# Pattern-Based Development of User-Friendly Web Applications

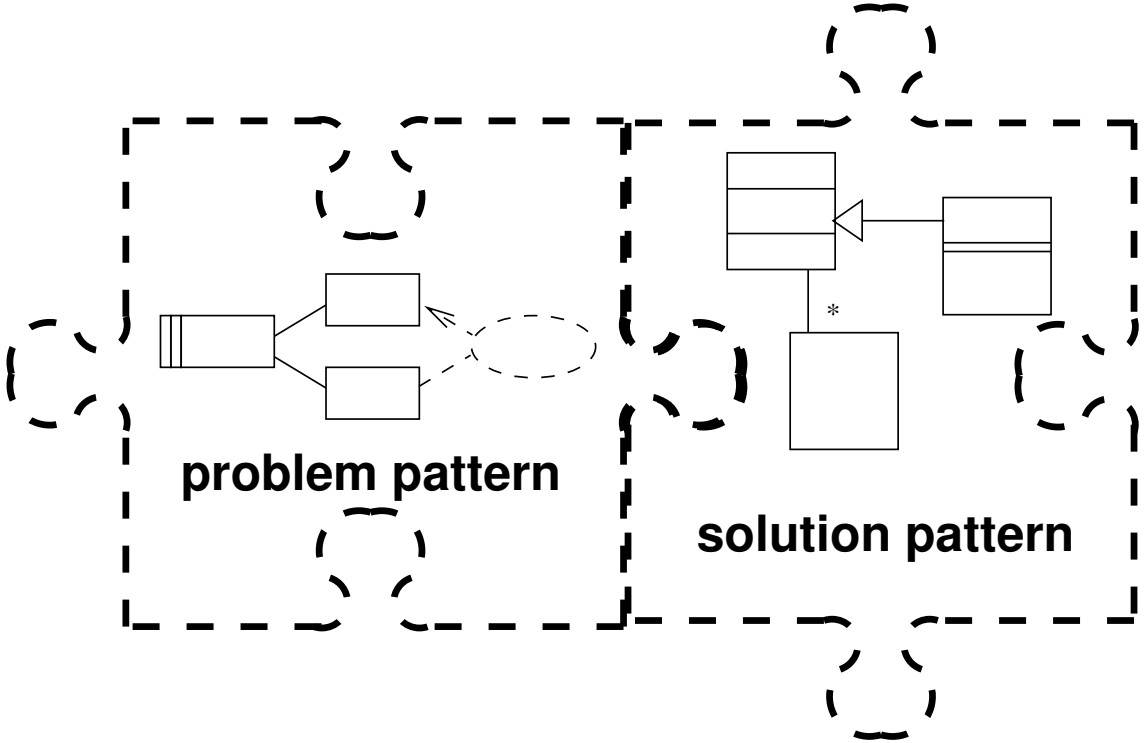
## Workshop on Model-Driven Web Engineering 2006
## Palo Alto, California, USA.

Dipl.-Inform. Ina Wentzlaff and Dipl.-Inform. Markus Specker

University Duisburg-Essen · Faculty of Engineering · Department of Computer Science
Software Engineering · Interactive Systems and Interaction Design

