Toward Plug and Play Medical Cyber-Physical Systems (Part 1)

Insup Lee
PRECISE Center
School of Engineering and Applied Science
University of Pennsylvania

EMSIG Autumn School
Copenhagen, Denmark
November 10, 2015

Penn Engineering

PRECISE
PENN RESEARCH IN EMBEDDED COMPUTING AND INTEGRATED SYSTEMS ENGINEERING

Medical Cyber-Physical Systems: PCA Case Studies

Insup Lee
PRECISE Center
School of Engineering and Applied Science
University of Pennsylvania

EMSIG Autumn School, Demark
November 10, 2015

Penn Engineering

PRECISE
PENN RESEARCH IN EMBEDDED COMPUTING AND INTEGRATED SYSTEMS ENGINEERING

Infusion Pumps

- Infusion pumps are medical devices that deliver fluids, (nutrients and medications) into a patient's body in a controlled manner
- Infusion pumps are used worldwide in patient care, as well as in the home

Patient-Controlled Analgesia (PCA)

- Purpose
  - Pain-relief treatment (opioids, e.g., morphine)
- Operation parameters
  - VTBI (Volume To Be Infused)
  - Basal rate
  - Bolus dose
    - additional amount of drug can be requested by the patient

Bolus-Request button

PHYSIOLOGICAL CLOSED-LOOP PCA

Closed Loop Safety Interlock

Another Use Case: PCA Monitoring

- Patients are commonly given patient-controlled analgesics after surgery
- Crucial to care, but numerous issues related to safety

A 49-year old woman underwent an uneventful operation (total abdominal hysterectomy and bilateral salpingo-oophorectomy). Postoperatively, the patient complained of severe pain and received intravenous morphine sulfate in small increments. She began receiving a continuous infusion of morphine via a patient controlled analgesia (PCA) pump. A few hours after leaving the PACU [post anethesia care unit] and arriving on the flow, she was found pale with shallow breathing, a faint pulse, and pinpoint pupils. The nursing staff called a "code", and the patient was resuscitated and transferred to the intensive care unit on a respirator. Based on family wishes, life support was withdrawn and the patient died. Review of the case implicated a PCA overdose. Delayed detection of respiratory compromise in patients undergoing PCA therapy is not uncommon because monitoring of respiratory status has been confounded by excessive nuisance alarms.

[hatcliff]

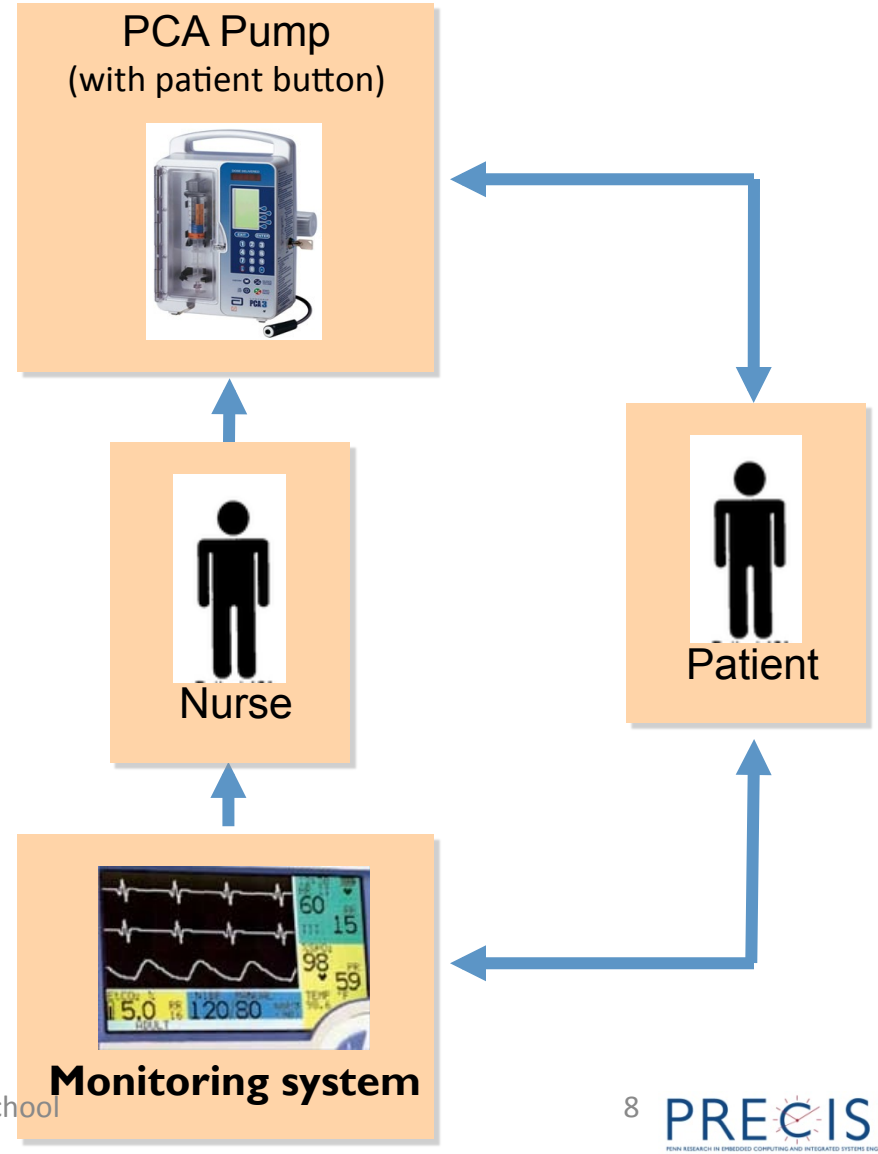
Causes of Overinfusion

- Incorrect dose
  - Varying sensitivity: hard to predict the right dose
    - Many hospitals disable basal infusion
- Excessive bolus
  - "PCA by proxy" makes the problem worse
- Free flow of medication

- Many of these causes cannot be mitigated by the device itself!

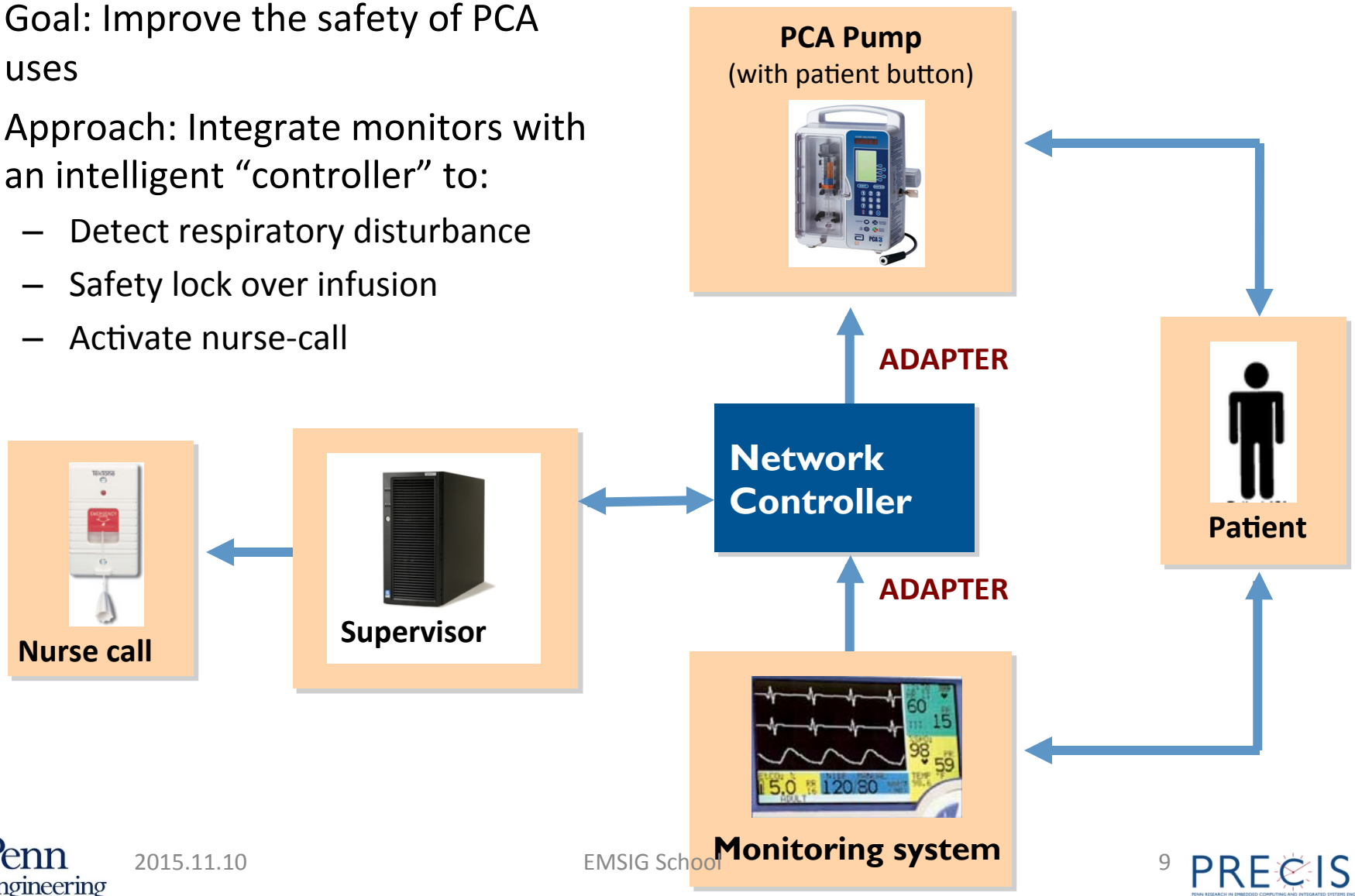
Patient Controlled Analgesia (PCA)

