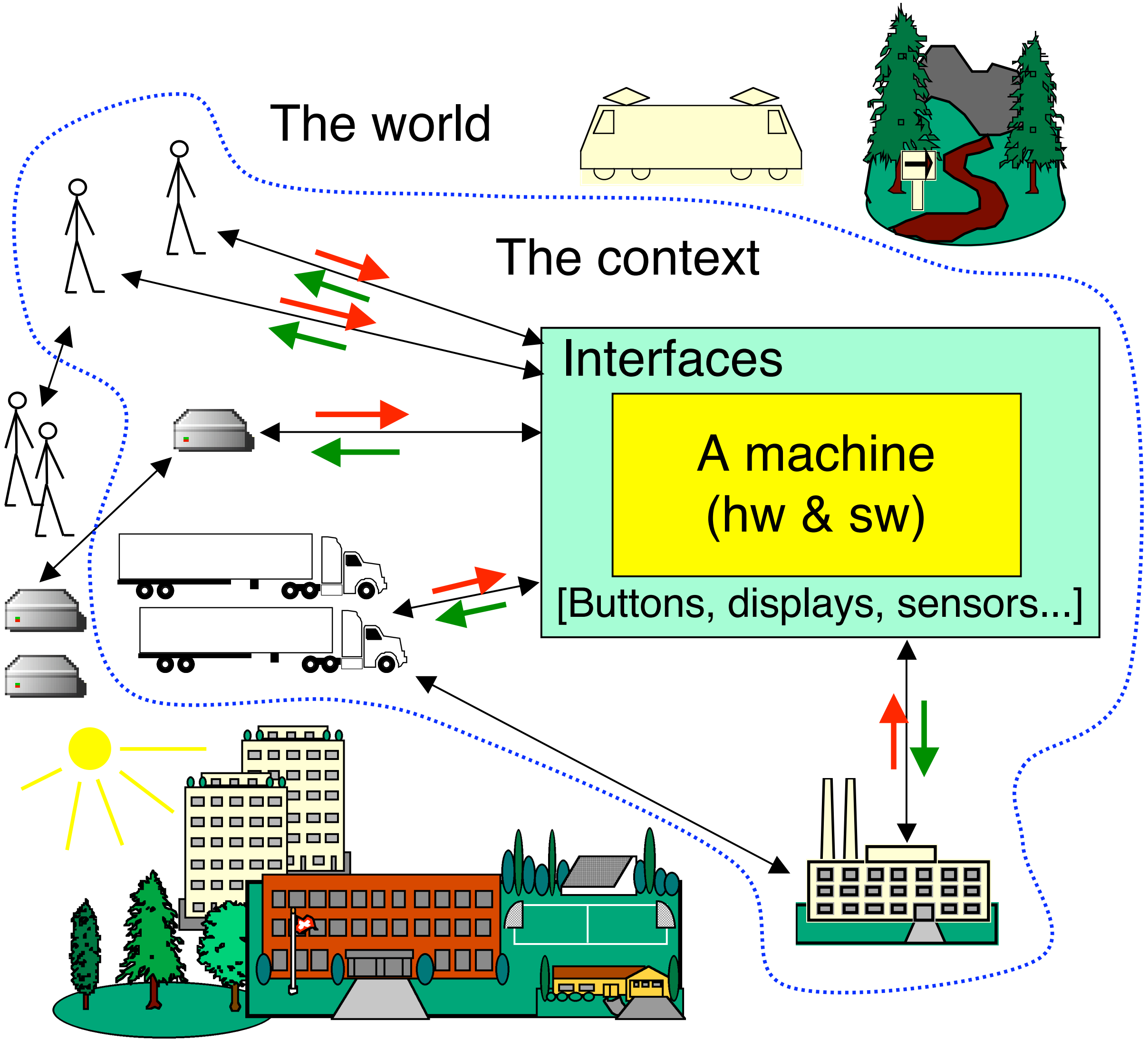
❍ Introduction
- Modeling software requirements
- Why UML is not the ultimate solution of the problem

❍ ADORA – A fresh look at object-oriented modeling of software
- Some basic problems and how ADORA solves them
- An overview of the language
- About visualizing ADORA models
- The ADORA tool
- Exploring new avenues: simulation and aspect-orientation

❍ Conclusions

❍ Demonstration of the ADORA tool prototype

# Introduction: modeling software requirements (1)



**The problem of describing requirements:**

- Identify the context
- Describe the stimuli (from the context)
- and the responses (to the context)
- and the restrictions (performance, qualities, constraints)

# Introduction: modeling software requirements (2)

Specifying requirements with models means

- Model the machine ↔ context interaction

  basically a set of relations

  + state

  [state: what the machine must know about the state of the world]

- Hence, add a model the machine's view of the world

  ... yielding a specification of the functional requirements

- Finally, add a specification of the restrictions

# UML does it all !??

- UML seems to satisfy all needs:
  It comprises sub-languages for nearly every modeling paradigm

But:

- Serious problems with UML 1.x as a requirements modeling language

  [Glinz (2000): Problems and Deficiencies of UML as a Requirements Specification Language. *IWSSD-10*. San Diego]

- Serious problems with UML 1.x as an architecture modeling language (not a topic of this talk)

- UML 2.0
  - solves some problems of UML 1.x (e.g. architectural modeling)
  - lets all the requirements modeling problems persist
  - makes some problems worse (e.g. the abundance of features)

# The ADORA approach

ADORA (Analysis and Description of Requirements and Architecture)

- is a new approach to object-oriented modeling of specifications
- on the basis of
  - Modeling with abstract objects
  - Hierarchical decomposition of models
  - An integrated model with views
  - An adaptable degree of formality
  - Contextual visualization of models

# Class modeling considered harmful (1)

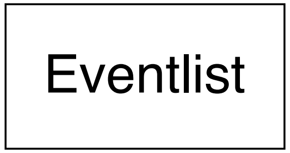
Example: Imagine an information system that supports control and dispatching of emergency operations (police, ambulance service,…)

- In every Operator Support component we need
  - the list of pending events
  - the event currently being handled
  - the list of processed events
- In the Archive component we have
  - a global event history
- All these items belong to the same class: Eventlist