July 11th, 2006

# Overview




**Markus Specker**

- Basic Concepts
- Analysis Step
- Patterns / HCI

**Ina Wentzlaff**

- Pattern Transformations
- Design Step
- UML / CBR

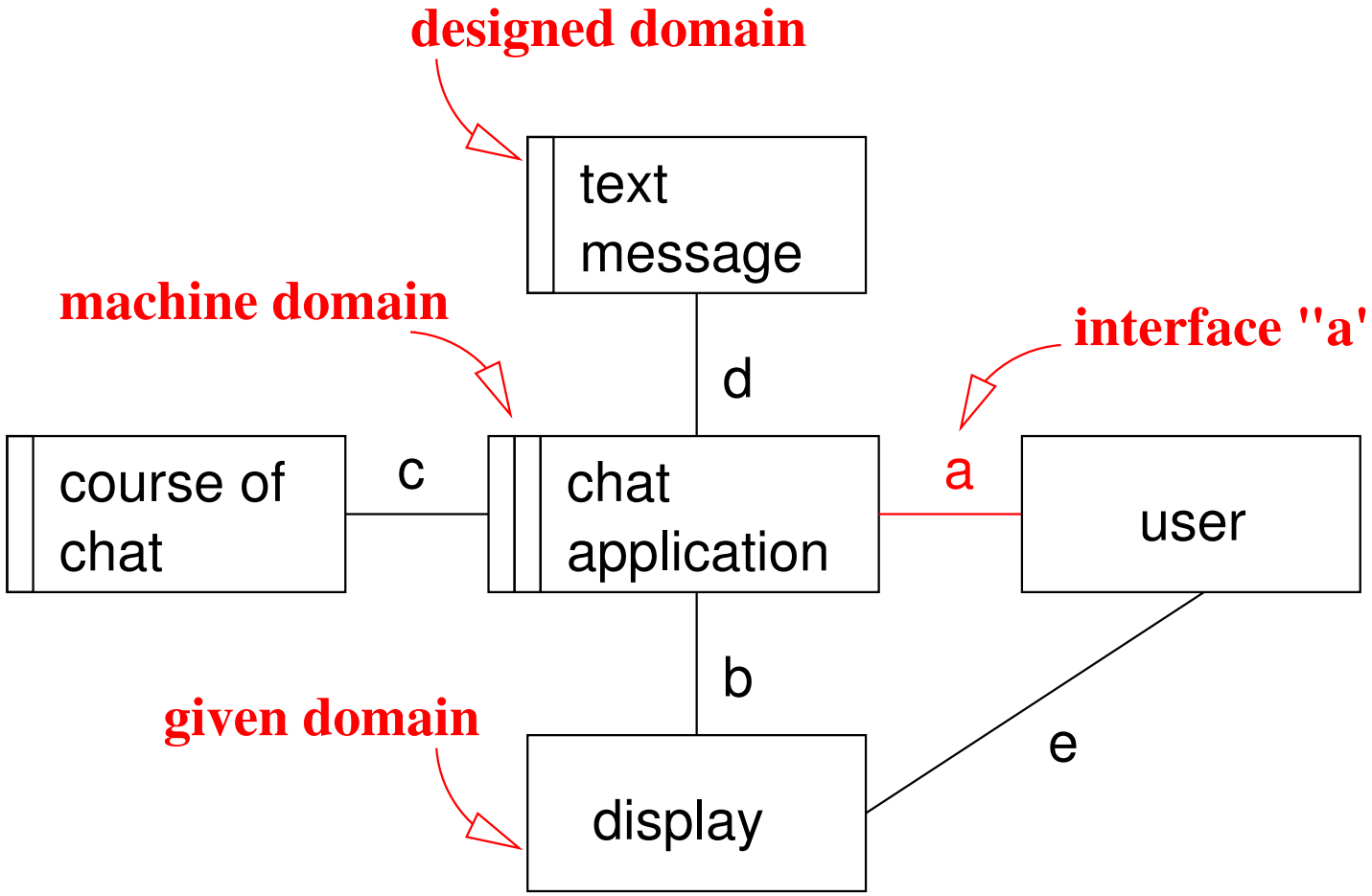
- Widespread usage of patterns
- Comparable or redundant pattern descriptions evolved
- Applicability of patterns through experience
- Gaining any benefits from that? **YES!** ☺
- Developing a chat application as an example
- Combining patterns from HCI with patterns from SE
- Role-based mapping problem patterns to solution patterns
- Achieving pattern-based software development method
- Especially, considering software quality aspects

- Starting from initial software development goal
- Detailing system mission by requirements
- Collecting relevant domain knowledge (facts and assumptions)
- Representing the overall problem situation in a **context diagram** [Jackson]

## System Mission: Chat Application

*"A text message-based communication platform shall be developed which allows multi-user communication via private I/O-devices."*

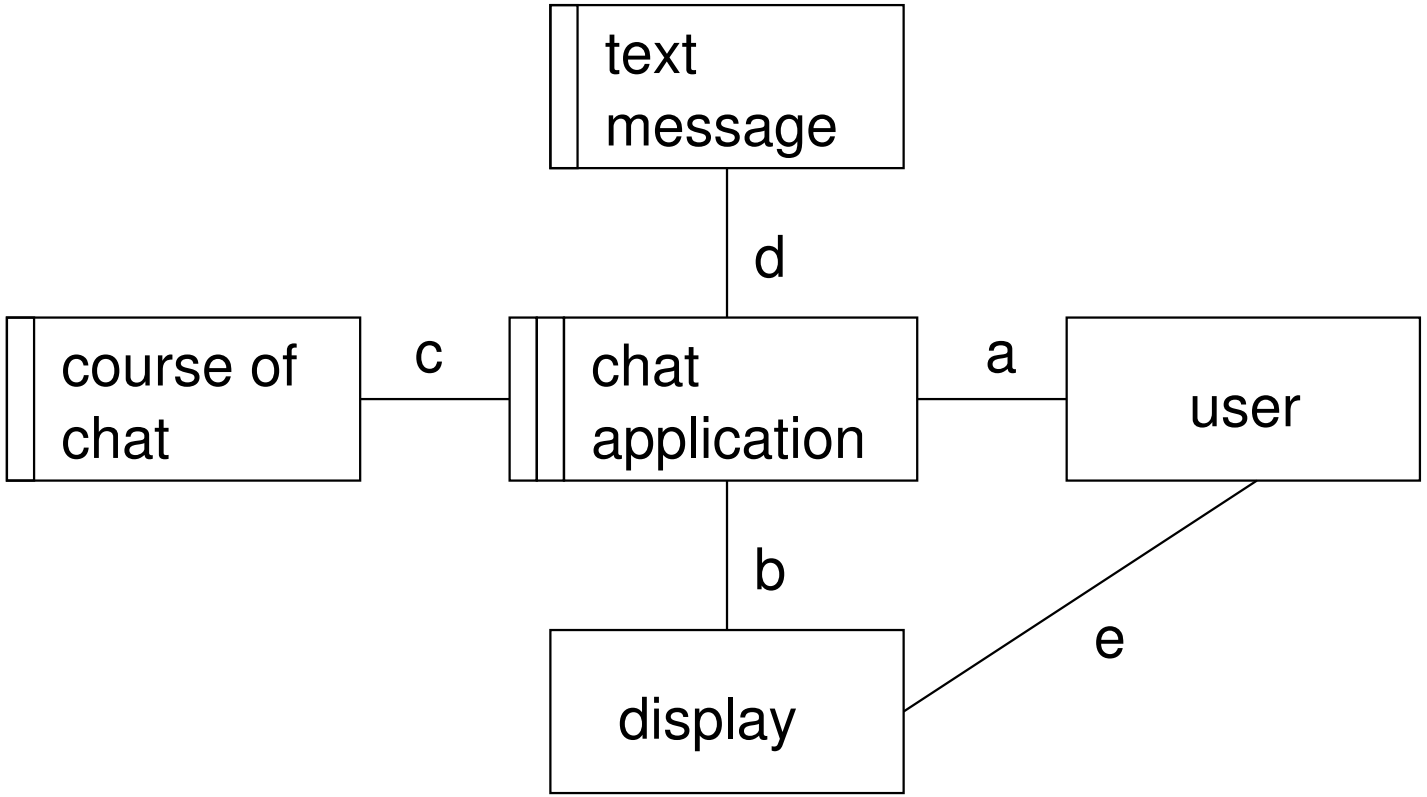
| | |
|---|---|
| R1 | All users can phrase text messages. |
| R2 | The phrased text messages are presented on a graphical display. |
| R3 | ... |
| R4 | ... |
| R5 | ... |
| R6 | ... |
| F1 | Users can only understand the course of chat, if the text messages are presented in the correct temporal order (First In - First Out (*FIFO*)). |
| A1 | Users will follow the course of chat on the display. |




