Aspect-Oriented Programming for Reactive Systems

Karine Altisen, Florence Maraninchi, David Stauch

Verimag
Aspect-Oriented Programming

- New approach in software engineering.
- Kiczales et al. Aspect-Oriented Programming. ECOOP’97
Aspect-Oriented Programming

– New approach in software engineering.
– Kiczales et al. Aspect-Oriented Programming. ECOOP’97
– Motivation: we cannot modularize all concerns in a object-oriented programming language. (some concerns “crosscut”).
– Express functionalities outside the object structure in aspects.
AspectJ Example

package pack;
class foo{
    public void visit(Object o){...}
    ...
}
class bar{
    public String visit(String s){...}
    ...
}

aspect logVisits {
    pointcut visits():
        execution(public * pack.*.visit(..));
    before(): visits() {
        System.out.println(
            thisJoinPoint.getSignature().toString());
    }
}

We specify:
- where we intervene: the pointcut
- what we do: the advice
Application to Reactive Systems

– Idea : apply aspect-oriented programming to reactive systems.
– AspectJ and other approaches are little formalized.
– Impossible to use the existing tools : we need a formal semantics.
Parallel Structure

- Synchronous languages have a parallel structure with broadcast.
- Parallel structure already powerful: “real” aspects.
- We want a simple language with a parallel structure.
- Subset of Argos.
Synchronous Automata

- Formal framework for our propositions.
- Example modulo-8-counter.
- Operators: parallel product (\(\parallel\)) and encapsulation (\(\setminus\)).
- 3-bit counter: \((B_1 \times B_2 \times B_3) \setminus \{b, c\}\).
Aspects for Reactive Systems: a Proposal

- New operators (with preservation of bisimilarity).
- Select states as join points.
- Advice: adding transitions:

\[ \text{join point} \]

- Advice transition

- Two kinds of advice:
  - toInit
  - recovery
Pointcut

- AspectJ pointcuts are syntax-based.
- We want something more semantic.
- Observer as pointcut.
- Mark joinpoints with a special output JP.
Proposition: toInit Aspect

- when an activation signal $\alpha$ is true, we go to a predefined state.
- $\alpha$: boolean expression of fresh signals and inputs.
- Specification of target state:
  - trace $\sigma$ on the inputs, starting in the initial state.
  - Add outputs $O$ to the new transitions.

- Example: reset (with empty trace).
Example: Pasteurizer

- Juice processing plant.

commands: openValve, closeValve

sensor:
cold

command:
Clean

commands:
PastOn, PastOff

input:
buffer tank

output:
buffer tank

Output

User Interface

button:
clean

button:
past

cleaner

sensor:
fullO

sensor:
emptyO

sensor:
emptyI

command:
Drain

Pasteurizer

Input buffer tank
Controller

- The pasteurizer can either pasteurize, or clean itself, or it is stopped.

<table>
<thead>
<tr>
<th>Buttons and sensors:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>clean</td>
<td>clean the pasteurizer (button)</td>
</tr>
<tr>
<td>past</td>
<td>pasteurize juice (button)</td>
</tr>
<tr>
<td>emptyI</td>
<td>the input buffer tank is empty</td>
</tr>
<tr>
<td>emptyO</td>
<td>the output buffer tank is empty</td>
</tr>
<tr>
<td>fullO</td>
<td>the output buffer tank is full</td>
</tr>
<tr>
<td>cold</td>
<td>the pasteurizer is cold</td>
</tr>
<tr>
<td>cleaned</td>
<td>the cleaner has finished cleaning</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Commands:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Clean</td>
<td>make the cleaner work</td>
</tr>
<tr>
<td>PastOn</td>
<td>switch the pasteurizer on</td>
</tr>
<tr>
<td>PastOff</td>
<td>switch the pasteurizer off</td>
</tr>
<tr>
<td>openValve</td>
<td>open the valve between the input buffer tank and the pasteurizer</td>
</tr>
<tr>
<td>closeValve</td>
<td>close the valve between the input buffer tank and the pasteurizer</td>
</tr>
<tr>
<td>Drain</td>
<td>empty the tanks</td>
</tr>
</tbody>
</table>
Pasteurizer – the Aspect

- A quality controller must be added.

commands:
- openValve, closeValve

sensor:
- fullO

Output buffer tank

command:
- Clean

PastOn, PastOff

commands:
- Drain

buffer tank

Output

buffer tank

Input buffer tank

sensor:
- emptyI

command:
- Drain

sensor:
- emptyO

Tester

sensor:
- qualityOk

Cleaner

output:
- cleaned

User Interface

sensor:
- fullO

button:
- clean

button:
- past
Pasteurizer – the Aspect

– When the quality is insufficient, clean the pasteurizer.
– Pointcut: we pasteurize, between pastOn and pastOff.

