CIS 700-3: Selected Topics in Embedded Systems

Insup Lee
University of Pennsylvania

October 11, 2004
Introduction

Course requirements

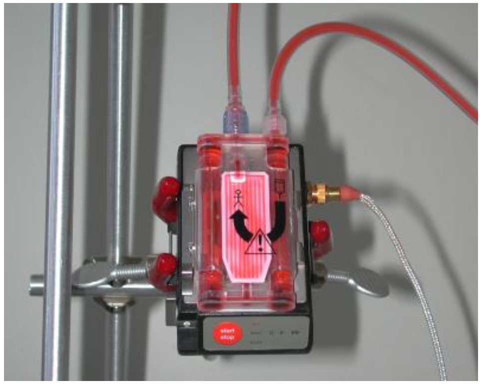
- Select a topic and then you are expected to
  - Do In-class presentation
  - Write a survey paper
  - Download a toolset and do a demo in class
- Partial paper listing at www.cis.upen.edu/~lee/04cis700
- Proceedings of RV'01, RV'02, RV '03, RV '04, WODA 2004.
- *Safeware*, Nancy Leveson, Addison Wesley,1995.

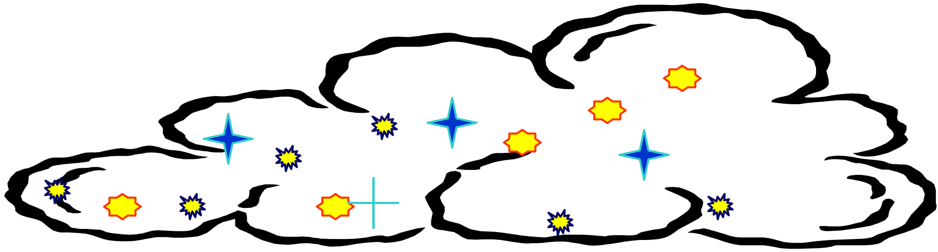
List of possible topics

- Foundations of run-time verification
- Probabilistic run-time verification
- Merging partial specifications
- Test generation from specifications, hybrid models
- Certification, CMM
- Safeware, by Nancy Leveson
- Model-carrying code
- Intrusion detection
  - Signature-based IDS, Model-based IDS
  - Anomaly-based intrusion IDS
- Application domains: medical devices, sensor networks, stateless PC
  - Modical device architecture and specification; e.g., infusion pump
  - Security in sensor networks
- Tools
  - Run-time verification: JPaX
  - Test generation: ASML
  - Software model checker: Bangor
  - Run-time concurrency analyzers
  - Etc.

Embedded Systems

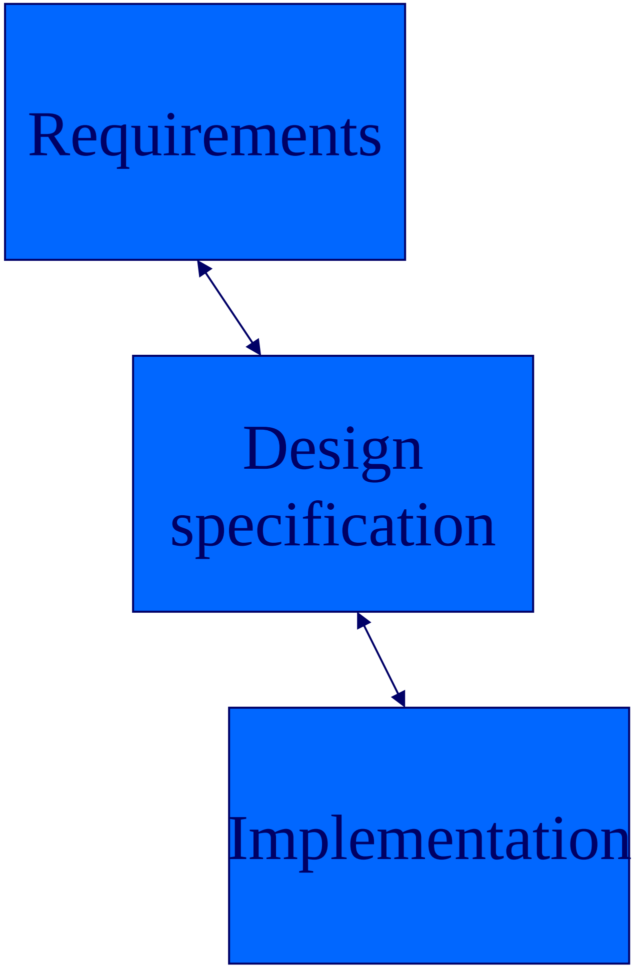
- An embedded system is a system
  - that interacts with (or reacts to) its environment, and
  - whose correctness is subject to the physical constraints imposed by the environment.
- Difficulties
  - Increasing complexity
  - Decentralized and networked
  - Resource constrained (e.g. power, size)
  - Safety critical
- Development of reliable and robust embedded software





Software Development Process

- Requirements capture and analysis
  - Informal to formal
  - Consistency and completeness
  - Assumptions and interfaces between system components
  - Application-specific properties
- Design specifications and analysis
  - Formal modeling notations
  - Analysis techniques
    - simulation, model checking, equivalence checking, testing, etc.
    - Abstractions
- Implementation
  - Manual/automatic code generation
  - Validation
    - Testing
    - Model extraction and verification
    - Run-time monitoring and checking
- Motivation & Objectives
  - make each step more rigorous using formal method techniques
  - narrow the gaps between phases



RTG: Real-Time Systems Group

- Goals:
  - To develop methods and tools for improving the reliability and quality of real-time embedded systems
  - To apply them to real world problems and applications
- Projects:
  - Modeling and analysis techniques
    - requirements capture and analysis: user requirements
    - design specification and analysis: systems and hardware/device platforms
    - Techniques
      - EMFS (Extended Finite State Machines)
      - CHARON (Hybrid systems: discrete and continuous)
  - Prototyping using simulator, code generator
  - Test generation for validation (of real implementation)
  - Runtime monitoring and checking
  - Validation and Certification
  - Real-time operating systems, e.g., resource management, scheduling
- Application domains
  - Wireless sensor networks
  - Medical devices
  - Stateless PC

