VERTAF: An Application Framework for the Design and Verification of Embedded Real-Time Software

Pao-Ann Hsiung, Shang-Wei Lin, Chih-Hao Tseng, Trong-Yen Lee, Jih-Ming Fu and Win-Bin See
IEEE Transactions on Software Engineering, 2004

2005. 12. 21
Presented by Sang-Uk Jeon
Contents

- Introduction
- Design & Verification Flow
  - UML Modeling
  - Scheduling
  - Formal Verification
  - Code Generation
- VERTAF Components
- Case Study
- Conclusion
Introduction (1/2)

- Designing embedded software

```
typedef struct {
  BOOL isCooking;
  BOOL (*isCooking)(OvenCtrl*);
  /* attributes and methods */
} OvenCtrl;

OvenCtrl ovenCtrl = { FALSE, OvenCtrl_method_isCooking, … };
```

- Code generation
- Formal verification

- Security?
- Performance?
- Time constraints?

Partially supported!!

Needs integrated frameworks!!
• VERTAF
  – Verifiable Embedded Real-Time Application Framework
  – Features
    • Modeling of embedded systems
      – Through well-defined UML semantics
    • Formal synthesis
      – That guarantees satisfaction of temporal/spatial constraints
    • Formal verification
      – Check the system satisfies user-given/system-defined properties
    • Code generation
      – Produces efficient portable code
Design & Verification Flow (1/2)

- Front End: Machine-independent software construction phase

![Flowchart]

- Timed Statecharts
- Class Diagram With Deployments
- Extended Sequence Diagrams
- UML Model
- Extended Timed Automata Generation
- Scheduler Generation
- Real-Time Petri-net Generation
- Model Check
- Specification satisfied
  - Yes
  - No
    - Display counter-example in UML model

- Yes
  - Schedulable
    - Yes
    - Display unschedulability information
  - No
Design & Verification Flow (2/2)

- Back End: Machine-dependent software implementation phase

UML Model
- Timed Statecharts
- Class Diagram With Deployments
- Extended Sequence Diagrams

Specification satisfied

Component Mapping

Scheduler Generation

Code Generation

Embedded Real-Time Software

Control flow
Data flow
Class diagram with deployment

- Two types of classes
  - Software classes
    - Specified by a designer from scratch
    - Reused from library components
  - Hardware classes
    - Represents supported hardware component
- Method types
  - Event-triggered methods
  - Time-triggered methods
    - Added keywords: period, deadline, one-shot
    - Started, stopped and restarted by actions in the statecharts
Timed statecharts

- Enhanced time-out specification
  - Multiple clocks, clock check, reset
- Another addition
  - Set of keywords associated with time-triggered methods
    - Start, stop and reset
Extended sequence diagrams
- Mainly used for scheduling the different tasks performed by objects
- Enhancements
  - Added control structures
    - Concurrency, conflict and composition
  - State-markers
    - Indicates a state of the object’s statechart
    - <Existing features in UML 2.0>
Design guidelines examples

- 1 statechart for the behavior of an object
  - No more than 4 levels of hierarchy
- A sequence diagram for one or more use-case scenario
- Ensure the logical correctness between
  - Class diagram and statecharts
  - Startcharts and sequence diagrams
Scheduling

- Generating Petri nets from UML sequence diagrams
- Scheduling algorithms
  - Without RTOS: Quasi-dynamic scheduling
    • Requires RTPN (Real-Time Petri Nets)
  - With RTOS: Extended quasi-static scheduling
    • Requires CCPN (Complex Choice Petri Nets)
Model Checking

- **SGM**
  - High-level model checker
  - Operated on state-graph
  - Includes
    - State-graph merger
    - State-reduction techniques
    - Dead state checker
    - TCTL model checker

- **Properties verified**
  - System-defined
    - Dead states
    - Deadlocks
    - Livelocks
  - User-defined
    - Specified in OCL

- **SGM**
  - OCL Constraints
  - Extended Timed Automata
  - Scheduler Automation
  - TCTL Properties
  - Global System State Graph
  - Model Checking Results
  - Counter-examples (Sequence diagrams)
Component Mapping

- Mapping classes to specific hardware
  - Automatic generation of configuration, make, header, and dependency files

- Issues in this phase

- Solution
  - Interactive approach
    - User is requested to choose from a list of available compatible device types
**Code Generation**

