Heterogeneous Modeling: Hybrid Systems

- Hybrid Models
- Languages and Verification Problems
  - Simulink and StateFlow
  - CheckMate
  - Charon
  - Masaccio
  - SHIFT
Motivation

- Hybrid Systems are becoming a major *modeling paradigm* for embedded systems
  - Capability of modeling controller and plant
  - Use of concurrent multiple levels of abstraction
- Difficult to verify and design
  - Combination of *continuous and discrete dynamics* of different types
  - Lack of “operationally strong” theoretical results
- Variety of tools and approaches *mutually incompatible* due to modeling differences
Foundations of Hybrid Model

- Used classic model by J. Lygeros, S. Sastry and C. Tomlin as basis

- Model consists of three parts:
  - Structure = sets, discrete and dynamical components
  - Time Bases = intervals over which behavior is continuous
  - Hybrid execution = rules according to which we have jumps and continuous flows

- Observations:
  - Non deterministic behavior allowed (needed)
  - Fixed interaction structure
Model 1: Hybrid Automata

1. Locations or modes (discrete states)
2. Edge
3. Guard: $g_i(x) \geq 0$
4. Jump transformation
5. Invariant: state may remain in $u$ as long as $x \in INV_u$

Initial condition
Continuous dynamics

$x \in INV_u$
$dx/dt = F_u(x)$

$u
x \in INV_u$
$dx/dt = F_u(x)$

$u'$
$x \in INV_{u'}$
$dx/dt = F_{u'}(x)$

CheckMate Hybrid Model Source: B. Krogh
System Specifications

Functional View for System Validation
Closed loop vehicle model

Driver → Vehicle
Key, Brake, Gas, Transm.
force, speed, acceleration, jerk, rpm, fuel consumption,...

Vehicle → Controller
 spark advance, injection time, throttle angle

Controller → Engine & Driveline
emissions, external noise, temperature, ...

Engine & Driveline
**INPUTS:**
- K - Key
- G - Gas Pedal
- T - Clutch Pedal & Gear Stick
- B - Brake Pedal
- C - Cruise Control

**OUTPUT:**
- n - Engine Speed
- FG - Generated Force
- VG - Vehicle Speed
- D - Comfort

**Diagram:**
- Stop
  - n=0
  - FG=0
- Startup
  - n=0
  - FG=0
- Idle
  - n=argmin(M_fuel)
  - FG=0
- Rpm Tracking
  - n=n(G)
  - FG=0
- Fast Negative Force Transient
  - max D
  - D < D_max
  - FG=FG(G,T,n)
  - f1(n) = 0 & G=0
  - G > 0 & B=1
  - G = 0 & C=1
  - f1(n,G) > 0
- Force Tracking
  - min f(D,M_fuel)
  - D < D_max
  - FG=FG(G,T,n)
  - f1(n) = 0 & G=0
- Fast Positive Force Transient
  - min D
  - M_fuel < M_max; D>D_min
  - FG=FG(G,T,n)
  - f1(n,G) > 0
- Speed Tracking
  - VG= VG(.)
- Idle & Trasm On
  - n=n(.)

**Equations:**
- \( f_I(n) = 0 \) & G=0
- \( f_I(n,G) > 0 \)
Model of Power-train

Simple?

Manifold
(continuous system)

Throttle opening angle

Engine subsystem

Spark timing

Torque

Drive-line
(continuous system with changing dynamics)

Clutch Insertion/Release

Gear change

Vehicle Speed
Combustion Process

<table>
<thead>
<tr>
<th>CRANKSHAFT ANGLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTAKE</td>
</tr>
<tr>
<td>120°</td>
</tr>
</tbody>
</table>
Engine and Drive-line
Engine and Drive-line
positive spark advance:
the spark is given before
the TDC between C and E

negative spark advance:
the spark is given after
the TDC between C and E

FSM for a single cylinder
Hybrid Model vs Mean-Value Model

- **Mean-Value Model:** accurate over a longer time window
  - regulation control problems
  - low performance transient problems

- **Hybrid Model:** cycle accurate
  - transient control problems
  - stability of delay-sensitive control algorithms
  - high performance control algorithms
Hybrid Systems Languages

- Simulation (Charon, Shift, Stateflow+Simulink)
- Formal Verification (Masaccio, Checkmate)
Outline

- **Hybrid Models**

- **Languages and Verification Problems**
  - *Simulink and StateFlow*
  - *CheckMate*
  - *Charon*
  - *Masaccio*
  - *SHIFT*
What is a simulator?