Modeling languages and tools

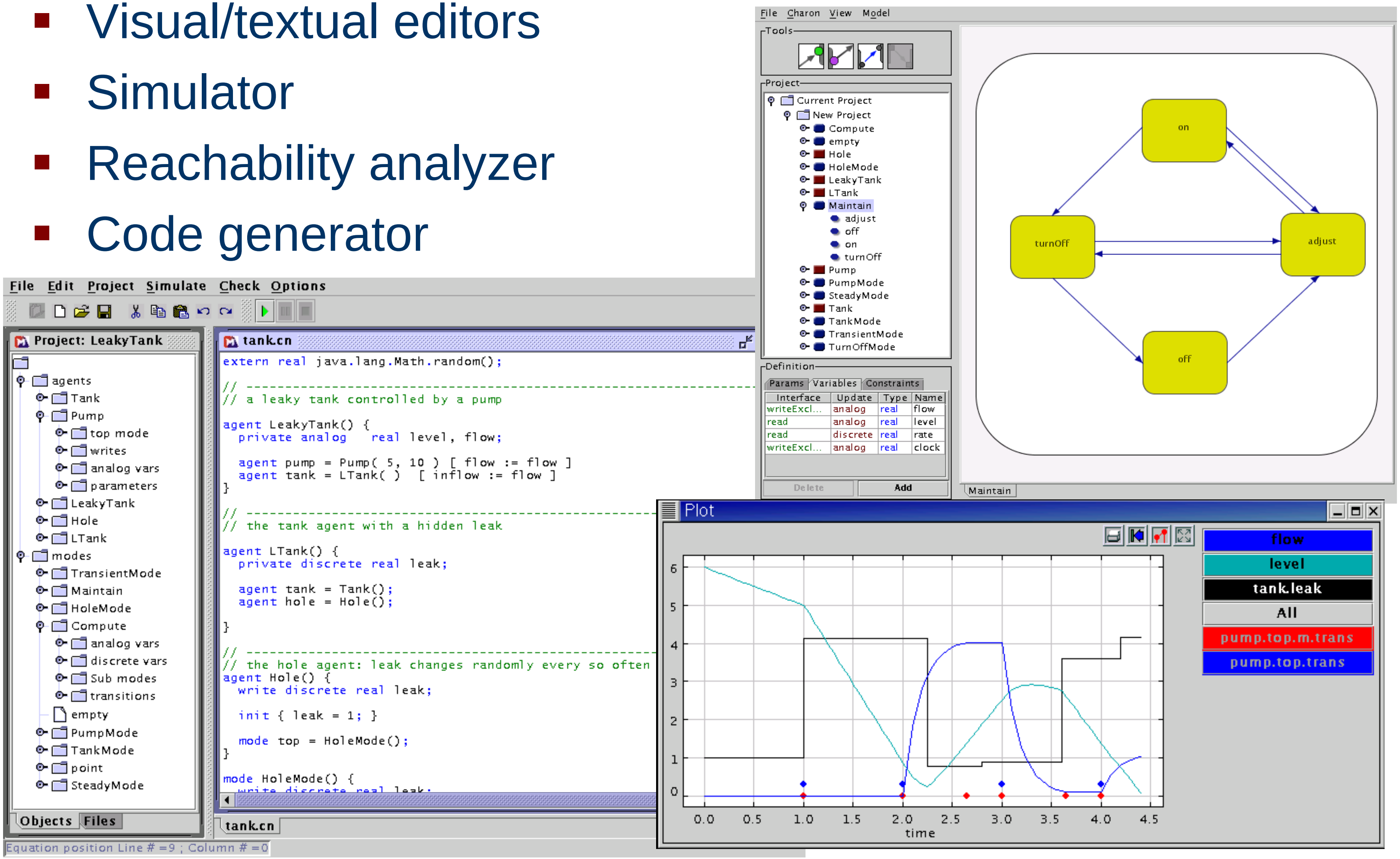
- ACSR
- CHARON
- EFSM

CHARON language

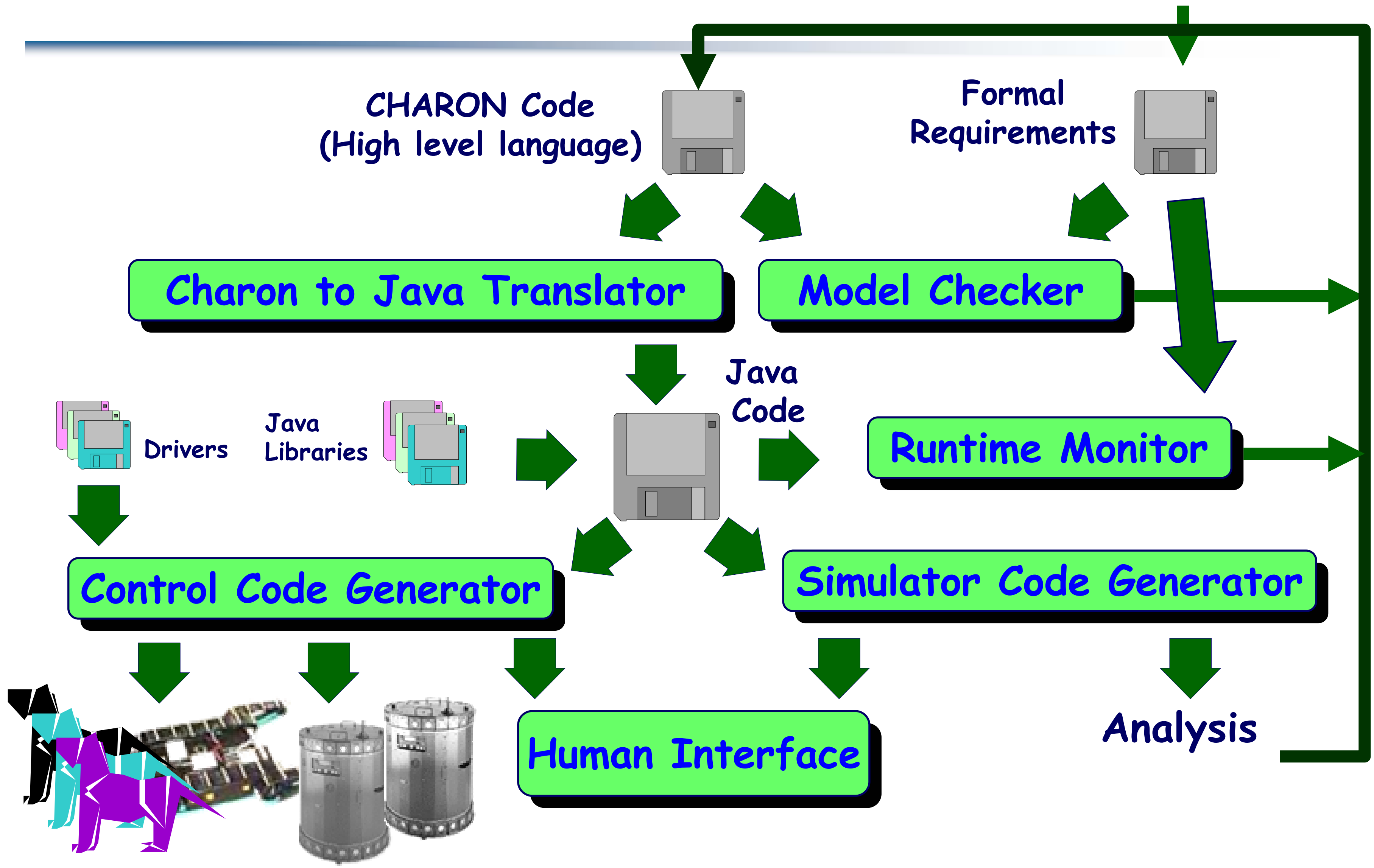
- Hierarchical modeling of concurrent embedded systems
  - Discrete computation, continuous environment
    - Avionics, automotive, medical device controllers
  - Architectural hierarchy
    - Communicating concurrent components
    - Shared variable communication
  - Behavioral hierarchy
    - Hierarchical hybrid state machines
    - Mode switches, interrupts, exceptions
- Formal compositional semantics enables rigorous analysis

