Improving the Extent and Reach of Service Oriented Systems

Licentiate Thesis

Martin Henkel

Overview

- Services and Service Oriented Computing
- Identified problems – Extent and Reach
- Contribution – improving:
  - Reach and Extent – Process abstractions
  - Extent – Analysis of architectural styles
  - Reach – Use of aspects
**Software Service?**

"A well defined work that can be offered by a provider to a consumer"

In a Software service the provider is represented by a software system.

Implications:
- A Software service is not used like a component
- Provider run-time responsibility

---

**Business and Technology Perspectives**

<table>
<thead>
<tr>
<th>Concepts</th>
<th>Benefits with services</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Business perspective</strong></td>
<td><strong>Technology perspective</strong></td>
</tr>
<tr>
<td>Business actors, Activities, Business documents, Agreements</td>
<td>Depict the business activities, automated business process support</td>
</tr>
<tr>
<td>Web Services, Legacy systems, XML documents</td>
<td>System integration, Message routing, Message transformation, Standards -WSDL -SOAP</td>
</tr>
</tbody>
</table>
Software Services - Vision

Use services as the fundament to support the large scale interconnection of existing and future systems, within a single business and across business boundaries.

Services – Vision and Problems

Extent

Infrastructure

Point-wise

Reach

Internal

External

Lack of abstraction and modularity

Lack of explicit service interaction descriptions
Thesis Contribution
- Instruments to improve reach and extent

Process Abstractions

An increased extent of services could be handled if the abstraction level is raised — thereby allowing services and their interactions to be handled on a business level:

Alignment criteria: The technical implementation must be able to depict the desired states in the business layer - Lossless realization of the business level

Obstacle: Constraints in system technology and legacy systems makes it impossible to totally disregard the technical concerns.
Realization – Example
(based on case from Sandvik)

Business service process
- Receive order
  - Confirm order
  - Plan shipment
  - Process order
- Send shipment advice

System constraints
S1 – Existing ERP service
S2 – The validation of order integrated into the ERP
S3 – Customers service capability differs

Technical service process
- Receive order
  - Process order
  - Plan shipment
  - Confirm order HTTP
  - Confirm order FTP
  - Send shipment advice SMTP
- Wait for notification

Example – Concurrency Transformation

Business level
A
B

Transformation pattern
Reordering concurrency

Technical level
A
B

Not a lossless realization - This use of the transformation pattern “Reordering concurrency” can cause an inconsistent business state (B can finish before A)
## More Rules for Lossless Realizations

<table>
<thead>
<tr>
<th>Realization type</th>
<th>Rules for a lossless realization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregation</td>
<td>Named activities in the technical process must be aggregated/mapped into a single activity name in the business process, otherwise it is not possible to determine which activity that are executing on the business level</td>
</tr>
<tr>
<td>Condition alteration</td>
<td>Conditions in the technical process must be designed such that the activity pre-conditions are the same or weaker in the technical process. Post-conditions in the technical process must be the same or stronger.</td>
</tr>
<tr>
<td>Extension/Exclusion</td>
<td>Exclusion of concepts in the technical process may not be introduced. However, extensions to handle technical issues might be introduced.</td>
</tr>
<tr>
<td>Condition change</td>
<td>Conditional control flow must be designed such that each (optional) path in the business process corresponds to at least one unique path in the technical process.</td>
</tr>
</tbody>
</table>

## Architectures for Process Abstractions

The alignment of business and technical perspectives are a logical view of the solution.

How to implement both technical and business issues in Service oriented systems?

Use the transformation patterns & rules to analyse a set of typical architectural styles.
Layered Architecture

Implement the business and technical issues in two (synchronized) layers:

```
Business level
```

```
Technical level
```

“Aspect Oriented” Architecture

Cater to technical needs by injecting aspects with technical features:

```
Business level
```

```
Generic technical aspects
```

```
Error handling
Security
Transactions
```
Domain Service Architecture

Isolate technical details into modules

Aspects for Services

Aspects can be applied to externalize a formerly internal service. Case example using the aspect oriented language AspectJ.

```java
public interface CreditCheckingServiceInterface {
    public boolean hasPaymentRemarks(String name);
    public boolean hasCreditHistory(String name);
    public boolean checkCreditForAmount(String name, int amount);
}
```
Example aspect: Reliability

Aspect defined using AspectJ:

```java
public aspect ReliabilityQoSAspect {
  pointcut reliabilityMethods() : (execution(public * CreditCheckingService.*(..)));

  after() throwing(Exception e): reliabilityMethods() { // Log error }
}
```

Summary of Contribution

<table>
<thead>
<tr>
<th>Extent</th>
<th>Point-wise</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infrastructure</td>
<td>Business</td>
</tr>
<tr>
<td></td>
<td>Patterns &amp; rules</td>
</tr>
<tr>
<td></td>
<td>Architectural styles</td>
</tr>
<tr>
<td></td>
<td>Technical</td>
</tr>
<tr>
<td></td>
<td>Aspects for Services</td>
</tr>
</tbody>
</table>

Reach

Internal | External