Process Aspect: Handling Crosscutting Concerns during Software Process Improvement

Ya-sha Wang

wangys@sei.pku.edu.cn
Outline

- Motivation
- Our solution
  - Process aspect model
  - Describing a process aspect
- Weaving process aspect into SPEM-based processes
- An example
- Related work
Motivation: New Concerns in SPI

- **Software Process Improvement** (SPI) contains actions taken to **change** an organisation's **processes**, so that they meet the organisation’s **business needs** and achieve its **business goals** more **effectively** – *ISO 15504*

- A frequently emerging situation in SPI...
  - New concerns need to be added
    - Be careful of Intellectual Property (IP) problems
    - Enhance customers’ involvement
    - Introduce systematic reuse into process
    - Develop more secure software
    - ...

Motivation: Building Blocks in Software Process

<table>
<thead>
<tr>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>E1</td>
<td>E2</td>
<td>C1</td>
</tr>
<tr>
<td>Business Modeling</td>
<td>Requirements</td>
<td>Analysis &amp; Design</td>
<td>Implementation</td>
</tr>
<tr>
<td>Test</td>
<td>Deployment</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Time

OpenUP
Motivation: Crosscutting Concerns

- When implementing certain new concerns...
  - Changes usually spread over many different building blocks of original processes
    - Be careful of Intellectual Property (IP) problems
    - Enhance customers’ involvement
    - Introduce systematic reuse into process
    - Develop more secure software
    - ...

Diagram:
- A new concern
- Processes
- Connections between new concern and processes
Two Tasks in OpenUP

management

- capture and communicate status
- handle exceptions and problems
- manage objectives

solution

- determine a strategy
- write source code
- evaluate the implementation

up_analyst - up_project_manager
- <<performs>>

up_manage_iteration
- <<performs>>
- <<mandatory>>

up_developer - up_tester
- <<performs>>

up_implement_solution
- <<mandatory>>
- <<input>>

up_usecase implementation - up_design
- <<mandatory>>
- <<output>>

up_work_item_list - up_iteration_plan
- <<mandatory>>
- <<output>>
IP as a Crosscutting Concern
Motivation: What’s Wrong

- What if we do NOT handle these crosscutting concerns...
  - How to easily understand the implementation of such a concern
  - What if the implementation of a concern change over time ??
  - What if there are several similar processes which also require these concerns ??
- We need to model the implementation of a crosscutting concern
- Just like it was in software, via Aspect
Outline

- Motivation
- Our solution
  - Process aspect model
  - Describing a process aspect
- Weaving process aspect into SPEM-based processes
- An example
- Related work
SPEM 2.0 (2/2)

- Software Process Engineering Metamodel (SPEM) 2.0
  - OMG, 2008-4-2
  - 27+ Related Research Papers
  - Strong and Wide Support in Industry
SPEM 2.0 (2/2)
A join point specifies one possibly changed element in an existing process model when applying a process aspect.

We define every task, role, work product and tool in SPEM 2.0 process models can be a join point.

AspectJ: The join point model provides the common frame of reference that makes it possible for execution of a program’s aspect and non-aspect code to be properly coordinated

- Method Call
- Event
- Reading / Writing a Field
Pointcut

- Pointcut is a set of join points on which a certain advice should apply, which results from certain filtering among join points.

- We define 4 designators to specify the filtering rules, for example:
  - `tasks_designators`: `tasks_criteria → {e | e.type = Task}`
  - `tasks_criteria = {(f,v) | (f, v.type) ∈ {(name, string), (description, string), (step, Step), (performer, Role), (input, WorkProduct), (output, WorkProduct), (used tool, Tool)}}`

\[ \forall e ∈ tasks\_designators(( f_1, v_1), (f_2, v_2),...(f_n, v_n)), \text{ we have } e.f_i = v_i, \ i ∈ [1, n] \]

- AspectJ: A pointcut is a set of join points, plus, optionally, some of the values in the execution context of those join points.
  - by parameter
  - by return value
  - by literal
Advice

- Advice defines which operations should be taken.

- An operation is represented as \((a,p)\), where \(a\) denotes the action to be taken, and \(p\) denotes the parameter for the action. Formally, we can define Advice as:

\[
\text{Advice} = \{(a,p) \mid (a, p.\text{type}) \in \{(\text{add\_attribute, string}), (\text{add\_role, Role}), (\text{add\_workproduct, WorkProduct}), (\text{add\_tool, Tool}), (\text{add\_task\_before, Task}), (\text{add\_task\_after, Task}), (\text{add\_unordered\_task,Task})\}\}
\]

- **AspectJ:** Advice is a method-like mechanism used to declare that certain code should execute at each of the join points
  - Jave Code Snippet
Correspondence Cases

Meaningful correspondence cases between a pointcut type and operations in its advice.