Charon toolset

- Visual/textual editors
- Simulator
- Reachability analyzer
- Code generator



CHARON Environment



Example: Four Legged Robot

- Control objective
  - $v = c$
- High-level control laws

$$\dot{x} = -v \qquad x \geq -\text{stride}/2$$

$$\dot{y} = -kv$$

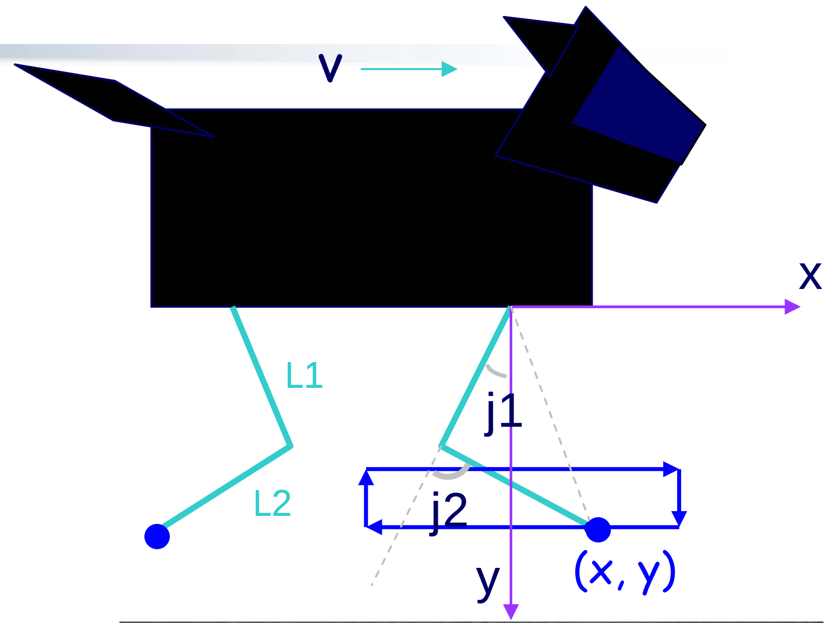
$$\dot{x} = kv \qquad x \leq \text{stride}/2$$

$$\dot{y} = kv$$

- Low-level control laws

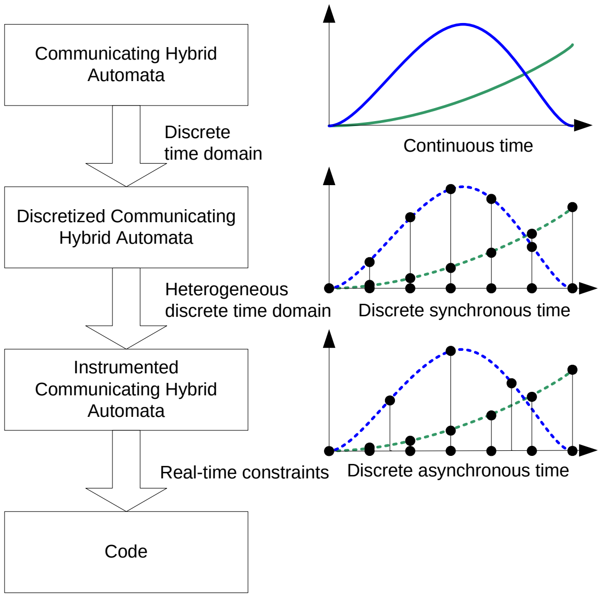
$$j_1 = \arctan(x/y) - \arccos\left(\frac{x^2 + y^2 + L_1^{\,2} - L_2^{\,2}}{2L_1\sqrt{x^2 + y^2}}\right)$$

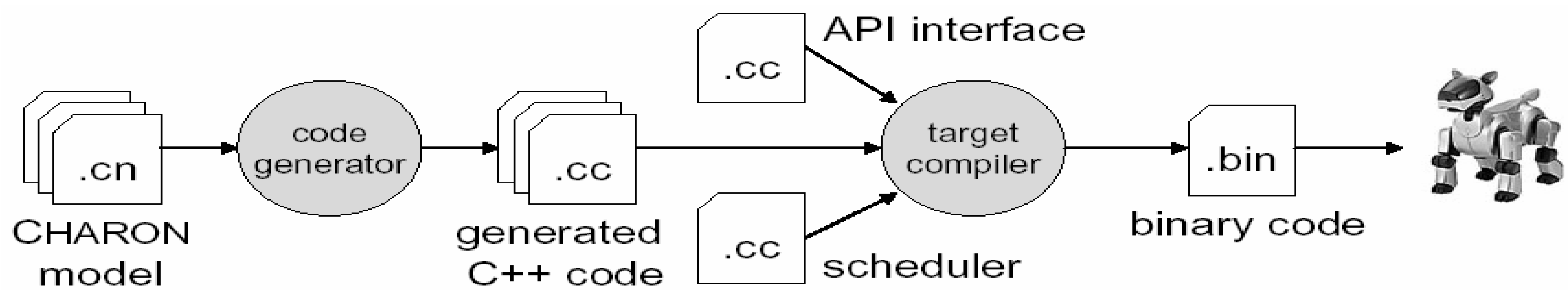
$$j_2 = \arccos\left(\frac{x^2 + y^2 + L_1^{\,2} - L_2^{\,2}}{2L_1L_2}\right)$$



CHARON Code Generator

- CHARON code generator translates CHARON models into C++ code
  - Each object of CHARON models is translated into a C++ structure
- Generated C++ code is compiled by the target compiler along with additional code
  - Run-time scheduler: invokes active components periodically
  - API interface routines: associates variables with devices
- Correctness of generated code



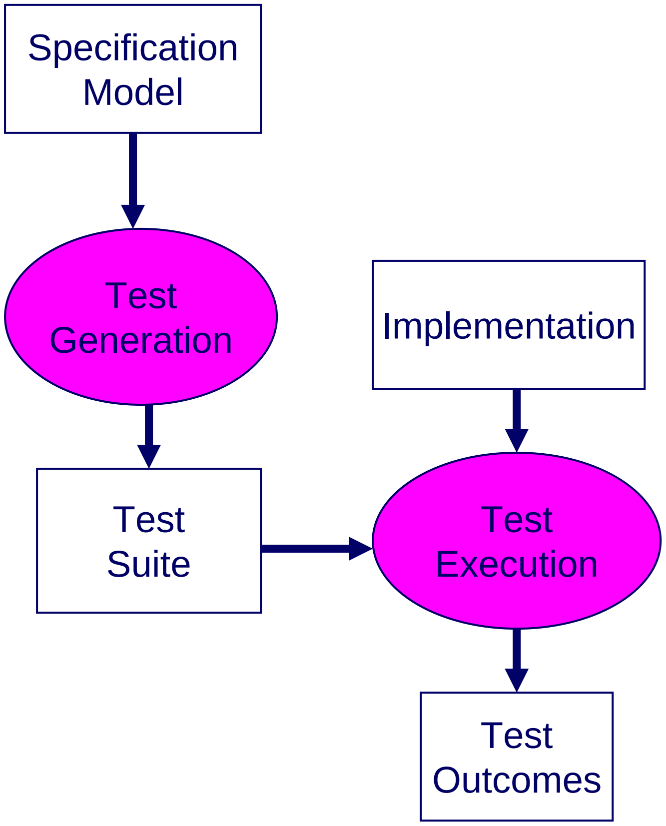


Bridging the gap between specification and implementation

- Model-based code generation and synthesis
- Model-based testing
- Software model checking
- Run-time monitoring and checking (i.e., run-time verification)

