



Pattern-based Modeling of Process-Driven SOAs

Uwe Zdun

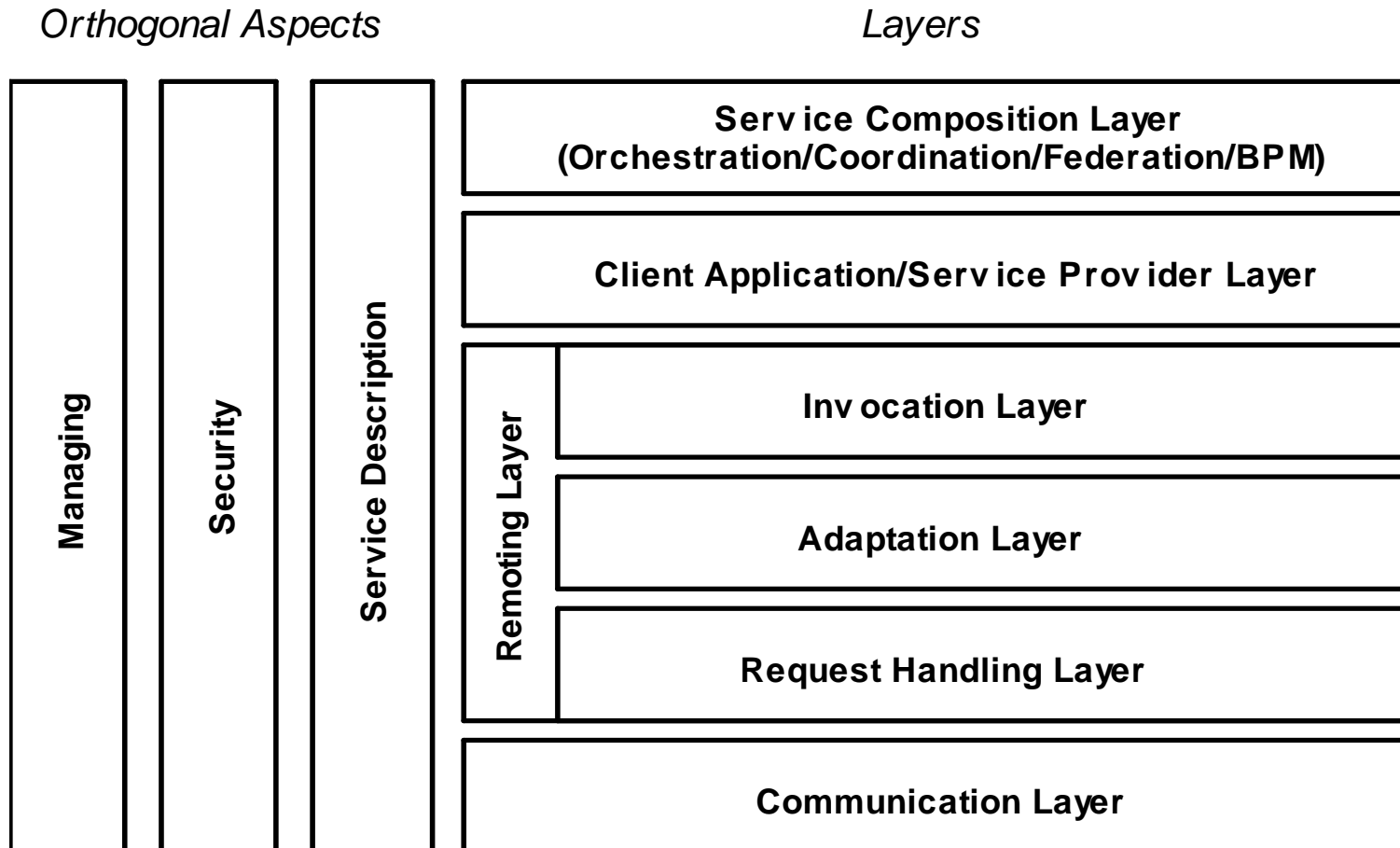
Distributed Systems Group
Institute of Information Systems
TU Wien

