Model-Integrated Toolchain for High Confidence Design

Graham Hemingway, Nicholas Kottenstette
Sandeep Neema, Harmon Nine, Joe Porter
Janos Sztipanovits, and Gabor Karsai
Vanderbilt University/ISIS

Part of the Multi-University Research Initiative on
High-Confidence Design for Distributed Embedded Systems
US AFOSR, FA9550-06-0312
Overall Design Flow

1. Requirement Specification
2. Functional Design
3. Component Design
4. System Arch. Design
5. HW Arch. Design
6. HW Pwr/Perf Est
7. Code Gen.
8. Alloc./Sched. Analysis
9. Latency/RT Analysis
10. SW Deployment
Focus of the toolchain

The toolchain turns the *functional controller design* into a *software implementation*: a collection of *integrated components* executed by a *robust component platform* that runs on a *system/hardware platform*. 
Toolchain challenges(1)

- Using the results of the functional controller design stage
  - Import functional design models into the toolchain
- Building software component/s from the functional models for a component platform on top of a hardware platform
  - Modeling the components that “wrap” the functional models, modeling the component architecture
  - Modeling the hardware platform
  - Mapping the software components onto the hardware platform
Toolchain challenges (2)

- Code generation
  - Functional code
  - Integration code (functional code/platform bridge)
- Scheduling
  - Design-time, static schedule computation
- Analysis and verification
  - Platform effects: generating simulation models from the functional models and platform models
    - *Functional model verification*
    - *Functional code verification*
- Execution
  - Robust execution on a component platform
Toolchain overview

Functional Design
Simulink/Stateflow

Analysis
• Platform effects
• Verification

Model-based Design
• Componentization
• Architecture modeling
• Deployment modeling

Model-based Code Generation

Schedule Generation

Schedule

SW Platform
ESMoL: Embedded System Modeling Language

- Components:
  - Synchronous dataflow network + Statecharts
  - Run at a fixed rate - periodic execution: 
    \[ x(k+1) = f(x(k),u(k)) \] – State update
    \[ y(k+1) = g(x(k),u(k)) \] – Output Update
  - Responsible for (periodic) I/O interactions

- Component architecture:
  - Components are scheduled according to their rates
  - Communication is facilitated by time-triggered messages

- Hardware platform:
  - Each node runs a cyclic, static, timed-triggered schedule
  - Nodes communicate via time-triggered messages
Functional design models into implementation models

Simulink models translated into ESMoL functional models

- *Syntactical* translation
- Only discrete-time, fixed, single-rate subsystems and Stateflow blocks
- Produces an XML representation of the Simulink/Stateflow model (suitable for other, XML-based tools)
Design models in ESMoL

- Component models: imported Simulink subsystems

- Software architecture models: dataflow diagram depicting signal flows and components

- Hardware architecture models: processor nodes, communication buses, sensors and actuators

- Deployment models: (TT) tasks hosting components, interfacing via messages, mapped to processor nodes
Code Generation

- Functional code generator
  - Preprocessing: type inference on the models
  - Translation: based on graph transformations (uses the GReAT toolsuite)
  - Retargetable code emitter: restricted/safe subset of C or Java [fixed-size arrays, no dynamic memory allocation]
  - 3 generators: Simulink, Stateflow, Embedded Matlab

- Integration code generator
  - Produces platform-specific integration code that ‘glues’ functional code to platform API-s.
Schedule Generation

Time-triggered schedule generation:
- Components are periodically executed tasks:
  - Read/Execute-Update/Write
  - Tasks are statically scheduled, do not preempt
- Communication is periodically executed
  - Read buffer/Transmit → Receive/Write buffer
  - Messages are statically scheduled, do not preempt
- Assumptions: fixed and known
  - Task rates and WCET
  - Message transfer times and dependencies
- Solution: The scheduling problem is translated into finite-domain constraint programming problem – the valuation of variables gives the schedule.

Based on:
**Off-Line Scheduling of a Real-Time System**
Klaus Schild, Jörg Würz
Analysis and verification

- Platform effects analysis
  - *TrueTime* model for platform
  - Generator to build Simulink model for functional model + platform model
  - Continuous time, high-precision Simulink simulation allows studying subtle platform effects

- Verification (*Leverage NASA MICTES Project - JPF*)
  - Functional model state reachability verification
  - Functional code verification

*Objective: To give feedback for the designer*
Execution: Time-triggered Platform

Low-end platform:
- AVR microcontroller with I/O devices
- Timer/clock-interrupt driven execution of tasks according to static schedule
- Tasks do not preempt, except in case of overruns
- If a task overruns its WCET, it is cancelled and the next task is started
- Tasks are scheduled with sufficient slack to allow for I/O interrupts
- IT-driven I/O communicates with component/tasks via shared buffers
Execution: Time-triggered Platform

High-end platform:
- OS (Linux), with limited timing accuracy (threads and nanosleep services)
- Time-triggered execution model for components
- Time-triggered messaging
  - Using UDP on isolated Ethernet
  - Using char IO on I2C (to AVR)

Schedule:
- Task A.1
- Task B.1
- Task A.2
- Task C.1
- Task A.3
- Task B.2
A Platform for Experimentation

Model-based Toolchain

TT/UDP/Ethernet

High-end platform (Gumstix/TT/Linux)

Low-end platform (Robostix/TT/RTOS)

I2C TT

High-end platform (Gumstix/TT/Linux)

Low-end platform (Robostix/TT/RTOS)

I/O lines (serial, parallel, PWM, analog, etc.)