a: {phraseTextmessage, sendTextmessage}
b: {showTextmessage, showCourseOfChat}
c: {registerTextmessage, CourseOfChat}
d: {editTextmessage, MessageText}
e: {followCourseOfChat}

**set of shared phenomena**

R1 All users can phrase text messages.

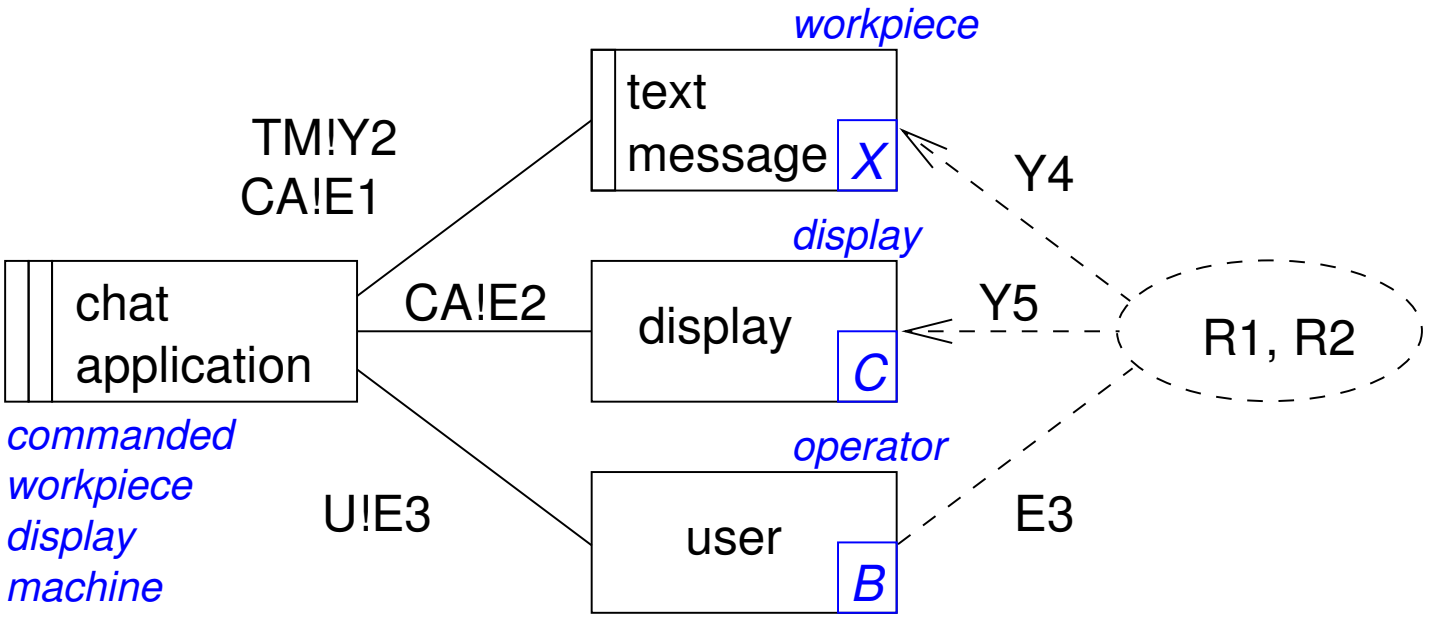



a: {phraseTextmessage, sendTextmessage}
b: {showTextmessage, showCourseOfChat}
c: {registerTextmessage, CourseOfChat}
d: {editTextmessage, MessageText}
e: {followCourseOfChat}

- Splitting complex problem into simple subproblems
- Knowledge-based projection
- accompanied by requirements and decomposition operators
- independent, parallel subproblems (separation of concerns) can be instances of **problem frames**