– Advice: switch off the pasteurizer, close the valve, start cleaning, and go to the state that corresponds to cleaning.
– emit PastOff, Clean and closeValve
– trace = clean.past
Pasteurizer – an Implementation

OFF

CLEANING

COOLING DOWN

EMPTYING

PASTEURIZING

WAITING

past.clean/PastOn, openValve

past/PastOff, Drain

clean/Clean

cleaned.cold

clean.cold

paste/emptyI+fullO

past/PastOff, Drain

emptyI.emptyO

clean/Clean

paste.emptyI.fullO

paste.closeValve, PastOff, Drain

paste.closeValve

D. Stauch
An Implementation – Aspect

- OFF
  - clean/Clean
  - clean/cold
  - cleaned.cold
  - cleaned.cold

- CLEANING
  - clean/Clean

- EMPTYING
  - qualOK/past
  - /closeValve, PastOff, Drain
  - qualOK/past.(emptyI+fullO)
  - /closeValve
  - emptyI.emptyO

- COOLING DOWN
  - qualOK/PastOff, Drain
  - qualOK/PastOff, Drain

- PASTEURIZING
  - past.clean/PastOn, openValve
  - /closeValve, PastOff, Drain
  - qualOK/past.emptyI.fullO
  - /openValve

- WAITING
  - qualOK/PastOff, Clean, closeValve
  - qualOK/PastOff, Clean, closeValve
Proposition : Recovery Aspects

- Jump backwards.
- we define certain states as “recovery states”.
  - specification similar to the pointcut’s.
- When $\alpha$ is true, go back to the last recovery state passed.
Recovery Aspect Weaving

- Automaton $\mathcal{M}$ memorizes which recovery was passed last, executes in parallel.
- Signals $\text{In} : \text{tell } \mathcal{M}$ when the program enters a recovery state.
- Signals $\text{Rec} : \text{tell the program which recovery state was passed last.}$
Example – Recovery Weaving

- A, B : recovery states \( R \)
- C : join point \( J \)
Example – Recovery Weaving

\[ \text{true.}(\overline{\text{rec}_B.\text{rec}_A.\alpha + \overline{\alpha}}) \]

\[ \text{rec}_B.\text{rec}_A.\alpha / \overline{\text{O}} \]

\[ x.y \]

\[ P \]

\[ q_0 \]

\[ A \in R \]

\[ x \]

\[ \text{in}_A \]

\[ \overline{\text{in}_B} \]

\[ B \in R \]

\[ y \]

\[ \text{rec}_B.\alpha / \overline{\text{O}} \]

\[ \overline{\text{M} \ \overline{\{ \text{in}_A, \text{in}_B, \text{rec}_A, \text{rec}_B \}}} \]

– Program after weaving:

\[ P \parallel \overline{\text{M} \ \overline{\{ \text{in}_A, \text{in}_B, \text{rec}_A, \text{rec}_B \}}} \]
Example: Blender

- Blends water with three juice concentrates to produce fruit juice.
- 3 juices with different water/concentrate relations.
Blender – the Aspect

– When the tank has been emptied, the user must specify which juice to produce next.
– The concentrate tube is then reconnected (expensive).
– Aspect : new input restart
  – activation signal : restart.empty.
  – applies when the tank has just been emptied.
  – the tube is not reconnected.
  – continue to produce the current juice.
Blender – the Aspect

- **Pointcut**: when emptying, i.e. when pump is true.
- **Recovery**: Just after the reconnection of the pipe, i.e. when cncted is true after cnctX has been emitted.
Blender – an Implementation
An Implementation – Aspect
An Implementation – Aspect
Pasteurizer – Extension

- **Extension**: cleaner also cleans tanks.
- **New requirement**: tanks must be emptied before cleaning.
- **Aspect must empty tanks**:
  - emit Drain to empty tanks.
  - wait for tanks to be empty, and clean.
- **Cannot be done with existing aspects**!
Extension – Advice Program

– Idea: include automaton on advice transitions.
– Specify advice program: automaton with special final state.

base program  advice program  woven program
Pasteurizer – Advice Program

- Poincut, target state and activation condition unchanged.
- Outputs: Drain, PastOff, closeValve

emptyI.emptyO/Clean
Pasteurizer – Aspect Application

- **OFF**
  - clean/Clean
  - clean/cold
  - cleaned/cold

- **CLEANING**
  - clean/Clean
  - clean/cold
  - cleaned/cold

- **EMPTYING**
  - qualOK/past
  - qualOK/past/emptyI.fullO
  - emptyI.emptyO

- **COOLING DOWN**
  - clean/Clean
  - clean/cold

- **PASTEURIZING**
  - past.clean/PastOn, openValve
  - past/PastOff, closeValve
  - qualOK/past
  - qualOK/past.emptyI.fullO
  - emptyI.emptyO

- **WAITING**
  - qualOK/PastOff, Drain, closeValve
  - qualOK/PastOff, Drain, closeValve
  - emptyI.emptyO/Clean
Conclusion

– Aspect language for Argos.
– New operator for Argos.
– Preserves trace equivalence, determinism, reactivity.
– Implementation exists.
– Perspectives :
  – Study other kinds of advice.
  – What is the effect of an aspect on a program?
  – Do we need aspects?