<table>
<thead>
<tr>
<th>Pointcut Type</th>
<th>Operation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task</td>
<td>(add_attribute, x)</td>
<td>Add a new attribute x for an existing task</td>
</tr>
<tr>
<td>Task</td>
<td>(add_role, x)</td>
<td>Relate a new role x with an existing task</td>
</tr>
<tr>
<td>Task</td>
<td>(add_workproduct, x)</td>
<td>Relate a new work product x with an existing task</td>
</tr>
<tr>
<td>Task</td>
<td>(add_tool, x)</td>
<td>Relate a new tool x with an existing task</td>
</tr>
<tr>
<td>Task</td>
<td>(add_task_before, x)</td>
<td>Add a new task x before an existing task</td>
</tr>
<tr>
<td>Task</td>
<td>(add_task_after, x)</td>
<td>Add a new task x after an existing task</td>
</tr>
<tr>
<td>Task</td>
<td>(add_unordered_task, x)</td>
<td>Add a new unordered task x into an existing task</td>
</tr>
<tr>
<td>Role</td>
<td>(add_attribute, x)</td>
<td>Add a new attribute x for an existing role</td>
</tr>
<tr>
<td>Role</td>
<td>(add_task, x)</td>
<td>Relate a new role x with an existing role</td>
</tr>
<tr>
<td>Role</td>
<td>(add_workproduct, x)</td>
<td>Relate a new work product x with an existing role</td>
</tr>
<tr>
<td>WorkProduct</td>
<td>(add_attribute, x)</td>
<td>Add a new attribute x for an existing work product</td>
</tr>
<tr>
<td>WorkProduct</td>
<td>(add_task, x)</td>
<td>Relate a new task x with an existing work product</td>
</tr>
<tr>
<td>WorkProduct</td>
<td>(add_role, x)</td>
<td>Relate a new role x with existing work product</td>
</tr>
<tr>
<td>WorkProduct</td>
<td>(add_workproduct, x)</td>
<td>Relate a new work product x with an existing work product</td>
</tr>
<tr>
<td>Tool</td>
<td>(add_attribute, x)</td>
<td>Add a new attribute x for an existing tool</td>
</tr>
<tr>
<td>Tool</td>
<td>(add_task, x)</td>
<td>Relate a new task x with an existing tool</td>
</tr>
</tbody>
</table>
Outline

- Motivation
- **Our solution**
  - Process aspect model
  - Describing a process aspect
- Weaving process aspect into SPEM-based processes
- An example
- Related work
  targetNamespace="http://www.w3school.com.cn" xmlns="http://www.w3school.com.cn">
  <xs:element name="ProcessAspect">
    <xs:complexType>
      <xs:element name="PointcutAndAdvice" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="pointcut">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="#name" type="xs:string"/>
                  <xs:element name="#type" type="xs:string"/>
                  <xs:element name="#attribute" type="xs:string"/>
                  <xs:element name="#parent" type="spem:element"/>
                  <xs:element name="#child" type="spem:element"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
          <xs:element name="advice">
            <xs:complexType>
              <xs:element name="add_attribute" type="spem:attribute"/>
              <xs:element name="add_role" type="spem:role"/>
              <xs:element name="add_artifact" type="spem:artifact"/>
              <xs:element name="add_tool" type="spem:tool"/>
              <xs:element name="add_task_before" type="spem:task"/>
              <xs:element name="add_task_after" type="spem:task"/>
              <xs:element name="add_unordered_task" type="spem:task"/>
              <xs:element name="add_relation" type="spem:relation"/>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:element>
</xs:schema>
Outline

- Motivation
- Our solution
  - Process aspect model
  - Describing a process aspect
- Weaving process aspect into SPEM-based processes
- An example
- Related work
Leveraging Process Aspect into SPEM-based Processes

We leverage the VariabilityElement class (in the method plugin package of SPEM2.0) to serve as the infrastructure for weaving process aspect.
Mapping Pattern 1

Add new attribute for existing task:
- basic_task
  - <<contributes>>
  - add attribute such as steps
- contributer_task

Add new attribute for existing role:
- basic_role
  - <<contributes>>
  - add attributes such as skills
- contributor_role

Add new attribute for existing product:
- basic_workproduct
  - <<contributes>>
  - add attributes such as sections
- contributor_workproduct

Add new attribute for existing tool:
- basic_tool
  - <<contributes>>
  - add attributes such as new help documents
- contributor_tool
Mapping Pattern 2

Relate new role with existing task.
Relate new artifact with existing task.
Relate new tool with existing task.
Relate new role with existing artifact.

Relate new task with existing role.
Relate new task with existing artifact.
Relate task role with existing tool.
Relate new role with existing artifact.