<http://www.infosys.tuwien.ac.at/staff/zdun>

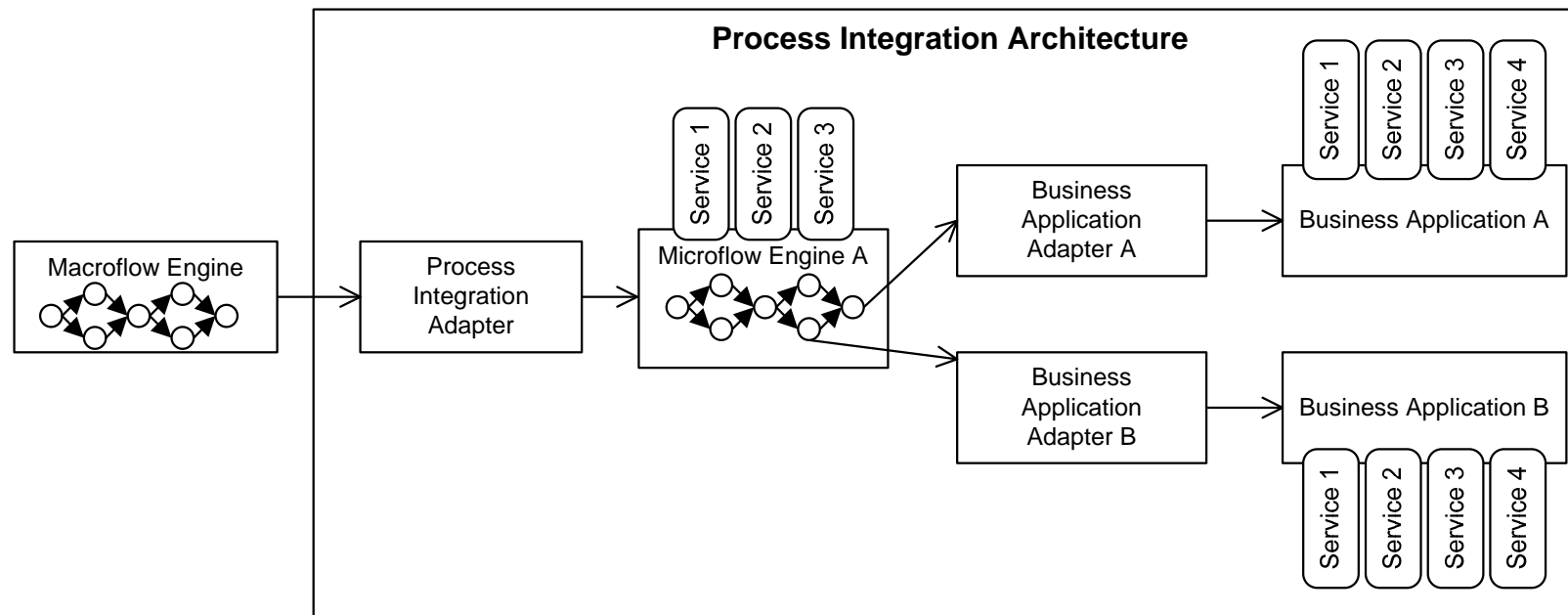
Services and SOA

- Service
 - Distributed object, accessible via the network
 - Public interfaces, no implementation details are needed for using the service
 - Self-contained, can be used independently
 - Platform- and protocol-independent
- SOA
 - Architectural concept or style in which services are used as the main constructs for composing distributed systems

