**Heterogeneous Modeling: Hybrid Systems**

- **Hybrid Models**
  - Automotive Powertrain

- **Languages and Verification Problems**
  - Simulink and StateFlow
  - CheckMate
  - Charon
  - Masaccio
**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

- locations or modes (discrete states)
- edge
- guard
- $e_i: g_i(x) \geq 0$
- invariant: state may remain in $u$ as long as $x \in \text{INV}_u$
- jump transformation
- initial condition
- continuous dynamics
- $x \in X_0$

$u$
$x \in \text{INV}_u$
$dx/dt = F_u(x)$

$u'$
$x \in \text{INV}_{u'}$
$dx/dt = F_{u'}(x)$

$x_u' \in J_i(x_u)$
System Specifications

Functional View for System Validation

DriverInTheLoop

System

EngOn/Off
Eng-RPM
Vehicle-Speed
Vehicle-Acceleration

Key
Transmission
Accelerator

K
T
A

EngState
Eng-RPM
Vehicle-Speed
Vehicle-Acceleration
Closed loop vehicle model

Driver

Key, Brake, Gas, Transm.

Vehicle

force, speed, acceleration, jerk, rpm, fuel consumption, ...

emissions, external noise, temperature, ...

Engine & Driveline

spark advance, injection time, throttle angle

Controller
**Inputs:**
- K - Key
- G - Gas Pedal
- T - Clutch Pedal & Gear Stick
- B - Brake Pedal
- C - Cruise Control

**Outputs:**
- n  - Engine Speed
- \( F_G \) - Generated Force
- \( V_G \) - Vehicle Speed
- D  - Comfort

**States:**
- **Stop**
  - \( n = 0 \)
  - \( F_G = 0 \)

- **Start Up**
  - \( n = . \)
  - \( F_G = 0 \)

- **Idle**
  - \( n = \text{arg min}(M_{fuel}) \)
  - \( F_G = 0 \)

- **Rpm Tracking**
  - \( n = n(G) \)
  - \( F_G = 0 \)

- **Fast Negative Force Transient**
  - \( n < n_{min} \)
  - \( (K = Off) \)

- **Force Tracking**
  - \( \tau < \tau_{max} \)
  - \( F_G = F_G(G,T,n) \)

- **Fast Positive Force Transient**
  - \( M_{fuel} < M_{max} ; D > D_{min} \)
  - \( F_G = F_G(G,T,n) \)

- **Speed Tracking**
  - \( V_G = V_G(.) \)

- **Idle & Transmission**
  - \( n = n(.) \)

**Decision Logic:**
- \( G > 0 \)
- \( G = 0 \)
- \( G < 0 \)

**Conditions:**
- \( T > 0 \)
- \( T = 0 \)
- \( G > 0 \)
- \( G < 0 \)
- \( G = 0 \)

**Variables:**
- \( f_i(n) = 0 \) & \( G = 0 \)
- \( f_j(n,G) > 0 \)
- \( f_l(n) = 0 \) & \( G = 0 \)
- \( f_m(G,T,n) \)

**Formulas:**
- \( n = \text{arg min}(M_{fuel}) \)
- \( \tau_{max} \)
- \( f(G,T,n) \)
- \( M_{fuel} < M_{max} \)
- \( D > D_{min} \)
Why Mixed Models of Computation in internal combustion engines control?

- Specifications

- Control variables

<table>
<thead>
<tr>
<th>input</th>
<th>value</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throttle valve</td>
<td>Continuous</td>
<td>Continuous</td>
</tr>
<tr>
<td>Fuel injection</td>
<td>Continuous</td>
<td>Discrete</td>
</tr>
<tr>
<td>Spark ignition</td>
<td>Discrete</td>
<td>Discrete</td>
</tr>
</tbody>
</table>

- Physical processes in the plant

- Description of the HW/SW implementation constraints
Model of Power-train

Manifold
(continuous system)

Throttle opening angle

Engine sub-system

Spark timing

Manifold pressure

Torque

Drive-line
(continuous system with changing dynamics)

Clutch Insertion/Release

Gear change

Vehicle Speed

Simple?
4-stroke engines

**intake**
- Valve open
- Fuel drawn into cylinder

**compression**
- Fuel mixture compressed

**power**
- Spark plug fires
- Burning fuel forces piston down

**exhaust**
- Valve open
- Burnt gas is pushed out

©1994 Encyclopaedia Britannica, Inc.
Engine and Power-train Model
\[ \dot{\alpha}(t) = a_\alpha \alpha(t) + b_\alpha V(t) \]

\[ \dot{p}(t) = a_p(n, p)p(t) + b_p(n, p)\alpha(t) \]

**Intake Manifold**

- \( V(t) \) - throttle motor voltage
- \( \alpha(t) \) - throttle angle
- \( n(t) \) - crankshaft speed
- \( p(t) \) - manifold pressure
Power-train dynamics

\[ \begin{bmatrix} \alpha(t) \\ n(t) \\ \omega_p(t) \end{bmatrix} = A \begin{bmatrix} \alpha(t) \\ n(t) \\ \omega_p(t) \end{bmatrix} + b T(t) + b_0 \]

- \( T(t) \) - generated torque
- \( \alpha(t) \) - torsion angle
- \( n(t) \) - crankshaft speed
- \( \theta(t) \) - crankshaft angle
- \( \omega_p(t) \) - wheel speed
- \( \zeta(t) \) - power-train dynamics
- \( \theta(t) \) - crankshaft angle
Engine and Power-train Model
Hybrid Systems in the Tagged-Signal Models (TSMs) Framework

Hybrid systems can be seen as formalisms for describing a complex system using mixed models of computation when a single one is not powerful, expressive or practical enough.

