GRATIS

Model-Integrated Development of Sensor Network Applications

Akos Ledeczi
Senior Research Scientist
Institute for Software Integrated Systems
Vanderbilt University

http://www.isis.vanderbilt.edu/projects/nest
→ Browsing and navigation capability in the library of visual components
→ Composition of TinyOS components and applications
→ Parsing existing source files and building the corresponding graphical models (GRATIS provides over 10k connections and objects as a library to the user). This reverse-engineering facility is necessary to keep the visual models and TinyOS source base in sync
→ Automatic generation of the nesC interface and configuration files as well as the skeleton of module files from the graphical models.
→ Capturing the design space of the application via explicit alternatives and supporting design space exploration by integrating the DESERT tool
→ Manual creation of interface automata models capturing the temporal behavior of the component that is observable through its interfaces.

→ Verification of Compatibility of Interface Automata using UPPAAL or Spin
→ Black Box Testing of the Temporal Behavior of Components
TinyOS

- Programmed in nesC, a C-variant
- An application is a network of components:
  - **Configurations**: assembly of modules and configurations
  - **Interfaces**: a set of functions representing commands and events
  - **Modules**: implementation of basic operations (they have associated C code)
- The whole application is configured and linked at compile time (no dynamic reconfiguration)
- The OS components are also compiled and linked to the application (no permanent code on the MOTEs)
- No dynamic memory allocations, no “real” multitasking
TinyOS components

- Handled Commands
- Signaled Events
- Used Commands
- Handled Events
→ Graphical representation of major TinyOS concepts
→ Easier to create, comprehend, and debug than textual equivalent
→ Automatic generation of the nesC interface and configuration files as well as the skeleton of module files from the graphical models.
→ Bidirectional translation: Parser generates the graphical representation of the large existing TinyOS code base
Design Space Exploration

DESERT: DEsign Space ExploRation Tool

- Capturing the design-space via explicit alternatives
- Capturing attributes of the components via parameters
- Capturing requirements via OCL contraints
- Configuring DESERT with the (exponentially large) design space and the constraints
- DESERT configures GRATIS models automatically
TinyOS Design Space Exploration

Design space models

Complete and pruned design space
Automatic Verification and Testing

- TinyOS interfaces do not describe the temporal and behavioral aspects of components
  - Native TinyOS components are not verifiably composable
- Hierarchical Interface Automata (HIA) formalism
  - Hierarchical extension of the Interface Automata of de Alfaro and Henzinger
    - TinyOS applications are inherently hierarchical
  - Adopted to the concurrency model of TinyOS
    - Pessimistic component compatibility
    - Non-preemptable states are introduced
  - Two hierarchical interface automata is compatible if no illegal state can be reached from the initial states in the composed automaton
- HIA is the formal foundation of temporal models of TinyOS interfaces, modules and configurations
HIA Models in Gratis II

- Integrated into Gratis II
  - HIA models are hierarchical state-transition diagrams
  - Natural extension of the TinyOS composition architecture
  - Interweaved with interface references
  - Interfaces define the actions of HIA models

- Used to automatically detect:
  - Incorrect wiring of components
  - Components not obeying the implicit contract of interfaces

- Configuration verification
  - Verifies that the composed modules and interfaces are compatible
  - Implemented in UPPAAL
  - The graphical HIA models are translated into UPPAAL models

- Module verification
  - Model verification of nesC code is hard (especially if obfuscated)
  - We automatically generate a wrapper around modules to verify the consistency between the implementation and its temporal model at runtime
Model Verification

- **Modeling temporal behavior**
  - I/O automata notation
  - Hierarchical states
  - Transient states
  - Resource interfaces
  - Library of temporal models
- **Model verification**
  - Flattening hierarchical state-machines
  - Transforming graphical models to UPPAAL models
  - Compatibility and resource usage verification with the UPPAAL model checker
Conclusions

• The TinyOS component model is much easier to comprehend and debug in a graphical manner
• Bidirectional model translator keeps graphical models and code in synch
• Composition verification is very helpful in debugging
• Design space modeling and exploration enables multi platform support

• Manual creation if HIA models are time consuming and hard
• GRATIS is still a work in progress