**Problem Frames [Jackson]**

*Patterns* classifying software development problems

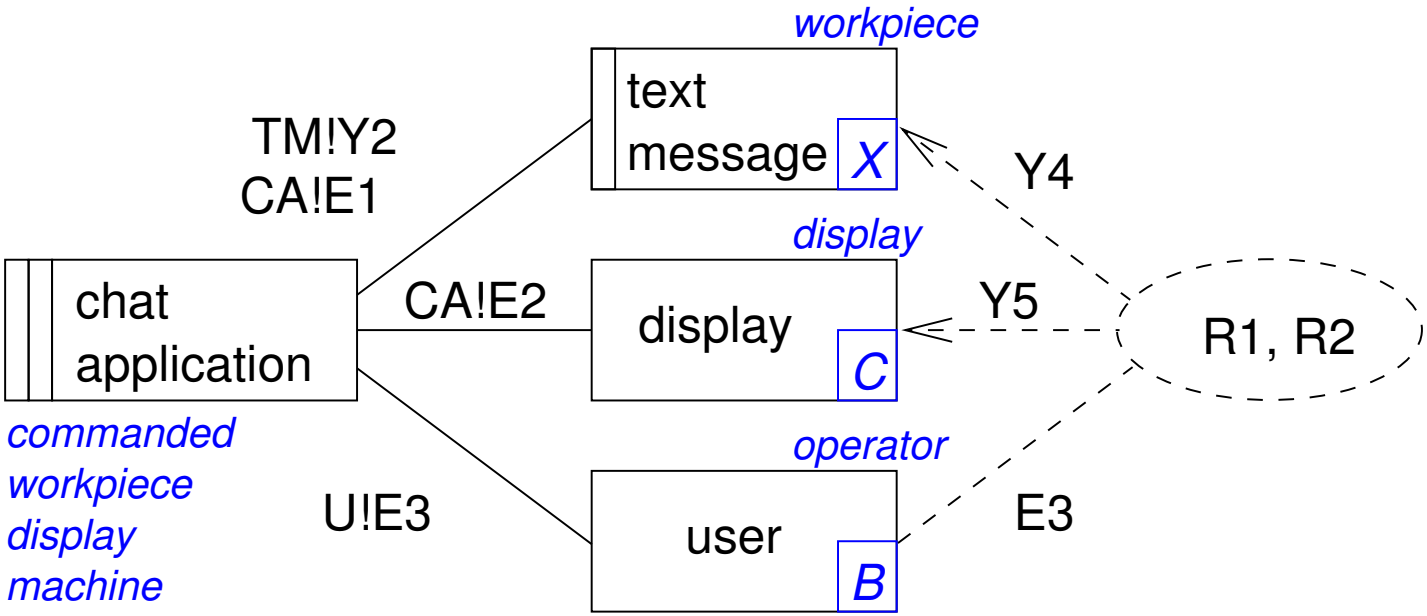



E1: editTextmessage
E2: showTextmessage
E3: phraseTextmessage /
"user phrases text message"
Y2: MessageText
Y4: "content of text message"
Y5: "display text message"

- Instance of problem frame: commanded workpiece display

R1 All users can phrase text messages.

R2 The phrased text messages are presented on a display.




E1: editTextmessage

E2: showTextmessage

E3: phraseTextmessage /
"user phrases text message"

Y2: MessageText

Y4: "content of text message"

Y5: "display text message"

- Functional representations lack of software quality aspects such as usability
- HCI design patterns [Rossi et al., Tidwell, van Welie]