- Given a mathematical model of the system, computes its evolution and its outputs under a pre-determined set of inputs.
- The mathematical model expresses heterogeneity and concurrency.
- The simulator computes the response of the model by mapping it onto the “device” used to carry out the computation.
- In general, the computing device has limited resources and is digital.
  - We must embed the model of time of the model into the model of the computing device that gives the “common denominator” (e.g., discretize time, synchronize).
  - We must map a set of concurrent processes into a sequential system (e.g., schedule execution of concurrent processes).
Hybrid Systems Simulation

- Integrator (hold)
- Invariants & Guards
- Sampling

FSM, Discrete Event and other MOCs
Hybrid System Simulation

A simulator for hybrid systems must capture different types of behaviors:

- Continuous Time
- Discrete Events
- FSMs …

and resolve the domain interface problems.
**Continuous Time**

- **Model of computation is DISCRETE TIME**
  - All variables are computed at each time point
    - no run-time scheduling decisions on variable computation
  - Time interval can be
    - fixed (bad for stiff systems), but no run-time decision
    - variable (sophisticated solvers have this)
      - Variable time step algorithm *predicts* a time step that will satisfy accuracy criterion based on previous behavior
      - After actual computation, step may be rejected because constraints are violated
      - Run-time scheduling
Discrete Domain

◆ Two basic techniques:

▲ Zero-time assumption:

 ▼ Static scheduling of computation
 ▼ Can be done off-line for maximum efficiency (cycle-based simulation)

▲ Components modeled with delay (Discrete Event Model).

 ▼ All components evaluated at the same time-point always (wasteful)
 ▼ Follow reaction to events: schedule components whose inputs have changed (assumes internal dynamics completely captured by pure delay) Selective-trace event-driven simulation.
For $f : S \to S$, define the semantics to be a fixed point of $f$

i.e. $s$ such that
“Synchronization” between domains:

- sample the continuous time interface variables
- integrate discrete event interface signals
- detect guards and invariants (zero crossing detection)
Simulator Architecture

- One simulator (e.g. Ptolemy)
  - different algorithms for each domain and unique scheduler

- \(N\) simulators (e.g. Simulink-StateFlow, Simulink-Bones, Simulink-VCC)
  - One simulator per domain (different schedulers per domain) and communication among simulators.

  Scheduler works by transferring control to simulator

  Much less efficient but easier to do!
**Invariant Detection**

◆ **An approach:**
  ▲ the discrete event simulator checks the conditions sampling the continuous time variables

◆ **Advantages:**
  ▲ easiest implementation
  ▲ strong separation between the two domains

◆ **Drawbacks:**
  ▲ high precision detection reached only with long simulation time.
  ▲ high inter-process communication overhead

◆ **Partial Solution:**
  ▲ Simulation look-ahead
Outline

- Introduction to WP
- Hybrid Models
- Languages and Verification Problems
  - Simulink and StateFlow
  - CheckMate
  - Charon
  - Masaccio
  - SHIFT
- Conclusions and Future Work
CheckMate

Simulink/Stateflow Front End (graphical editing, simulation)

Threshold-event-driven Hybrid Systems (TEDHS)

Source: B. Krogh
The CheckMate Model: TEDHS

Three parts:

- **Switched Continuous System (SCS)**, that takes in the discrete-valued input $u$ and produces continuous state vector $x$ as output into TEG.

- **Threshold Event Generator (TEG)**, produces an event when a component of $x$ crosses a corresponding threshold from the specified direction (rising, falling, or both) and feeds FSM.

- **Finite State Machine (FSM)**, whose output, in turn, drives the continuous dynamics of the SCS.
Deriving DES Models from Hybrid System Models

Given a hybrid system:

1. Consider behavior *only* at event times
2. Compute reachability between sets of continuous states
3. Perform analysis/control synthesis using the resulting transition system
4. If necessary, refine sets of states & return to 2.

*Source: B. Krogh*
CheckMate

Simulink/Stateflow Front End (graphical editing, simulation)

Threshold-event-driven Hybrid Systems (TEDHS)

Conversion

Polyhedral-Invariant Hybrid Automaton (PIHA)

Initial Partition

Flow Pipe Approximations

Quotient Transition System

Partition Refinement

ACTL Verification

Source: B. Krogh
The Polyhedral Invariant Hybrid Automaton

A PIHA is a hybrid automaton with the following restrictions:

- The continuous dynamics for each location is governed by an ordinary differential equation (ODE).
- Each guard condition is a linear inequality (a hyper-plane guard).
- Each reset condition is an identity.
- For the hybrid automaton to remain in any location, of the hybrid system all guard conditions must be false. This restriction implies that the invariant condition for any location is the convex polyhedron defined by conjunction of the complements of the guards. This gives rise to the name polyhedral-invariant hybrid automaton.
CheckMate Summary

- Integrated with Matlab/Simulink/StateFlow
- Limited semantics to simplify analysis and allow formal verification
- Uses Simulink constructs to enter data
- Based on reachability analysis to abstract continuous away
- Can perform simulation, partial and complete verification
- Computationally complex...
Outline

- Hybrid Models
- Languages and Verification Problems
  - Simulink and StateFlow
  - CheckMate
  - Charon
  - Masaccio
  - SHIFT
What is Charon?

