Resource Constrained Embedded Systems

- Wide perspective – RCES: high level programming language technology for embedded software.
- Narrower – SCOPE: efficient code synthesis for reactive concurrent control algorithms
  - aware of usage of resources (mainly memory)
  - meeting space constraints
  - control the trade-off between speed and size
- Concretely:
  - UML is a promising framework for that
  - Source language: UML-like statecharts
  - Target language: ISO C99 (perhaps more)

Compile-time Scope Resolution for Statecharts Transitions

Andrzej Wasowski and Peter Sestoft
30th September 2002

An Optimization for Statecharts Compiler

Content:
- Environment:
  - visualSTATE tool
  - visualSTATE language
- The problem
  - Multitarget transitions
  - Dynamic scopes problem
- The solution: an algorithm
- Evaluation
  - Basic properties of the algorithm
  - Relation to standard UML
  - A bit on compile-time analysis

IAR visualSTATE

- Industrial CASE tool for development of embedded software
  - UML-like statechart language
  - design environment
  - model-checker
  - animating debugger
  - code generator
- Compilation scheme:

Remark: Moving some work from run-time to compile-time (across the dashed line) is a fundamental software optimization approach.
VisualSTATE Statecharts

- State hierarchy:
  - parallel and sequential decompositions
  - The root is an and-state
  - Basic states (leaves) are and-states
  - State type alternation
  - Orthogonal states: NCA is an and-state.
- Entry/exit actions.
- Transitions:
  - condition side: event + guard
  - executable side: action + targets

Transitions guards:
\[ g ::= \text{true} | g \land s | g \land \neg s \]
where \( s \) stands for any state name.

Textual notation for transitions:
\[ t : [e \ pos \ neg] / a : s_1...s_k \]
- \( t \) optional rule name,
- \( e \) triggering event,
- \( pos \) must-be-active states,
- \( neg \) must-be-inactive states,
- \( a \) action,
- \( s_i \) targets

Differences from standard UML:
- no fork and join transitions,
- generalized multiple targets

Multitarget Transitions: example

- UML conditions on targets relaxed
- Enter a state orthogonal to source of the transition

Scope of Firing a Transition

- Two transitions on the left fire within region \( C \) (the scope)
- Scope is important because it determines exit and entry actions
- Multiple targets yield multiple scopes
- Scopes for the left transition are regions \( B \) and \( C \)
  - \( B \) is the scope for target \( E \)
  - \( C \) is the scope for target \( G \)
Scope of Firing a Transition (II)

- Targets statically annotated with scopes:
  \[ t_1 : [ e \{D, F\} \{\} ] / a : [B] E \{C\} G \]
  \[ t_2 : [ f \{F\} \{\} ] / - : [C] G \]
  \[ t_3 : [ f \{G\} \{E\} ] / - : [C] F \]
- Cannot always be done
  - The scope occasionally depends on current configuration.

Dynamic Scope: example

- Three legal configurations activating the transition.
- All contain \( D \).
- Also contain one of \( F \), \( H \) or \( I \)

Scope of target \( E \) is always \( B \)
Scope of target \( H \) depends on active configuration of \( C \)....

Dynamic Scope: example (II)

- Dynamic scope can only be identified at runtime.
- Detection algorithm is complicated
  - efficiency suffers
  - quality/security issues (trusted code base)
- Also all normal transitions with static scopes suffer (the majority).
- If dynamic scopes are bad – get rid of them!
  - Identify dynamically scoped transitions
  - Remove them from the model
  - Add new, equivalent, statically scoped transitions.
  - Use scope annotations at runtime
The Problem and The Solution (II)

The problematic transition in our example:

![Hierarchy structure diagram]

can be rewritten with two rules:

\[
\begin{align*}
&\{ e \{D,F\} \} / a : [B]E [C]H \\
&\{ e \{D,G\} \} / a : [B]E [G']H
\end{align*}
\]

Adding extra positive conditions can ensure static scopes.

Let’s make it automatic ...

Algorithm: overview

- Describe hierarchy as a boolean formula
  - For each and-state \( s \) and children \( s_1, \ldots, s_k \) conjoin
    \[ (s \Rightarrow s_1 \land \cdots \land s_k) \land (\neg s \Rightarrow \neg s_1 \land \cdots \land \neg s_k) \]
  - For each or-state \( s \) and children \( s_1, \ldots, s_k \) conjoin
    \[ (s \Rightarrow s_1 \text{ XOR} \cdots \text{ XOR} s_k) \land (\neg s \Rightarrow \neg s_1 \land \cdots \land \neg s_k) \]
  - Conjoin a simple term (root), where root is the top state of the hierarchy.
- Restrict it with the transition’s guard.
- Eliminate irrelevant variables.
- Check the number of satisfiable assignments:
  - no solutions: transition will never fire
  - single solution: determine the static scope
  - multiple solution: the scope is dynamic

Identify Branch Exclusions

Guard propagation ensures a regular shape of solutions.
Characteristics

- Can entirely be performed at compile time
- Multiplies transitions only occasionally
- Multiplicity is small (and bound by depth of the hierarchy)
- Preserves the semantics
  - New guards are stronger than original
  - Newly added transitions are mutually exclusive
  - Disjunction of new guards is equivalent to original guard.
  - Other components of transition (action, targets) remain unmodified.
- Can be conveniently combined with other model transformations
  - guard minimization, transition compaction, message elimination, etc
- Demands a boolean logics SAT-solver
  - We use Binary Decision Diagrams (BDDs)
  - Implementation Buddy/Muddy

Applications for UML

- Multitarget transitions more efficient than UML broadcasts
  - at least two microsteps are needed in message passing
- Multitarget transitions perform similar communication task as message passing.
  - RTC semantics allows to replace message passing with multitarget transition
- Conclusion: multitarget transitions may play role in compact runtime representations for statechart models.

Efficiency

- The problem solved is substantially smaller than typical model-checking problems:
  - Only static structure is considered (no time progressing).
  - Only a subset of states needs to be represented.
  - The number of solutions is bound by the depth of hierarchy.
- 2.5s to compile a 200 transitions model (SCOPE, all incurred translation cost included)
Advocating Compile-time Analysis

- We moved scope resolution algorithm from runtime to compile time.
- A fundamental approach in compiler optimizations.
- Is it possible to propose more shifts like that?
  - Concurrent transition compaction
  - Sequential transition compaction
  - Collapsing of entry/exit rules.
  - ...
- Model-checking ...

Questions?