GRAF: Graph-based Runtime Adaptation Framework

SEAMS 2011
Waikiki, Honolulu, Hawaii, USA

Mahdi Derakhshanmanesh¹, Mehdi Amoui², Greg O’Grady²
Jürgen Ebert¹, Ladan Tahvildari²

¹ Institute for Software Technology, University of Koblenz-Landau, Germany
² Software Technologies Applied Research (STAR) Lab, University of Waterloo, Canada
Agenda

I. Introduction
II. Graph-based Runtime Adaptation Framework (GRAF)
III. Case Study: Jake2
IV. Conclusions and Future Work
Introduction

What is this talk all about?
Motivation

Engineering self-adaptive software is difficult
- How to expose sensors & effectors?
- How to build the adaptation manager?
- How to encode adaptation steps?
- ...

→ Using generic frameworks can help!
Challenges and Goals

Support two engineering paths
- **Migration** towards self-adaptivity
- **Development** from scratch

Realize adaptation approach using existing model-driven techniques
Model-Centric Approach

Reflective Architecture

Adaptation of model first

Five adaptation scenarios at runtime
What is the Graph-based Runtime Adaptation Framework (GRAF)?
Background

Developed to support the migration of non-adaptive software towards self-adaptive software systems [ADET11, Der10]

- Adaptivity via querying, transforming, and interpreting a runtime model
- Extendable, flexible architecture
Graph-based Runtime Adaptation Framework (GRAF)

Adaptation Management Layer
- Rule Engine
- Adaptation Rules
- Control Panel

Runtime Model Layer
- Schema & Constraints
- Model Manager
- Runtime Model
- Model History

Adaptation Middleware Layer
- State Variable Adapters
- Model Interpreter
- StateVar
- Sync StateVar
- Interpretation Point
- Atomic Action

Adaptable Software
- Adaptable Elements
- Original Elements

External Controller

Managed Software
Adaptable Software

Contains business logic

Can be constructed
- by migration
- from scratch

Must expose essential interfaces to framework
Adaptable Software Interfaces

Access to
- state variables
- points in control flow
- atomic actions

Behavior description in runtime model (subset of UML activity)
Adaptation Middleware Layer

Graph-based Runtime Adaptation Framework (GRAF)

- Adaptation Management Layer
  - Rule Engine
  - Adaptation Rules
  - Control Panel

- Runtime Model Layer
  - Schema & Constraints
  - Model Manager
  - Runtime Model

- Adaptation Middleware Layer
  - State Variable Adapters
  - Model Interpreter

Adaptable Software

Propagation of state variable values
- to runtime model
- back to adaptable software

Execution of behavioral (sub-)models
Runtime Model Layer

Encapsulation of runtime model for
- queries
- transformations
- constraint validation

Model history as a trace of runtime model states
Adaptation Management Layer

Graph-based Runtime Adaptation Framework (GRAF)

Adaptation Management Layer
- Rule Engine
- Adaptation Rules
- Control Panel

Runtime Model Layer
- Schema & Constraints
- Model Manager
- Runtime Model

Adaptation Middleware Layer
- State Variable Adapters
- Model Interpreter
- StateVar
- Sync StateVar
- Interpretation Point
- Atomic Action

Adaptable Software
- Adaptable Elements
- Original Elements

Repository of predefined adaptation rules
- **Event** (change type)
- **Condition** (query)
- **Action** (transformation)

MAPE-loop [IBM05]
Graph-based Runtime Adaptation Framework

Adaptation Management Layer
- Rule Engine
- Adaptation Rules
- Control Panel

Runtime Model Layer
- Model Manager
- Model History
- Runtime Model

Adaptation Middleware Layer
- State Variable Adapters
- Model Interpreter

Adaptable Software
- Adaptable Elements
- Original Elements

GReQL + Generated Java API (using JGraLab)

UML Class Diagrams + query language (GReQL)

TGraphs

Java Reflection

JBoss AOP

Java Annotations
Case Study: Jake2

How did we put GRAF into practice?
Jake2 Screenshot

Artificial player (Bot)

Human player
Adaptation Requirement

“As the game is in progress, bots must adapt their **attack behavior**, according to the expertise of the human player.”
Jake2 and GRAF Setup Tasks

Identify how to
- measure “expertise“ → health, score
- adjust game difficulty → cloning

Prepare Jake2 (migration process [ADET11])
- annotate code, write adaptation rules, …
Extract of Behavioral Model
Adapted Behavioral Model
# Setup and Memory Utilization

## System Variations

<table>
<thead>
<tr>
<th>System Variations</th>
<th>S0</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atomic Actions &amp; Annotations</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Start GRAF &amp; Runtime Model</td>
<td></td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Interpretation</td>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>Adaptation Rules</td>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

## Metrics

<table>
<thead>
<tr>
<th>Metrics</th>
<th>S0</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Used (MB)</td>
<td>103</td>
<td>104</td>
<td>174</td>
<td>174</td>
<td>179</td>
</tr>
<tr>
<td>Max Committed (MB)</td>
<td>112</td>
<td>112</td>
<td>214</td>
<td>213</td>
<td>226</td>
</tr>
<tr>
<td>#Loaded Classes</td>
<td>3799</td>
<td>3799</td>
<td>5648</td>
<td>5660</td>
<td>5754</td>
</tr>
</tbody>
</table>
Execution Times (checkAttack)
Conclusions & Future Work

What are the lessons learned, and which areas need further research?
Conclusions

Model-centric runtime adaptivity is feasible
- **Query / Transform / Interpret** approach
- Separation of concerns

Current technology is well-suited
- Special need for **performance & tools**
Future Work (excerpt)

Tool support, e.g., Eclipse workbench

Extension of the rule engine

Generalization of the runtime model
  - Which model types are (when) suitable?
References


[ERW08] Jürgen Ebert, Volker Riediger, and Andreas Winter. Graph Technology in Reverse Engineering, the TGraph Approach. In Rainer Gimnich, Uwe Kaiser, Jochen Quante, and Andreas Winter, editors, 10th Workshop Software Reengineering (WSR 2008), volume 126 of GI Lecture Notes in Informatics, pages 67–81, Bonn, 2008. GI.


Runtime Model Schema (excerpt)

- **Activity**
  - behaviorDescription
  - behaviorDescriptions
  - availableActions
  - nodes
  - sourceOutgoings
  - targetIncomings

- **RuntimeModelRoot**
  - reifiedData

- **StateVariable**
  - guardCondition: String

- **Flow**
  - sourceOutgoings
  - targetIncomings

- **Action**
  - declaringElementId: String

- **Node**
  - CallAction
  - OpaqueAction
  - ControlNode
  - ObjectNode

- **InitialNode**
- **FinalNode**
- **DecisionNode**
- **JoinNode**
- **Pin**