Charon is a high-level modeling language and a design environment for hybrid systems reflecting the current state of the art both in formal and object oriented methods (UML).

- Architectural Hierarchy (Agents)
- Behavioral Hierarchy (Modes)

Charon toolkit

- Syntax-directed editor
- Parser and type checker
- Global simulator
- Plotter (from Ptolemy)
Language Summary

- Individual components described as agents
- Individual behaviors described as modes
- Support for concurrency
  - Shared variables as well as message passing
- Support for discrete and continuous behavior
- Well-defined formal semantics
**Continuous Behavior in Charon**

**Differential Constraints**
- \( \text{write Position robot\_Pos;} \)
- \( \text{diff diffStop \{d(robot\_Pos.x)=0.0; d(robot\_Pos.y)=1.0;\}} \)

**Algebraic Equations**
- \( \text{write real robot\_EST;} \)
- \( \text{read x;} \)
- \( \text{alge cont\_EST \{ robot\_EST = \text{foo}(x) + \text{bar}(x); \}} \)

**Invariant Constraints in Modes**
- \( \text{inv invTUCost \{ lub <= x <= gub; \}} \)
Simulation in Charon

- In the present approach, a program-specific simulator is generated from the Charon program.

- Each object of the Charon program is converted into an executable Java object.

- Together with a program-independent core, these objects implement behavior of the program (Compiled-Code simulator).
Future Extensions

- Graphical input language
- Modular simulation
- Model Checker
Outline

- Hybrid Models
- Languages and Verification Problems
  - Simulink and StateFlow
  - CheckMate
  - Charon
  - Masaccio
  - SHIFT
The FRESCO Project
(Formal Real-Time Software Components)

Hybrid System Model

MASACCIO: correctness by formal verification against requirements

Time-Safe Code

GIOTTO: correctness by schedulability analysis against resources
No formal connection between requirements, model, and resources:
- expensive development cycle iterates all stages

No exact correspondence between model and code:
- difficult to upgrade code
- difficult to reuse code
Embedded Software Design: UCB and PARADES Vision

Design Verify

Model

Compilation (analysis, optimization, and code generation)

Code
Hierarchical Hybrid Modules

Time-Triggered Blocks of C Code

Compile

Synthesize

Refine

Model-check

DESIGN

REQUIREMENTS

MODEL

CONSTRAINTS

PROGRAM

EXECUTABLE

ARCHITECTURE SCHEDULER COMMUNICATION

given

SLDL 1

MASACCIO

GIOTTO

GIOTTO-ASC

RTOS 1
Semantics:

Component = interface + behaviors

Interface (the “statics”):
- Variables: input/output, discrete/continuous (data)
- Locations: entry/exit (control)

Behavior (the “dynamics”):
- Jumps: all variables may change (instantaneous)
- Flows: continuous variables evolve (real-valued duration)
Masaccio & Charon: an informal comparison

Charon's hierarchy:
- architectural -> agents -> parallel composition
- behavioral -> modes -> parallel & serial comp

Masaccio's hierarchy:
- both architectural & behavioral -> components -> parallel & serial comp.

Features:
- Charon -> Simulation; more developed
- Masaccio -> Formal Verification; few papers and few applications; focusing on Giotto at the moment
Outline

- Introduction to WP
- Hybrid Models
- Languages and Verification Problems
  - Simulink and StateFlow
  - CheckMate
  - Charon
  - Masaccio
  - SHIFT
- Conclusions and Future Work
Motivation: California PATH Smart AHS (Automated Highway Systems)

3. **Semantics:** similar to other languages, but with extensions for creating and deleting components (i.e. hybrid sub-systems) **dynamically**.

5. **Syntax:** C-like (component types akin to struct types in C).
- **SHIFT** = *Hybrid System Tool Interchange Format*

- Programming language for describing dynamic networks of hybrid automata.

- Hybrid systems are components: can be created, interconnected and destroyed as the system evolves.

- Components may evolve independently, or interact through their inputs, outputs and exported events. *The interaction network itself may evolve.*