

Modeling Systems with ADORA

Ledri Thaqi
Faruk Acibal

Outline

- Introduction
- Key Concepts of ADORA
- The ADORA Language
- Contextual Visualization
- Validation
- ADORA vs. UML

Introduction

- Goal: Modeling Requirements & Architectural Design
- Historically various problems in models
- UML with intention of creating industry standard
 - Still problems present with UML
- ADORA wants to tackle all these problems
- Analysis and Description Of Requirements and Architecture
- Primarily designed for Requirements Specification & Logical Level System Design
- ADORA's basic concepts are different than UML

Key Concepts of ADORA

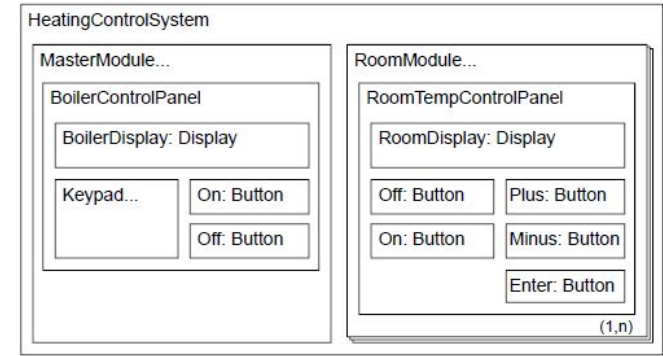
1. Working with abstract objects (instead of classes)
2. Structuring the system being modeled with hierarchical decomposition
3. Using an integrated model (instead of collections of models)
4. Allowing to specify different parts with varying degrees of formality
5. Visualizing models in context (show abstractions of surrounding context)

The ADORA Language

- Basic hierarchical object structure (base view) + set of aspect views
- Aspect Views
 - Structural View
 - Behavioural View
 - Functional View
 - User View
 - Context View

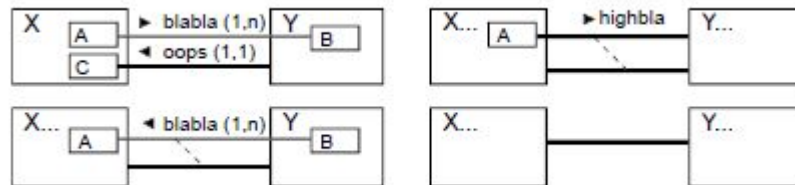
Base View

- Objects and object sets
 - Abstract representation(s) of instance(s)
 - Consists of *properties* and *parts*
 - *Properties are attributes, relationships, operations or standardized properties*
 - *Parts are other objects, object sets, states and scenarios*
- *Types of objects* -> allows tool support
- *Standardized properties* allowed
- Name and structure of properties are user-definable -> flexibility



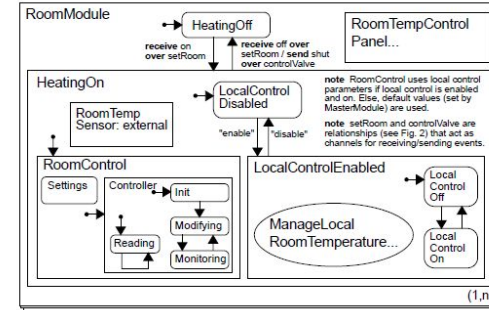
Structural View

- Adds information about relationships between objects
- Associations
 - Directed binary relationships
 - Whenever A accesses public attribute of B or invokes operation of B
- Abstract relationships
 - Once associations exist in the more detailed views
 - Has only a direction iff all underlying relationships have the same direction
- Information Flow
 - Also modeled with abstract relationships



Behavioural View

- Adds dynamic system behaviour
- Statechart-based state machine hierarchy
- Allows behaviour on all levels of abstraction
- Events are only broadcasted within the object
 - Send it to needed receivers
 - Allows for synchronized systems within, but asynchronization between systems
- Write in the state transition arrows or as table
 - Big models require table to keep it readable



State Transition Tables for Controller

Init, Monitoring → Modifying				
IN LocalControlEnabled	Y	•	Y	•
IN LocalControlEnabled LocalControlOn	Y	N	Y	N
ActualTemp > Settings.CurrentTemp(now)	N	•	Y	•
ActualTemp < Settings.CurrentTemp(now)	Y	•	N	•
ActualTemp > Settings.DefaultTemp(now)	•	N	•	Y
ActualTemp < Settings.DefaultTemp(now)	•	Y	•	N
send open over control/valve	✓	✓		
send close over control/valve			✓	✓
Modifying → Monitoring				
180 s IN Modifying	Y			
Reading → Reading				
10 s IN Reading	Y			
self ReadSensor/value	✓			

