Frameworks and Tools for High-Confidence Design of Adaptive, Distributed Embedded Control Systems

- MURI Project Overview -

Edited and presented by

Janos Sztipanovits, ISIS-Vanderbilt University
Jonathan Sprinkle, UC Berkeley

Safe and Secure System Airworthiness Certification Research and Development Symposium
Dayton, OH
September 12-14, 2006
Team

- UC Berkeley
  Edward Lee, George Necula, Shankar Sastry, Claire Tomlin (Co-PI)

- CMU
  Edmund Clarke, Bruce Krogh (Co-PI)

- Stanford
  Stephen Boyd (Co-PI)

- Vanderbilt
  Gabor Karsai, Janos Sztipanovits (PI)
Research Areas

- Hybrid and Embedded Systems
- Model-Based Design and Verification
  Source Code
- Source Code Verification and Testing
- Advanced Tool Architectures
- Testing and Experimental Tool Validation
Embedded Control System Design Flow

- Requirement Specification
- Functional (Controller) Design
- Software Architecture
- Component Design
- SW Deployment
- HW Arch. Design
- System Arch. Design
Design Flow: Tools and Analysis

- Requirement Specification
- Control Design
- Software Architecture
- HW Arch. Design
- Component Design
- System Arch. Design
- HW Pwr./Perf. Est.
- SW Deployment
- Latency/RT Analysis
- Code Gen
- Verif.
- SW Mod/Sim
- Hardware Mod/Sim
- Architecture Mod/Sim
- SW Arch. Design
- HWA
- Evidence
Overall Undertaking

- Development of component technologies in all areas
- Incrementally building a tool chain for a selected domain (UAV flight and mission control)
- Demonstration of control software development with the tool chain
- Experiments
Objective: Define the control laws to meet requirements
Platform: SL/SF-like modeling language, (Ptolemy 2; GME)
Tools: SL/SF Model Builder+Simulator (Ptolemy 2)

Requirements - Functional Design Mapping (DSMLSL/SF)
Control Design: Approaches

Goal: Design controller behavior satisfying all requirements

- Mathematical model of the Plant
- Design of a lin. or non-lin. controller satisfying stability/performance requirements
- Simulations/refinement

- Embedded Systems Modeling and Deep Compositionality
- Hierarchies of Robust Hybrid and Embedded Systems
- Verification and Validation of Conservative Approximations
- Adaptive Control Architectures for Uncertainty Handling

- Quantization, finite word length, round-off errors
- Modality
- Limited resources, resource sharing
- Concurrency models, scheduling
- Limited communication bandwidth, networking

- Uncertain dynamics, unknown non-linearities
- Fault effects, sensing errors
- Fault adaptive control
- Robust analysis, (SDP, LMI), Simulations
Addition to the Design Flow

Requirement Specification

Control Design

Component Design

Software Architecture

System Arch. Design

HW Pwr/ Perf Est

HW Arch. Design

Latency/RT Analysis

Code Gen.

Verif.

Alloc./Sched.

Analysis

Arch Mod/Sim

Functional Mod/Sim

Arch Mod/Sim

Software Architecture

Code Gen.

Verif.

Alloc./Sched.

Analysis

Arch Mod/Sim
Objective: Optimize the SW architecture by selecting a component model and by allocating functions to components.

Platform: MoC-s

Tools: GME, GReAT, DESERT, Ptolemy-2, …

Functional Architecture - SW Architecture Mapping
Software Architecture Verification

**Goal:** design software architecture using well understood composition platforms that allow verification of properties using analysis or “correct-by-construction” property guarantees.

**Embedded Software Composition Platforms**
- Heterogeneous MoC-s
- Actor Models
- Ptolemy-II based runtime support
- Formally specified semantics
- Compositional semantics for heterogeneous systems
Addition to the Design Flow

- Requirement Specification
- Control Design
- Functional Mod/Sim
- Component Design
- HW Arch. Design
- HW Pwr/Perf Est
- Software Architecture
- System Arch. Design
- Latency/RT Analysis
- Code Gen.
- Verif.
- Alloc./Sched. Analysis
- Deployment
**Objective:** Design and implement SW for components satisfying behavior defined by control laws.