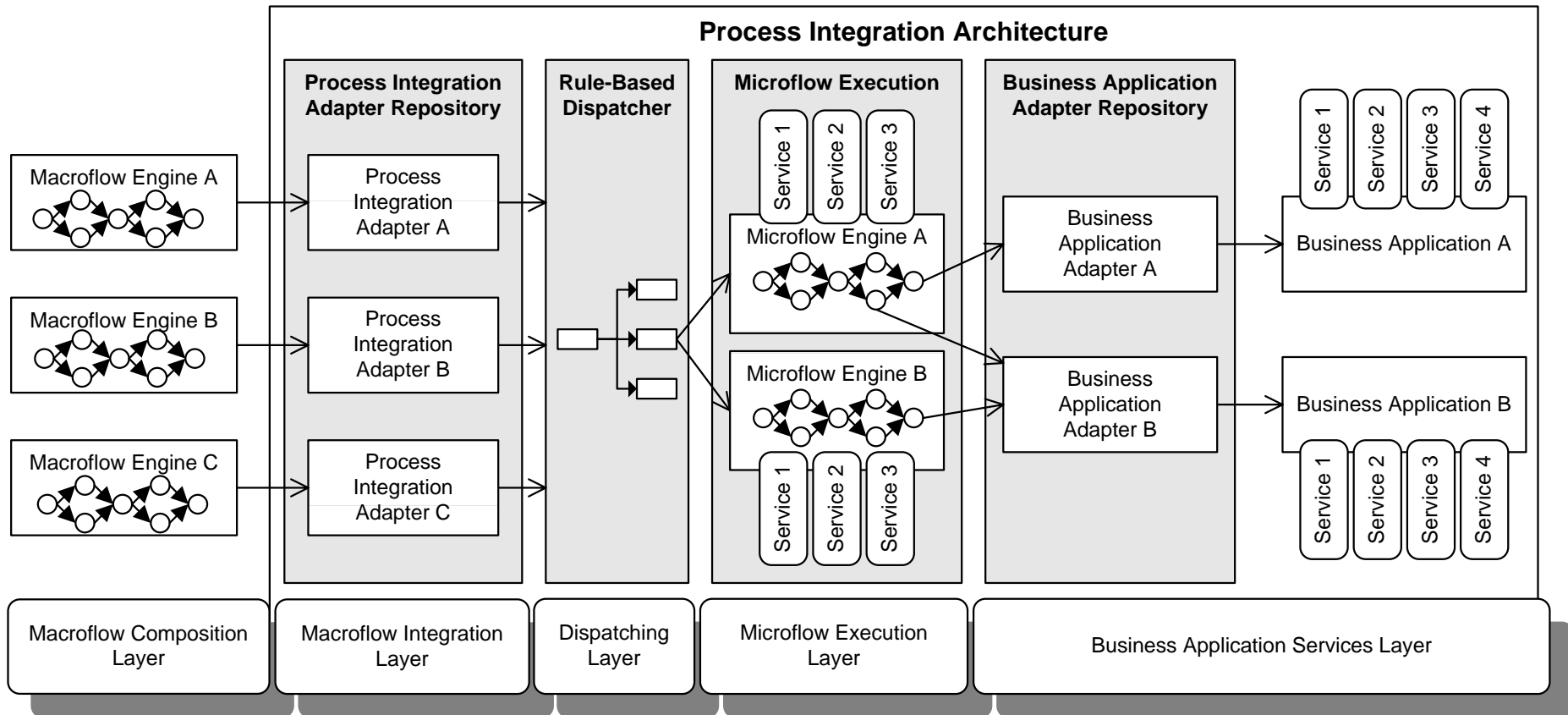
SOA: Layer Architecture



Process-driven SOA: Example of a small-scale architecture



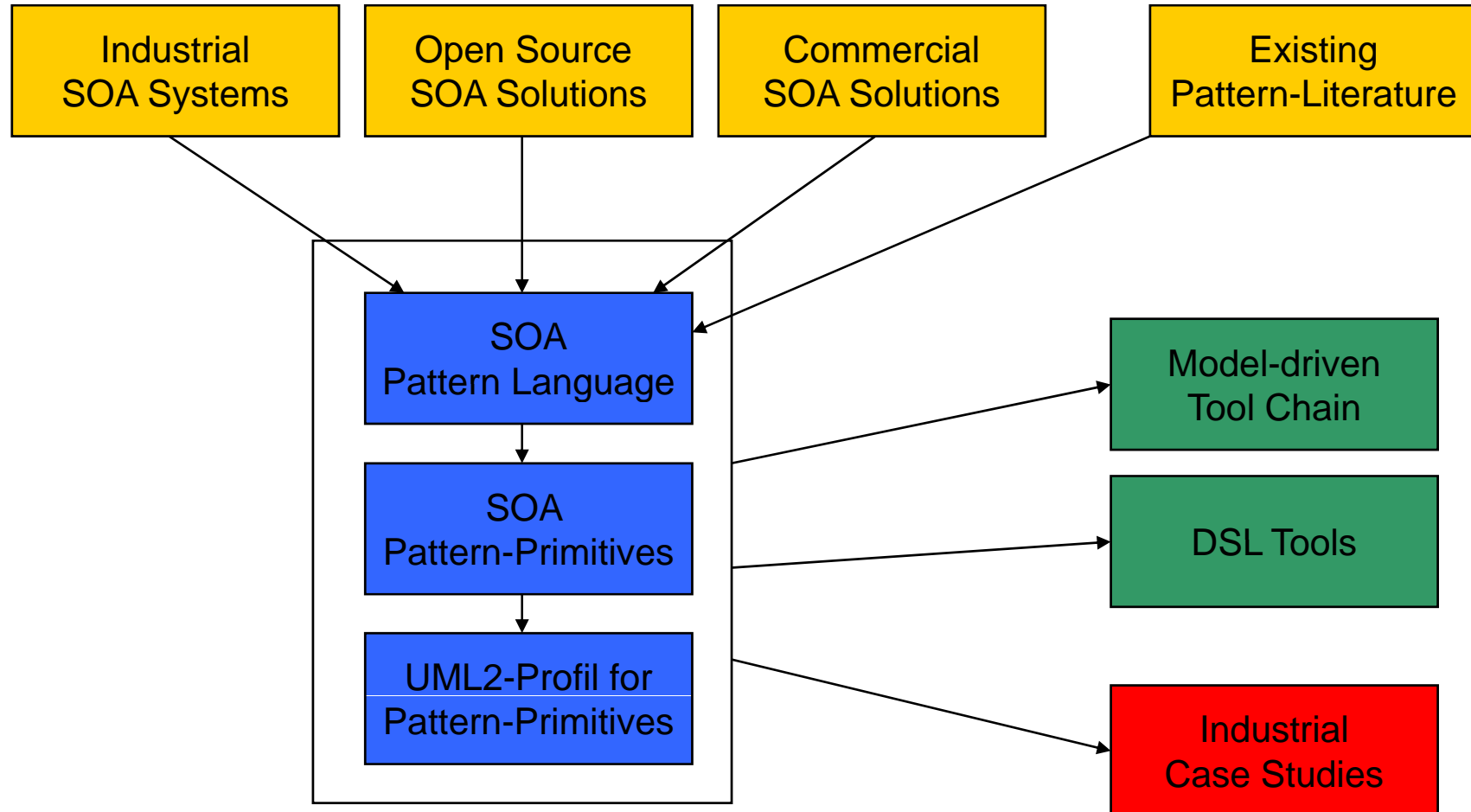
Prozessgetriebene SOA



Problem in Detail

- SOA Concepts imply integration of heterogeneous technologies and concepts; not only:
 - vendor-specific concepts or models
 - approaches in the context of a single composition technique
- On the one hand, SOA should be modeled following best practices, but on the other hand those are only informally documented
 - Process-driven SOA models are hard to understand and consistency is hard to guarantee
 - Hence, design decisions are violated over time
 - Hence, model-driven or model-based development is not really well supported
- Various stakeholders, such as business analysts, developers, architects, management, customs require different information from the models