# Class modeling considered harmful (2)

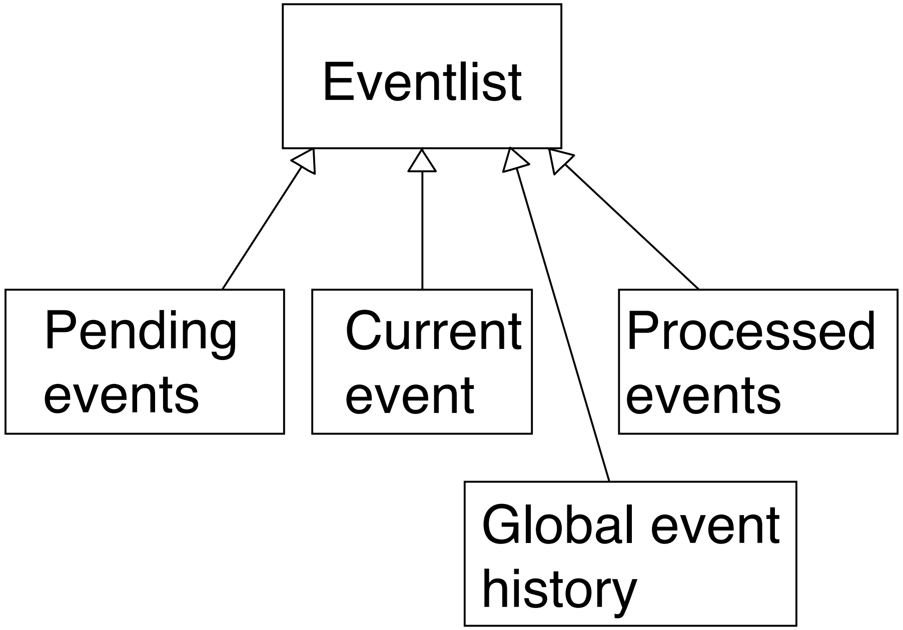
In a class model we have to model

either


Eventlist

Bad: does not model essential elements of the problem

or


Eventlist

Pending events

Current event

Processed events

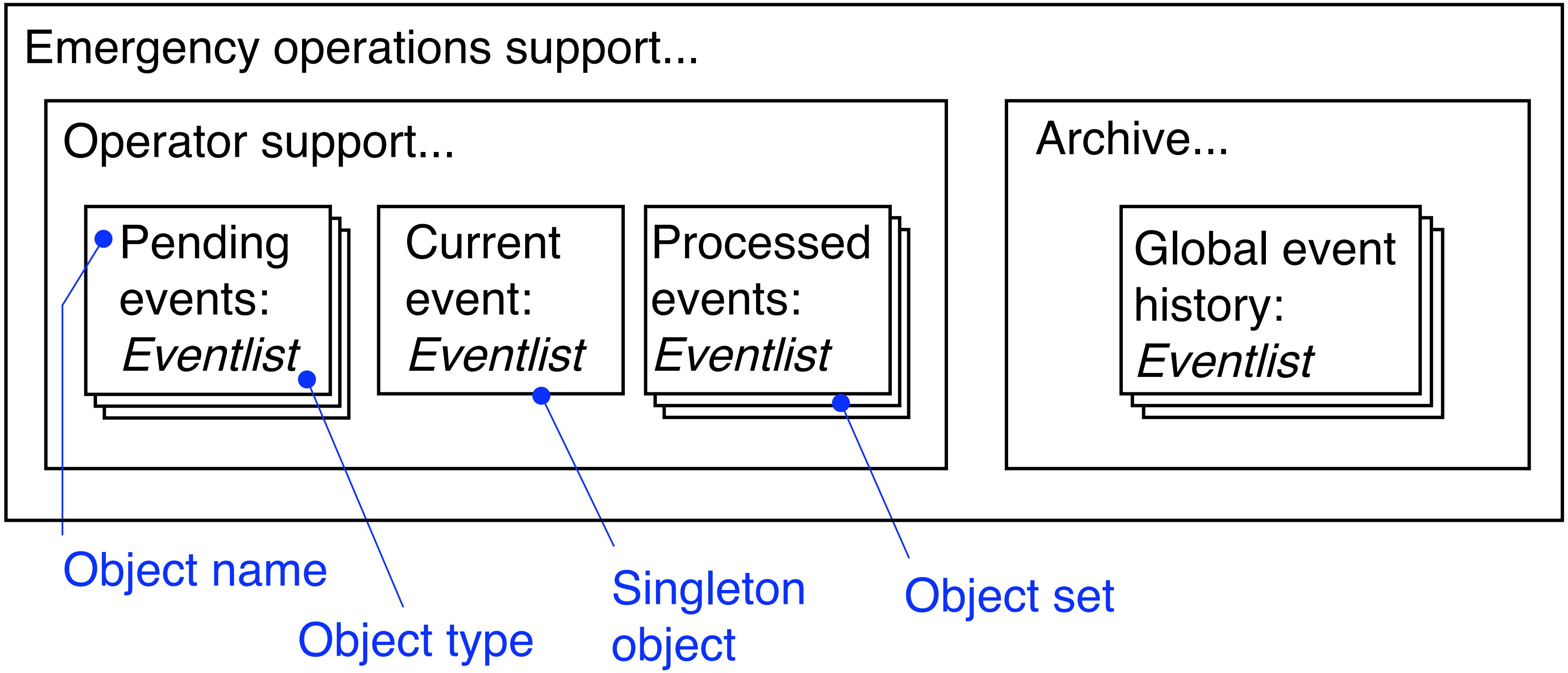
Global event history

Unnatural: subclasses are structurally identical

# Class modeling considered harmful (3)

- Class models do not work
  - when more than one object of the same class has to be modeled
  - when collaboration between objects have to be modeled
- Class models cannot be decomposed hierarchically
  - What is the semantics of a class containing other classes?
  - What happens when different objects of a class belong to different parts of a system?
- Subclassing is a workaround, no solution

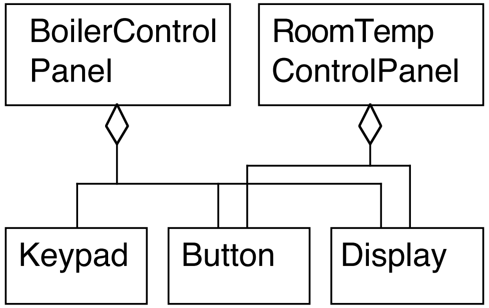
# Abstract objects: how ADORA does it


Emergency operations support...
Operator support...
Pending events: *Eventlist*
Current event: *Eventlist*
Processed events: *Eventlist*
Archive...
Global event history: *Eventlist*
Object name
Object type
Singleton object
Object set