**Platform:** Component Implementation Languages (Java, C++, Other..)

**Tools:** Generators (RT-Workshop; GReAT), Compilers, WCET Analyzers
Software Component Verification

**Goal:** prove that the component software behaves as intended under all foreseeable operating conditions.

- **Component Behavior Model**
- **Component Implementation**
- **Metamodeling**
- **DS Generator**
- **Meta Generator**
- **Semantic Anchoring**
- **DS Generation**
- **Model-based code generation**
- **Model Integrated Computing**
- **Code Analysis**
- **Model refinement**
- **Model verification**
- **Model compilation or hand coding**
- **Static analysis**
- **Test-based verification**

**Automated Source Code Verification and Testing**
- Model-based test generation
- Advanced static analysis

**Model Integrated Computing**
- Metamodeling
- Model-based code generation
- Meta-model-based testing of code generators
Objective: Design System configuration that meets cost/reliability/power requirements.

Platform: Comm-links; RTOS, Comp.

Middleware Tools: GME, RTOS, Comp. Middleware tools
Objective: Optimize System architecture by allocating SW components to RTOS Tasks and Communication Channels.

Platform: Composition Model

Tools: GME, DESERT, Timing Analysis,...
Deliverables

- Composable tool architecture
  - Open Tool Integration Framework
  - Prototype Tool Chain
- Testing and Experimental Validation
  - Single Platform Flight Control
  - Formation Control
  - Mission Management
Research Areas

- Hybrid and Embedded Systems
- Model-Based Design and Verification
- Advanced Tool Architectures
- Testing and Experimental Tool Validation
Focus Area 1: Hybrid and Embedded Systems (HES)

Verification and control design using hybrid systems

Embedded software design
HES 1: Embedded systems modeling and deep compositionality

- Deep compositionality
  - Ability to derive properties of a composite system solely from the properties of the components and the interconnection
  - Functionality, efficiency, accuracy, stability, timing, resource usage
  - Derive necessary and sufficient conditions on when behavior is preserved by composition

- Interfaces for hybrid components
  - Facilitate bottom-up assembly by providing a compatibility check for components that originate from different vendors
  - Facilitate top-down design by providing contracts that can be handed to independent designers of individual components

- Assume-guarantee reasoning for hybrid systems
  - A component meets its requirements when the environment it interacts with does likewise
HES 2. Hierarchies of robust hybrid and embedded systems

- Robust hybrid automata:
  - Formal models, which specify the behavior of the system up to a level of precision determined by the extent of the uncertainties
  - Ability to rapidly partition high dimensional hybrid state spaces to coarse grained accuracy, ensuing refinements for finer grained accuracy
  - Within each partition, provide robust control schemes to optimize according to performance criteria
  - QoS: “quality level parameters”
Example: Specifier/Implementer Interface

- Specifier (control/system designer) and Implementer (hardware/software/networking implementer) interact through an interface.
- Interface consists of:
  - reference design
  - certificate of performance
- Specifier provides reference design and certificate.
- Implementer implements the design using whichever method s/he wishes, provided the implementation passes the certification.
- Certificate is used to judge imperfect implementations: allows margin in implementation of reference design.
- Certificate is formulated using Lyapunov theory and implemented using LMIs and convex optimization.
Example (cont): state feedback controller, with LQR cost specification

- The plant is described as \( x(t + 1) = Ax(t) + Bu(t) \) with \( x(0) = x_0 \).

- The controller is given by

- The reference controller is the LQR optimal feedback controller, with double precision floating point coefficients.

- The implementer implements a fixed point controller, with as few bits as possible, but with the guarantee that the LQR cost is, at most, \( \epsilon \)-suboptimal.

- A controller passes the certification if:

\[
(1 + \epsilon)(A + BK)^T P^\text{nom} (A + BK) - (1 + \epsilon)P^\text{nom} + Q + K^T R K \geq 0 \\
(A + BK^\text{nom})^T P (A + BK^\text{nom}) - P + Q + K^\text{nom}^T R K^\text{nom} = 0.
\]
Example (cont): Results

- $\Phi(K)$ is the number of bits required to express $K$
HES 3: Verification and validation of conservative approximations

Verification
A mathematical proof that the system satisfies a property

Reachable set
States for which the property does not hold

Controller synthesis
Design of control laws to guarantee that the system satisfies the property
1. Always remain outside **Unsafe set**
   States in **Reachable set** will eventually reach **Unsafe set**
   (despite any possible control effort)

2. Always remain inside **Initial set**
   States in the **Safe set** will always remain in **Initial set** provided a particular control is used on the boundary
HES 3: Current work on tools for verification of hybrid control systems