- Patient presses button, pump delivers opiod
- Nurse monitors patient's respiratory state.
  - If there is a problem manually intervene

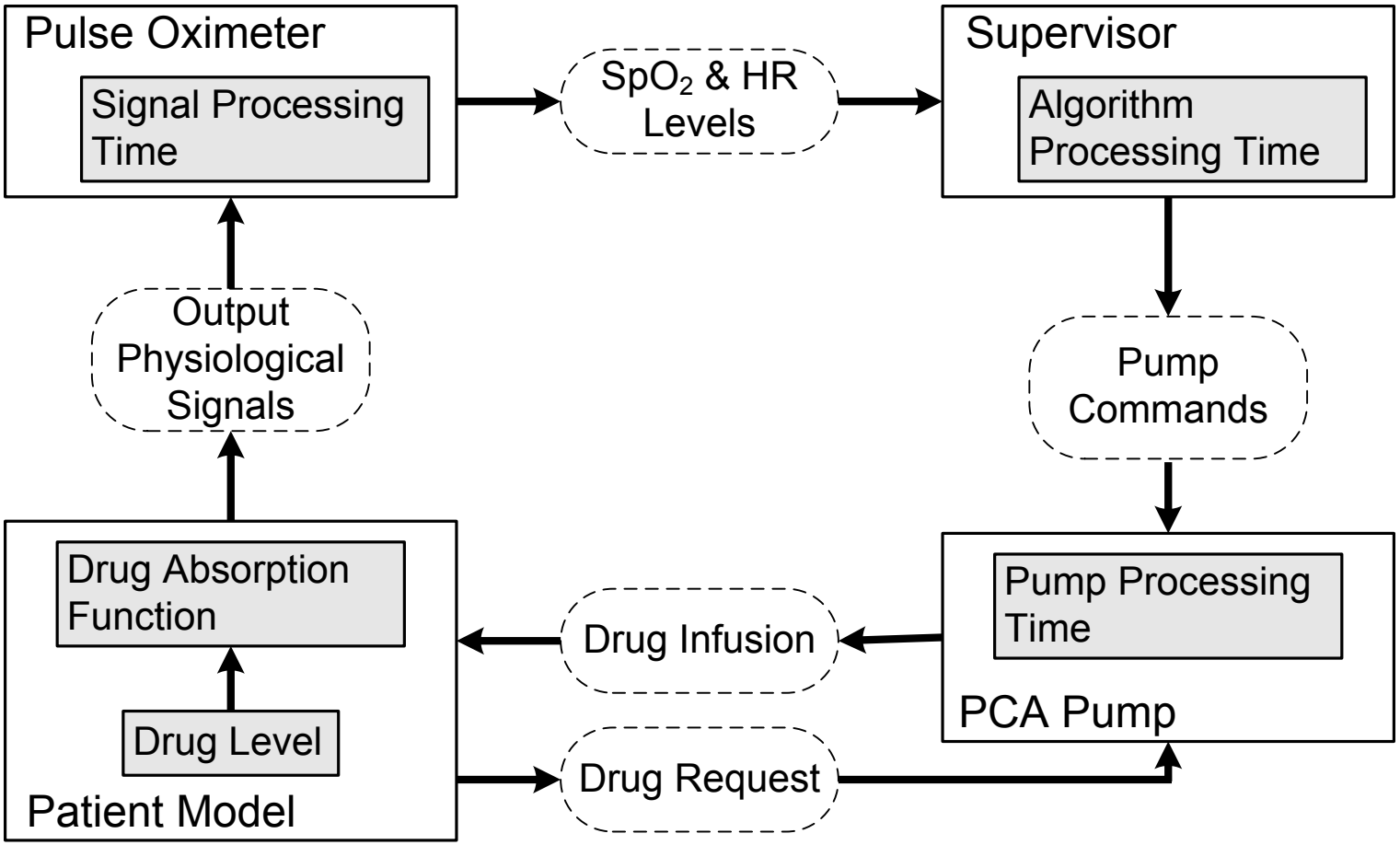


PCA Closed-loop System

- Goal: Improve the safety of PCA uses
- Approach: Integrate monitors with an intelligent "controller" to:
  - Detect respiratory disturbance
  - Safety lock over infusion
  - Activate nurse-call