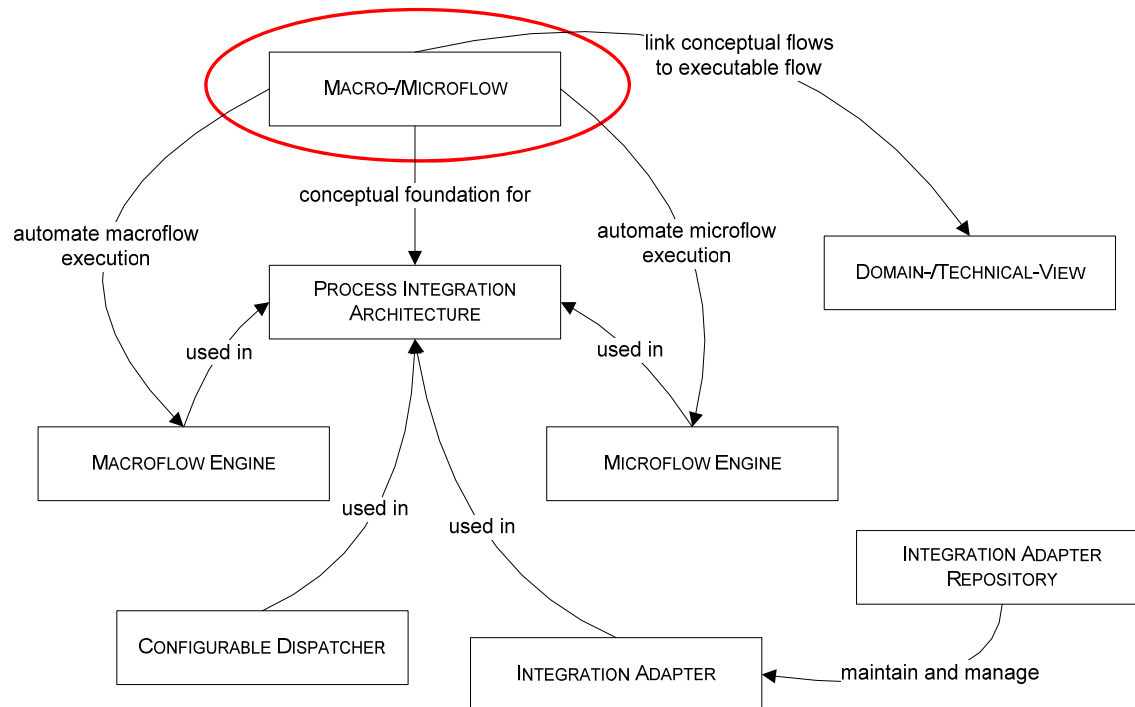
Approach: Overview



Patterns and Pattern Languages

- *Software Patterns:*
 - Systematic reuse strategy for design knowledge
 - A pattern is a description of a generic design decision that expresses the relationship between a context, a recurring problem, and a proven solution to that problem, considering a set of forces
 - Consequences and forces of the design decision
- *Pattern Language:*
 - Collection of patterns that solve problems in a particular domain
 - Focus on the relationships of the patterns

Pattern Language for Process-driven SOAs (Small Excerpt)



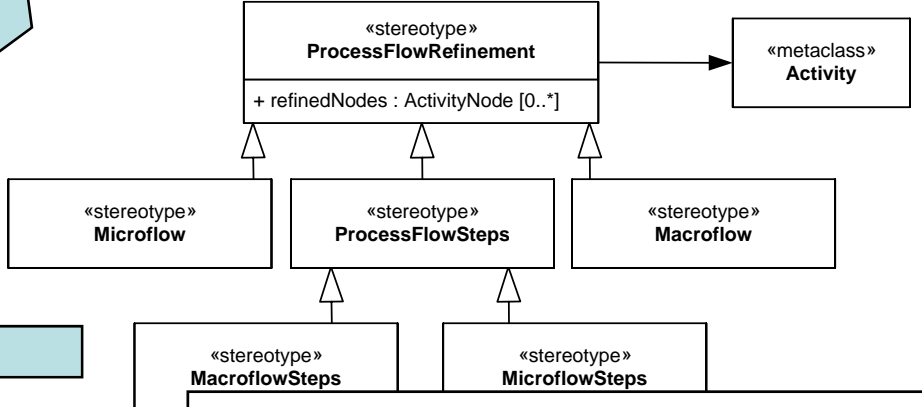
- Patterns are only informally documented
- Patterns describe not only a specific SOA solution but an entire design space
- Pattern-Primitives: Precisely specified modeling element in the patterns