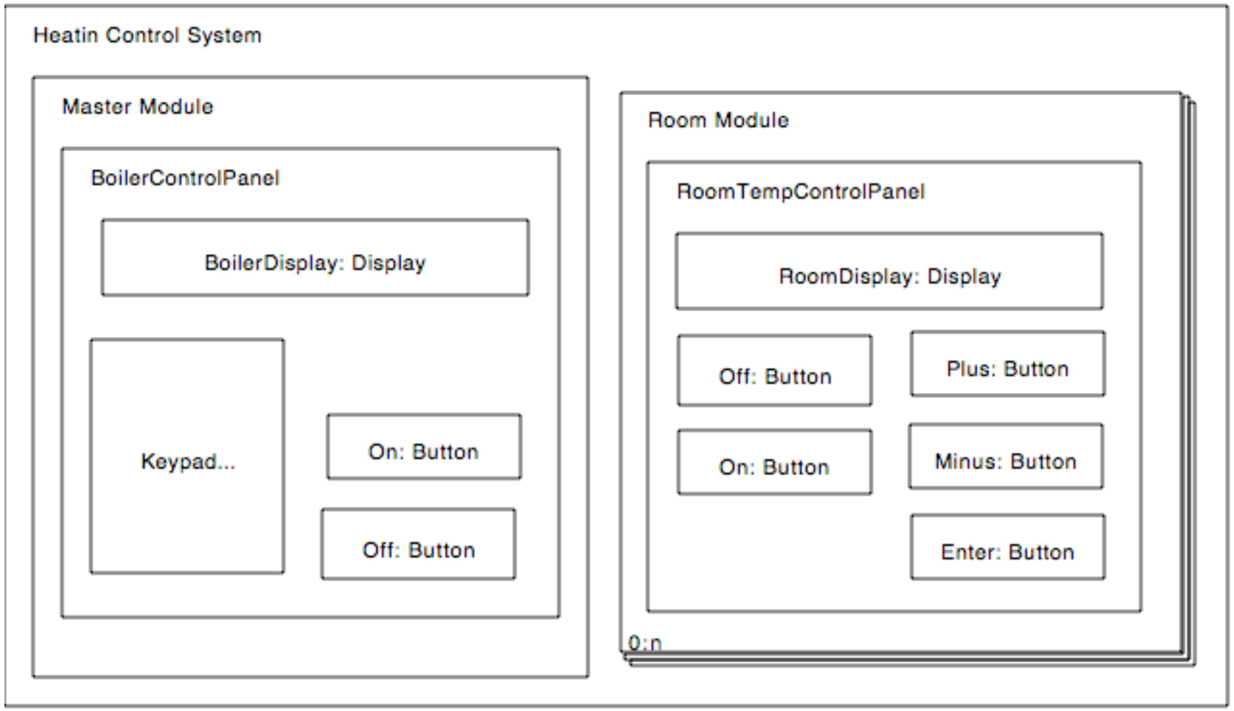
# Hierarchical decomposition of models

Example: A distributed heating control system

What UML can do



What ADORA does instead



# Decomposition in modeling languages

Looking back

- Structured Analysis had it
- Entity-Relationship-models never got it
- Object-oriented models inherited the problem from ER-models
- Containers (à la UML packages) do not suffice

Why do we need decomposition for specifications?

- Making large specifications manageable
- Distributing work
- Understanding large models

# An integrated model with views

- UML is a collection of models (class diagrams, class descriptions, object diagrams, sequence diagrams, collaboration diagrams, state diagrams, activity diagrams, use case diagrams, use case descriptions, component diagrams, packet diagrams,...)

- A nightmare if you want to achieve consistency, completeness, traceability…

- ADORA avoids this problem by

  …integrating all these aspects into a single, coherent model

  ...ensuring usability and readability by providing

  - Views
  - Hierarchical decomposition

# The ADORA view concept

- The Base view: Objects and object sets
  + hierarchy
  + annotations

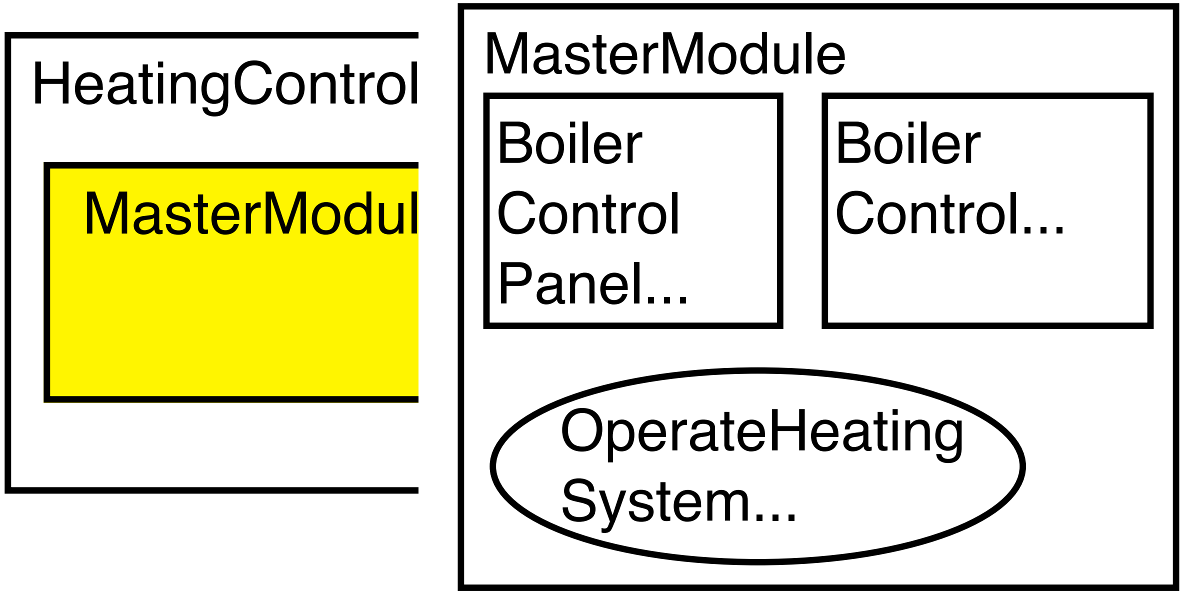
- Combined with zero or more of the following views
  - Structural view: static relationships and relationship abstractions
  - Behavioral view: dynamic behavior expressed with a statechart-like state machine hierarchy
  - Functional view: detailed definition of an object (attributes, methods)
  - User view: User-system interaction modeled with scenarios
  - Context view: how a system is embedded in its environment