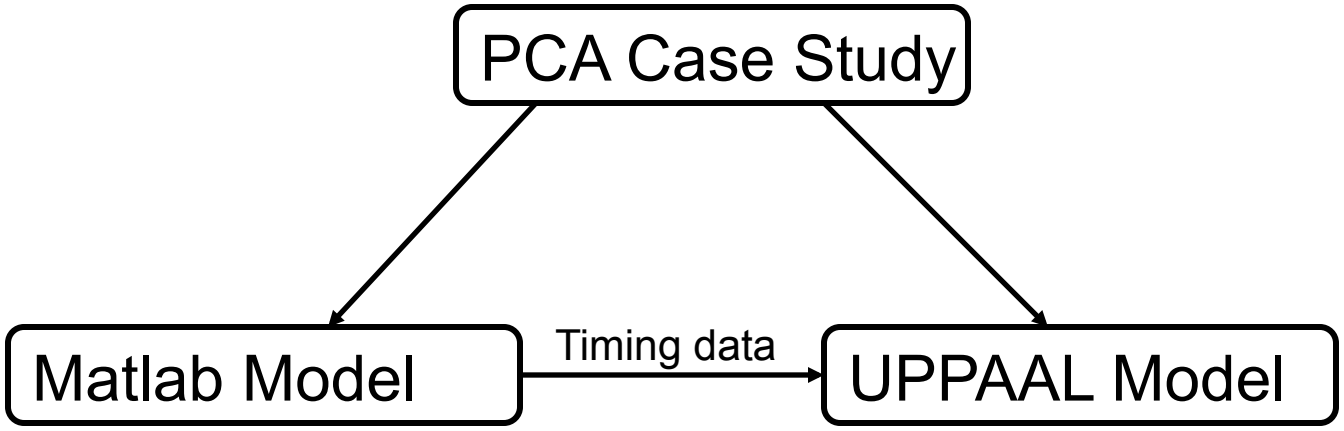


Control Loop



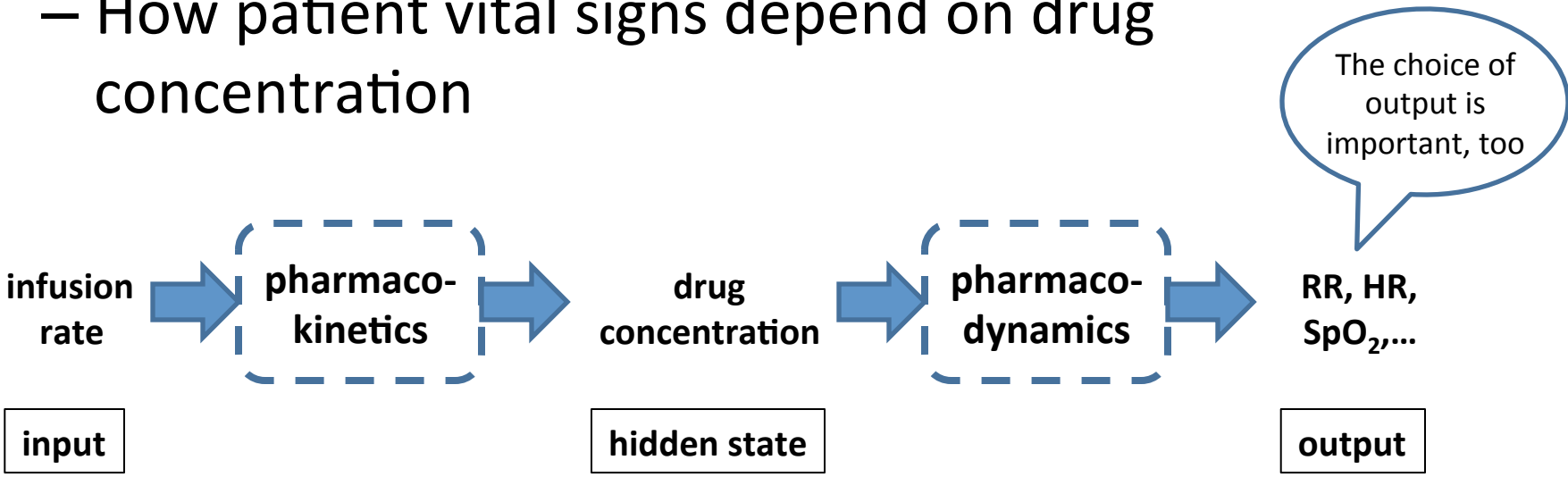
Modeling approach

- Matlab/Simulink captures detailed dynamics
- Simulation provides timing data to tune the more abstract UPPAAL model
- Formal verification in UPPAAL



Patient Modeling

- Pharmacokinetics:
  - How infusion rate affects drug concentration in the bloodstream
- Pharmacodynamics:
  - How patient vital signs depend on drug concentration



Patient Model

- Derived from pharmacokinetics model for intravenous delivery of anesthetic drugs

$$\begin{bmatrix}\dot{C}_1\\ \dot{C}_2\\ \dot{C}_3\end{bmatrix} = \underbrace{\begin{bmatrix}-(k_{12}+k_{13}+k_{10}) & k_{21} & k_{31}\\ k_{12} & -k_{12} & 0\\ k_{13} & 0 & -k_{31}\end{bmatrix}}_{\mathbf{A}} \begin{bmatrix}C_1\\ C_2\\ C_3\end{bmatrix} + \underbrace{\begin{bmatrix}\frac{1}{V_1}\\ 0\\ 0\end{bmatrix}}_{\mathbf{B}} I$$

$$dl = \underbrace{\begin{bmatrix}1 & 0 & 0\end{bmatrix}}_{\mathbf{C}} \begin{bmatrix}C_1\\ C_2\\ C_3\end{bmatrix}$$

Modeling Patient specific behavior – model with uncertain parameters

$$k_{ij} \in \left[\hat{k}_{ij} - \Delta k_{ij}, \hat{k}_{ij} + \Delta k_{ij}\right]$$

$$V_1 \in \left[\hat{V}_1 - \Delta V, \hat{V}_1 + \Delta V\right]$$

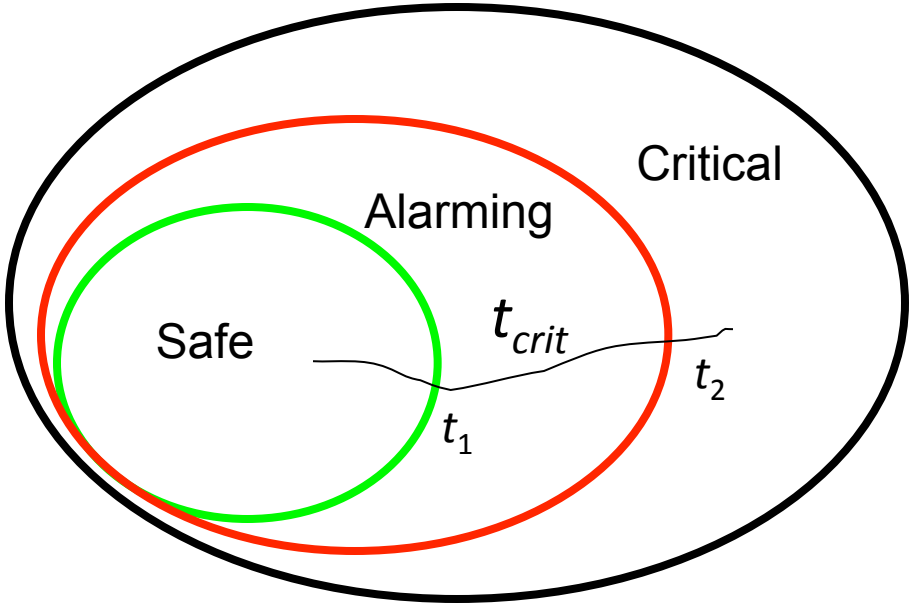
- Pharmacodynamics is much more complex
  - Not modeled in this case study