Pattern-Primitiv: Macro-Microflow Refinement

MACRO-MICROFLOW

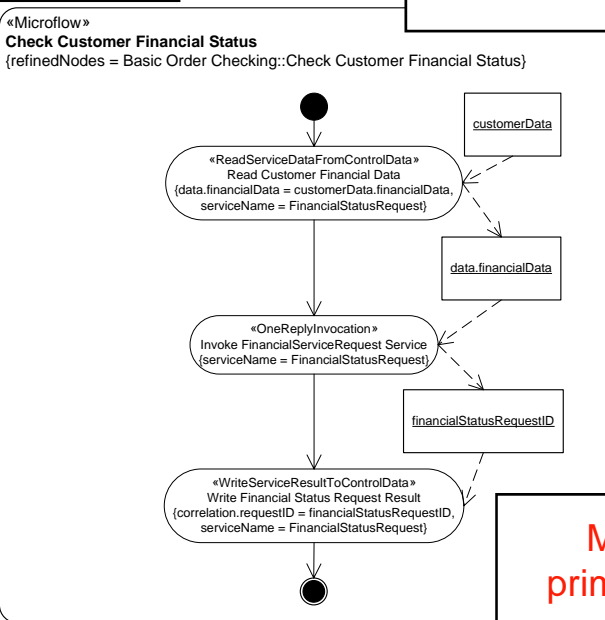
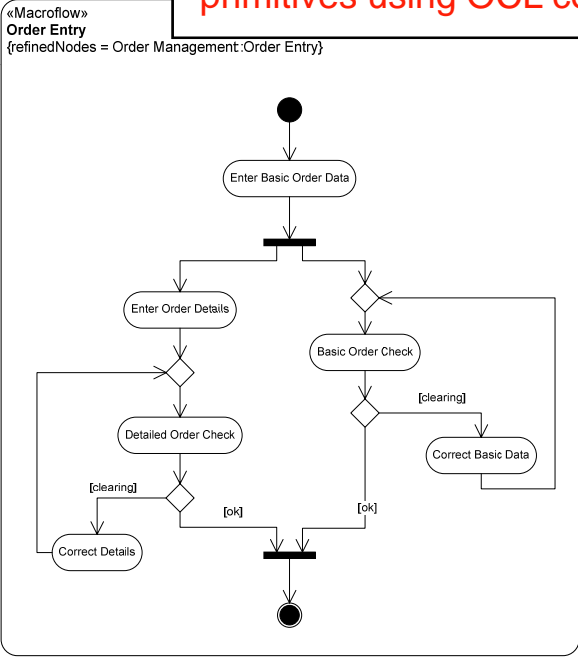
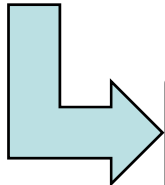
Identifying formalizable aspects in the patterns:
strict separation of Macro – Microflow

```
context Microflow inv:
  self.refinedNodes->forall(rn |
    Macroflow.allInstances()->exists(a |
      a.baseActivity = rn.activity) or
    MacroflowSteps.allInstances()->exists(a |
      a.baseActivity = rn.activity) or
    Microflow.allInstances()->exists(a |
      a.baseActivity = rn.activity))
```



