Analyzing Robustness of UML State Machines

Steffen Prochnow, Gunnar Schaefer, Ken Bell and Reinhard von Hanxleden

Department of Computer Science and Applied Mathematics
Real-Time Systems and Embedded Systems Group
Christian-Albrecht Universität zu Kiel

MARTES’06, October 2006
Introduction

Motivation

• realistic Statecharts possess high complexity
  • size
  • side effects
  • misunderstanding

• potential errors can be subtle and hard to locate for humans
• tools provide restricted facilities to avoid modeling errors
Introduction

Motivation

• realistic Statecharts possess high complexity
  • size
  • side effects
  • misunderstanding

• potential errors can be subtle and hard to locate for humans

• tools provide restricted facilities to avoid modeling errors

Purpose

• formulate profiles of robustness rules as a Statechart modeling style guide

• avoid errors, improve readability and maintainability

• establishment of automatic Statechart analysis in a highly configurable tool
Modeling Errors with Statecharts

*Humans tend to digress, err, and diversify.*
Modeling Errors with Statecharts

Humans tend to digress, err, and diversify.
Style Checking in Statecharts

Error prevention:
- human code review
- dynamic testing
- Model Checking
- Style Checking
Style Checking in Statecharts

Error prevention:
- human code review
- dynamic testing
- Model Checking
- Style Checking

Statechart Robustness:
- syntactic and semantic style
- gather from element correlation

Steffen Prochnow, Gunnar Schaefer, Ken Bell and Reinhard von Hanxleden
Style Checking Tools for Statecharts

Mint/Guideline-Checker:
- related to Matlab/Simulink/Stateflow
- trivial graphical and syntactic checks

State Analyzer:
- related to Statemate
- automated theorem proving
- Problem Specific

Rule Checker:
- related to UML
- checking with Java and OCL
- interpreting OCL
A Statechart Style Guide

- operational instructions for humans and configuration for automated analysis
- set of 41 wellformedness-, syntactic, and semantic rules
- defines a subset of the language Statechart
A Statechart Style Guide

- operational instructions for humans and configuration for automated analysis
- set of 41 wellformedness-, syntactic, and semantic rules
- defines a subset of the language Statechart

Syntactic Rules
A Statechart Style Guide

• operational instructions for humans and configuration for automated analysis
• set of 41 wellformedness-, syntactic, and semantic rules
• defines a subset of the language Statechart

Syntactic Rules

Semantic Rules

Steffen Prochnow, Gunnar Schaefer, Ken Bell and Reinhard von Hanxleden
Checking: The Environment

**Kiel Integrated Environment for Layout**

- modeling environment to explore the visualization and intuitive comprehend complex reactive systems
- provides a simulation based on dynamic focus-and-context
- KIEL’s generic concept of Statecharts can be adaptated to specific notations and semantics
- imports, visualizes, and simulates Statecharts created with Esterel Studio, Stateflow, UML tools via XMI format
- Statechart synthesis from textual languages (e.g. Esterel)
- structural Statechart optimization for compactness and readability

Steffen Prochnow and Reinhard von Hanxleden.

Comfortable Modeling of Complex Reactive Systems.

Steffen Prochnow, Claus Traulsen, and Reinhard von Hanxleden.

Synthesizing Safe State Machines from Esterel.
Checking: The Plug-In

Syntactical Checks/Wellformedness:

- adopted OCL to KOCL

```java
rule UML13CompositeStateRule1 {
  declarations {
    message "A composite state can have ...";
  }
  constraint {
    context ORState or Region;
    "self.subnodes->select(
      v| v.oclIsTypeOf(InitialState))-> size<=1";
  }
  fails {message;}
}
```
Checking: The Plug-In