Model-based testing

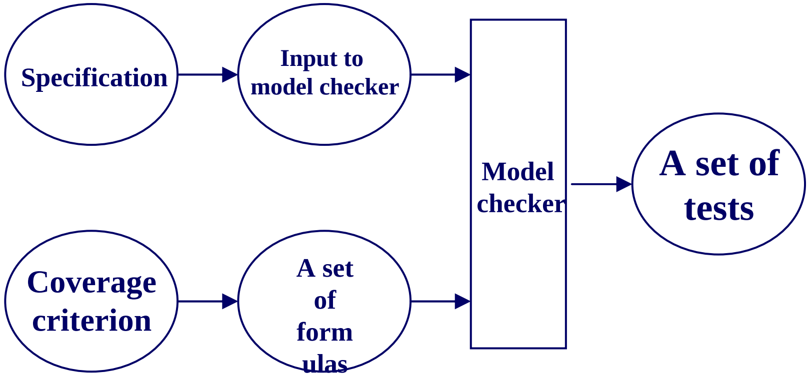
- Narrowing the gap between the model and implementation
- Testing remains the primary validation technique
- Model-based test generation adds rigor to testing:
  - Provide test suites based on a formally verified model
  - Conventional testing coverage criteria applied to the *model*
- Determines whether an implementation conforms to its specification
- Two main steps
  - Test generation from specification model
  - Test execution of implementation

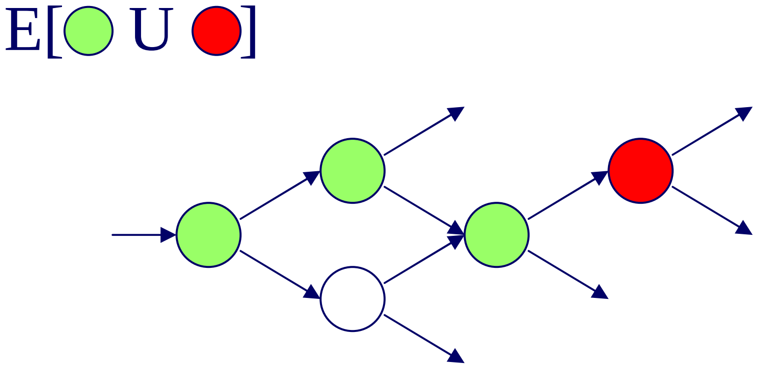


Model-based test generation

- Developed a framework for test generation:
  - Model is Extended Finite-State Machines (EFSM)
  - Coverage Criteria
    - control-flow (e.g., state coverage, transition coverage)
    - data-flow (e.g., all-def, all-use coverage)
  - Test generation using model checker
  - Covert test sequences to scripts for test execution

- Basis for conformance metrics

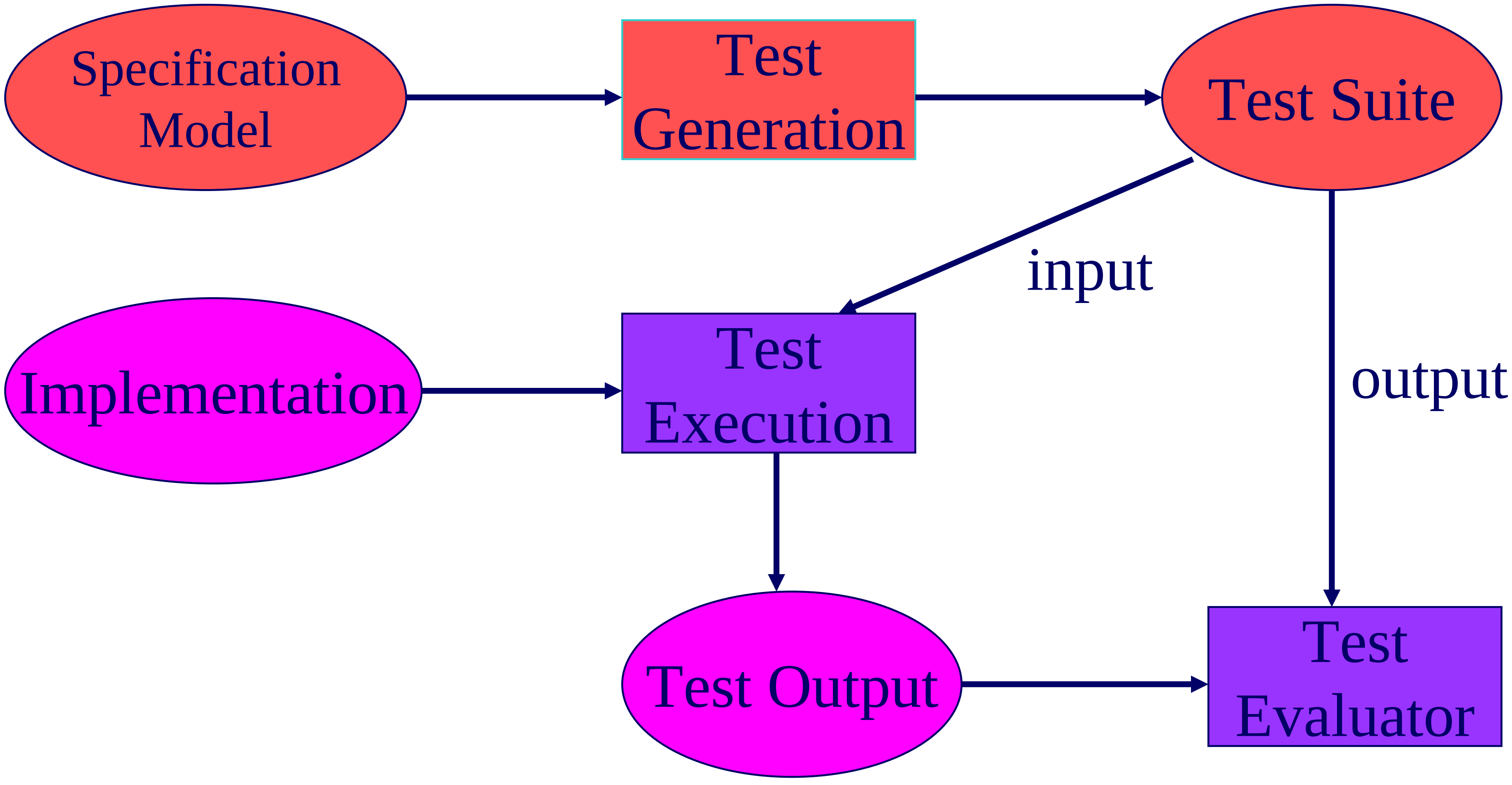




Testing-based Validation

- Determines whether an implementation conforms to its specification
  - Hardware and protocol conformance testing
  - Widely-used specifications
    - Finite state machines and labeled transition systems
- Two main steps
  - Test generation from specifications
    - What to test, how to generate test
  - Test execution of implementations
    - Applies tests to implementations and validates the observed behaviors