Precise specification of the primitives using OCL constraints

Extension of the UML2 meta-model



Modelling systems using the primitives as modeling constructs

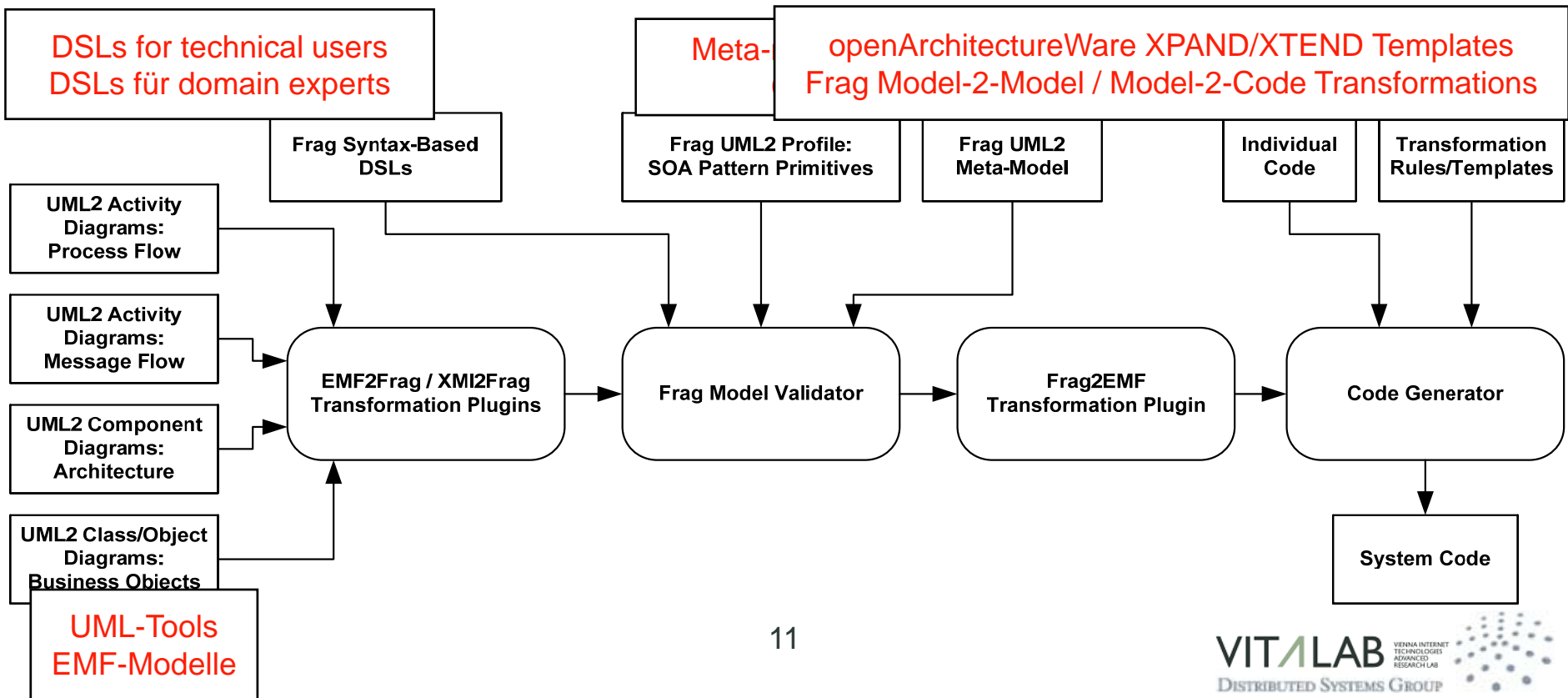
Model-driven Tool Chain

```

MMM::Stereotype create ProcessFlow
  -attributes {
    refinedNodes UML2::ActivityNode
  }
MMM::Stereotype create Microflow
MMM::Stereotype create Macroflow
MMM::Stereotype create ProcessFlow
MMM::Stereotype create MicroflowSt
MMM::Stereotype create MacroflowSt

«DEFINE Activity FOR orchestration::SimpleActivity-»
  «EXPAND SimpleActivity FOR getActivityByName(this.name)-»
«ENDEFINE»

«DEFINE SimpleActivity FOR bpel_collaboration::Invoke-»
  <invoke name="«name»" partnerLink="«getPartnerLink()»"
    outputVariable="«getOutput()»" inputVariable="«getInput()»"
    portType="«getInterface()»" operation="«getOperation()»"/>
«ENDEFINE»
  