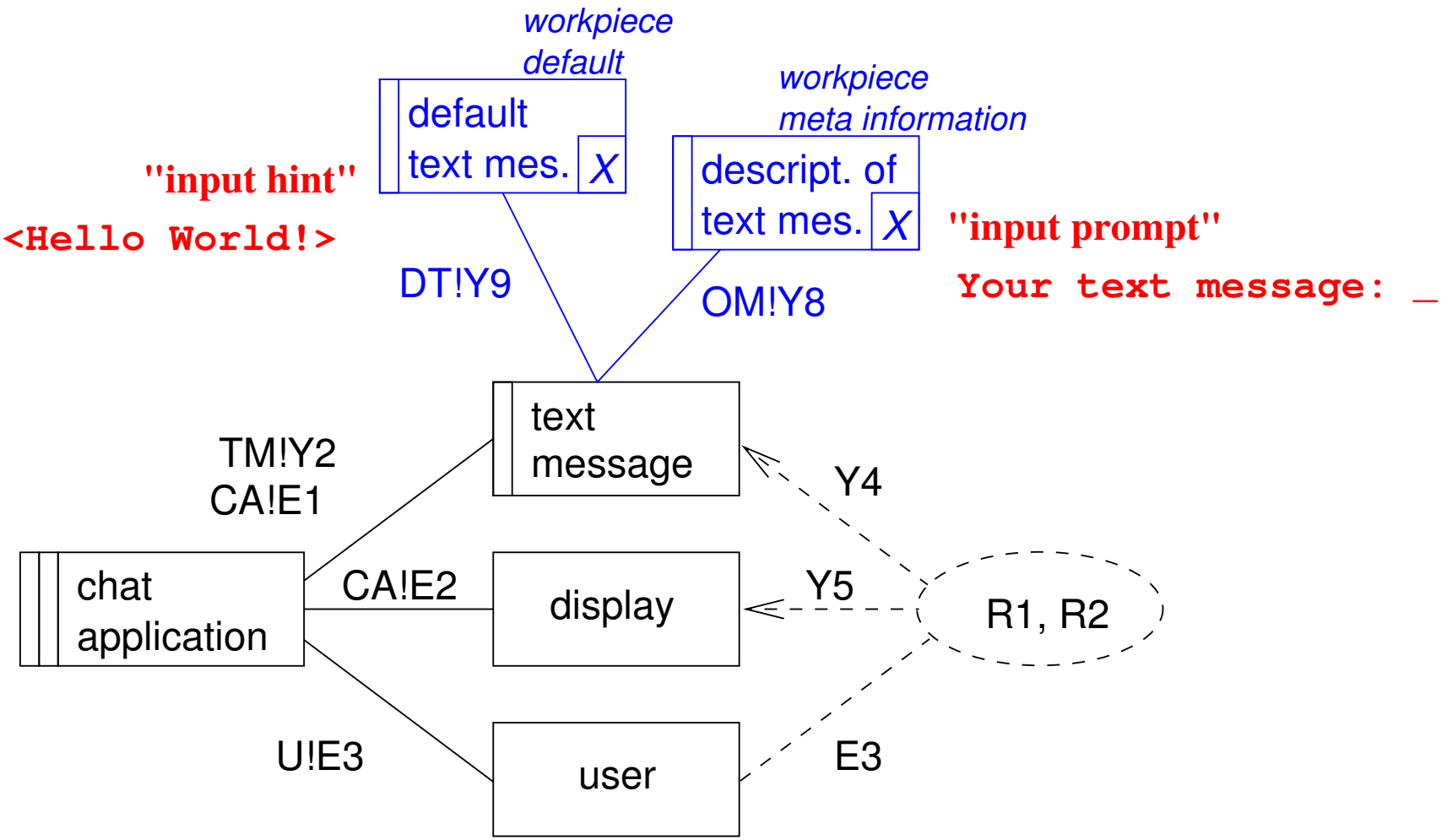
**Input Hints**

*Place a sentence to explain what is required*

**Input Prompt**

*Prefills telling the user what to do*

- Extending problem patterns of the software analysis phase by ***problem description*** *of some HCI design patterns*
- Pointing up software quality demands as user-friendliness
- Pointing out *where* to apply respective patterns




E1: editTextmessage

E2: showTextmessage, showMetaInfo, showDefault

E3: phraseTextmessage / "user works on text message"

Y2: MessageText, Meta, Default

Y4: "content of text message"

Y5: "show text message"

Y8: TextMessageMetaInfo

Y9: MessageTextDefault

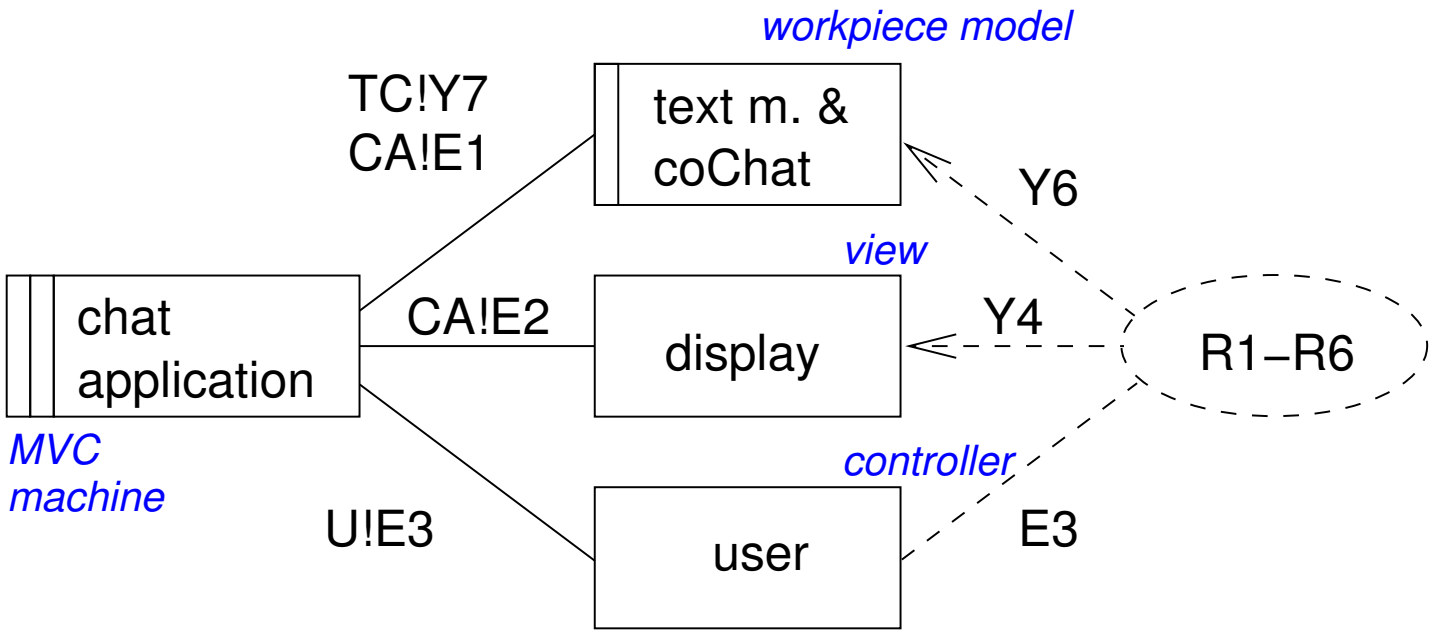
## Pattern-based software development process

