Hybrid Systems: From Models to Code

Tom Henzinger
UC Berkeley

French Guyana, June 4, 1996
$800 million embedded software failure
Mars, December 3, 1999
Crashed due to uninitialized variable

$4 billion development effort
40-50% system integration & validation cost
Sources of Complexity

- concurrency
- real time
- heterogeneity

A hybrid system consists of multiple continuous (physical) and discrete (computational) components that interact with each other in real time.

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: Our Vision

ITR Kickoff / Chess 7

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
Continuous (Euclidean) Systems

State space: $\mathbb{R}^n$
Dynamics: initial condition + differential equations

Room temperature:

$$x(0) = x_0$$
$$x'(t) = -K \cdot x(t)$$

Analytic complexity.

Discrete (Boolean) Systems

State space: $\mathbb{B}^m$
Dynamics: initial condition + transition relation

Heater:

Combinatorial complexity.
The Curse of Concurrency
300,000 latches

10^{11} stars

10^{100,000} states
Hybrid Systems

State space: $\mathbb{B}^m \times \mathbb{R}^n$
Dynamics: initial condition + transition relation + differential equations

Thermostat:
- off: $x' = -Kx$
- on: $x' = K(H-x)$

$x \leq l$ \quad $x \geq u$

$x \leq U$ \quad $x \geq L$

Hybrid Automata
Hybrid Automata

far
\[ x' \in [-50, -40] \]
\[ x \geq 1000 \]
\[ x = 100 \]
\[ x : \in [2000, \infty) \]

near
\[ x' \in [-50, -30] \]
\[ x \geq 0 \]
\[ x = 0 \]

past
\[ x' \in [30, 50] \]
\[ x \leq 100 \]

Hybrid Automata

up
\[ y' = 9 \]
\[ y \leq 90 \]
\[ y = 90 \]

open
\[ y' = 0 \]
\[ y = 0 \]

down
\[ y' = -9 \]
\[ y \geq 0 \]
\[ y = 0 \]

closed
\[ y' = 0 \]
\[ y = 0 \]

raise? lower?
raise?
lower?
Hybrid Automata

Controller

Requirements

Safety: $\forall \Box ( x \leq 10 \Rightarrow \text{loc}[\text{gate}] = \text{closed} )$

Liveness: $\forall \Box \forall \Diamond ( \text{loc}[\text{gate}] = \text{open} )$

Real time: $\forall \Box z := 0. ( z' = 1 \Rightarrow \forall \Diamond ( \text{loc}[\text{gate}] = \text{open} \land z \leq 60 ) )$

Verification and failure analysis by model checking (e.g., HyTech).
Two Problems with Hybrid Automata

1. Scalability
   Possible solutions:
   - hierarchy (MASACCIO)
   - assume-guarantee decomposition (interfaces)

2. Robustness
   Possible solutions:
   - ε-variability
   - discounted future

MASACCIO
Hierarchical Hybrid Automata

Crossing blocked? obstacle bool → RealCrossing → start

RealCrossing

- x: real
- y1: real
- y2: real
- Distance of train from gate
- Angle of left gate
- Angle of right gate (closed: y=0, open: y=50)
1. Scalability
   
   Possible solutions:
   - hierarchy (MASACCIO)
   - assume-guarantee decomposition (interfaces)

2. Robustness
   
   Possible solutions:
   - $\varepsilon$-variability
   - discounted future

The Robustness Problem

Hybrid Automaton $\xrightarrow{X}$ Property

slightly perturbed automaton
The Robustness Problem

Hybrid Automaton

\[ x = 3 \]

Safe

The Robustness Problem

Hybrid Automaton

\[ x = 3 + \varepsilon \]

Unsafe
A Possible Solution of the Robustness Problem: Metrics on Traces

Instead of consider

A More Radical Solution of the Robustness Problem: Discounting the Future

\[ \text{value(Model,Property)}: \text{States} \rightarrow \{\text{Yes, No}\} \]

\[ \text{value(Model,Property)}: \text{States} \rightarrow \mathbb{R} \]
value(Model, Property): States → {Yes, No}

\[ \text{value}(m, \Diamond T) = \mu X. (T \lor \text{pre}(X)) \]

discountedValue(Model, Property): States → \( \mathbb{R} \)

\[ \text{discountedValue}(m, \Diamond T) = \mu X. \max(T, \lambda \cdot \text{pre}(X)) \]

discount factor \( 0 < \lambda < 1 \)

**Robustness Theorem:**

If \( \text{discountedBisimilarity}(m_1, m_2) > 1 - \varepsilon \),

then \( |\text{discountedValue}(m_1, p) - \text{discountedValue}(m_2, p)| < f(\varepsilon) \).