- Level Set Methods (Ian Mitchell)

- CHECKMATE (Alongkrit Chutinan)
  [http://www.ece.cmu.edu/~webk/checkmate/](http://www.ece.cmu.edu/~webk/checkmate/)

- Heterogeneous Verification (Rajesh Kumar)

- PHAVer (Goran Frehse)

Applications to:
- Steer-by Wire (Jim Weimer)
- Autopilot logic (Meeko Oishi)
- Collision avoidance (Ian Mitchell, Alex Bayen)
Example: Collision avoidance systems

Differential game formulation:
Compute the set of states for which, for all possible maneuvers \( (d) \) of the red aircraft, there is a control action \( (u) \) of the blue aircraft which keeps the two aircraft separated.

http://www.cs.ubc.ca/~mitchell/ToolboxLS/
Tests at Moffett Federal Airfield

North (m)

East (m)

Separation distance (m)

EEM alert
Above threshold

Normal
OwnShipSpd=31.6 m/s, OthShipSpd=31.0 m/s
OwnShipHdg=-40.4 deg, OthShipHdg=-19.8 deg
time=364.9 s

Line up for take off
Tests at Edwards Air Force Base

[DARPA/Boeing SEC Final Demonstration: F-15 (blunderer), T-33 (evader)]
Conservative approximations: scalability of hybrid system verification

- Use simplified models and/or set representations to perform the reach set computations

**Model**

- high-order model
- weakly-coupled model
- nonlinear model

**Set representation**

- Full-dimensional calculation
- Low-dimensional calculations

**Model Order Reduction**

- reduced-order model
- decoupled subsystems

**Decomposition**

- trajectory piecewise -linear model

**Piecewise Linearization**

- Counterexample guided abstraction refinement
HES 4: Adaptive control architectures for uncertainty handling

- Adaptive control can deal with parametric uncertainty, faults, environmental and adversarial uncertainty.
- Develop a mathematical theory of fault and disturbance adaptive control, with the adaptation managed at design-time and run-time:
  - Parametric hybrid system ID
  - Reinforcement learning schemes
- For model checking adaptive systems, develop new methods for representing the behaviors of such systems for multiple histories, histories defined by equivalence classes of environmental behaviors.
1. Embedded Systems Modeling and Deep Compositionality
2. Hierarchies of Robust Hybrid and Embedded Systems
3. Verification and Validation of Conservative Approximations
4. Adaptive Control Architectures for Uncertainty Handling
Research Areas

- Hybrid and Embedded Systems
- Model-Based Design and Verification
- Advanced Tool Architectures
- Testing and Experimental Tool Validation
Focus Area 2: Model-based Software Design and Verification

**MSD-1. Model-Integrated Computing (MIC)** *(Karsai, Lee, Sztipanovits)*
- Develop formal, metamodel-based semantic foundations for domain-specific modeling languages (DSML), based on the concept of semantic anchoring, and model transformations, two core technologies in model-based design, in the form of the theory and tool infrastructure, and incorporate results in the Model-Integrated Computing (MIC) tool suite.

**MSD-2. Embedded Software Composition Platforms** *(Lee, Karsai, Sastry, Sztipanovits)*
- Develop a heterogeneous software composition platform that offers middleware support for a well-defined suite of models of computations (MoC), incorporating dynamic type checking for system-level types and seamless interfaces towards underlying systems platforms such as Time Triggered Architecture and towards higher-level modeling environments.

**MSD-3. Automated Source-code Verification and Testing** *(Clarke, Necula)*
- Develop new static analysis techniques for programming languages widely used in embedded software development such as C, C++, and Java.

**MSD-4. Model-Based Runtime Testing and Verification** *(Krogh, Tomlin, Clarke, Sztipanovits)*
- Develop algorithms for the runtime, passive conformance testing of system behavior to a set of approximate models. Integrate the algorithms in the runtime architecture.
MSD 1: Domain-Specific Model-based Development

Configured Domain-specific Modeling Tools:
- Architecture Models
- Behavior Models
- Deployment Models
- Fault Models

Metaprogrammable Tool Infrastructure:
- Model Builders (e.g. GME)
- Model Transformation Tools (e.g. GReAT)
- Tool Integration Frameworks (e.g. OTIF)

Semantic Foundations:
- Semantic Anchoring Environment (SAE)
- Semantic elements
- Analysis tools

Basic Research Challenge
MSD 1: The Problem of Semantics

Simple visual models often have ambiguous semantics.