Patient Model Outputs

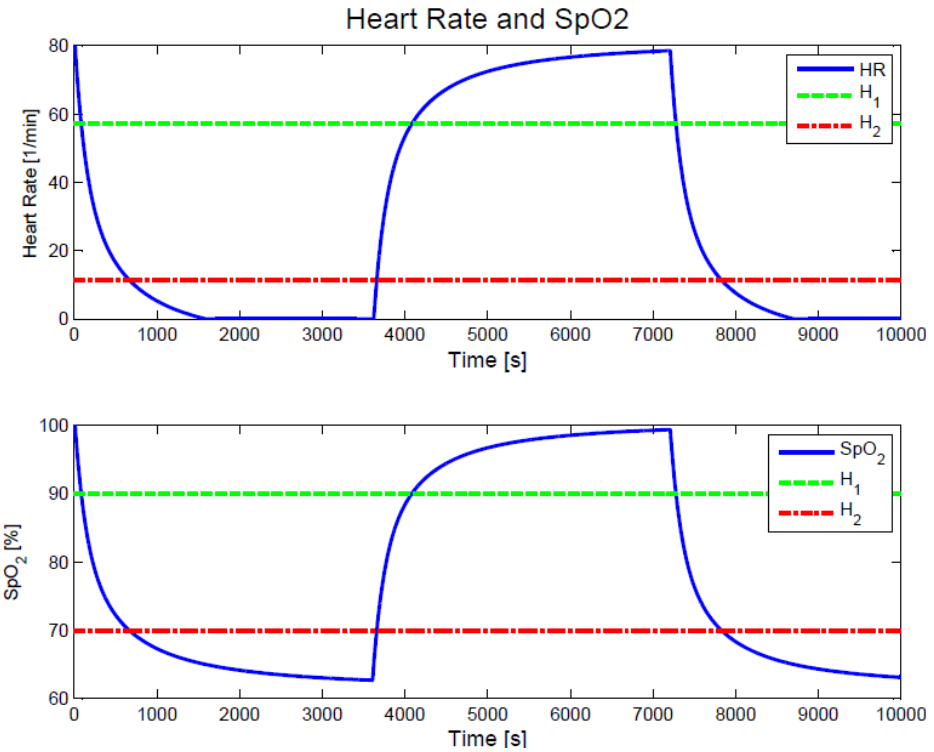
SpO_2 level and heart rate

$$c_{min} + a_1 e^{-\lambda_1 t} + a_2 e^{-\lambda_2 t} + a_3 e^{-\lambda_e t}$$

Patient Critical Regions

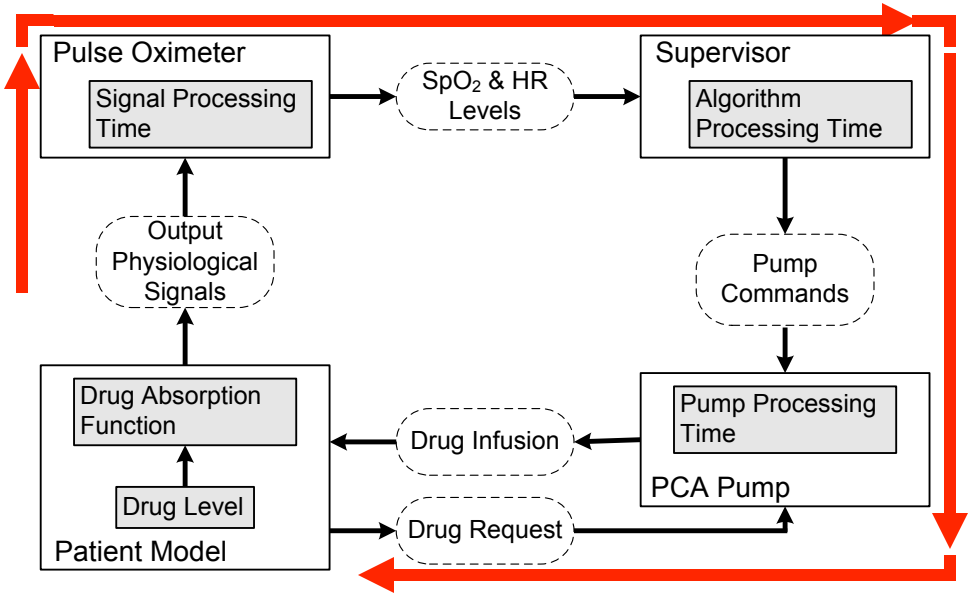


Patient Response to Drug



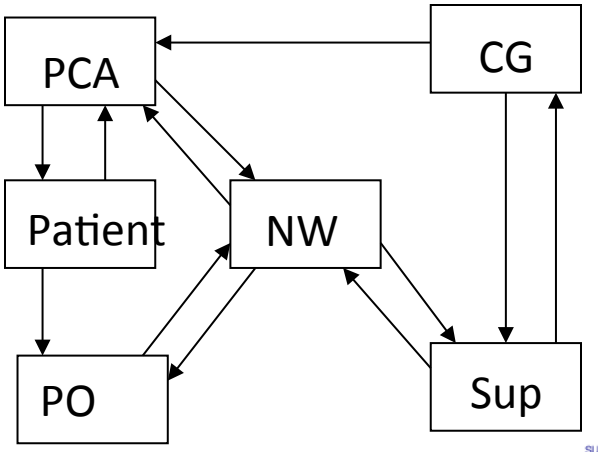
Key Safety Property

Pump stops in time if total delay <= t_{crit}



Total delay is the sum of:

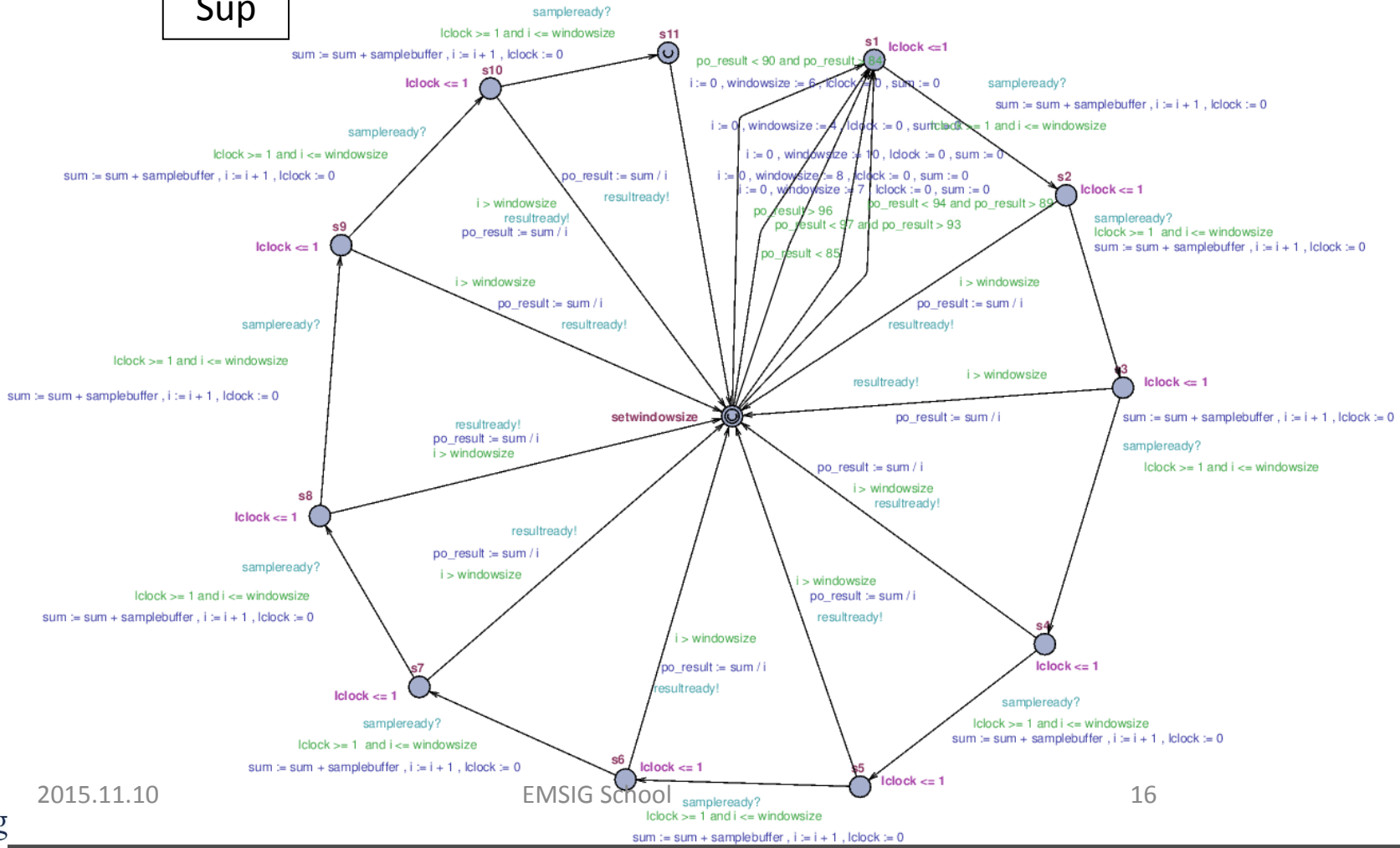
- tPOdel: worst case delay from PO (1s)
- tnet: worst case delay from network (0.5s)
- tSup: worst case delay from Supervisor (0.2s)
- tPump: worst case delay from pump (0.1s)
- tP2PO: worst case latency for pump to stop (2s)
- tcrit: shortest time the patient can spend in the alarming region before going critical



UPPAAL Model

Pulse Oximeter module:

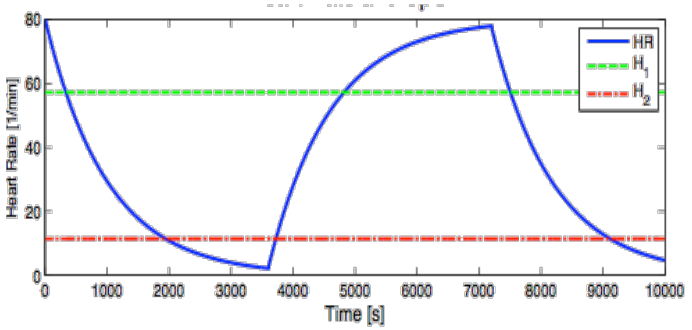
- Averages samples in a window; size of window depends on the measured value => variable delay

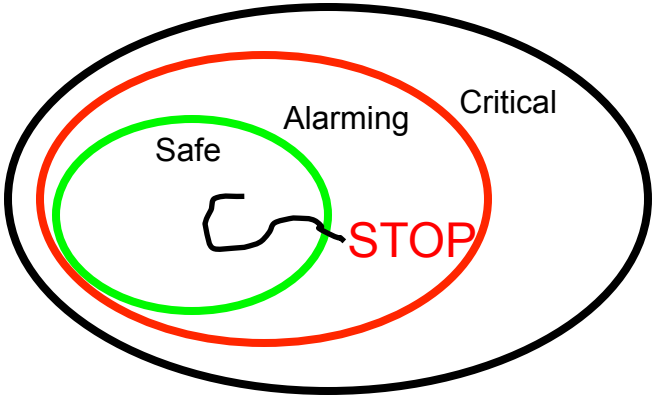


Properties verified with UPPAAL

- Once SpO2 drops below pain threshold, it eventually goes back up

  A[] (samplebuffer < pain_thresh -> A <> samplebuffer >= pain_thresh)