```



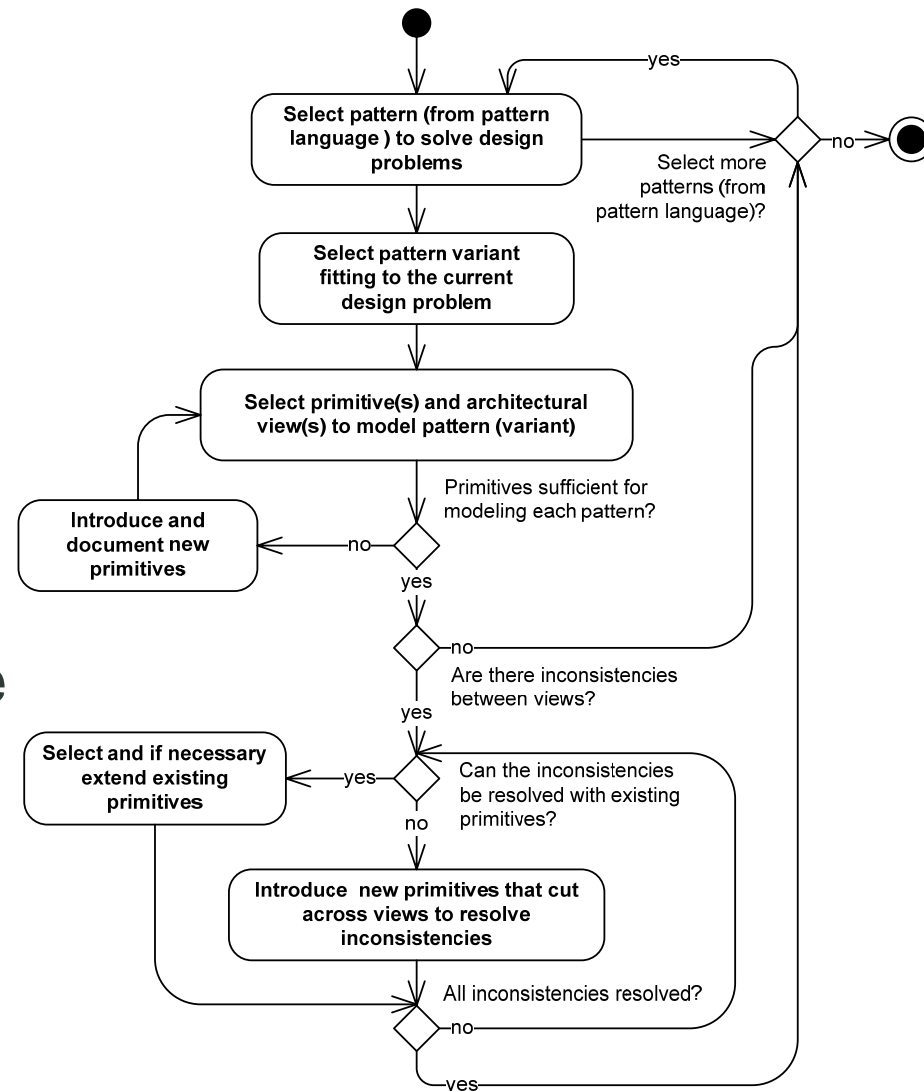


Inconsistencies across architectural views

- Application of a pattern is usually modeled in a single architectural view
- Potential inconsistencies between the views, as the different patterns sometimes entail conflicting structural or behavioral semantics

Solution: Architecting Process using Patterns and Primitives

- Architecting process based on patterns and their modeling through primitives
- Demonstrated in the domain of process-driven SOA
- Using two sets of pattern primitives that are suitable for two architectural views:
 - Process Flow View
 - Component-and-Connector View



Conclusions

- Technology-neutral approach to describe process-driven SOAs based on proven practices
- Informality and variability of the patterns resolved using primitives approach
- Model-driven approach supports
 - platform-neutral view and generation of schematic code
 - traceability
 - extensibility
- Domain-specific languages for supporting various stakeholders
- View-based approach for supporting separation of concerns

Thanks for your attention!



Uwe Zdun

Distributed Systems Group
Institut für Informationssysteme
TU Wien

<http://www.infosys.tuwien.ac.at/staff/zdun>