- 3-tier approach for code generation

<table>
<thead>
<tr>
<th>Scheduler</th>
<th>Application Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quantum Framework</td>
<td></td>
</tr>
<tr>
<td>( \mu ) C/OS</td>
<td>Linux, eCOS, and other POSIX-like OS</td>
</tr>
<tr>
<td>( \mu ) HAL</td>
<td></td>
</tr>
</tbody>
</table>

- A scheduler with temporal monitor
- Middleware Layer
- Hardware abstraction layer
- Supported:
  - ARM, StrongARM
  - 8051, Lego RCS
Case Study (1/3)

- Two application examples
  - Avionics application
    - 24 tasks
    - 45 objects were identified
  - AICC (Autonomous Intelligent Cruise Controller)
    - 12 tasks
    - 21 objects were identified

- Development time

<table>
<thead>
<tr>
<th></th>
<th>Initial development with VERTAF</th>
<th>Second development without VERTAF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avionics (2 designers)</td>
<td>1 week</td>
<td>5 weeks</td>
</tr>
<tr>
<td>AICC (3 designers)</td>
<td>5 days</td>
<td>20 days</td>
</tr>
</tbody>
</table>
## AICC Tasks

<table>
<thead>
<tr>
<th>Index</th>
<th>Task Description</th>
<th>Object</th>
<th>Period</th>
<th>Exec. Time</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Traffic Light Info</td>
<td>SRC</td>
<td>200</td>
<td>10</td>
<td>400</td>
</tr>
<tr>
<td>2</td>
<td>Speed Limit Info</td>
<td>SRC</td>
<td>200</td>
<td>10</td>
<td>400</td>
</tr>
<tr>
<td>3</td>
<td>Proceeding Vehicle Estimator</td>
<td>ICCReg</td>
<td>100</td>
<td>8</td>
<td>100</td>
</tr>
<tr>
<td>4</td>
<td>Speed Sensor</td>
<td>ICCReg</td>
<td>100</td>
<td>5</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>Distance Control</td>
<td>ICCReg</td>
<td>100</td>
<td>15</td>
<td>100</td>
</tr>
<tr>
<td>6</td>
<td>Green Wave Control</td>
<td>ICCReg</td>
<td>100</td>
<td>15</td>
<td>100</td>
</tr>
<tr>
<td>7</td>
<td>Speed Limit Control</td>
<td>ICCReg</td>
<td>100</td>
<td>15</td>
<td>100</td>
</tr>
<tr>
<td>8</td>
<td>Coordination &amp; Final Control</td>
<td>Final_Control</td>
<td>50</td>
<td>20</td>
<td>50</td>
</tr>
<tr>
<td>9</td>
<td>Cruise Switches</td>
<td>Supervisor</td>
<td>100</td>
<td>15</td>
<td>100</td>
</tr>
<tr>
<td>10</td>
<td>ICC Main Control</td>
<td>Supervisor</td>
<td>100</td>
<td>20</td>
<td>100</td>
</tr>
<tr>
<td>11</td>
<td>Cruise Info</td>
<td>Supervisor</td>
<td>100</td>
<td>20</td>
<td>100</td>
</tr>
<tr>
<td>12</td>
<td>Speed Actuator</td>
<td>EST</td>
<td>50</td>
<td>5</td>
<td>50</td>
</tr>
</tbody>
</table>

(단위 : ms)
AICC Call-graph

- Traffic Light Info (SRC)
- Speed Limit Info (SRC)
- Preceding Vehicle Estimator (Distance Sensor)
- Speed Sensor (EBC)
- Distance Control
- Green wave Control
- Coordination & Final Control
- Final Control
- Cruise Switches (Main Instrument Controller)
- ICC Main Control
- Cruise Info (Main Instrument Controller)
- Speed Actuator (EST)
Conclusion

● VERTAF
  – An application framework for embedded real-time systems development
  – Integration of three technologies
    • Software component reuse
    • Formal synthesis
    • Formal verification

● Future works
  – Consider more features of real-time application
    • Network delay, network protocols, online task scheduling
  – Enhance abstraction technique for model checking
Critique

- **Pros**
  - Addresses broad parts in embedded software development
  - The way of conducting case study is feasible

- **Cons**
  - Insufficient examples
    - New notations in UML are not shown in the paper
  - No detail description
    - Due to addressing broad parts
  - Infeasible schedulability analysis
    - Due to partial nature of UML sequence diagrams
  - No worst-case analysis
  - Not address low-level logic description