Add task after existing task.
Add task before existing task.
Add task in existing activity.
Relate new role with existing task.
Composing Basic Patterns

= Pattern 1 + 2
Dealing with Redundancy

- A simple example
**The Merge Algorithm**

<table>
<thead>
<tr>
<th>Merge Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Let</strong> E: list of elements in the aspect-method-plugin-package</td>
</tr>
<tr>
<td><strong>Let</strong> e.contributee: the element pointed by e with a 'contribute' relation</td>
</tr>
<tr>
<td><strong>Let</strong> e.addProperty(p): adding property p into e</td>
</tr>
<tr>
<td><strong>Let</strong> relation(e1,e2): relation between e1 and e2</td>
</tr>
<tr>
<td><strong>Let</strong> e1.addRelatedElement(e2,r): adding relation r between e1 and e2</td>
</tr>
<tr>
<td><strong>Let</strong> E.deleteElement(e): deleting element e itself and all the links from e or to e</td>
</tr>
</tbody>
</table>

s1: For each e1 of E
s2: For each e2 of E
s3: If (e1 ≠ e2) and (e1.contributee = e2.contributee)
s4: For each p of e2.properties
s5: e1.addProperty(p)
s6: For each re of e2.relatedElements
s7: e1.addRelatedElement(re, relation(e2,re))
s8: E.deleteElement(e2)
Proof 1: Semantics of E equal Semantics of E'

Let E': list of elements in the aspect-method-plugin-package after Merge.

<table>
<thead>
<tr>
<th>Proof 1: Semantics of E equal Semantics of E'</th>
</tr>
</thead>
<tbody>
<tr>
<td>For ∀ property p contributed from E to M, let p ∈ x, x is an element of E. During Merge, (a) If x acts as e1 in s3, obviously s4 to s8 do not touch p; (b) If x acts as e2 in s3, s5 will add p into e1 and e1.contributee = M, p is still contributed from E' to M; (c) If x does not satisfy s3, obviously there is no other place in Merge that changes p So, p is still contributed from E' to M In the same way, we can have: for ∀ related element re contributed from E to M, re is still contributed from E' to M Meanwhile, E' does not introduce any new property or related element that does not belong to E As a result, Semantics of E equal Semantics of E'</td>
</tr>
</tbody>
</table>
Proof 2: E' has the least simplicity

There are two types of elements in E: some that directly contribute to the original process (let they be contributors), and others that act as related elements of contributors (let they be related_elements). If we define the simplicity of E as number of contributors + number of related elements, we can proof that E' has the least simplicity.

<table>
<thead>
<tr>
<th>Proof 2: E' has the least simplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proof 2: E' has the least simplicity</td>
</tr>
<tr>
<td>In E', the number of contributor = the number of contributee, which is the possible least. Otherwise there will exist one contribute without any contributor, which will change the semantics of the whole Aspect.</td>
</tr>
<tr>
<td>Meanwhile, s7 and s8 do not change the number of related_elements, so the number of related_elements stays unchanged after Merge. Therefore, we can get that the sum of contributor number and related_elements number, which is the simplicity defined above, has the least possible value in E'.</td>
</tr>
<tr>
<td>That is to say, E' has the least simplicity</td>
</tr>
</tbody>
</table>
Outline

• Motivation
• Our solution
  • Process aspect model
  • Describing a process aspect
• Weaving process aspect into SPEM-based processes
• An example
• Related work
<ProcessAspect>
   <!--...
   <pointcutAndAdvice>
   <pointcut>
      <_name>implement solution</_name>
      <_type>OpenUP::Task</_type>
   </pointcut>
   <add_attribute>
      <spem::Step>
         <spem::Step::Name>detect potential IP problem</spem::Step::Name>
         <spem::Step::Description>
            Detecting potential IP problem is carried out by a code scanner...
         </spem::Step::Description>
      </spem::Step>
   </add_attribute>
   <add_artifact>risk patterns</add_artifact>
   <add_artifact>potential problem list</add_artifact>
   <add_tool>code scanner</add_tool>
   </add_attribute>
   </pointcutAndAdvice>
<!--...-->
Supporting Tool (1/2)
Supporting Tool (2/2)
Outline

- Motivation
- Our solution
  - Process aspect model
  - Describing a process aspect
- Weaving process aspect into SPEM-based processes
- An example
- Related work
Related work

- **General position paper on software process aspect**

- **Software process aspect at execution time**
  - Oren Mishali, Shmuel Katz, Using aspects to support the software process: XP over Eclipse, Proceedings of the 5th international conference on Aspect-oriented software development, March 20-24, 2006, Bonn, Germany

- **Workflow aspect**
  - Anis Charfi, Mira Mezini. Aspect-Oriented Workflow Languages. OTM Workshops 2006: Montpellier, France
Future work

- advice.operation
  - replace_xxx
  - delete_xxx
- Dealing with conflict