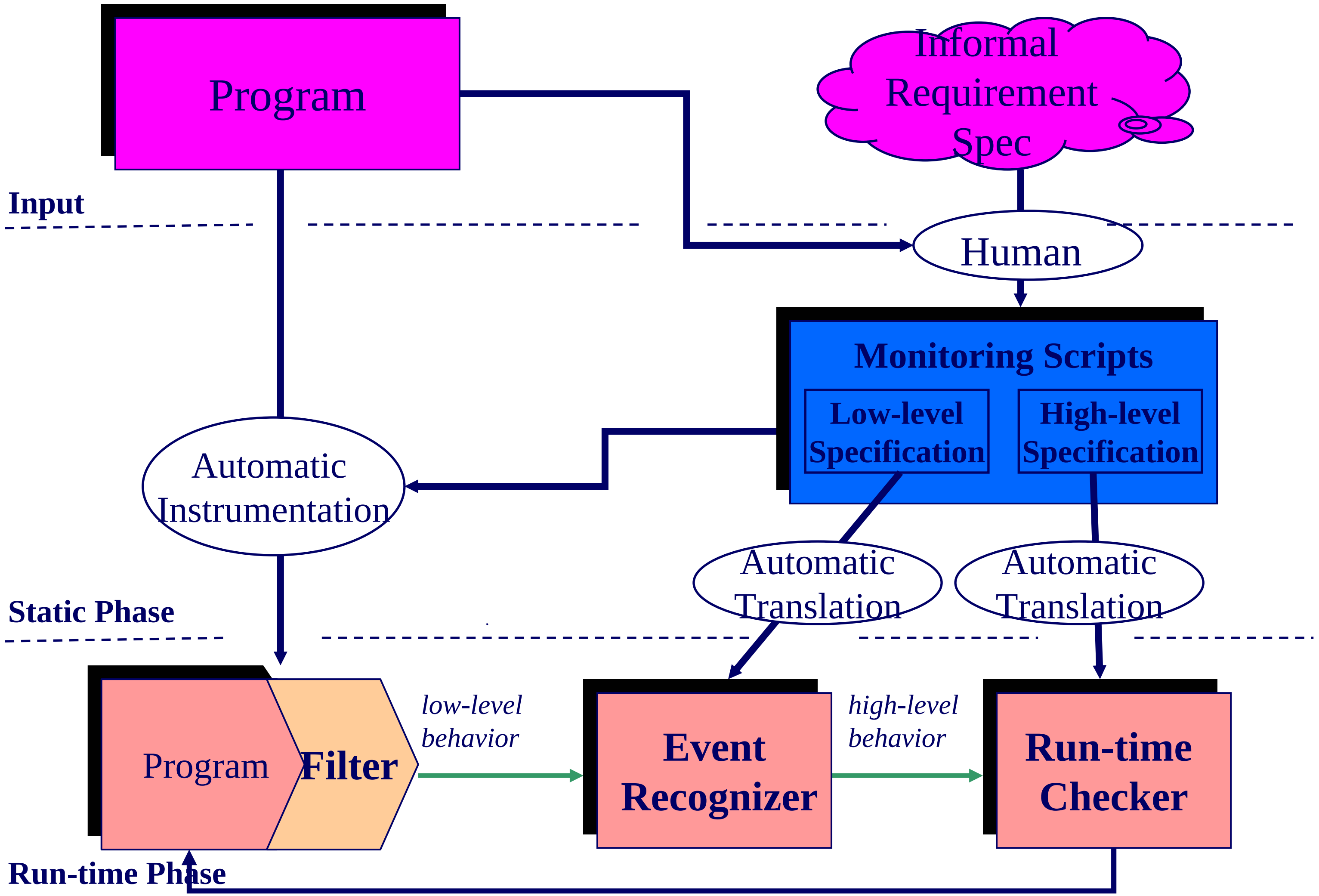
Model-based testing



Run-time verification and checking

- Run-time monitoring and checking (MaC) w.r.t. formal specification
- Ensures the runtime compliance of the current execution of a system with its formal requirement
  - **detect** incorrect execution of applications
  - **predict** error and **steer** computation
  - **collect** statistics of actual execution
- Complementary methodology to formal verification and program testing
- Prevention, avoidance, and detection & recovery

The MaC Framework



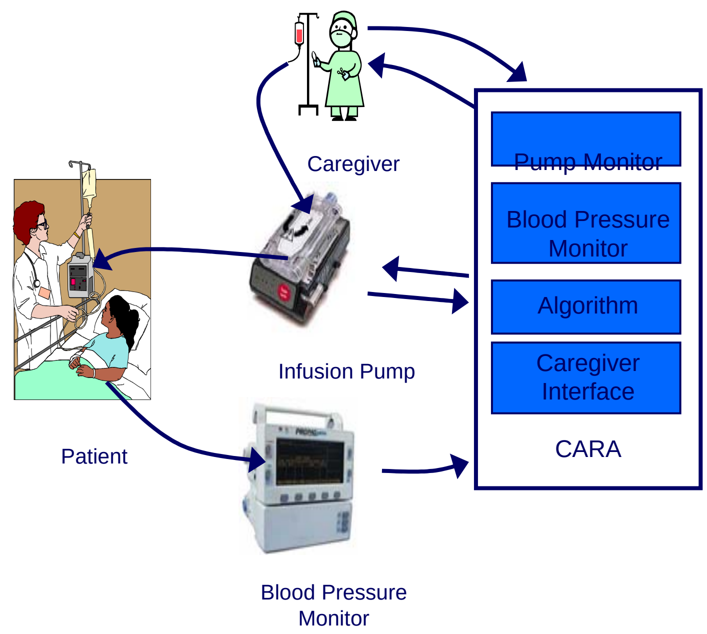
Case Studies

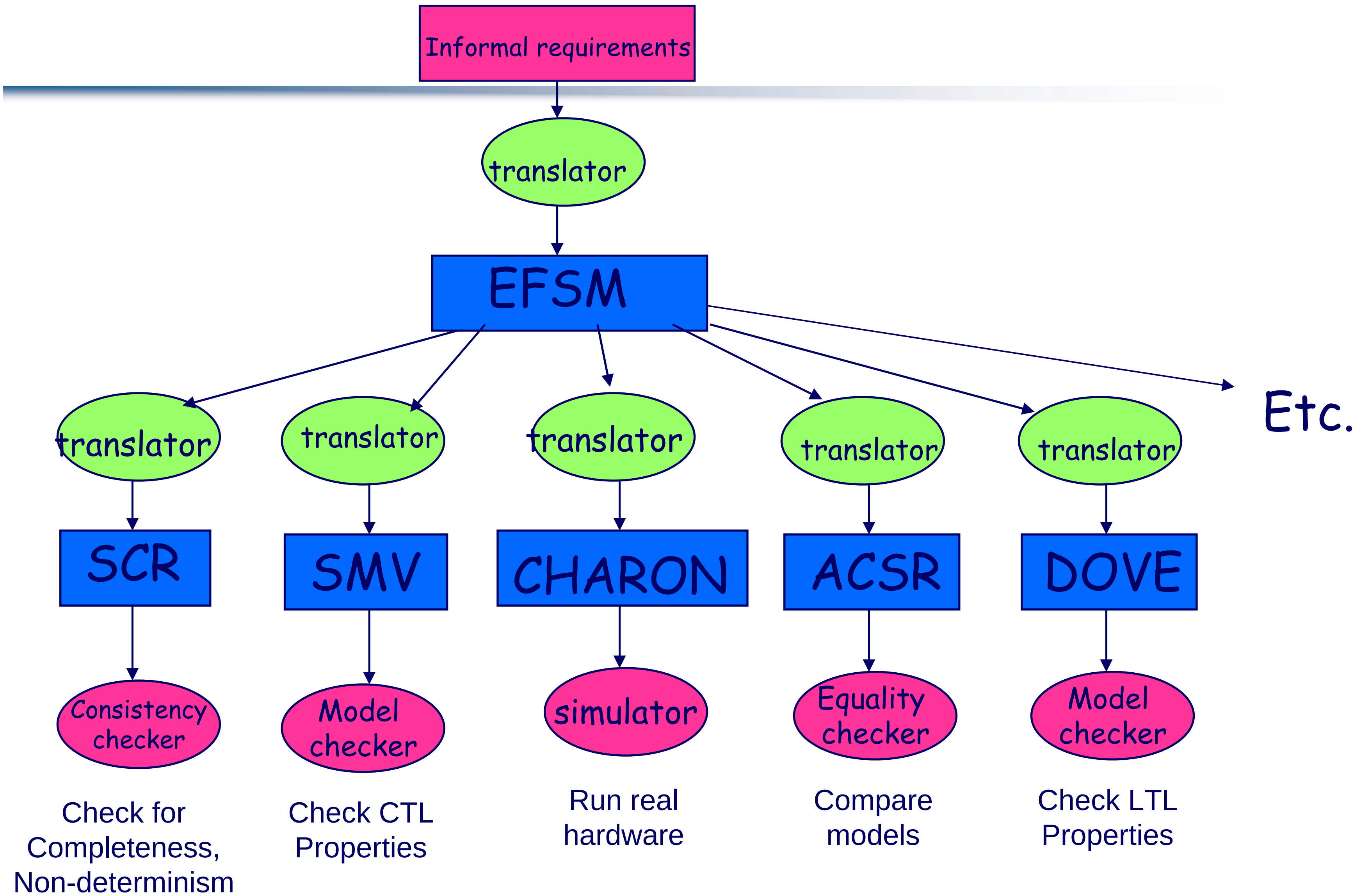
Experience/case studies in medical devices

