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

3

Business and Technology Perspectives

	Concepts	Benefits with services
Business perspective	Business actors, Activities, Business documents, Agreements	Depict the business activities, automated business process support
Technology perspective	Web Services, Legacy systems, XML documents	System integration, Message routing, Message transformation, Standards -WSDL -SOAP

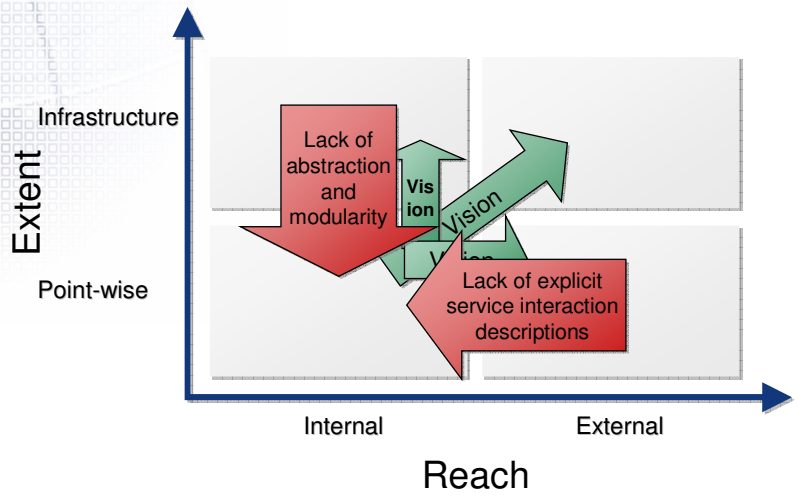
4

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*.

5

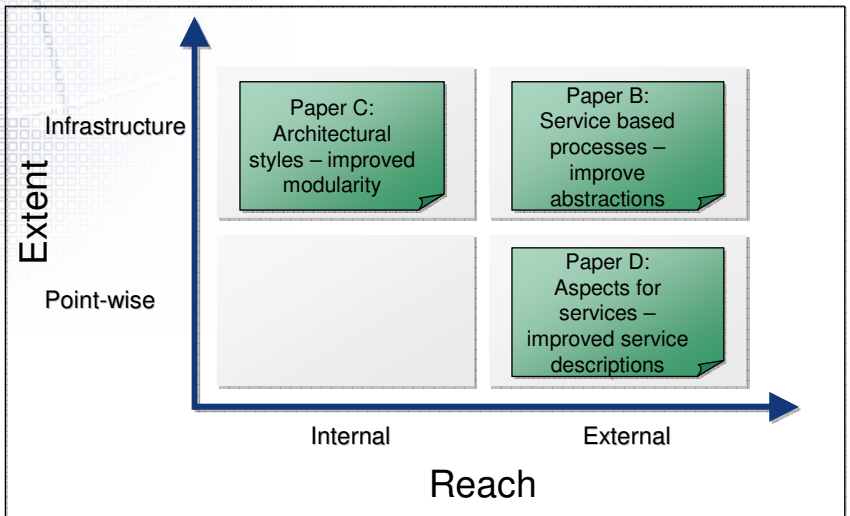
Services – Vision and Problems



6

Thesis Contribution

- Instruments to improve reach and extent



Paper A: General framework

Journal of Integrated Design and Process Science, Vol. 8, No. 1, pp 129-141, March 2004.

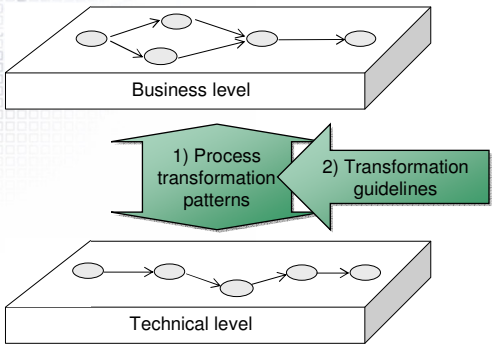
7

Process Abstractions

Paper B

International Conference on Service Oriented Computing (ICSOC'04), ACM Press, New York, USA, November 15-18, 2004.

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:

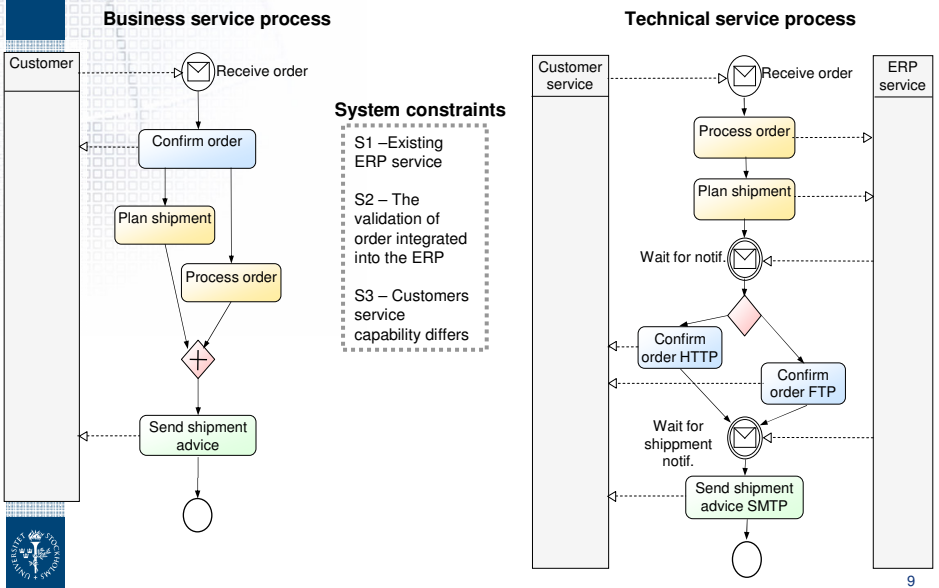


Alignment criteria: The technical implementation must be able to depict the desired states in the business layer - *Lossless* realization of the business level

Obstacle: Constraints in system technology and legacy systems makes it impossible to totally disregard the technical concerns.