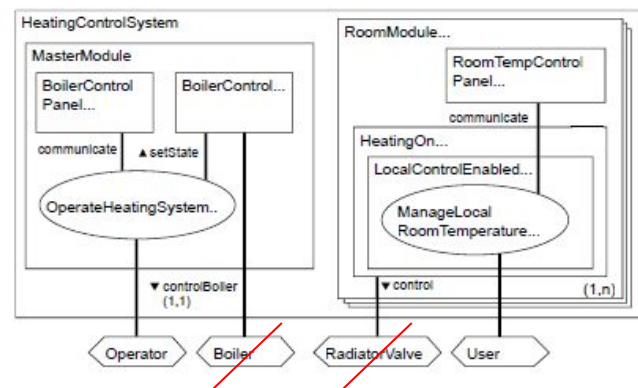
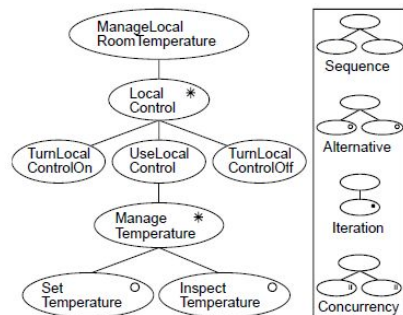
Functional View

- Adds properties that have not been defined by the type
- Not combined with other views -> has its own textual separate form
- Syntax supports from informal text to logic formulae

```
object Settings
part of RoomControl
provides    Actual Temp;
            "Operations to inspect / manipulate control intervals (consisting
            of start time and desired temperature), both default and user-
            defined"
requires    //nothing
type
    TempIntervals is list of TempInterval;
    TempInterval is structure of (start: Time, temp: Temperature)
attribute    //public attributes
    ActualTemp: Temperature
var          //private attributes
    DefaultIntervals: TempIntervals; //default temperature settings
    UserSetIntervals: TempIntervals //user-defined temperature settings
syncoperation CurrentTemp (t: Time): Temperature
pre    1 ≤ t ≤ 24*60 //minutes
post    with usi = UserSetIntervals;
        //returns desired temperature at current time t
        CurrentTemp = usi[i].temp and usi[j].start ≤ t and
        not exists j • (usi[j].start < usi[i].start ≤ t)
end CurrentTemp
syncoperation DefaultTemp (t: Time): Temperature
    "Same as CurrentTemp, but returns current default value"
end DefaultTemp
operation RevertToDefault
    post    UserSetIntervals' = DefaultIntervals
end RevertToDefault
"Settings must also provide operations for setting and deleting intervals and
for browsing the currently defined intervals."
end Settings.
```

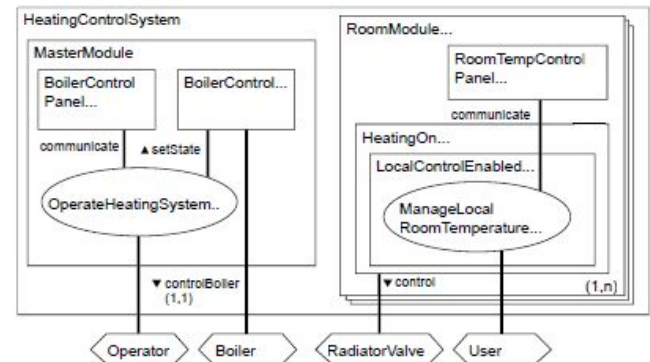
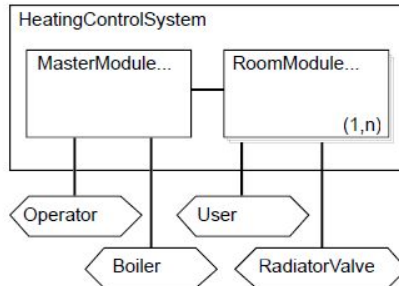
User View

- Adds actors which interact with scenarios and all interactions between scenarios and objects
- Base View has ovals for scenarios
- Scenarios require exactly one starting point & one regular exit point
 - Only exception handling for other exits
- Jackson style diagrams for scenario composition



