Model checking Non-Functional Requirements for Interface Specifications

Florian Kammüller and Sören Preibusch

Institut für Softwaretechnik und Theoretische Informatik

FIT’08 Budapest, 5. April 2008
Software Ubiquitous in Safety Critical Systems
Software Ubiquitous in Safety Critical Systems

Importance of Specification and Verification of rich Interfaces
Our Approach: Requirements Feedback Loop

- Feedback loop between requirements and design phases
  1. Requirements: Object-Z specification $M$ (functional), Temporal Logic formulas $\rho$ (non-functional)
  2. Translate to SMV: $\phi(M)$, $\phi(\rho)$
  3. Modelcheck result (Consistency and $\phi(M) \vdash \phi(\rho)$)
  4. Use feedback for improvement of requirements and iterate
Overview

1. Background

2. Translation from Object-Z to SMV

3. Case Study TWIN-Elevator

4. Feedback Loop: Example

5. Conclusions
Specification in Object-Z

- Extension of specification language Z by objects:
  - OO-features: Classes, objects
  - Interfaces (by visibility lists)
  - Inheritance, polymorphism

- Formal specification of Interfaces in a very general way

- Full predicate calculus available to specify invariants and operations

⇒ Rich interfaces (for functional requirements) can be explicitly specified

⇒ Non-functional requirements integrated through temporal logic and feedback loop
SMV Modelchecking

- SMV is a classical Modelchecker for CTL and LTL
- SMV’s input language similar to programming language (Verilog style)
- Supports simple datatypes and modules
Translation from Object-Z to SMV

- *Profit from syntactic similarities*
  \[ \Rightarrow \text{Translation identifies: types(bool), constants, classes, instantiation} \]

- *Watch out for semantical differences*
  \[ \Rightarrow \text{Special: state transition, operations, operation composition} \]

\[
\begin{align*}
\text{plus} & \quad \Delta(\text{var}) \\
\Delta(\text{var}) & \quad \text{var}' = \text{var} + 1 \\
\text{minus} & \quad \Delta(\text{var}) \\
\Delta(\text{var}) & \quad \text{var}' = \text{var} - 1
\end{align*}
\]

\[
\begin{align*}
\text{next(\text{var}) := var + 1;} & \quad \Rightarrow \text{SMV: "redefining \text{var}"}
\end{align*}
\]

\[
\begin{align*}
\text{next(\text{var}) := var - 1;} & \quad \Rightarrow \text{SMV: "redefining \text{var}"}
\end{align*}
\]
Special Translation for Operations: Stimulus

In O-Z operations are offered, must be invoked by environment
Additional variable mimicks operation stimulus explicitly in SMV

\[ \Delta(\text{var}) \]

\[
\begin{array}{c}
\text{var} > 0 \\
\text{var}' = \text{var} - 1
\end{array}
\]

\[
\text{minus_pre} : \text{boolean}; \\
\text{minus_pre} := (\text{var} > 0);
\]

\[
\text{next}(\text{var}) := \text{case} \{ \\
\text{plus_stimulus} \& \text{plus_pre} : \text{var} + 1; \\
\text{minus_stimulus} \& \text{minus_pre} : \text{var} - 1; \\
\text{default} : \text{var}; \} ;
\]
Translation from Object-Z to SMV

• Translation Soundness
  • Translation is a homomorphism $\phi$ of the boolean lattices of Object-Z and SMV
  • State change preserves $\phi$: both formalism contain logical representation of “change”
    • In Object-Z: pre-state $s$, post-state $s'$
    • In SMV: pre-state $s$, post-state $\text{next } s$

\[ OZ \text{ system step } \]
\[ s \xrightarrow{\phi} s' \]
\[ \Downarrow \]
\[ s \xrightarrow{\text{SMV system step}} \text{next } s \xrightarrow{\phi} \]

• Translation *not* complete:
  • Quantification only for finite domains
  • OO notion of self not supported by SMV modules

... but sufficiently practical
The TWIN Problem and ThyssenKrupp Solution
Feedback Loop: Example from TWIN

- Example requirement: cabin’s movements are restricted by boundaries shaft:
  \[ \text{LevelGround} \leq \text{system.curr_level} \leq \text{LevelTop} \]

- Translation into SMV:
  \[
  \text{CabinStaysInShaft}: \text{assert} \quad \begin{align*}
  & \quad \text{G} (\text{LevelGround} \leq \text{system.curr_level}) \\
  & \quad \land (\text{system.curr_level} \leq \text{LevelTop});
  \end{align*}
  \]

[cont’d]
Feedback Loop: Example from TWIN [cont’d]

- SMV detects violation of safety requirement:

  ⇒ Operation \texttt{MoveUp} responsible for illegal change of \texttt{curr\_level}

  ⇒ Add precondition \texttt{curr\_level < LevelTop}
Results of the Case Study

- All safety properties of TWIN system could be successfully verified
- Observations:
  - Feedback loop enables detection of inconsistencies
  - Specification is abstract description of interfaces reducing to behaviour relevant for safety, e.g. DSC

actual implementation of cabin selection is irrelevant
⇒ this part of the behaviour is *not* part of interface
Summary and Lessons Learned

• Summary:
  • Interface specification with Object-Z
  • Feedback loop to requirements for quality:
    • Translation to SMV
    • Modelcheck specification
  • Case study TWIN-elevator shows feasibility
• Object-Z: formal specification “traditional way”
  • Stepwise, iterated specification refinement
  • Elegant and abstract (in comparison to SMV)
  • Temporal logic together with Object-Z seems natural
  • Pragmatism of translation; so far no restriction