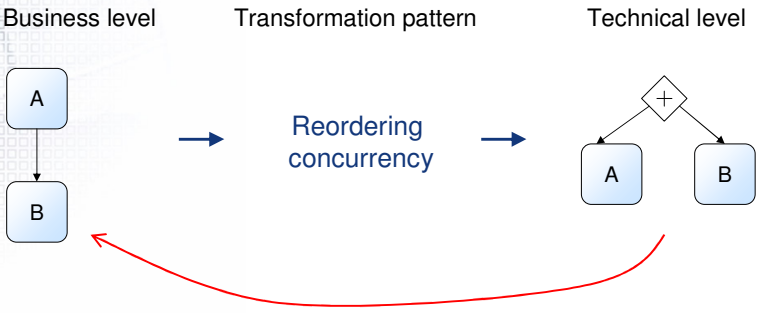
8

Realization – Example (based on case from Sandvik)



9

Example – Concurrency Transformation



Not a *lossless realization* - This use of the transformation pattern "*Reordering concurrency*" can cause an inconsistent business state (B can finish before A)

10

More Rules for Lossless Realizations

Realization type	Rules for a lossless realization
Aggregation	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
Condition alteration (Pre & post-conditions)	*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.
Extension/Exclusion Of information	*Exclusion* of concepts in the technical process may not be introduced. However, extensions to handle technical issues might be introduced.
Condition change (branching)	*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.

Paper C

Architectures for Process Abstractions

The Nordic Conference on Web Services (NCWS'04), Växsjö, Sweden, November 22-23, 2004.

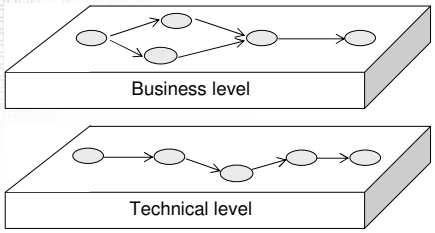
The alignment of business and technical perspectives are a logical view of the solution.

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

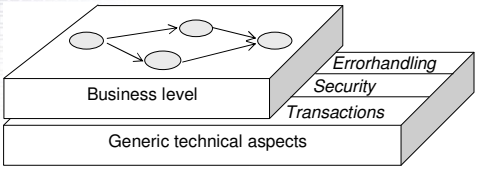
Layered Architecture

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



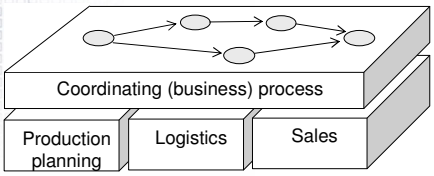
"Aspect Oriented" Architecture

Cater to technical needs by injecting *aspects* with technical features:



Domain Service Architecture

Isolate technical details into modules



Paper D

Aspects for Services

INTEROP-ESA'05, Springer Verlag, Geneva, Switzerland, February 23 – 25, 2005.

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

```
public interface CreditCheckingServiceInterface
{
   public boolean hasPaymentRemarks(String name);

   public boolean hasCreditHistory(String name);

   public boolean checkCreditForAmount(String name, int amount);
}
```

Example aspect: Reliability

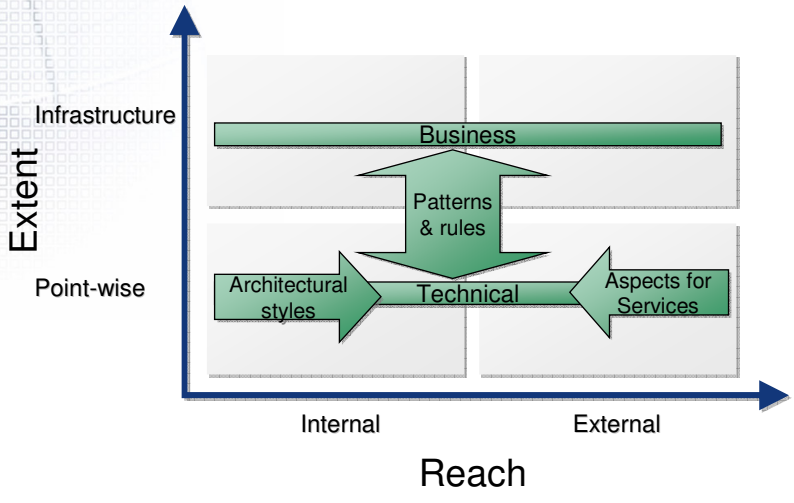
Aspect defined using *AspectJ:*

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


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

17

Summary of Contribution



18