

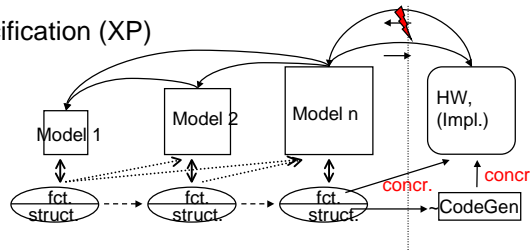
Model Based Testing in Evolutionary Software Development

Rapid System Prototyping (RSP'01), 06/27/01

Alexander Pretschner, Heiko Lötzbeyer, Jan Philipps
Software & Systems Engineering
Technische Universität München, Germany

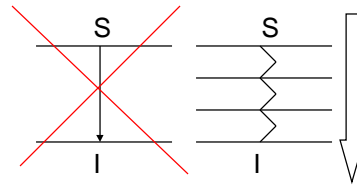
Introduction

- Reactive Embedded **Systems**
 - Hierarchic components with AutoFocus
- Model Based Incremental **Development Process**
 - Executable **behavior models** right from the beginning
 - Model: high-level, graphical, (formalized) **language**
 - Cf. Rational Unified Process, Cleanroom SW Engineering, Extreme Programming/Modeling
- Test cases
 - **Given** as part of the specification (XP)
 - **Generated** according to a test purpose: functional/structural
 - regression testing



Formal Methods and the Real World

- Unambiguous **notations!?**
- „stepwise refinement“?
 - **Non-monotonous** development
 - Round-trip engineering?
 - Emergence is not invertible
 - Imprecision vs. executability
- **Validation?!**
 - Proof systems (Isabelle, PVS), Model Checkers (SMV, SPIN)
 - **Testing!** E.g., smart cards
 - Scalability
- **Refactoring** (of pipe&filter architectures!)
 - Structural and behavioral
- Undoubted **successes** in chip design!
- Models and systems

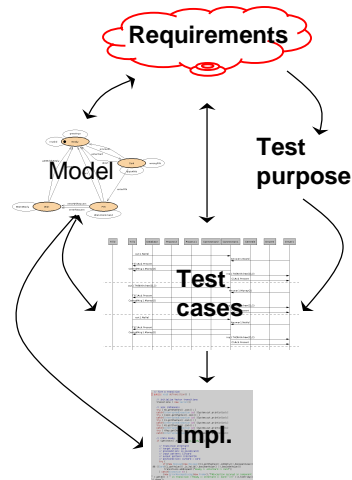


Overview

- Model Based development
- Generation of test cases with Constraint Logic Programming
 - AutoFocus
 - Generating test cases as a **search problem**
- Technical
 - **Scalability** by interaction
 - CLP vs. **Model Checking**
 - Compositionality
- Methodological
 - “Lifting” coverage criteria
 - Coverage/Complexity **metrics** for models
 - Input languages

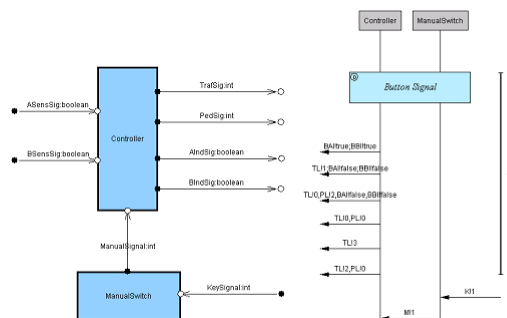
Model Based Development

- High-level (graphical) input **languages**
 - Simple, formalized semantics
- **Validation** on the level of models
 - Simulation; customer interaction
 - Automatized (testing)
- **Code generators**
 - No early lock-in to specific language
 - May prove to be not strong enough
- HW/SW Codesign
 - **Simulations**