Further Advantages of Discounting:

- **approximability** because of geometric convergence (avoids non-termination of verification algorithms)
- applies also to probabilistic systems and to games (enables reasoning under uncertainty and control)
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

---

The History of Computer Science:
Lifting the Level of Abstraction

**High-level languages:**
Programming to the application

Requirements
focused code

Compilation

**The “assembly age”:**
Programming to the platform

Resource
focused code

- Traditional high-level languages abstract time.
- This abstraction is unsuitable for real-time applications, which are still programmed in terms of platform time (“priority tweaking”).
- GIOTTO: Real-time programming in terms of application time.
1. Concurrent Periodic Tasks:
   - sensing
   - control law computation
   - actuating

2. Multiple Modes of Operation:
   - navigational modes (autopilot, manual, etc.)
   - maneuver modes (taxi, takeoff, cruise, etc.)
   - degraded modes (sensor, actuator, CPU failures)
MASACCIO
GIOTTO Time-Triggered Programming

**Mode 1**
- Task S: 400 Hz
- Task C: 200 Hz
- Task A: 1 kHz

**Mode 2**
- Task S: 400 Hz
- Task C: 200 Hz
- Task A': 1 kHz
- Task A**: 1 kHz

**Mode 3**
- Task S: 400 Hz
- Task C: 200 Hz
- Task A: 2 kHz

**Mode 4**
- Task C': 100 Hz
- Task A: 1 kHz

**Condition 1.2**
- Task S: 400 Hz
- Task C: 200 Hz
- Task A: 1 kHz

**Condition 2.1**
- Task S: 400 Hz
- Task C: 200 Hz
- Task A: 1 kHz

Host code e.g. C
Glue code Giotto

**Functionality.**
- No time.
- Atomic.
- Sequential.

**Timing and interaction.**
- Real time.
- Reactive.
- Concurrent.

This kind of software is reasonably well understood.

The software complexity lies in the glue code.
Achieving Verifiability and Compositionality in GIOTTO: The FLET (Fixed Logical Execution Time) Assumption

**Embedded Programming in GIOTTO**

The programmer specifies sample rate $d$ and jitter $j$ to solve the control problem at hand.

The compiler ensures that $d$ and $j$ are met on a given platform (hardware resources and performance); otherwise it rejects the program.
Implementing the FLET Assumption

Contrast the FLET with Standard Practice
Advantages of the FLET and GIOTTO

- **predictable** timing and value behavior (no internal race conditions, minimal jitter)
- **portable, composable** code (as long as the platform offers sufficient performance)

Research Agenda

**From Hybrid Models**
- robust hybrid models (tube topologies, discounting)
- model checking for hierarchical and stochastic hybrid models
- multi-aspect assume-guarantee decomposition of hybrid models (interface theories for time, resources, fault tolerance)

**To Embedded Code**
- distributed schedulability analysis and code generation
- on-line code modification and fault tolerance
Credits

Scalable and Robust Hybrid Systems: Luca de Alfaro, Arkadeb Ghosal, Marius Minea, Vinayak Prabhu, Marcin Jurdzinski, Rupak Majumdar

GIOTTO: Ben Horowitz, Christoph Kirsch, Rupak Majumdar, Slobodan Matic, Marco Sanvido

Collaborators of the FRESCO Project

-Alex Aiken on time-safety analysis of embedded code
-Karl Hedrick on Giotto implementation of electronic throttle control
-Edward Lee on Giotto modeling and code generation in Ptolemy
-Edward Lee on rich interface theories as type theories for component interaction
-George Necula on model checking device drivers
-George Necula on scheduler-carrying embedded code
-Alberto Sangiovanni-Vincentelli on synthesis of protocol converters from interfaces
-Alberto Sangiovanni-Vincentelli and Shankar Sastry on platform-based design of a helicopter flight control system using Giotto
-Shankar Sastry on hybrid automata