- Smooth transition from problem patterns to solution patterns
- Role-based mapping of problem frames with architectural styles and design patterns

## Maintaining identified quality aspects

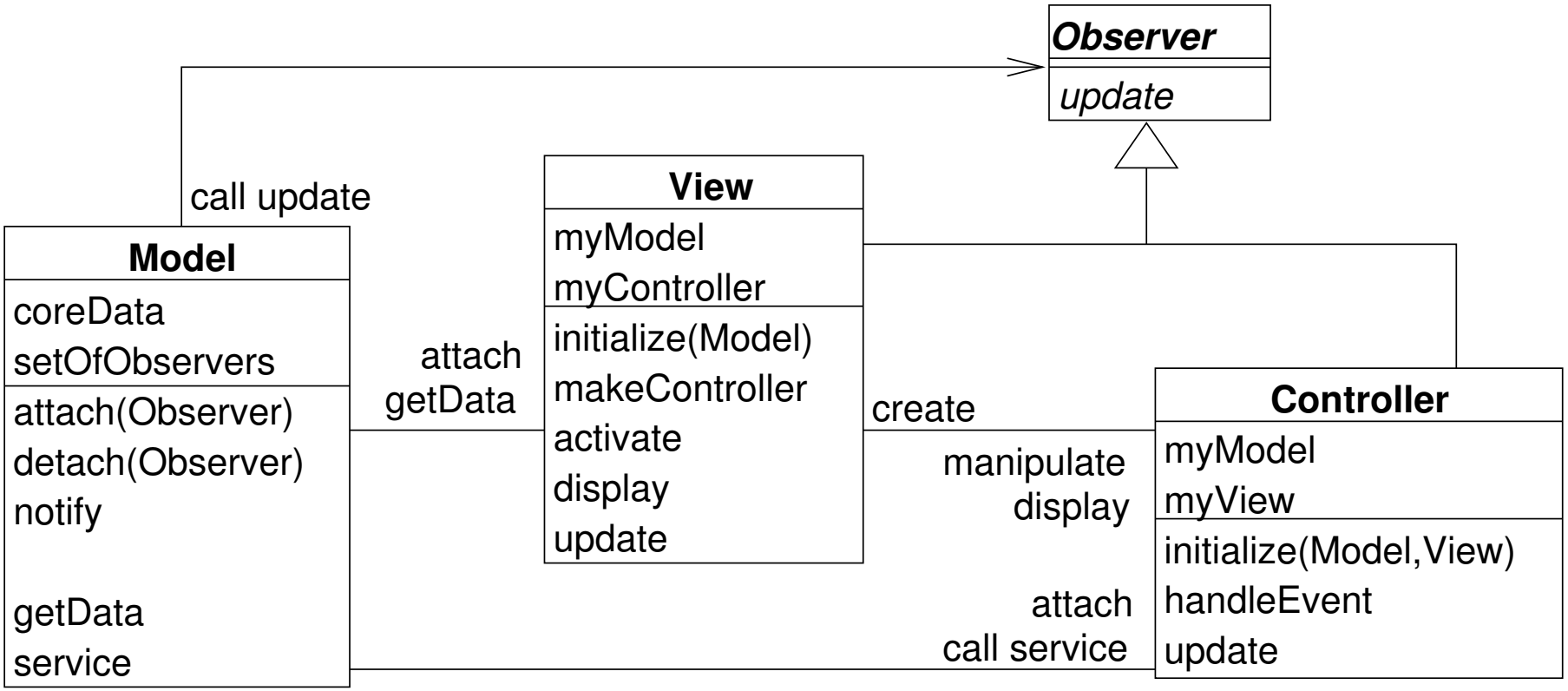
- Each found frame instance assigns values to accordant subsolution patterns
- Unified Modeling Language applicable
- Composing final software out of several subsolutions