*Is the communication between C1 and C2 synchronous or asynchronous? Not clear.*

The semantics of a modeling language is often specified in imprecise, human language...
MSD 1: Machine-readable, precise specification: Semantic Anchoring

Semantic Anchoring of DSML-s

- The “Semantic Units” are selected abstract semantics such as MoC-s
- DSML-s or their aspects are anchored to the MoC-s using transformations
- The “Semantic Units” are specified in a formal framework

Semantic units developed: Extended FSM, Timed automata
MSD 1: Verification of Model Transformations

- Design model-> { Analysis model ; Execution model }
  - Analysis on the models + executable code
- Missing piece: assurance that M/T is correct

1. No general notion of ‘correctness’ --- only correct w.r.t. some given property
2. Generate a ‘certificate’ for each transformation ‘instance’

*Based on work by Schumann, Denney, Fisher (NASA/ARC)*
Example: reachability in a finite-state system
- Property: design model is in a bisimulation relationship with the analysis model
- Input: Matlab/Stateflow, Output: SPIN/Promela

A. Narayanan and G. Karsai: Towards Verifying Model Transformations, GT-VMT 06 Workshop, at ETAPS 06, to appear in ENTCS.
The composition platform provides the basic abstractions composing systems from interacting components. Implements ‘Models of Computation’ that formally specify how components interact.
MSD 2: Embedded Software Composition Platform

- **Key:** Abstraction heterogeneity, compositionality
  - *Not* a traditional RTOS

- **Implementation constraints:**
  - Thin layer above existing RTOS/MW/VM implementations
  - ‘Composition Machine’

- **Platform semantics is formally specified (ASML?)**
  - Executable spec: can simulate the platform
  - Platform is abstract
    - Formally analyzable

- **Potentially multiple MoC-s**

- **Both event-triggered/asynchronous and time-triggered/synchronous components**

- **Implementation:**
  - On existing execution platforms
  - Benchmarking provides concrete platform metrics
MSD 3: Source-code Verification and Testing

Reasons source code needs to be verified:

- platforms often include primitive components as ‘black boxes’ written in some procedural language
- errors in the translation of the design into source code
- design becomes the specification
- details of the implementation not represented in design models
MSD 3: Source-code Verification and Testing

Extensions of current work (Edmund Clarke)

- Verifying that (manual) memory management patterns are followed in the code
- Checking of temporal safety and liveness properties using automatically generated finite abstractions of the code. Pointer aliasing necessitates the use of other approaches (e.g. separation logic), in combination with existing static analysis techniques.
- Bringing static analysis techniques closer to assembly code
  - Annotate assembly code with invariants derived from high-level source. Compile annotation into assertions about stack locations and registers.
  - Assembly code verifier (see Java byte code verifier)
- Model-based test generation (see below)
  - Correct-by-construction vs. needs-to-be-tested code
MSD 3: The MAGIC Tool:
Counterexample-Guided Abstraction Refinement

C Program → Abstraction
Abstraction Guidance → Improved Abstraction Guidance

Abstraction → Abstract Model

Abstract Model → Verification
Verification Valid? → Counterexample Valid?
Counterexample Valid? → Abstraction Refinement

Abstraction Refinement → Abstraction

Verification Valid? → Yes
System OK

Verification No → Abstraction

Abstraction No → Abstraction

Abstraction Guidance → Improved Abstraction Guidance

Abstraction Guidance → Yes
MSD 3: Translation Validation: Link to Model-based Design

- In Model-based Design,
  - Required behaviors are modeled using high-level languages like StateFlow.
  - Model-based generators are used for generating the code.

- Model checking can be used for analyzing the high-level models.

- Important to know if the generated code is consistent with high-level models.
  - Translation Validation (investigated by Pnueli et al.); no definitive solution yet
  - Clearly, this is an important direction for future research.
Objective: To achieve necessary and sufficient coverage from testing, given the knowledge of what is known to be "correct by construction"

- Proposal: Develop formal methods to:
  - identify the specifications that need to be tested (i.e., those properties not guaranteed by design-time verification); and
  - generate tests cases for these specifications.