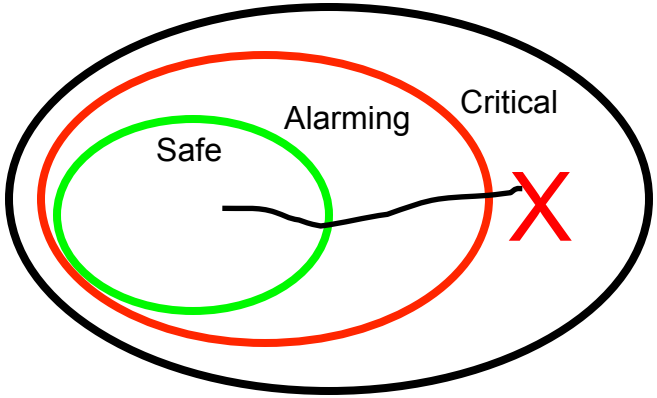


- The pump is stopped if patient enters alarming

A[] (samplebuffer < alarm_thresh ->
A<> (PCA.Rstopped V PCA.Bstopped)

- The patient can not go into the critical region

  A[] (samplebuffer >= critical)



Effects of unreliable network

- Problem:
  - The pump may not receive stop commands
- Solution:
  - Send a ticket: permission to run for a certain period of time
- Open-loop stability
  - We need to determine how long the pump can run without endangering the patient

$$\Delta t_{safe} \leq \tilde{t}_{safe} = \frac{1}{||\tilde{\mathbf{A}}||} \ln \left(\frac{|H_2^{\mathrm{SpO_2}} - h_{cur}|/\mathrm{SpO_2}_{gain}}{||\tilde{\mathbf{C}}|| \cdot \left(||\tilde{x}_0|| + \frac{||\tilde{\mathbf{B}} u_i||}{||\mathbf{A}_{min}||} \right)} + 1 \right)$$

Patient Modeling Challenge

- We have proved safety with respect to a model
- One of the risks of model-based development:
  - How good is the model?
- There usually is some agreement on the model
  - Less agreement on parameter ranges
- Narrow parameter ranges => some patients do not fit the model
- Wide parameter ranges => less effective model
  - Pump will shut down too soon for most patients
  - Tradeoff between patient safety and patient happiness?

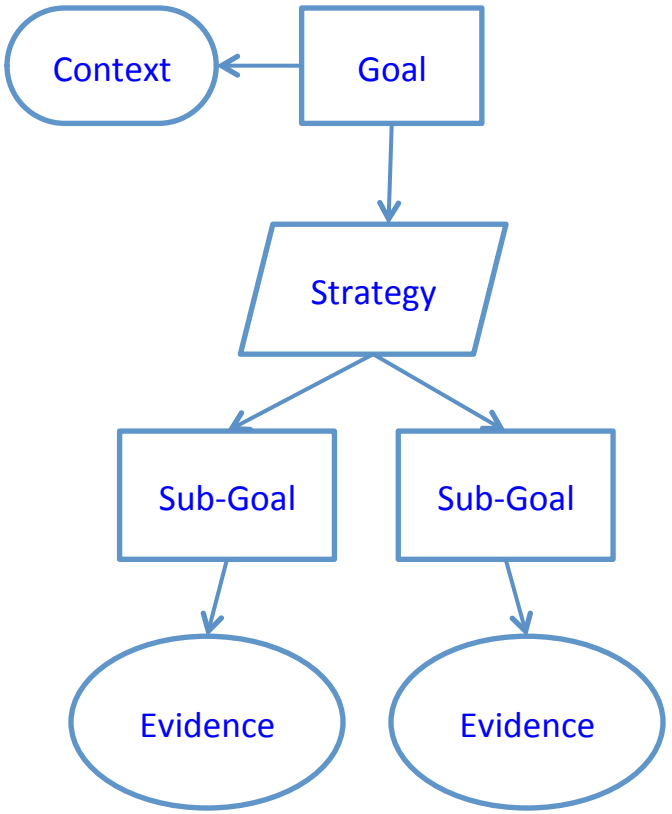
Evidence-based certification

- Suggested by: *Software for Dependable Systems: Sufficient Evidence? D. Jackson, M. Thomas, and L.I. Millett, Eds., National Academies Press, 2007*

- Evidence-based certification
  - How do we organize and evaluate evidence?
  - Assurance cases?

Assurance Cases

- To construct an assurance case we need to:
  - make an explicit set of claims about the system (safety, security, reliability, performance, etc.)
  - produce the supporting evidence
  - provide a set of arguments that link the claims to the evidence
  - make clear the context, including assumptions and judgments underlying the arguments
- Safety case is a special kind:
  - Claims are limited to safety

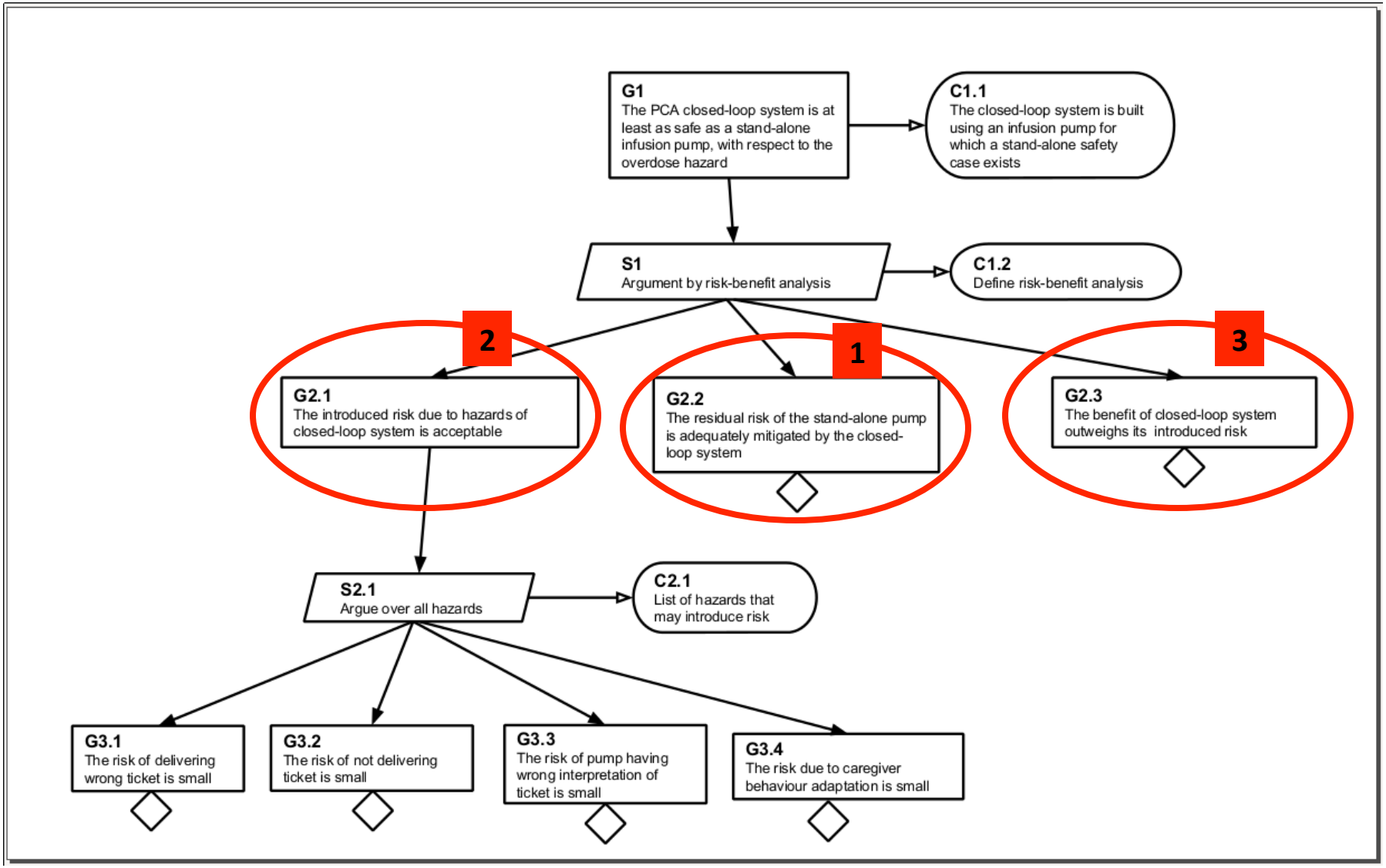


> Argument without Evidence is unfounded
> Evidence without Argument is unexplained
>
> - Time Kelley, 2008