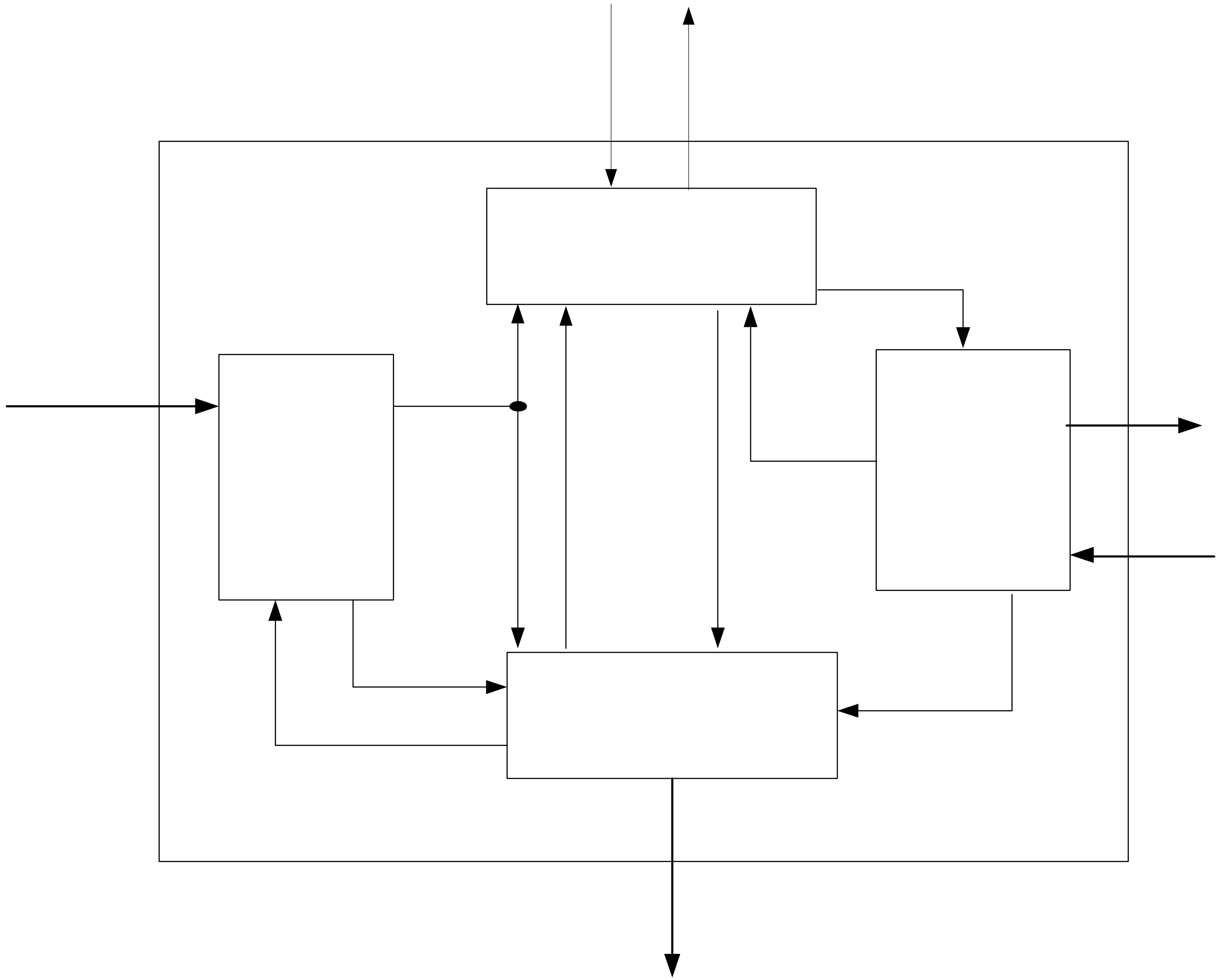
- CARA infusion pump system
  - Requirements modeling and analysis
  - Design specification and analysis
  - Hardware in-the-loop simulation
- Blood bank policy and DBSS
  - Extracting formal models from FDA guidelines
  - Test generation from models
  - (evaluation of DBSS for conformance to the FDA guidelines)
  - (testing DBSS)

CARA case study

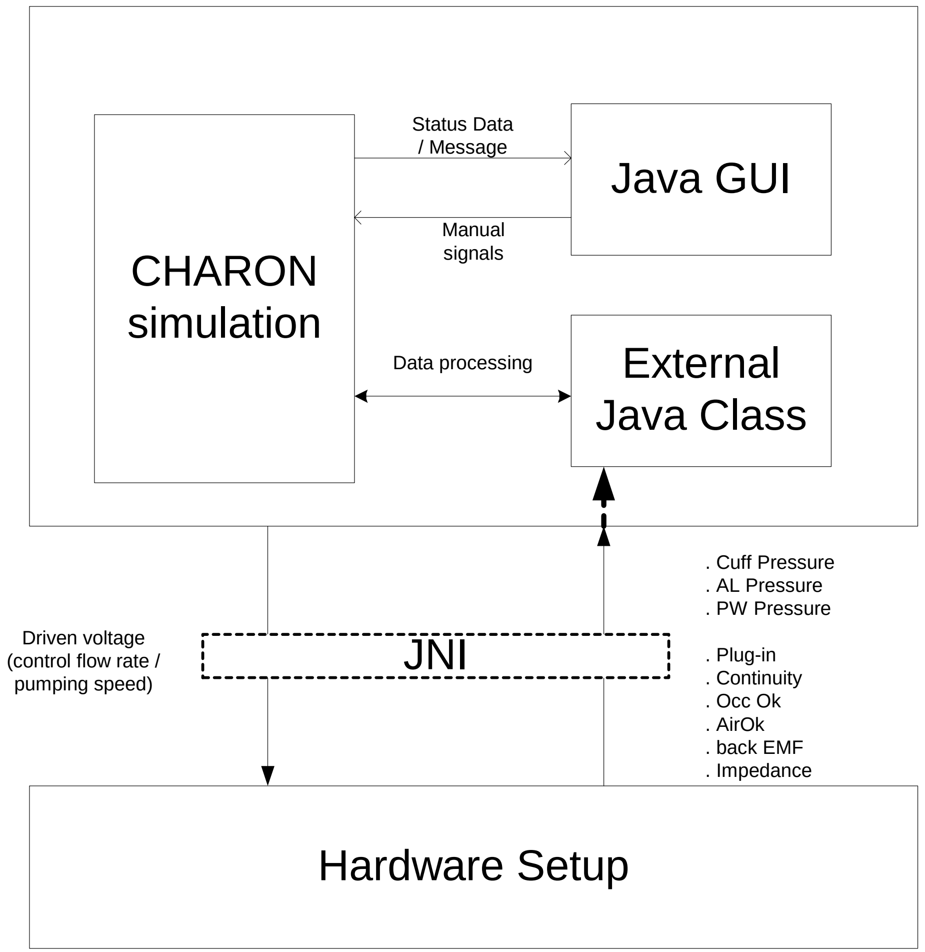
- The CARA (Computer Assisted Resuscitation Algorithm) infusion pump control system is being developed by WRAIR (Walter Reed Army Institute of Research)
- Goals:
  - Study applicability of state-of-the-art formal techniques for development of safety critical embedded systems
  - System modeling from requirements
  - Formulation and checking of properties on models
    - General properties
    - Specific safety properties (from requirements)