- Types and the type hierarchy are defined and visualized separately

# Visualizing hierarchical models

Zooming into MasterModule

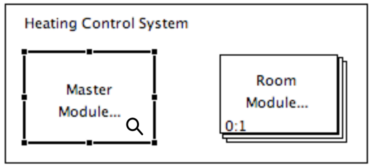
Traditional visualization would yield (explosive zooming):



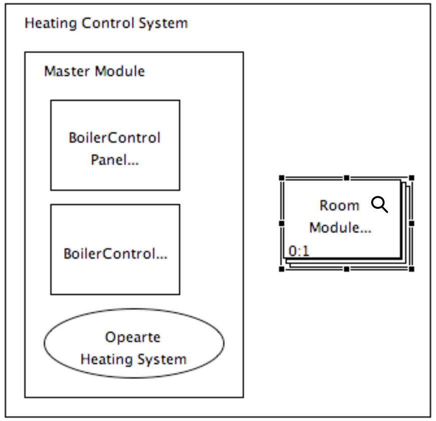
# Contextual visualization in ADORA

Successively zooming in:

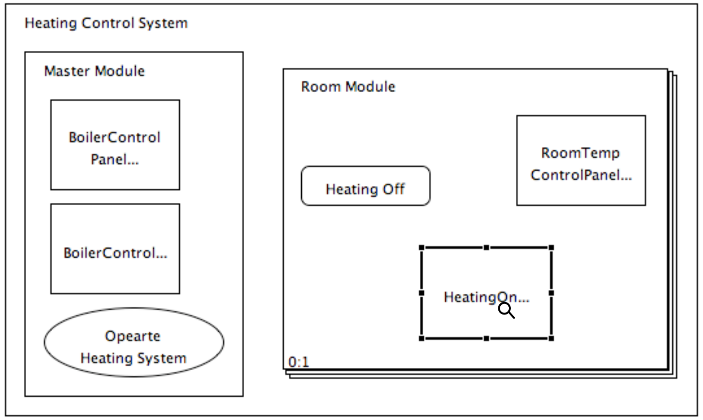
①



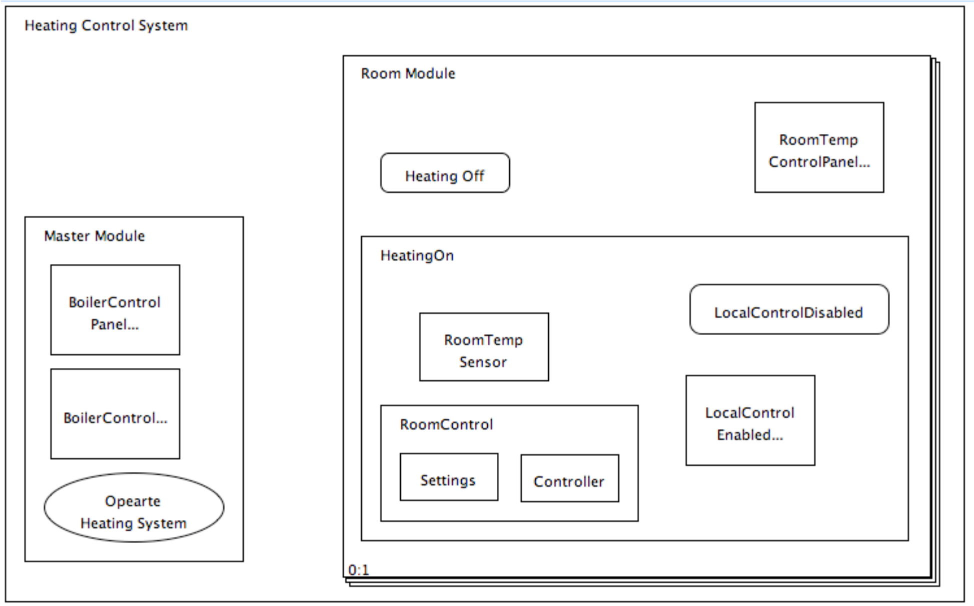
②



③



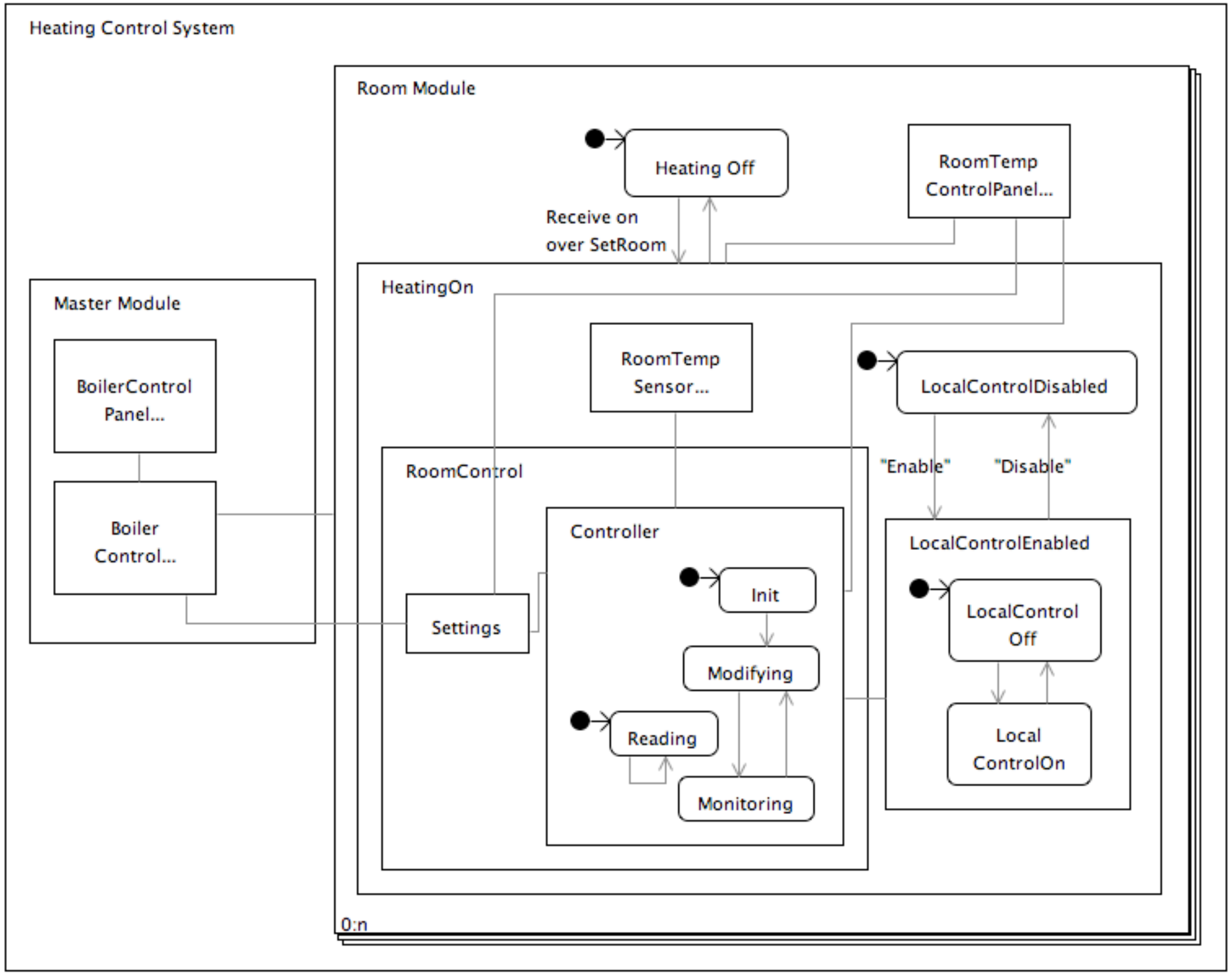
④