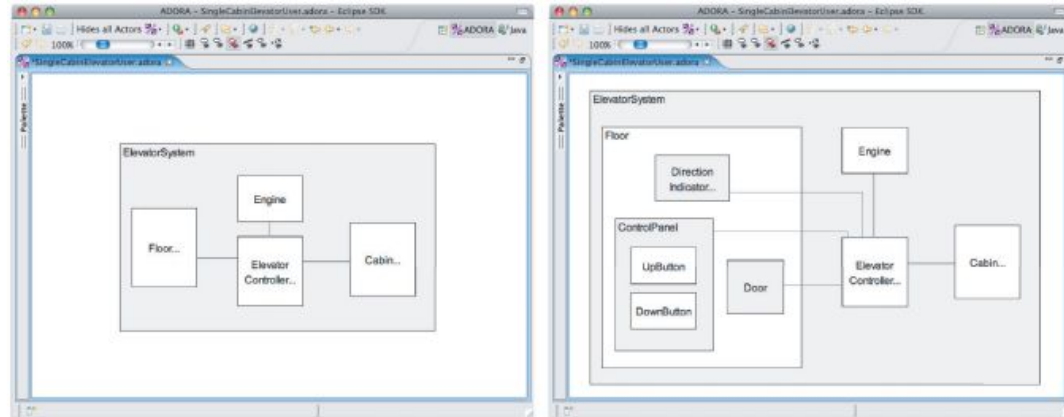
Context View

- Adds actors and objects in the environment and their relationships with the system
- Allows modelling rich context
 - Relationships between actors or objects in the system environment
- External objects
 - Black boxes embedded in the system



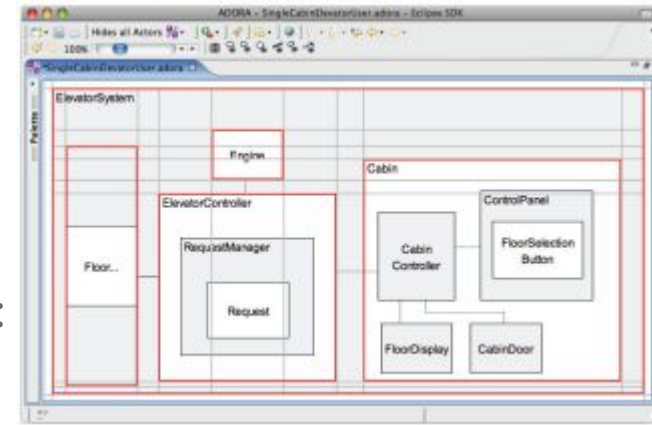
Contextual Visualization

- Use of not only explosive zooming but also fisheye views
- Global context and local detail in one view
- Based on hierarchical structure of an ADORA model
- Enable views with more than one focus
- Physical navigation
- Logical navigation
- Zooming-in and zooming-out
- Fisheye Zoom algorithm



Contextual Visualization–Fisheye Algorithm

- Adapts layout accordingly to zooming-in or zooming-out
- Rearranges the layout by resizing and moving nodes
- It provides:
 - No overlapping of nodes
 - Multiple points of interest zoomed in at the same time
 - Permitting model editing
 - Running in real time
- Layout arrangements useful in following situations:
 - Navigation in the hierarchy
 - Smart editing
 - Filtering nodes



Validation

Fundamental qualities of a specification language:

- Easy to comprehend
- Users must like it

Validation of ADORA through an experiment with the following goals:

- Determine the comprehensibility of an ADORA specification
- Determine the acceptance of the fundamental concepts of ADORA

Both qualities were validated on its own and compared with an UML specification

Experiment

- Partial specification of a distributed ticketing system both in ADORA and UML
- Questionnaire consisting of two parts:
 - Objective part aimed to measure the comprehensibility of an ADORA model (30 questions)
 - Subjective part tested the acceptance of ADORA vs UML (14 questions)
- Two groups of PhD students answered anonymously
- The results:
 - Percentage of errors when reading a model was 9.9% for ADORA and 18.1 % for UML
 - 87.8% of participants who answered about ADORA were confident about their answers compared to 76.1 % for the UML model
 - An ADORA specification is easier to comprehend than a UML specification

ADORA vs. UML

- Differ too much in their basic concepts
- UML has major problems and weaknesses as a requirements specification tool
- ADORA has a concept of an integrated, hierarchically decomposable model
- UML has a concept of a flat, mostly non-decomposable collection of models
- Integration of ADORA concepts in UML requires major changes in metamodel
- ADORA approach on its own separately from UML

Thank you for your attention !
Questions ?