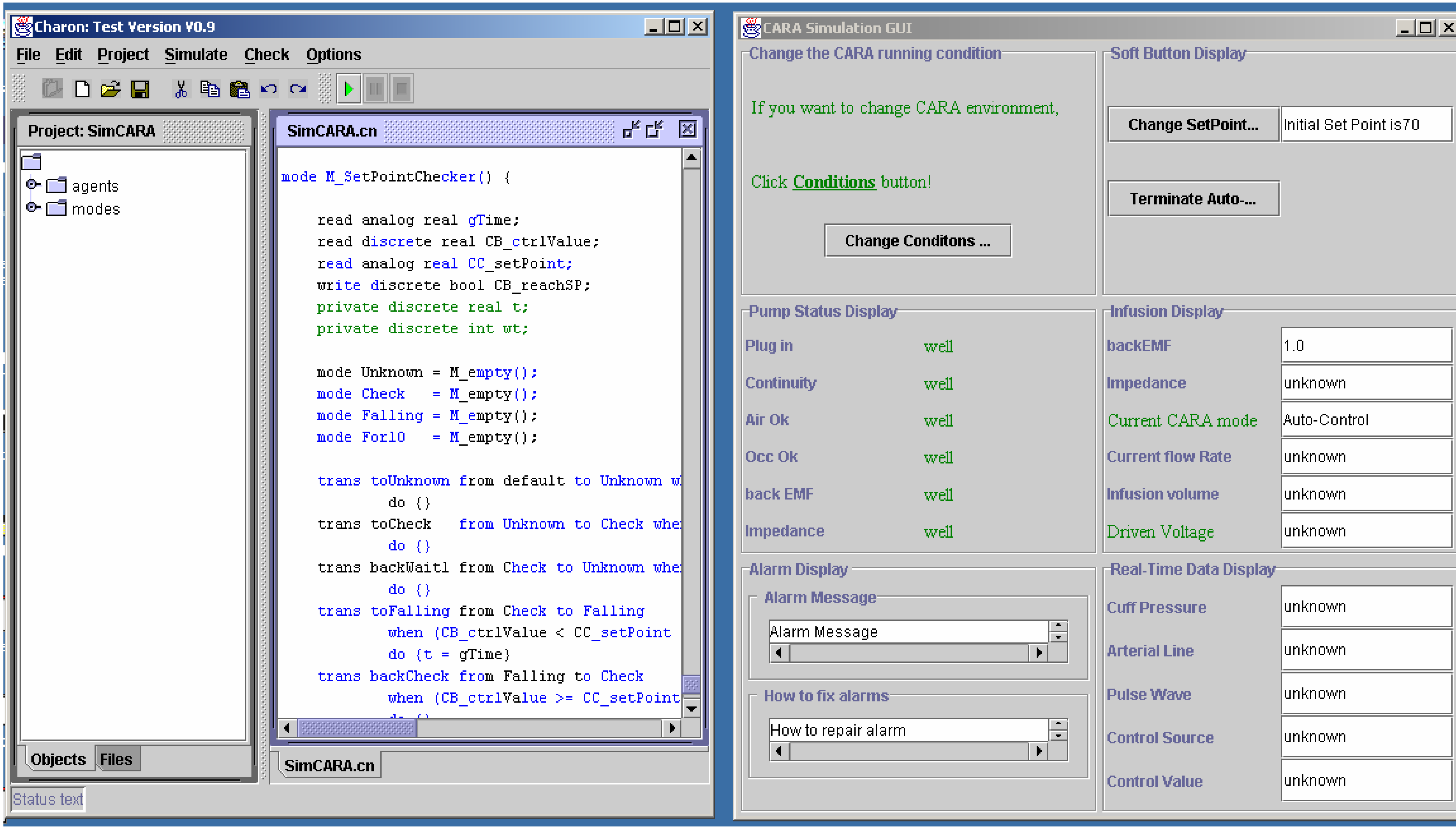



Informal requirements
translator
EFSM
translator
translator
translator
translator
translator
Etc.
SCR
SMV
CHARON
ACSR
DOVE
Consistency checker
Model checker
simulator
Equality checker
Model checker
Check for Completeness, Non-determinism
Check CTL Properties
Run real hardware
Compare models
Check LTL Properties



Interfaces of CARA Simulation




Charon: Test Version V0.9
File Edit Project Simulate Check Options
Project: SimCARA
agents
modes
Objects Files
Status text
SimCARA.cn
mode M_SetPointChecker() {
read analog real gTime;
read discrete real CB_ctrlValue;
read analog real CC_setPoint;
write discrete bool CB_reachSP;
private discrete real t;
private discrete int wt;
mode Unknown = M_empty();
mode Check = M_empty();
mode Falling = M_empty();
mode For10 = M_empty();
trans toUnknown from default to Unknown w
do {}
trans toCheck from Unknown to Check whe
do {}
trans backWait1 from Check to Unknown whe
do {}
trans toFalling from Check to Falling
when (CB_ctrlValue < CC_setPoint
do {t = gTime}
trans backCheck from Falling to Check
when (CB_ctrlValue >= CC_setPoint
CARA Simulation GUI
Change the CARA running condition
If you want to change CARA environment,
Click Conditions button!
Change Conditons ...
Soft Button Display
Change SetPoint...
Initial Set Point is70
Terminate Auto-...
Pump Status Display
Plug in well
Continuity well
Air Ok well
Occ Ok well
back EMF well
Impedance well
Infusion Display
backEMF 1.0
Impedance unknown
Current CARA mode Auto-Control
Current flow Rate unknown
Infusion volume unknown
Driven Voltage unknown
Alarm Display
Alarm Message
Alarm Message
How to fix alarms
How to repair alarm
Real-Time Data Display
Cuff Pressure unknown
Arterial Line unknown
Pulse Wave unknown
Control Source unknown
Control Value unknown

Hardware in-the-loop Simulation

- We connected the CHARON Simulator and GUI to the hardware setup.
- The hardware consists of four components:
  - M100 Infusion Pump
  - 2 1000mL flasks
  - Pressure Sensor
  - A/D interface