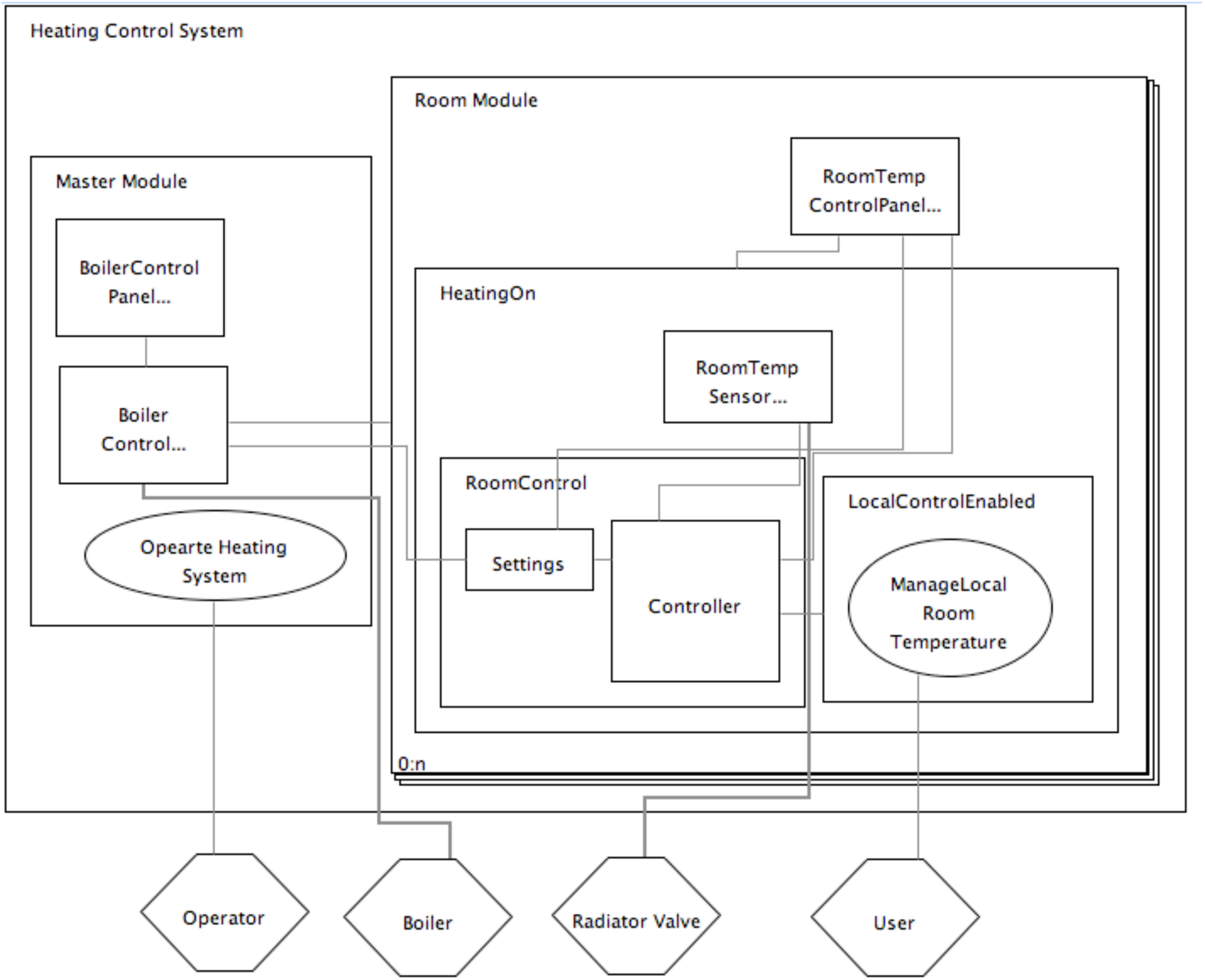
# Combining the base view with other views

Structural view: relationships

Behavioral view: states&transitions



# The context view and the user view



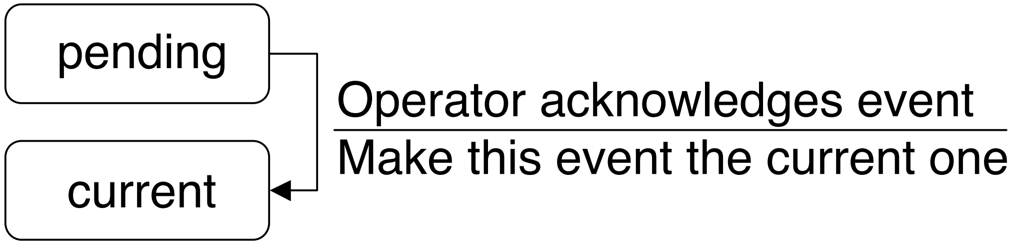
# Adaptable degree of formality

ADORA provides a consistent framework for specifying problems

...informally:

**object** HeatingControlSystem...
**purpose** "Provide a comfortable control for the heating of a building with several rooms."
**end** HeatingControlSystem.