Real-time Simulation Platform ‘Virtual plant’
Towards A Development Paradigm

- **All Simulated**
  - Simulated Application
  - Simulated Platform
  - Simulated Plant + Environment

- **Real Platform**
  - Simulated Application
  - Simulated Plant + Environment
  - Sim/API Adapter

- **Simulated Plant**
  - Real Application
  - Simulated Plant + Environment
  - Sensor Signal i/f
  - Actuator Signal i/f

- **Real System**
  - Real Application
  - Real Platform
  - Physical System
  - Sensors Actuators
  - Environment

API

Sim/Sim if

Sensor Signal i/f

Actuator Signal i/f
Status and next steps

- ‘Closing the loop’ …
- Tests and experiments

- Supporting other Models of Computation
  - Event-driven, asynchronous
- Integration of heterogeneous MoC-s
  - Problem/domain-specific ‘bridges’
- Fault management
- Integrated toolchain: from virtual prototyping to physical implementation
BACKUP SLIDES
Prototype toolchain elements

**Matlab/Simulink/Stateflow**
- Single rate subsystems
- Synchronous Dataflow semantics
- Event-triggered charts

**ESMoL Modeling Tool (GME)**
- Componentization
- Architecture

**Scheduler**
- Constraint-based generation of task and bus message schedules for a time-triggered platform

**CSP-based Scheduler**
- Resource allocation (Scheduling)

**Time-Triggered Platform**
- Multiple processors connected via a time-shared bus
- Tasks are cyclic, time-triggered
- Message receive/send happens before/after task release/finish

**ESMoL**
- Simulink/Stateflow import
- Additional aspects for components, architecture, and deployment
- Code generation for
  - Dataflow (Simulink/SDF) models
  - Statechart (Stateflow) models
  - Platform interface code

**Platform**
- Code generation for
  - Dataflow (Simulink/SDF) models
  - Statechart (Stateflow) models
  - Platform interface code
Design rationale for prototype toolchain (1)

The connection towards Simulink/Stateflow
- Simulink/Stateflow is the industry standard
- SDF and (restricted) Statechart semantics is well-defined and widely used
- Could be substituted in later stages of the project

The ESMoL language
- Software components and architectures and deployment had to be captured in models and integrated with the functional models.
- Not all features of Simulink/Stateflow are supported – only a ‘safe’ subset.
- Dataflow (Simulink/SDF) model: scheduling based on the time-triggered paradigm (\( t_k \) is determined by an off-line scheduler)
  - \( \text{receive}(t_k) \rightarrow \text{execute}() \rightarrow \text{send}(t_k+1) \)
- Extensible towards other models of computation
Embedded System Modeling Language
Design rationale for prototype toolchain (2)

**Code generation**

- **Dataflow/SDF code generation:**
  - Explicit type inference (if Simulink model is not fully typed)
  - Graph transformation into an intermediate code format (C-like, Abstract Syntax Graph)
  - Printing C code (or Java, or ...)?

- **Stateflow code generation:**
  - Follows Stateflow semantics (state transitions)
  - Graph transformation into an intermediate code format (C-like, Abstract Syntax Graph)
  - Printing C code (or Java, or ...)?

- Both code generators are extensible/backend can be replaced
The code generator is formally specified as a programmed graph transformation system. This allows reasoning about the correctness of the transformation itself.

Support for verification:
The code generation could insert verification conditions (derived from the models) into the generated ASG.

The result of the transformation is an abstract syntax graph that allows ‘printing’ the executable code in various languages.

Print
C source code
Design rationale for prototype toolchain (3)

Scheduler
- Explicit, design-time generation of cyclic time-triggered schedules for tasks and messages
- Constraint-based scheduling approach

The Platform
- Robust, timed execution of tasks on a network of processors
- Time-triggered approach:
  - Nodes schedulers are time-synchronized
  - Tasks are run cyclically released at specific points in time
  - Messages are transferred at specific points in time
- Tasks:
  - Receive(t_k) $\rightarrow$ execute() $\rightarrow$ Send (t_k+1)
- Task: single rate, multiple components
- Components == Simulink subsystems
- Messages == input and output dataflows (signals) of subsystems
The model is translated into a scheduling problem:
**Input:** set of tasks with desired rates, set of messages with desired source/destination tasks and rates  
**Output:** task release times (in a cyclic schedule)  
**Formulation:** Constraint Satisfaction Problem (equalities and inequalities) over integers.

**Support for certification:** Off-line scheduling of time-critical tasks and messages ensures correct temporal behavior.
Realization

Modeling/Simulation Environment (Simulink/Stateflow) → Model Editing Environment (ESMoL)

Mdl2Mga

Simulation-based verification

Dataflow → Stateflow → System

Simulink Code Gen → Stateflow Code Gen → Scheduler Conf Gen

Symbolic verification (TBD)

C code → C code → TT Schedule Conf
Platforms

TTTech

- MPC 555 micros
- TTP/C comm
- TTTech Software tools
- Fault-tolerance

Gumstix/Robostix

- Linux + AVR micro
- TT Virtual Machine on Linux/UDP + FreeRTOS
- No fault tolerance (yet)
**TT Virtual Machine**

**Step 1:**
DEVS model of the TT scheduler

**Step 2:**
Prototype on POSIX interface
- Embedded Linux hosts
- Isolated Ethernet network (UDP)
- High-precision timers

**DEVS: (Discrete-Event Systems)**
Finite-State Machines with
- Continuous time model for timed transitions
- Communication/triggering via discrete events
Abstract model, has C++ simulator implementation

**TT Tasks**

**TT Sched**

**TT Comm**

**Kernel**

**Ethernet (TT, shared bus)**