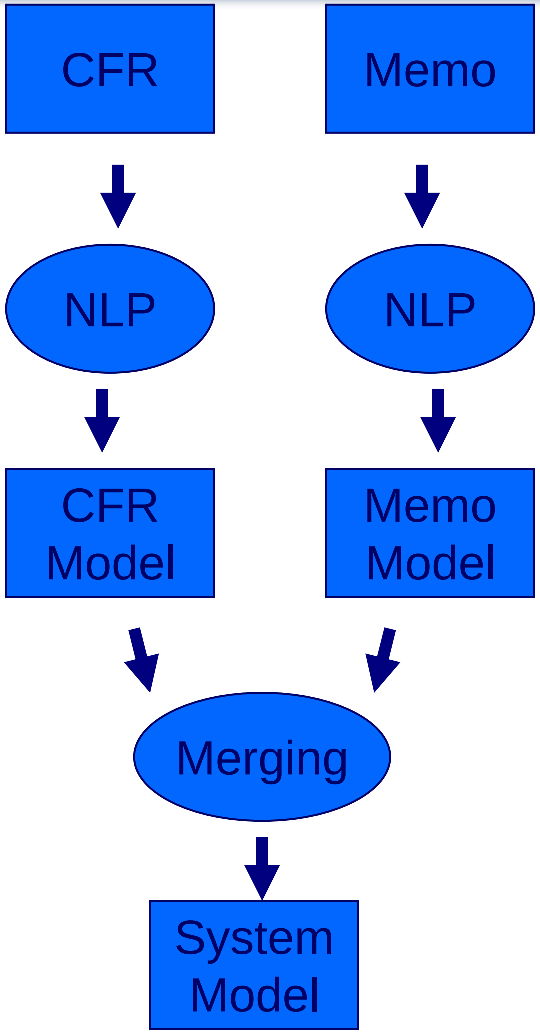
Blood Bank Case Study

- The FDA Code of Federal Regulations (CFR) requirements are complemented by an explanatory guidance memo.
- Extract formal models from documents and then analyze for
  - errors such as incompleteness;
  - inconsistencies between documents; and
  - requirements traceability and maintenance.
- DBSS (Defense Blood Standard System) is the system used by the Army to keep track of their blood supply.

Errors found include:

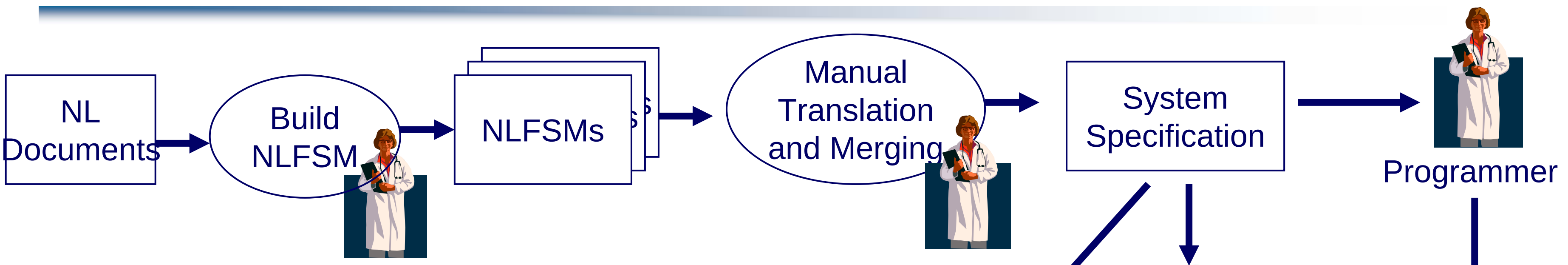
- Inconsistency
- Incompleteness

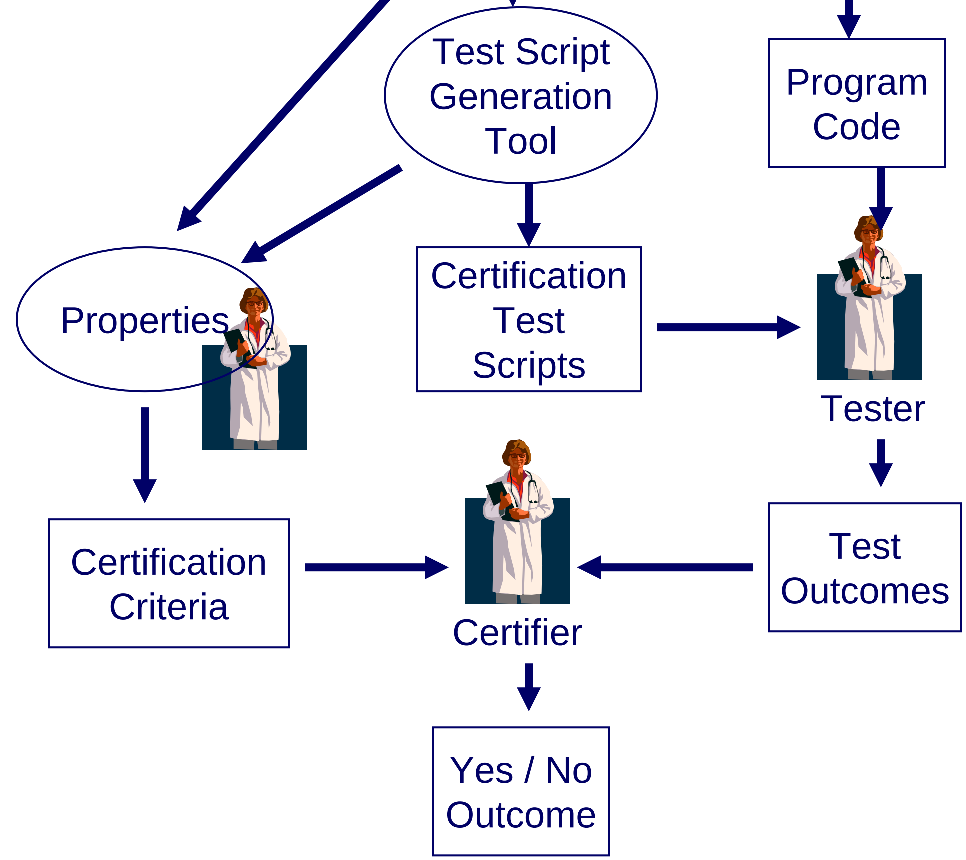
Our approach



- CFR and Memo documents are translated into formal models.
- Merge multiple models into a single model to
  - Verify using formal methods techniques
  - Generate test suite
- Working on semi-automatic way to extract models using NLP techniques
- Army's DBSS

Policy Modeling and Verification





1. Write NL Requirements
2. Extract formal System Specification (EFSMs)
3. Programmer implements system
4. Create Test Scripts
5. Tester runs scripts on implementation
6. Certifier uses test results and properties to decide if implementation passes

The HASTEN Project

- High Assurance Systems Tools and ENvironments (HASTEN)
- Develop techniques and tools for “end-to-end” software engineering of embedded systems
  - Requirements capture
  - Specification, analysis, simulation
  - Implementation generation and validation: code generation, testing
  - Deployed system monitoring, checking, and steering
- Integrated use of tools and artifacts
  - Vertical integration: multiple uses of models
  - Horizontal integration: multiplicity of techniques
- Case Studies and Tech Transfers

Opportunities and Challenges

- Modeling challenges
  - Semi-automatic extraction of formal models from informal docs
  - Composition of partial, heterogeneous models
- Open-source requirements and models
  - Multiple use and sharing of modeling artifacts
  - Assess to domain experts & Model validation
  - Certification based on models
  - Benchmarks for tool evaluation
- Support for system integration
  - Applying model-based techniques to legacy code
  - Extracting behavioral interfaces
  - Compositional real-time scheduling framework
- Certification challenges
  - Metrics based on formal method foundations

The End.