…semi-formally:



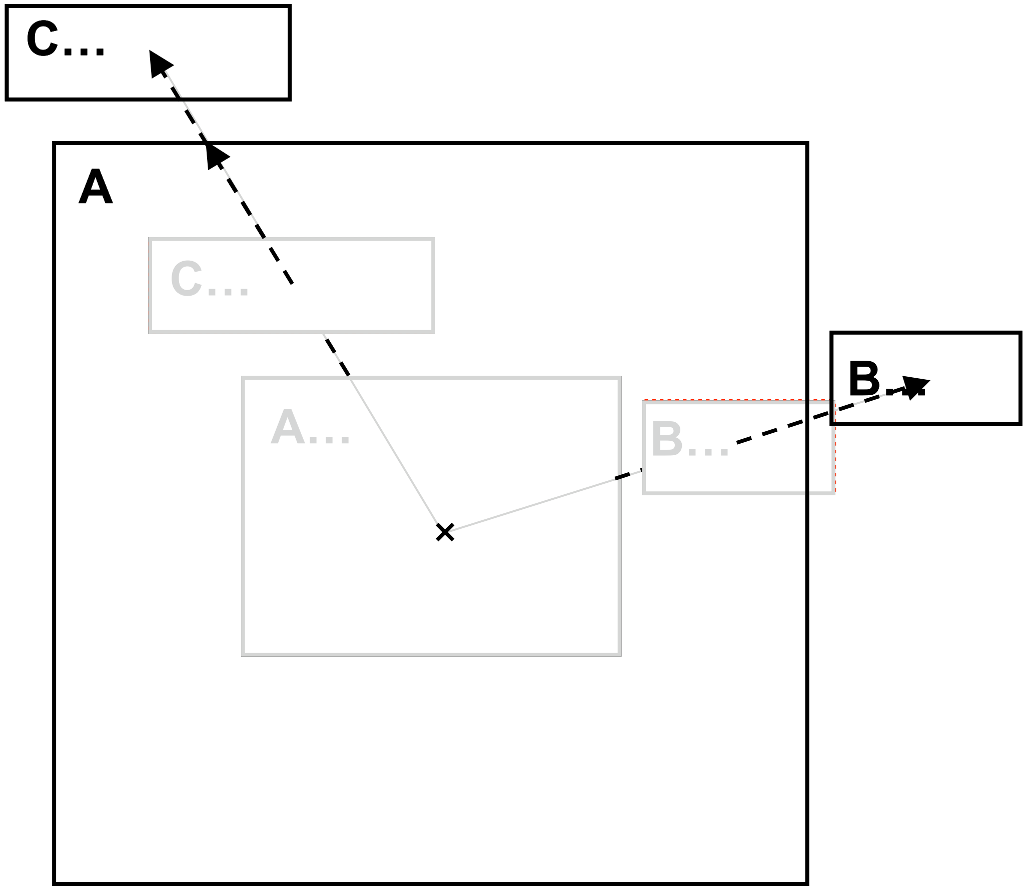
…or formally:

behavior and functionality can be described formally

# Contextual visualization

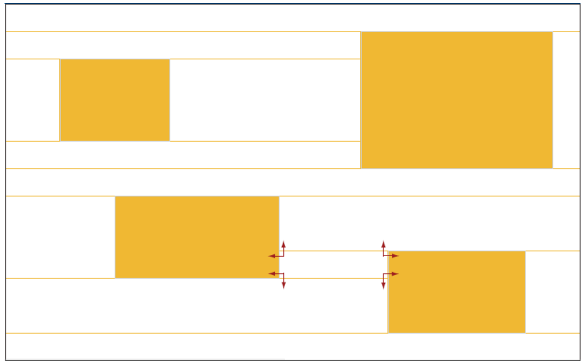
- Principal ideas
  - Use fisheye views for visualization
  - Visualize according to the decomposition structure
- Integrates local detail and global context in a single view
  - eases orientation
  - minimizes cognitive overhead for navigation in the model
  - supports the inherent abstraction mechanisms in the object model
- Works on any given layout, adjusting it incrementally and preserving it as far as possible
- User may re-arrange a layout without losing these rearrangements when zooming

# The Layout algorithm – principal idea



# Line Routing

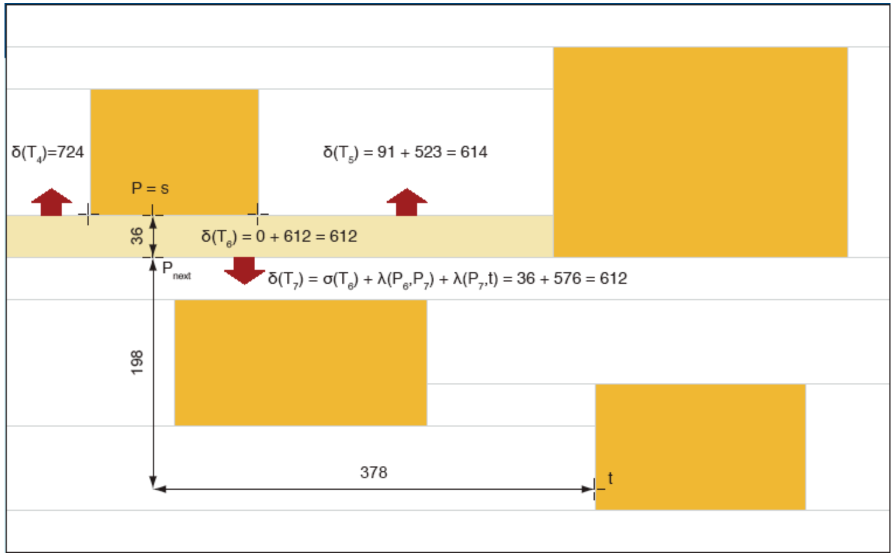
- Dynamic diagram generation requires dynamic line routing
- Existing algorithms
  - don't route in real time (e.g. Lee's algorithm used in VLSI design)
  - or don't preserve the given arrangement of nodes



