Model-Based Self-Adaptation of Service-Oriented Software Systems

GK Workshop 2010
Schloss Dagstuhl, June 2, 2010

Thomas Vogel
Research School on Service-Oriented Systems Engineering
System Analysis and Modeling Group
Motivation

- **Continuous adaptation** of software to keep its value for the user (Laws of Software Evolution) [Lehman, 1996]
- (Increasing) **complexity** of software systems [Northrop et al., 2006]
- Maintenance & administration costs [Sterritt, 2005, Sommerville, 2007]
Motivation

- **Continuous adaptation** of software to keep its value for the user (Laws of Software Evolution) [Lehman, 1996]
- (Increasing) **complexity** of software systems [Northrop et al., 2006]
- Maintenance & administration costs [Sterritt, 2005, Sommerville, 2007]

**Self-Adaptive Software [Cheng et al., 2009]**

*Systems that are able to adjust their behavior in response to their perception of the environment and the system itself.*

⇝ **Autonomic Computing** [Kephart and Chess, 2003]
Concepts originating from the control engineering discipline
[Kokar et al., 1999, Diao et al., 2005]

Self-healing/-optimization/-protection/-configuration
[Lin et al., 2005]
Service-Oriented Computing... [Papazoglou et al., 2007]

...promotes the idea of assembling application components into a network of services that can be loosely coupled to create flexible, dynamic business processes and agile applications.

- Composition of loosely-coupled services → modularity
- Self-containment of services (well-defined interfaces/contracts)
- Dynamic binding

→ Basic support for architectural adaptation at runtime
→ Suitable abstraction mechanism for self-adaptation [Nitto et al., 2008]
In our broad vision of MDE, models are not only the primary artifacts of development, they are also the primary means by which developers and other systems understand, interact with, configure and modify the runtime behavior of software.

[France and Rumpe, 2007]
Managing EJB-based Services
Managing EJB-based Services
Abstract Runtime Models

complex
detailed
platform-specific
solution space
Abstract Runtime Models

complex
detailed
platform-specific
solution space

vs.

less complex
abstract
platform-independent
problem space
Abstract Runtime Models

Metamodel for a Source Model

complex
detailed
platform-specific
solution space

Metamodel for a Target Model

less complex
abstract
platform-independent
problem space

vs.

Thomas Vogel | GK Workshop 2010 | 2 June
Different runtime models for monitoring [Vogel et al., 2010]
- performance,
- exceptions and
- architectural constraints,

Incremental, bidirectional model synchronization based on **Triple Graph Grammars** (TGG).
Runtime Model Synchronization

Adaptation

Monitoring
Current and Future Work

- Model-driven development + runtime management
- Distributed managed and managing systems
  $\Rightarrow$ distributing models and MDE techniques

[Nitto et al., 2008]
Models at Runtime

**model@run.time** [Blair et al., 2009]

A **model@run.time** is a causally connected self-representation of the associated system that emphasizes the structure, behavior, or goals of the system from a problem space perspective.

- Causal connection $\rightarrow$ reflection [Maes, 1987]
- Higher levels of abstraction and **problem space** perspective vs. low level models based on the **solution space** as in reflection
- Integrated into an MDE development approach: relation of runtime models to models from the development phase
Related Work

**Architectural model** as a runtime representation:

- One-to-one mapping between implementation classes and model elements [Oreizy et al., 1998]
- Focused on one concern of interest [Caporuscio et al., 2007, Dubus and Merle, 2006, Morin et al., 2009]
- All concerns of interests [Garlan et al., 2004]
Is one runtime model enough?

**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)
- Reusability of autonomic managers
Failure Target Metamodel

- Abstract and platform-independent model
- Architecture + occurred failures: self-healing
- Simplified as three associations are hidden
Model Transformation

Figure: Generic Model Transformation System

- Transformation vs. Synchronization
- Unidirectional vs. Bidirectional
- Bidirectional synchronization based on **Triple Graph Grammars** [Giese and Wagner, 2009, Giese and Hildebrandt, 2008]
Declarative rules

Automatic generation of operational rules

Abstraction gap between models: manually written code "extending" the rules for adaptation

→ MDE simplifies the development of maintaining several runtime models
Declarative rules
Automatic generation of operational rules
Abstraction gap between models: manually written code “extending” the rules for adaptation

MDE simplifies the development of maintaining several runtime models
Future Work

**MDE for Self-Adaptive Systems**
- Connect development phase with the runtime phase
- Development (requirements, design, . . . ) & runtime models
- Elaborating on model-driven managing elements
- Operational environment/context

**Large-scale, distributed system**
- Distributed managed and managing elements
- Decentralized mgmt tasks [Papazoglou and Georgakopoulos, 2003]
- Distributing models and MDE techniques
- Local autonomy vs. global consistency/goals [Kramer and Magee, 2007]
References I


References III

Service-Oriented Computing: State of the Art and Research Challenges.
Computer, 40(11):38–45.

Software Engineering.
Addison Wesley, 8 edition.

Autonomic computing.

Adaptation and abstract runtime models.

Model-Driven Architectural Monitoring and Adaptation for Autonomic Systems.

Incremental Model Synchronization for Efficient Run-Time Monitoring.