Argument Strategy for Closed-Loop PCA

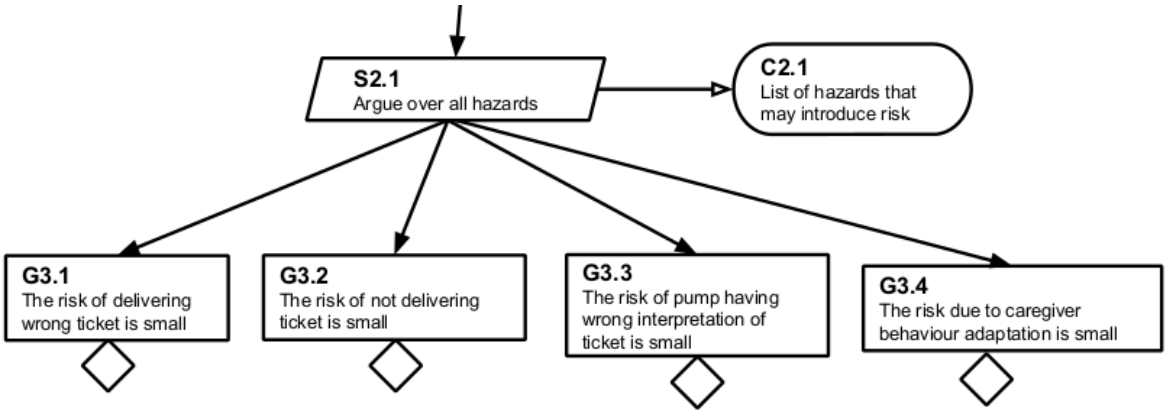
- Identify residual risk
  - Not all hazards are eliminated
- Argue about added hazard mitigation
- Identify added risk
  - New hazards
  - New sources of existing hazards
- Argue that reduction of residual risk outweighs new risks

Safety Case: Top Level

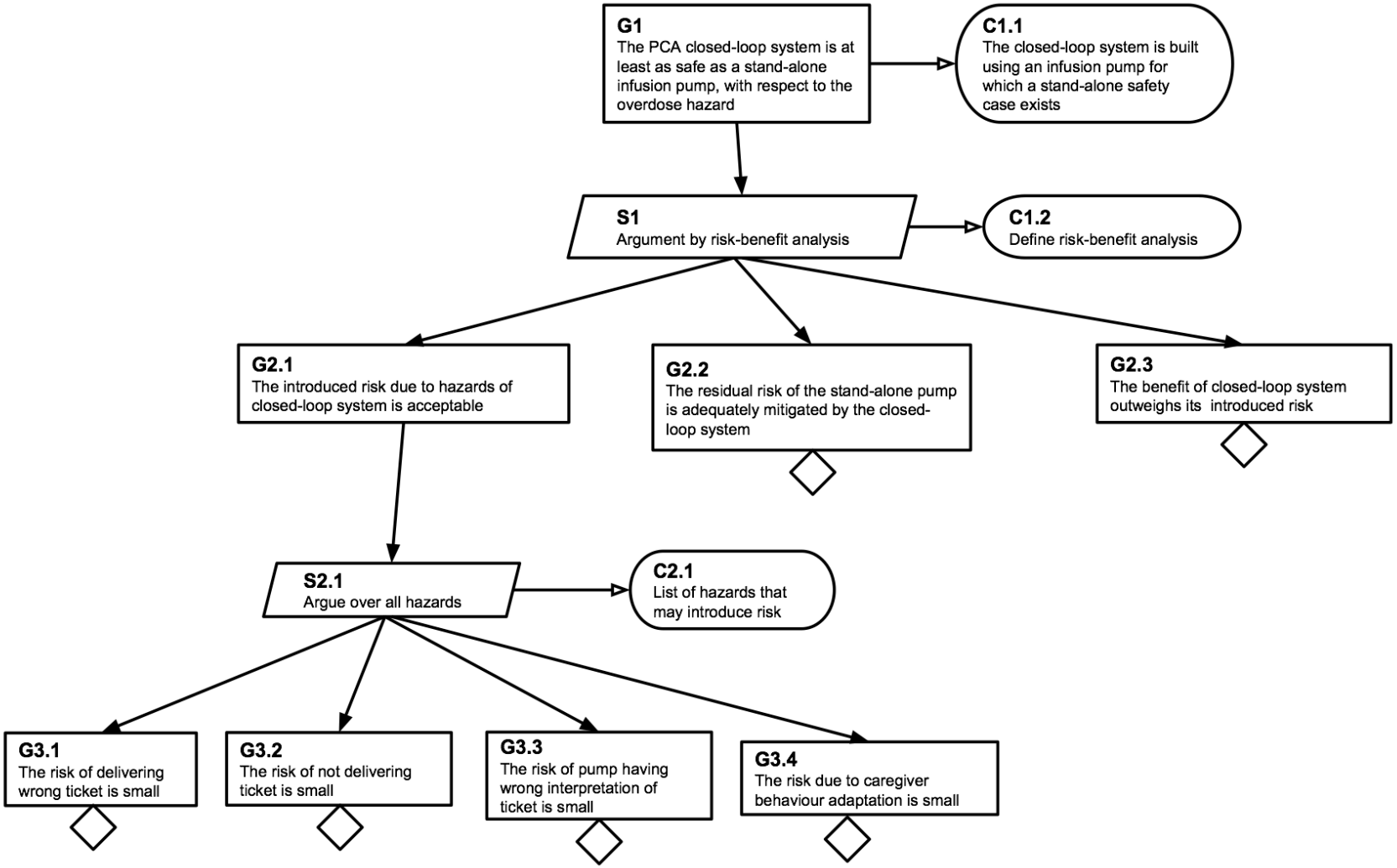


New Risks

- Sensor failures
- Controller failures
- Network failures
- Pump failures
- Human factors



The Assurance Case Structure of Closed-loop PCA System



MODEL-BASED DEVELOPMENT OF GENERIC PCA (PATIENT CONTROLLED ANALGESIC)

Infusion Pump Safety

- During 2005 and 2009, FDA received approximately 56,000 reports of adverse events associated with the use of infusion pumps
  - 1% deaths, 34% serious injuries
  - 87 infusion pump recalls to address safety problems
- The most common types of problems
  - **Software Defect**
  - User Interface Issues
  - Mechanical or Electrical Failure

U.S. Food and Drug Administration, Center for Devices and Radiological Health. White Paper: Infusion Pump Improvement Initiative, April 2010

PCA Hazards

- Overinfusion
  - Opioids can cause respiratory distress
    - the patient can stop breathing
- Air in line
  - Air bubbles entering blood stream with medication
- Underinfusion
  - Can limit effectiveness of pain management

Hazards -> Safety Requirements

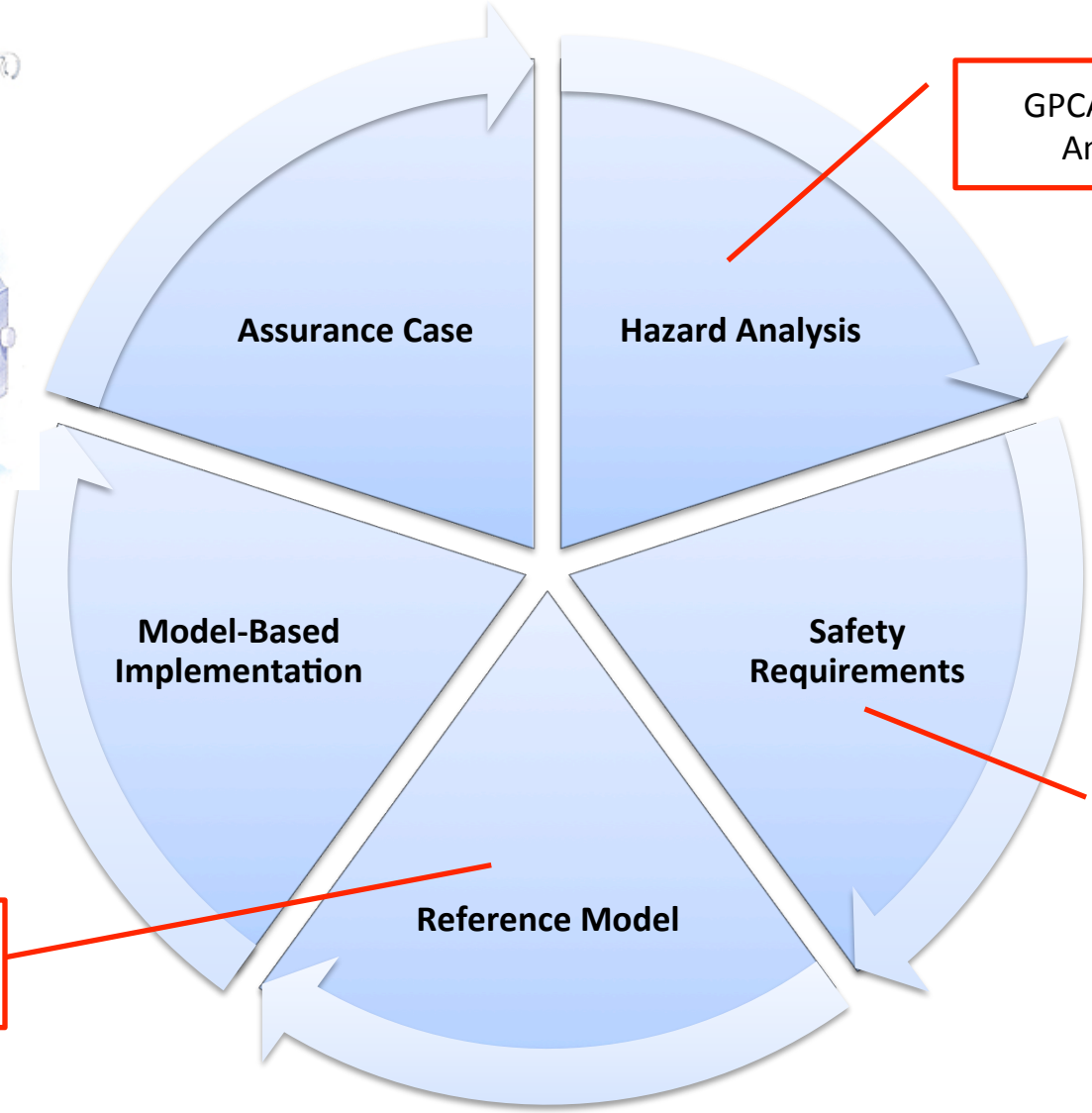
- Prescribed dose cannot be exceeded
- Prescribed rate is closely adhered to
- When an alarm is raised, the pump should be stopped quickly enough
- Minimum interval between boluses should be enforced