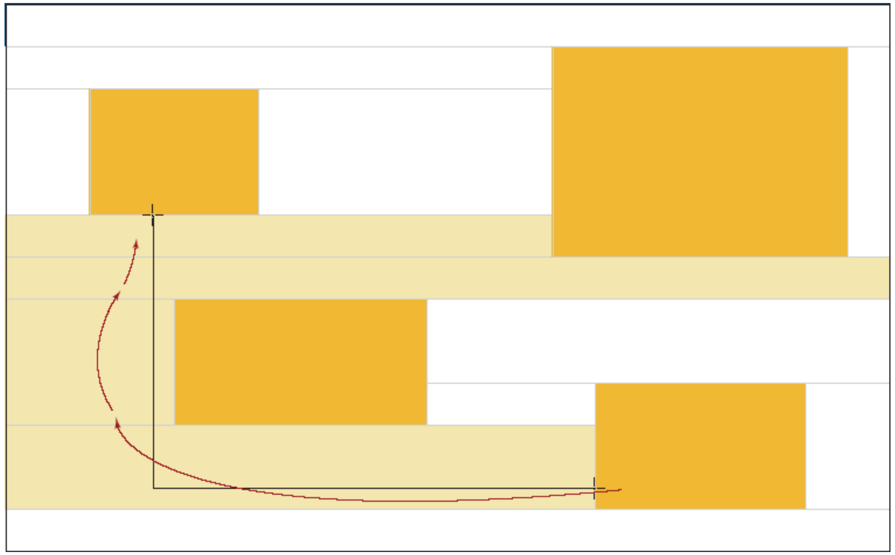
- Concepts:
  - Represent free space with maximum horizontal tiles instead of a uniform grid of cells
  - Adapt Lee's algorithm to this data structure, making it fast enough for real time routing
  - Compute lines in two decoupled steps
  - 1. Determine the tiles that the shortest path goes through
  - 2. Calculate the actual line within these tiles

# Calculating a line

Step 1: Calculate a shortest/ cheapest path from source to target


$\delta(T_4)=724$
$\delta(T_5) = 91 + 523 = 614$
$P = s$
36
$\delta(T_6) = 0 + 612 = 612$
$P_{next}$
$\delta(T_7) = \sigma(T_6) + \lambda(P_6,P_7) + \lambda(P_7,t) = 36 + 576 = 612$
198
378
t

Step 2: Calculate the actual line, e.g. as polyline or spline



# The ADORA tool

- Initially a hand-made model editor implemented in Java
- 2006 completely re-implemented as an Eclipse plug-in
- Supports drawing & navigating
- No code generation

- Both runtime and code easily available under an open-source license

# Exploring new avenues

- Simulation of models that are neither formal nor complete
- Aspect-oriented modeling

# Simulation of models in ADORA

- Motivation
  - Evolutionary modeling requires early and frequent model validation
  - ➪ Reviewing becomes too expensive
  - ➪ Classic simulation techniques are not applicable, because models are incomplete and semi-formal
- Concepts
  - Develop a technique for simulating incomplete, semi-formal models
  - Re-validate changed models by regression simulation
  - Let the modeler interactively specify missing behavior or functionality in a simulation run
  - Let regression simulation nevertheless run automatically
  - Use simulation traces for visualizing failed simulation runs and localizing defects in the model