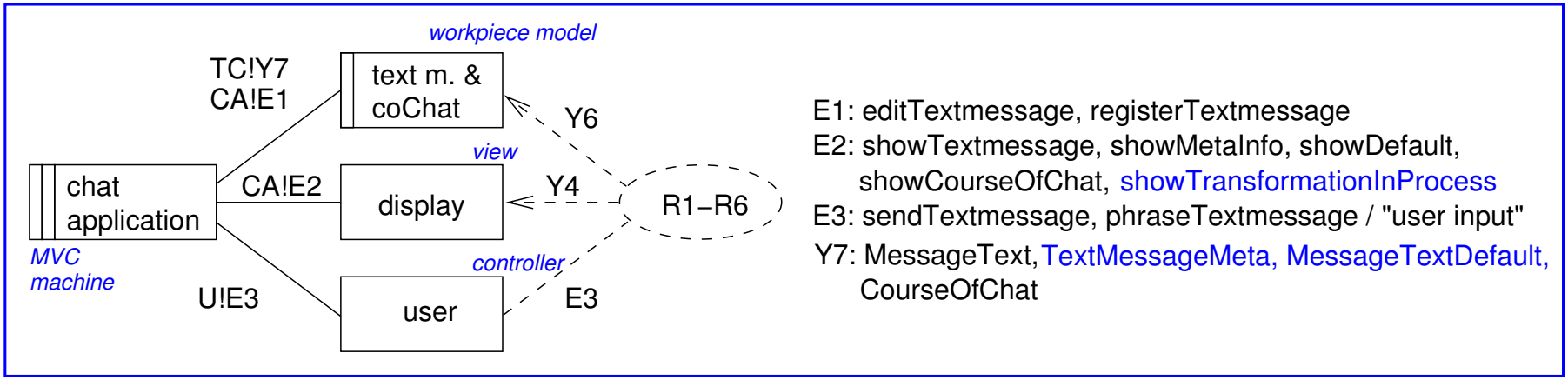
- AFrame variant [Rapanotti et al.]

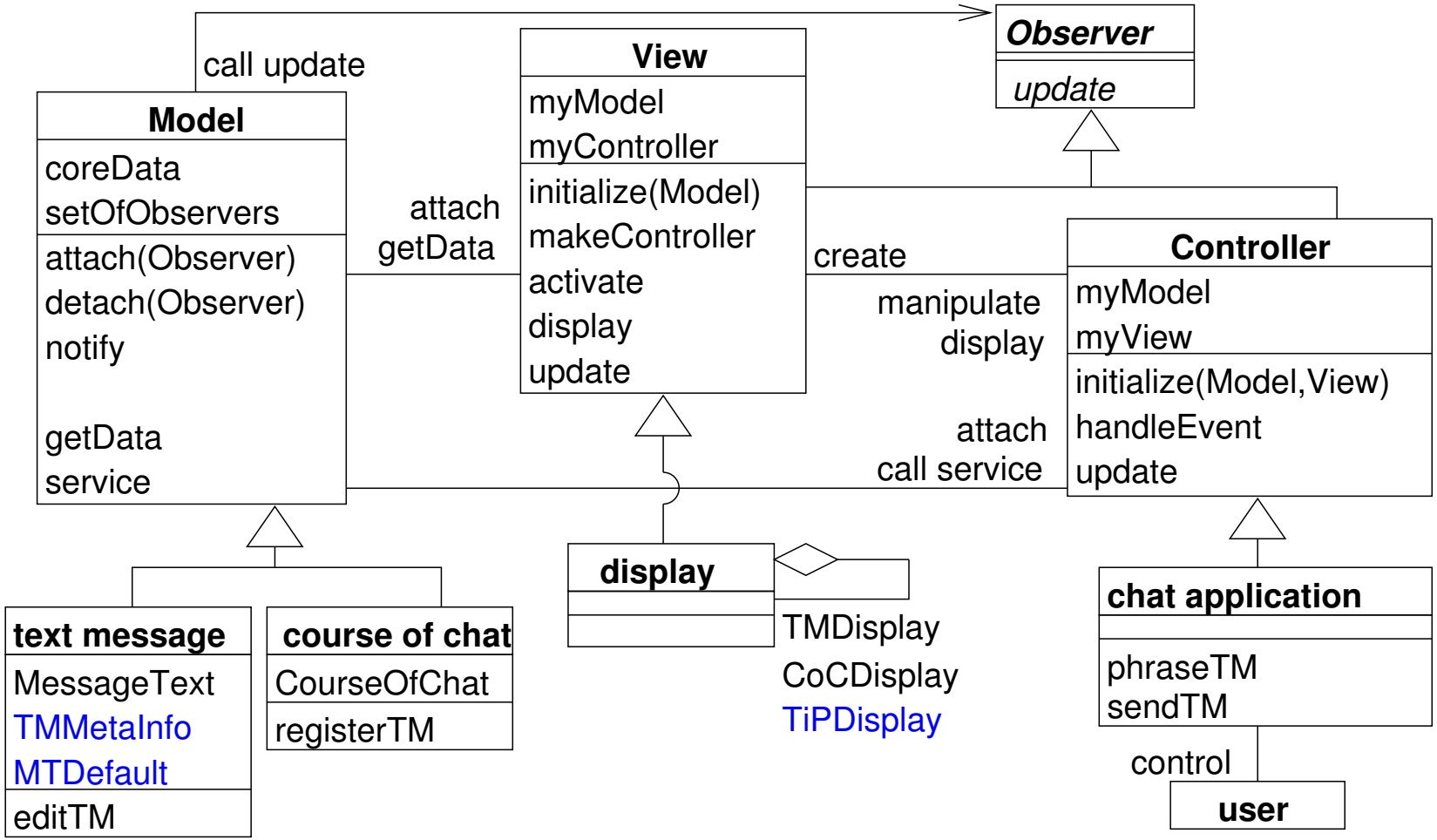



E1: editTextmessage, registerTextmessage
E2: showTextmessage, showMetaInfo, showDefault, showCourseOfChat, showTransformationInProcess
E3: sendTextmessage, phraseTextmessage / "user input"
Y4: "show text message and course of chat"
Y6: "content of text message and course of chat"
Y7: MessageText, TextMessageMeta, MessageTextDefault, CourseOfChat