High Assurance Development

- Use formal methods for modeling, verification, and code generation
- GPCA (Generic PCA) project
  - Develop a set of artifacts
    - Design documents, models, verification results, code, etc.
  - Community resource to apply and compare various development methods
  - Inform FDA on model-based development practices
  - http://rtg.cis.upenn.edu/medical/gpca/gpca.html

Generic PCA (GPCA) Project

PCA Infusion Pump



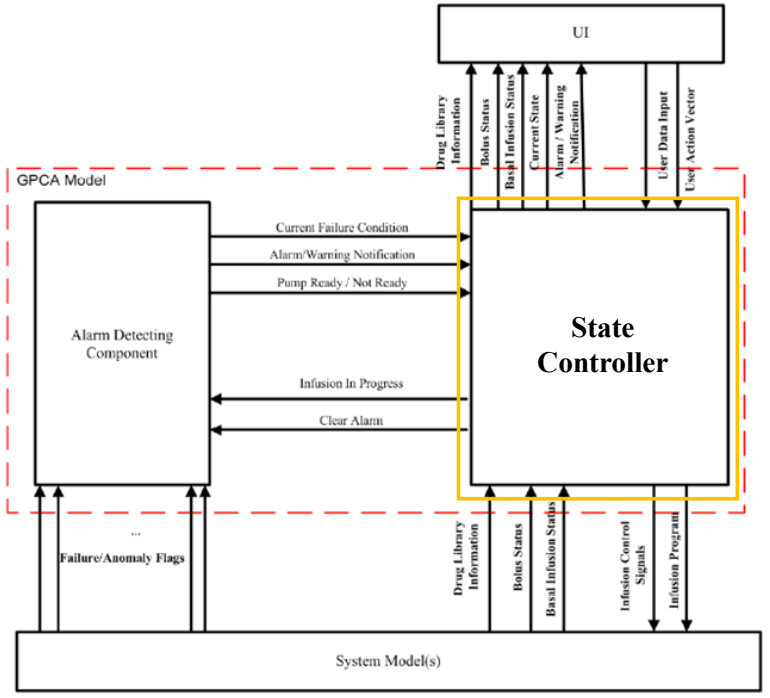
GPCA Hazard Analysis

GPCA Safety Requirements

GPCA Model

FDA's GPCA Model

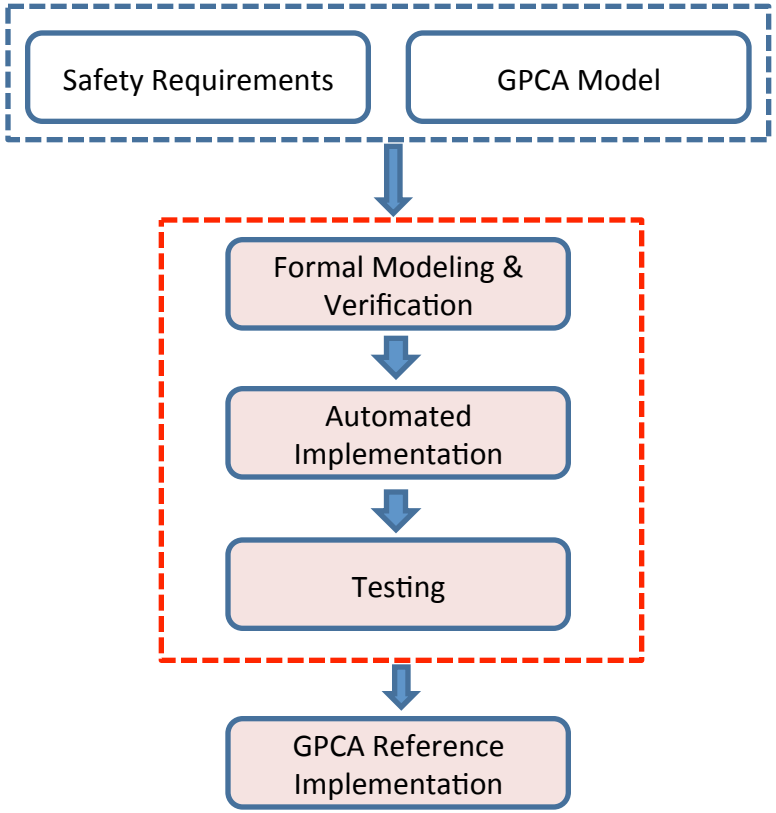
- An abstract representation of software used in a typical PCA infusion pump.
- The model is built in Simulink and Stateflow.
- State Controller
  - Describes a drug administration process such as parameter setting and bolus request.
- Alarm Detecting Component
  - Check hardware conditions and process alarm on any hardware failure.
- GPCA Environment
  - User Interface
  - System model
    - The GPCA model interacts with pump hardware such as motor and sensors through the System Model.