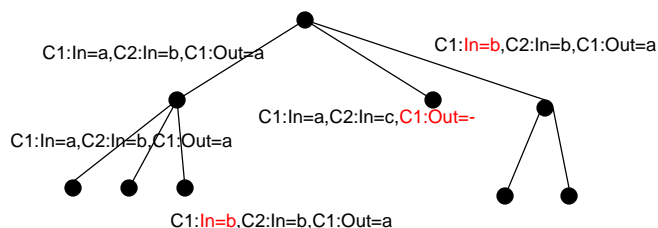
Reactive Systems with AutoFocus

- Network of **hierarchical components**
 - Synchronous communication over typed and directed channels
- Behavior for bottom level components
 - Extended **finite state machines** with I/O, function definitions, pattern matching: $In?inc(X):X<5:cnt=cnt+X:Out!sig(cnt)$
- Sequence Diagrams for specification/simulation
- Code generation, **Validation**, Requirements Tracing



In search of test cases

- Test cases from test purposes?
 - Structural: “cover all transitions”; (kind of) MC/DC
 - Functional: “Decrement C to zero”
- **Enumerate** I/O-Traces (up to a certain length)

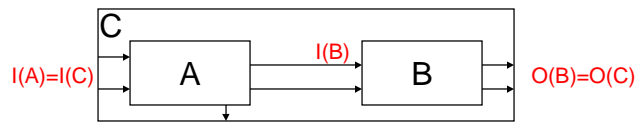


- Search **strategy**: depth/breadth/best first, tabu, interleaving

Why CLP?

- Enumeration by **symbolic execution**
 - Input values are guessed (free variables are bound)
- Pruning the search tree
 - “Slicing” the model interactively without altering the model
⇒ key to **scalability**
- Representation of **possibly infinite sets of states/traces**
 - E.g., $C1:In \in \{a,b\}, Out=a$, or $C1:\{x|x>5\}$
- “Meaningful” test cases
 - How to **instantiate** sets of traces?
 - Heuristics (cf. array boundaries)
- Search strategies easy to implement
 - E.g., A*-like by means of a good **transition ordering** (heuristics?)

Compositional test case generation



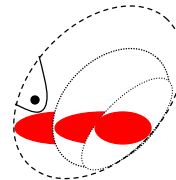
- Generate test cases for A, B (units), C: $\{TA\}$, $\{TB\}$, $\{TC\}$
- Use output of $\{TA\}$ as (putative) input to B $\Rightarrow \{TB2\}$
 - Not always possible (e.g., B requires specific timing)
- Use input of $\{TB\}$ as (putative) output of A $\Rightarrow \{TA2\}$
- Then: $I(\{TA\})$, $I(\{TA2\})$, $O(\{TB\})$, $O(\{TB2\}) \Rightarrow \{TC2\}$
- **Good/bad** test cases

Further Research

- Input **languages**
 - Temporal logics, Sequence diagrams, Automata, Constraints
- Certification
 - Coverage on models in itself rather boring (and to be defined)
 - How to convert it into **coverage on source code level** (DO-178B)?
 - Knowledge of code generators may help
- Coverage/complexity **metrics**
- Relationship with (infinite) **Model Checking**
 - Mixed discrete-continuous systems
- Definition of an “**increment**”
- **Architectures** reflecting the need for testing

More Open Problems

- How to obtain a test **case** from a test **purpose**?
 - If a **model** satisfies an invariant/universal assertion, does the **implementation** also?
 - Coverage criteria are independent of functional requirement
 - „Approximation“ metrics? Concept of continuity/topology?
- Good/general **best-first** heuristics?
- Specified requirements are met – what about **forgotten** ones?
 - Again: „good“ and „bad“ test cases
- **Generalized BDDs** with Constraints?
 - Abstracting sets of visited states = concretizing the system
 - Iterating levels of abstractions – Monte Carlo?



Conclusions

- Test case generation at the interface of **theory and practice**
- Model based **development process**
 - Increments
- Constraint Logic Programming: **Scalability**
 - **Interactivity**
 - **High-level** formalism
 - Symbolic storage of states
 - Constraint instantiations
 - Search strategies
- Lots of problems to be tackled!