Syntactical Checks/Wellformedness:
- adopted OCL to KOCL
- transformation into executable Java code

```
rule UML13CompositeStateRule1 {
  declarations {
    message "A composite state can have ...";
  }
  constraint {
    context ORState or Region;
    "self.subnodes->select(
      v| v.oclIsTypeOf(InitialState)) -> size<=1";
  }
  fails {message;}
```

Semantical Checks:
- using of a theorem prover (CVC Lite)
  - e.g. detecting a non-dwelling state: \((e_1 \land c_1) \land (e_2 \land c_2)\)
- implementation of JNI communication with SWIG
Checking: The Plug-In

Syntactical Checks/Wellformedness:
- adopted OCL to KOCL
- transformation into executable Java code

```java
rule UML13CompositeStateRule1 {
  declarations {
    message "A composite state can have ...";}
  constraint {
    context ORState or Region;
    "self.subnodes->select(v| v.oclIsTypeOf(InitialState))-> size<=1";}
  fails {message;}}
```

Semantical Checks:
- using of a theorem prover (CVC Lite)
- e.g. detecting a non-dwelling state: \(((e_1 \land c_1) \land (e_2 \land c_2))\)
- implementation of JNI communication with SWIG

Steffen Prochnow, Gunnar Schaefer, Ken Bell and Reinhard von Hanxleden

Analyzing Robustness of UML State Machines – 9/11
Demo: Error Checking

Steffen Prochnow, Gunnar Schaefer, Ken Bell and Reinhard von Hanxleden

Analyzing Robustness of UML State Machines – 10/11
Summary & Conclusion

Contributions:

• Comprehensive Statechart Style Guide
• Syntactic and Semantic analyses
• Transformative Approach for OCL usage

Conclusion:

• OCL sufficient for most of our checks
• OCL rule specification is much faster than programming
• OCL doesn’t fit all intended Statechart analyses: theorem proving was necessary

We look for realistic models to apply our checks!

thanks!

questions or comments?

Steffen Prochnow, Gunnar Schaefer, Ken Bell and Reinhard von Hanxleden

Analyzing Robustness of UML State Machines – 11/11
Summary & Conclusion

Contributions:

• Comprehensive Statechart Style Guide
• Syntactic and Semantic analyses
• Transformative Approach for OCL usage

Conclusion:

• OCL sufficient for most of our checks
• OCL rule specification is much faster than programming
• OCL doesn’t fit all intended Statechart analyses: theorem proving was necessary
Summary & Conclusion

Contributions:

- Comprehensive Statechart Style Guide
- Syntactic and Semantic analyses
- Transformative Approach for OCL usage

Conclusion:

- OCL sufficient for most of our checks
- OCL rule specification is much faster than programming
- OCL doesn’t fit all intended Statechart analyses: theorem proving was necessary

We look for realistic models to apply our checks!

thanks!

questions or comments?

Steffen Prochnow, Gunnar Schaefer, Ken Bell and Reinhard von Hanxleden
Appendix: SWIG Workflow

(a) The SWIG Workflow

Figure: Interfacing of KIEL and the CVC Lite Library via JNI and SWIG.
Appendix: Further Rules

**Syntactic Rules**

**Semantic Rules**
Appendix: Bibliography

Miltiadis Moutos, Albrecht Korn, and Carsten Fisel.
Guideline-Checker.
Studienarbeit, University of Applied Sciences in Esslingen, June 2000.

Christian Scheidler.
Systems Engineering for Time Triggered Architectures.

Martin Mutz and Michaela Huhn.
Automated statechart analysis for user-defined design rules.

David M. Beazley.
SWIG: An easy to use tool for integrating scripting languages with C and C++.

Clark W. Barrett and Sergey Berezin.
CVC Lite: A new implementation of the Cooperating Validity Checker Category B.

Steffen Prochnow and Reinhard von Hanxleden.
Comfortable Modeling of Complex Reactive Systems.

Steffen Prochnow, Claus Traulsen, and Reinhard von Hanxleden.
Synthesizing Safe State Machines from Esterel.