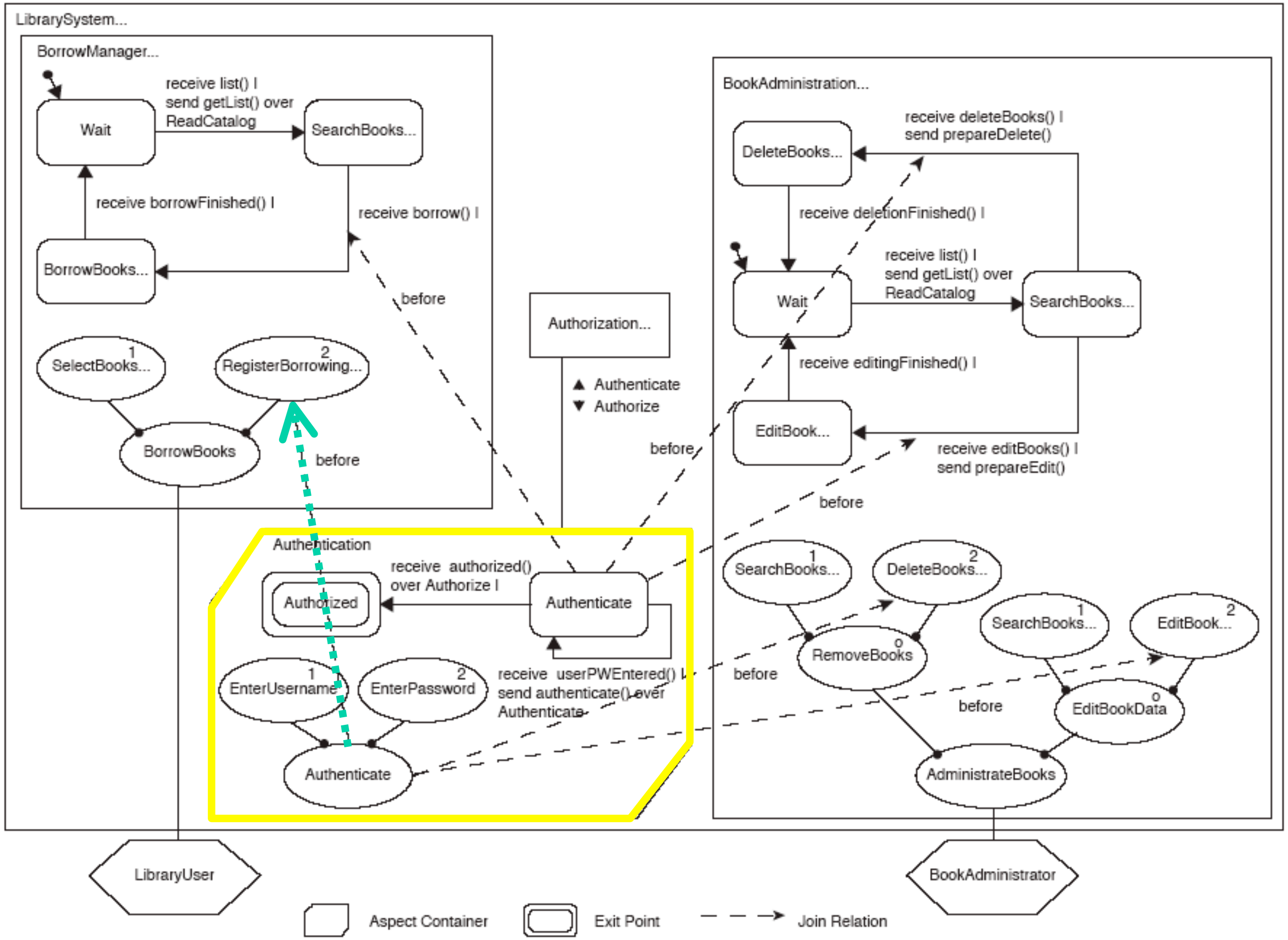
# Aspect-oriented modeling

- Motivation
  - Model crosscutting requirements separately and integrate (weave) them automatically into the base model on demand
- Concepts
  - Extend ADORA by so-called aspect containers that contain model fragments describing crosscutting functionality and behavior
  - Explicitly model join points (no obliviousness)
  - Define formal model weaving semantics
  - Let the ADORA tool generate weaved models on demand, using its capabilities for generating and incrementally adapting diagrams

# Aspect-oriented modeling – example

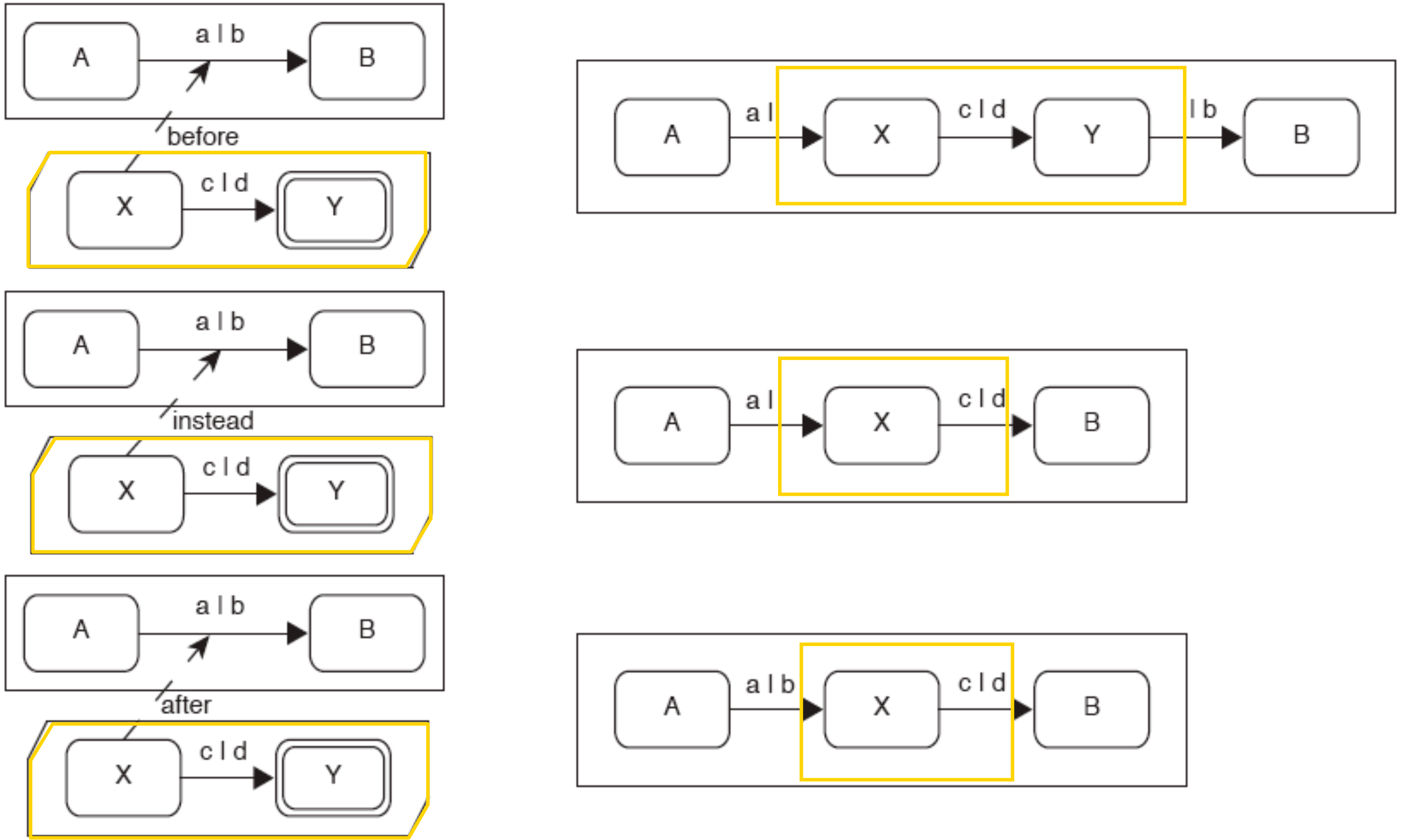
Join relation

Aspect container



# Aspect-oriented modeling – example – 2

## Weaving semantics for statecharts



# State of work

## Current state

- Definition of language finished
- Prototype ADORA tool is available

## Problems

- Tool development very time-consuming
- Still lots of minor problems that impede usability
- Major unsolved problem: stability of generated layouts

## Plans

- Solve the tool problems
- Gain experience from application in real projects
- Do we need it all? Towards a simpler modeling language
- Investigate further issues: process, how to get from goals to models, ...

# Conclusions

- There is life beyond UML.
- Hierarchical object modeling with an integrated model
  - yields a powerful approach to object-oriented specification
  - solves major problems plaguing UML and related approaches
  - could make a real difference in practical application ... but that is yet to be proved
  - opens promising new research directions.