- An event \( e \in V \times T \): \( V \) is the set of values and \( T \) is the set of tags e.g.
  - universal time (\( T \) is the set of real numbers)
  - discrete time (\( T \) is a totally ordered discrete set),

- a signal is a set of events,

- a process with \( N \) channels is a subset of the set of \( N \)-tuples of signals.
Models of Computation

- A *Finite State Machine (FSM)* is a synchronous TSM process in which the tags take values in $\mathbb{N}$ and the inputs, outputs and states take values on finite sets.

- A *Sequential System (SS)* is a synchronous TSM process in which the tags take values in $\mathbb{N}$ and the inputs, outputs and states assume values on infinite sets.

- A *Discrete-Event System (DES)* is a timed TSM process in which the tags are order-isomorphoric with $\mathbb{N}$ (and denote instants of time).

- A *Continuous-Time System (CTS)* is a timed TSM process in which the tags take values in a connected set on which a metric is defined (and denote instants of time).

- A *Discrete-Time System (DTS)* is a synchronous DES.
Hybrid Tagged-Signal Model of a Single Cylinder

Engine cycle

\[ s_{k+1}^i = \delta(s_k^i, u_k^i) \]

\[ \phi_k^i = \lambda(s_k^i, u_k^i) \]

Air intake

\[ m^i(t) = w[p(t), n(t)] \]

Torque profile

\[ T^i(t) = g_{\phi_k}(y_k^i, \phi(t)) \]

Torque generation delay

\[ z_{k+1}^i = f_{\phi_k}(z_k^i, \begin{bmatrix} \phi_k^i \\ m_k^i \\ q_k^i \end{bmatrix}) \]

\[ y_k^i = h_{\phi_k}(z_k^i, \begin{bmatrix} \phi_k^i \\ m_k^i \\ q_k^i \end{bmatrix}) \]
Engine Cycle (FSM)

◆ positive spark advance:
the spark is given before the TDC between the compression and expansion strokes.

◆ negative spark advance:
the spark is given after the TDC between the compression and expansion strokes.
Torque Generation Delay (SS)

Sequential System

\[ y^i_k = (m^i, q^i, \varphi^i) = (m^i_{k-2}, q^i_{k-2}, 180^\circ - \phi^i(t_{k-1})) \]

Torque Profile (CTS)

Continuous-Time System

\[ T^i(t) = g_{o_k}^i(y^i_k, \phi^i(t)) \]

\[ T^i(t) \]

real profile

spark

piece-wise profile

I \hspace{1cm} BS \hspace{1cm} PA \hspace{1cm} AS \hspace{1cm} H

k - 2 \hspace{1cm} k - 1 \hspace{1cm} k \hspace{1cm} k + 1
Engine and Power-train Model

intake manifold

power-train

cylinders

CTS

CTS

SYNC

\[ V_a(t) \rightarrow p(t) \rightarrow n(t) \rightarrow \text{SYNC} \rightarrow \text{cylinders} \rightarrow T_1(t) \rightarrow T(t) \rightarrow \zeta(t) \]
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
Outline

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

FSM, Discrete Event and other MOCs

- Integrator (hold)

Inputs

Interface

Continuous Time

Outputs

- Invariants & Guards
- Sampling

$\int$
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.
**Zero-time Loops**

For $f : S \rightarrow S$, define the semantics to be a **fixed point** of $f$

i. e. $s$ such that

$f(s) = s$
**Synchronization Problem**

- “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 continuos 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
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
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**
  - `write Position robot_Pos;`
  - `diff diffStop {d(robot_Pos.x)=0.0; d(robot_Pos.y)=1.0;}`

- **Algebraic Equations**
  - `write real robotEST;`
  - `read x;`
  - `alge contEST { robotEST = foo(x) + bar(x); }`

- **Invariant Constraints in Modes**
  - `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
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
Embedded Software Design: Current State

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

Diagram:
- Model
  - Design
  - Verify
- Code
  - Compilation (analysis, optimization, and code generation)
Hierarchical Hybrid Modules

Time-Triggered Blocks of C Code

Model-check

Synthesize Refine

Compile

given

DESIGN

MODEL

REQUIREMENTS

CONSTRANTS

PROGRAM

ARCHITECTURE

SCHEDULER COMMUNICATION

EXECUTABLE

RTOS 1

MASACCIO

GIOTTO

GIOTTO-ASC

SLDL 1

ATL
MASACCIO

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