Adaptation and Abstract Runtime Models

5th Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2010)
Cape Town, South Africa, 3-4 May 2010

Thomas Vogel and Holger Giese
System Analysis and Modeling Group
Hasso Plattner Institute
University of Potsdam
Self-Adaptive Systems

Figure: Feedback Loop [Kephart and Chess, 2003]

Separation of managing and managed elements
→ Runtime representation of the running managed system
**Architectural model** as a runtime representation:

- One-to-one mapping between implementation classes and model elements [Oreizy et al., 1998]
- All concerns of interests like performance, costs, failures etc. [Garlan et al., 2004]
Motivation & Related Work

**Pros**
- Easing the connection between the model and the running system
- Avoiding the maintenance of several models

**Cons**
- Complexity of the model (all concerns + low level of abstraction)
- Platform- and implementation-specific model (solution space)
- Limited reusability of autonomic managers
Multiple **Target Models**
- More abstract
- Focused on specific concerns

→ Reduced complexity
→ Problem space oriented
→ Leveraging reusability of models and managers across managed systems

- Maintenance of target models by a model transformation engine
- Incremental, bidirectional model synchronization

[Vogel et al., 2009]
Case Study for EJB: Source Metamodel

Adaptation and Abstract Runtime Models
Form Source to Target Metamodel

Black box views
Form Source to Target Metamodel

Platform-specific view
• Black-box view on component types and components
• Abstract and platform-independent model
• Focused on one problem space: architecture + occurred failures
Runtime Model Synchronization

Adaptation and Abstract Runtime Models

Monitoring

Challenges

[Vogel et al., 2010]
Challenge

- Desired abstraction gap between source and target model impedes the bidirectional model synchronization [Hettel et al., 2008, Stevens, 2010]
- Refinement of abstract target model changes to source model changes $\rightarrow$ architecture refinement [Moriconi et al., 1995, Garlan, 1996]
- Case study: white box (source model) vs. black box (target model) views on component types and components
(1) Refinement for Adaptation
(1) Refinement for Adaptation

**Solution:** Factories (cf. [Gamma et al., 1995])

- Operating on the source model (no abstraction gap)
- Invoked on target models
- Pragmatically extends the transformation engine
(2) Restrictions to Adaptation

Challenge

- Interfacing autonomic managers with target models → **How** changes are performed on a model?
- Definition of allowed changes on abstract target models → **What** changes can be performed on a model?

Solution

- Solution similar to adaptation operators in *Rainbow* [Garlan et al., 2004]
- For each target metamodel: specification of specific actions a manager can perform on a target model for adapting the system
(3) Ordering of Adaptation Steps

Challenge

- Structural adaptation involving a set of atomic changes/steps
- **Synchronizing a set of target model changes in one run to the source model, and then in one run to the system** transaction
- Interactions esp. dependencies among different steps
- Different orders for target model, source model or system changes
- Overwriting of changes and losing of intermediate changes
- **Consistency** of the system affected by not suitable orders
(3) Ordering of Adaptation Steps

**Solution**: 3 options

1. **Target Model Usage**
   - Triggering of intermediate synchronizations by managers at runtime
   - Example: \( c_1, \text{sync}, c_2, \text{sync} \)

2. **Transformation Engine**
   - Design of rules using application contexts or constraints
   - Example: \( c_1 || c_2 \) on target model, but constraint/context of rule for \( c_2 \) is not fulfilled until rule for \( c_1 \) has been applied \( \rightarrow c_1 \) before \( c_2 \) on source model

3. **Causal Connection between Source Model and System**
   - Generic ordering of changes for executing them on the system depending on the types of changes
   - Example: stop comp, remove conn and comp, deploy comp, create conn, set parameter values, start comp
Conclusion & Future Work

Conclusion

• Multiple and abstract models for monitoring and adaptation
• Reusability of models and managers across managed systems
• Runtime model synchronization to maintain multiple models

Future Work

• Concurrent adaptations by different managers on different models
  → Coordination to balance competing adaptations and concerns
• Distributed setting
  → Distributed, generic, and incremental model synchronization
References

    Design Patterns - Elements of Reusable Object Oriented Software.
    Addison-Wesley, 32 edition.

    Style-Based Refinement for Software Architecture.

    Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure.

    Model Synchronisation: Definitions for Round-Trip Engineering.

    The Vision of Autonomic Computing.

    Correct Architecture Refinement.

    Architecture-based Runtime Software Evolution.

    Bidirectional model transformations in QVT: semantic issues and open questions.

    Model-Driven Architectural Monitoring and Adaptation for Autonomic Systems.

    Incremental Model Synchronization for Efficient Run-Time Monitoring.