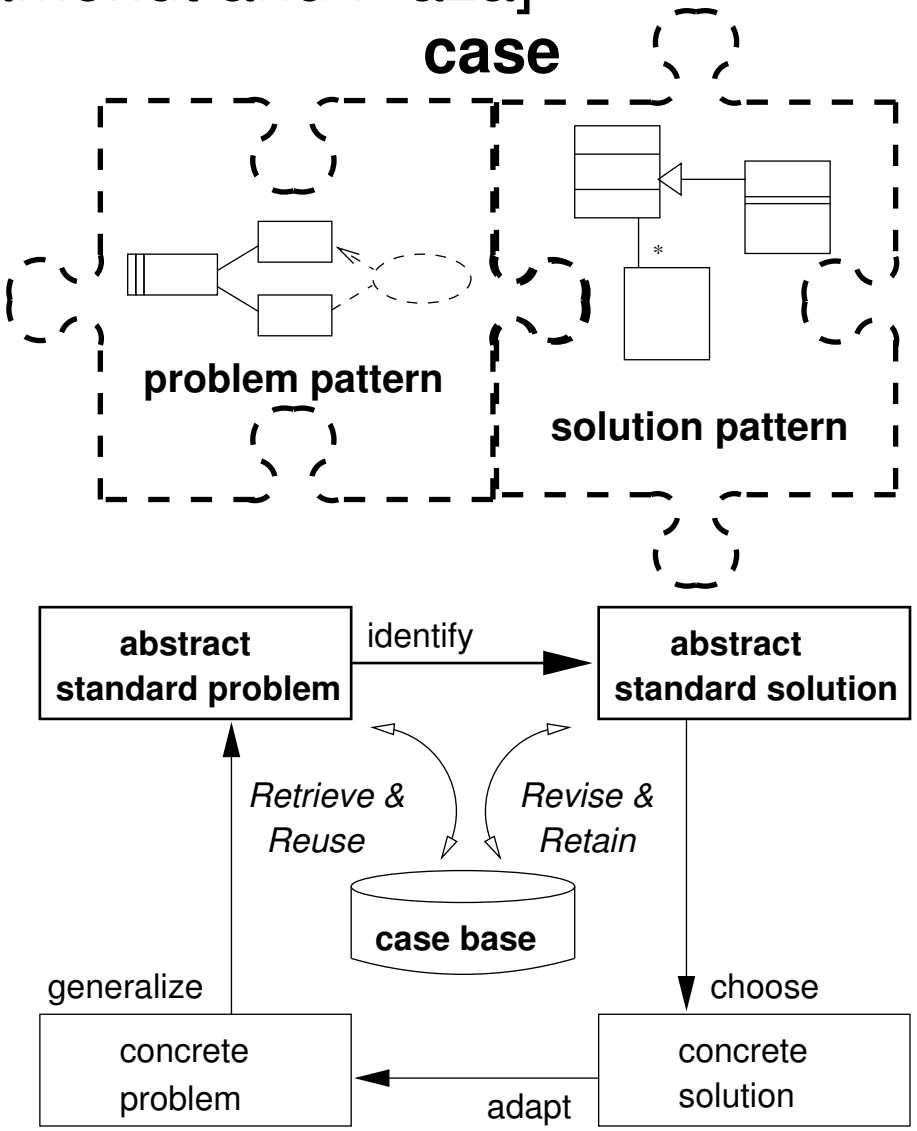
- Model-View-Controller architectural style [Buschmann et al.]




workpiece model
TC!Y7
CA!E1
text m. &
coChat
Y6
view
chat
application
CA!E2
display
Y4
R1–R6
MVC
machine
controller
U!E3
user
E3
E1: editTextmessage, registerTextmessage
E2: showTextmessage, showMetaInfo, showDefault,
showCourseOfChat, showTransformationInProcess
E3: sendTextmessage, phraseTextmessage / "user input"
Y7: MessageText, TextMessageMeta, MessageTextDefault,
CourseOfChat


call update
Observer
update
View
myModel
myController
initialize(Model)
makeController
activate
display
update
Model
coreData
setOfObservers
attach(Observer)
detach(Observer)
notify
getData
service
attach
getData
create
manipulate
display
attach
call service
Controller
myModel
myView
initialize(Model,View)
handleEvent
update
display
TMDisplay
CoCDisplay
TiPDisplay
text message
MessageText
TMMetaInfo
MTDefault
editTM
course of chat
CourseOfChat
registerTM
chat application
phraseTM
sendTM
control
user

- Methodological assistance by case-based reasoning (CBR) process [Aamondt and Plaza]



**Contribution:** linking patterns of software analysis with patterns of software design enables

- identification, representation and maintainance of software quality aspects
- guidance where to apply respective patterns
- value-adding software development process based upon patterns

**Vision:** aspect-oriented development of core functionality and user interface supported by our pattern-based quality software development approach

- sandwich-integration of GUI widgets and source code
- use of component technology for our purposes
- evolution of the chat application example

```
if (questions == true)
  please.ask();
else
  goto.nextTalk();
```