- Approach:
  - model features introduced by the translations of designs into implementations
  - identify vulnerable points in the implementation and generate specifications for these vulnerabilities
  - generate test/s that will provide full coverage without testing “correct-by-design” features
  - also generate the specifications for source code
Research Areas

- Hybrid and Embedded Systems
- Model-Based Design and Verification
- Advanced Tool Architectures
- Testing and Experimental Tool Validation
Focus Area 3: Advanced Tool Architectures

The foundation of model-based tool architectures is the formal abstract syntax and semantics of modeling languages. Meta-programmable toolkits perform scheduling, resource management, optimization, type inference, type checking, code generation and other verification functions.
For components in embedded systems, the concept of interface theories as foundation for behavioral type systems is much more powerful than, for example, the type systems used to capture static structure in object-oriented design. Rather than relying on informal documentation to declare temporal properties, concurrency, and other dynamic properties such as valid ordering of method invocation, we are working on providing interface theory tools for embedded systems design.
Today, embedded software designers use a brittle design process of tweaking low-level facilities of a real-time operating system, until the system seems to work. In general-purpose computing, the virtual machine concept has been effective at permitting software developers to focus on problem-level design rather than platform-specific idiosyncrasies. We are developing virtual embedded machines for embedded systems design.
Components for Embedded Systems

Today’s component technology does not allow for the tightly-coupled interaction necessary for small components to meet real-time deadlines and other system resource constraints. We are working developing on a notion of reusable embeddable components that will substantially impact the design of embedded systems in industrial practice.
Research Areas

- Hybrid and Embedded Systems
- Model-Based Design and Verification
- Advanced Tool Architectures
- Testing and Experimental Tool Validation
TEV Tasks

- TEV 1 Principled Design of Verified Software for Single Aircraft (Sastry, Tomlin, Sztipanovits, Krogh)
- TEV 2 Multiple Aircraft Experiments: AF Operations in Urban Environments (Karsai, Sastry)
- TEV 3 Mission Control Experiments (Tomlin, Sastry, Karsai)
TEV1-2: Moving OCP to embedded linux: Embedded Open Control Platform

- Autonomous vehicle systems are very complex, software included.
- The Open Control Platform allowed for a modular approach to the software design, and particularly allowed control designers to not worry about the system software.
- This modular approach can allow individual (and less complex) components to be verified independently.
- Current effort is aimed at enabling it to be used in small embedded systems.
- Also Supported by Boeing and ESCHER.
OCP History

- **OCP Core Developers**
  - Provided insulation between ITAR testbeds and uncleared TDs
  - Prepared OS (and kernel) specific build scenarios for TDs

- **Technology Developers** used this isolation to produce algorithms for certain testbeds
  - Berkeley: pursuit evasion games, landing controllers

- Expected the platform computer would be the build computer as well...
The Embedded Open Control Platform (EOCP)

OCP provides an insulation layer between software-based control algorithms and the testbed/platform/OS on which they run.
TEV1-2 Plans

- Development and deployment of UAV and other autonomous systems will require comprehensive software tools.
- The developed tool suite will allow many existing hybrid system tools to be leveraged for Simulation and Verification.
- The EOCP project will enable modular system software development and will be compatible with other tools directly.
2 UAVs on Force Protection Mission: Protect a Road

- Commander interacts with staff and two UAV controller teams to define mission
- Assignment of tasks to assets is fixed before mission starts
- Controller teams interact as required for cueing
- Teams report progress periodically and request guidance as required
- UAVs have awareness of their own state and health
TEV 3: N UAVs on Force Protection
Mission: Protect a Large Area

- Commander/staff have trouble interacting with UAV teams
- Dynamic assignment of tasks to assets difficult
- Controller teams overwhelmed trying to interact with each other
- Cognitive overload of commander
- All UAVs require significant monitoring and controlling
TEV 3: Challenge Problem Mini-Investigators for Deep Surveillance and Target ID

**Commander**
- Define missions
- Maintain SA
- Provide guidance
- Handle problems

**Hen**
- Interpret mission
- Allocate tasks
- Provide picture
- Interact w/ human
- Adjustable autonomy

**Chicks**
- Some delivered by Hen
- Handle tasks
- Report status
- Self organize
Transition Approach

- Tools are disseminated through the ESCHER Repository (Boeing, Raytheon, GM)
- AFRL connections
  - John Bay, AFRL/IF
  - Ray Bortner, AFRL/VACC
- CerTA Project (Boeing/UCB)
- Future Combat Systems Program (Boeing/VU)