The System Architecture of GPCA Model

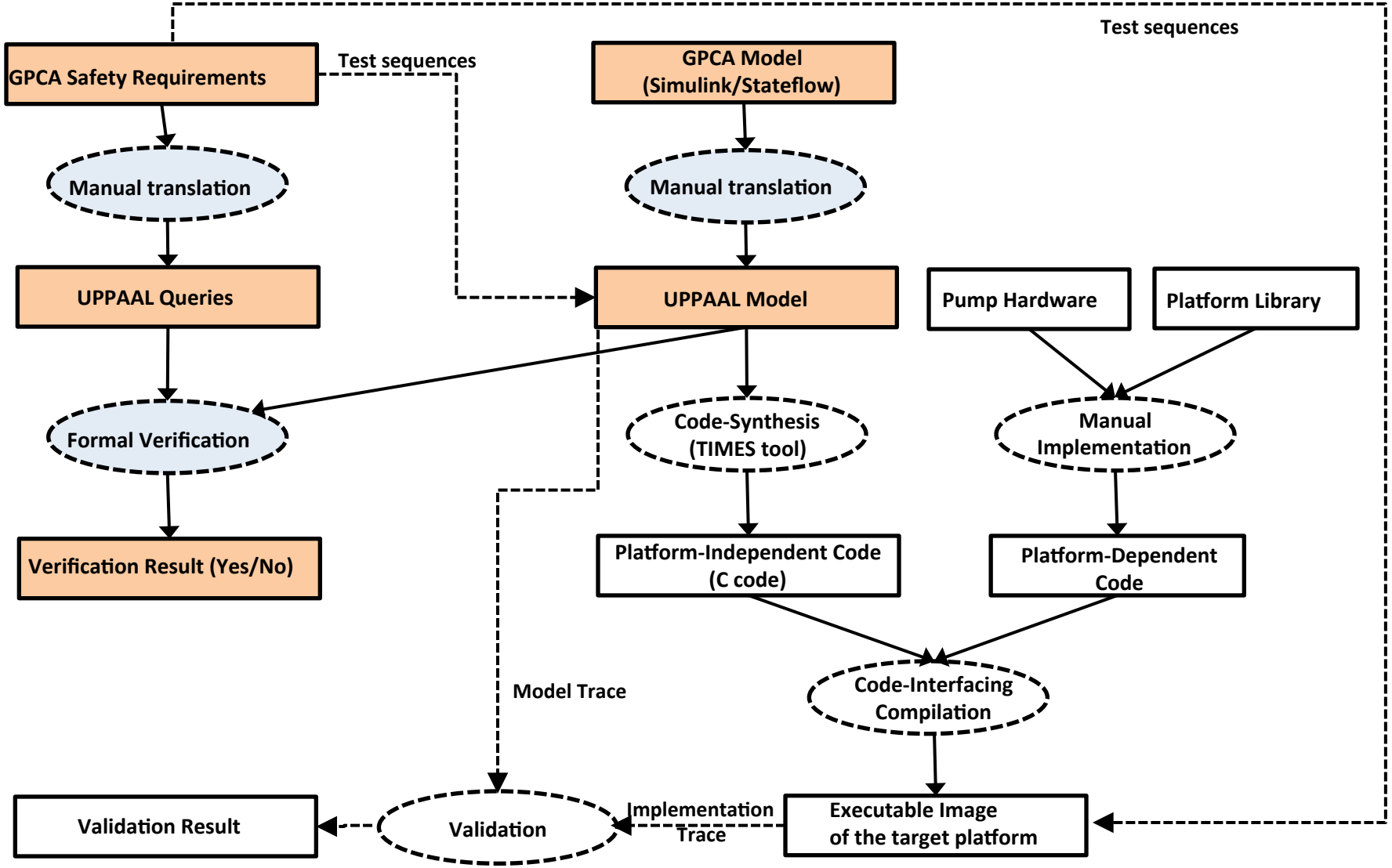
GPCA reference implementation

- FDA initiated
  - GPCA Safety Requirements
  - GPCA Model (Simulink/Stateflow)
- Goal: Develop a GPCA reference implementation
- Provide evidence that the implementation satisfies the safety requirements
  - Property verification
  - Code synthesis
- Organize evidence for certification
  - Assurance cases for safety
  - Confidence cases
- All artifacts to be available as open source
  - http://rtg.cis.upenn.edu/gip.php3

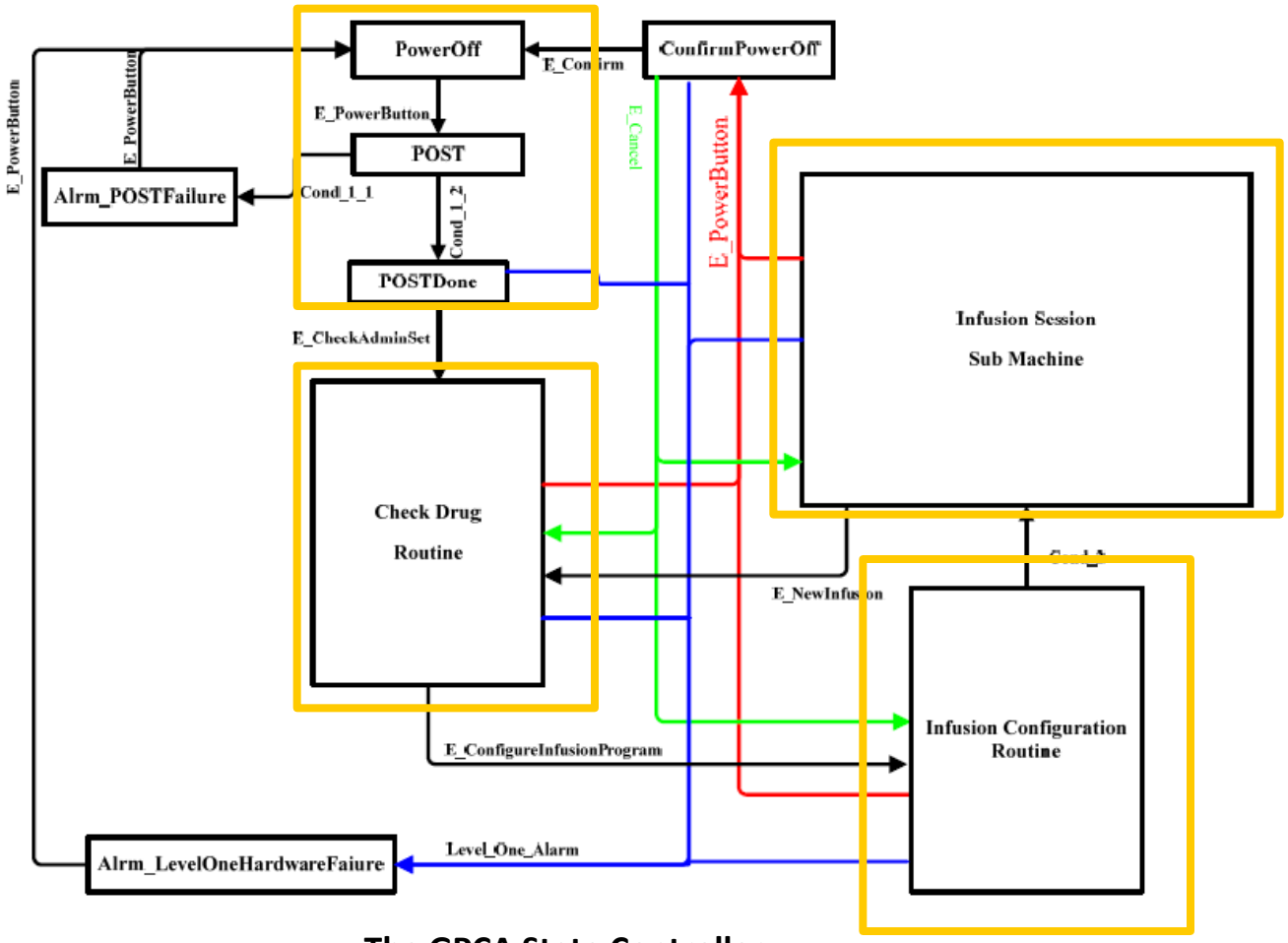


Model-Based Development of GPCA Reference Implementation

Phase 1: Formal Verification



Formalization of the FDA's GPCA model

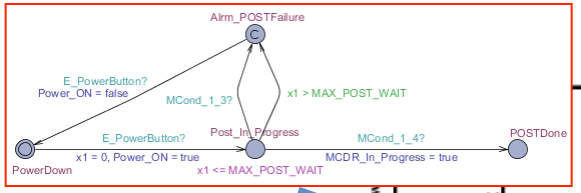


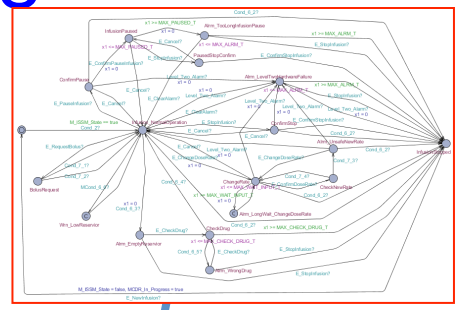
The GPCA State Controller

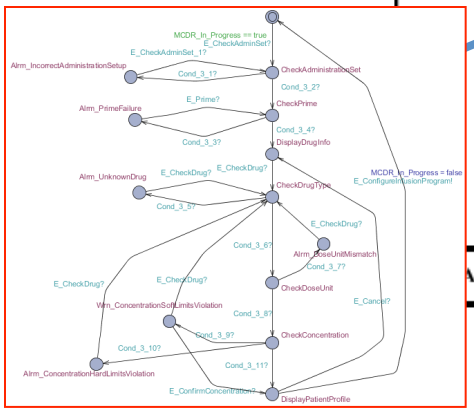
Penn Engineering

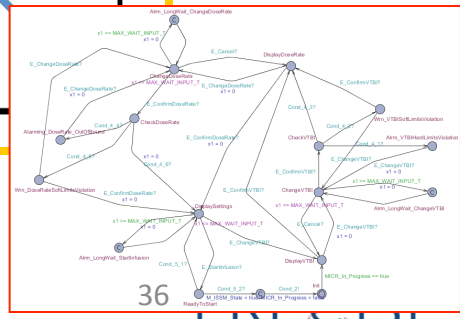
PRECISE

Formalization of the FDA's GPCA model





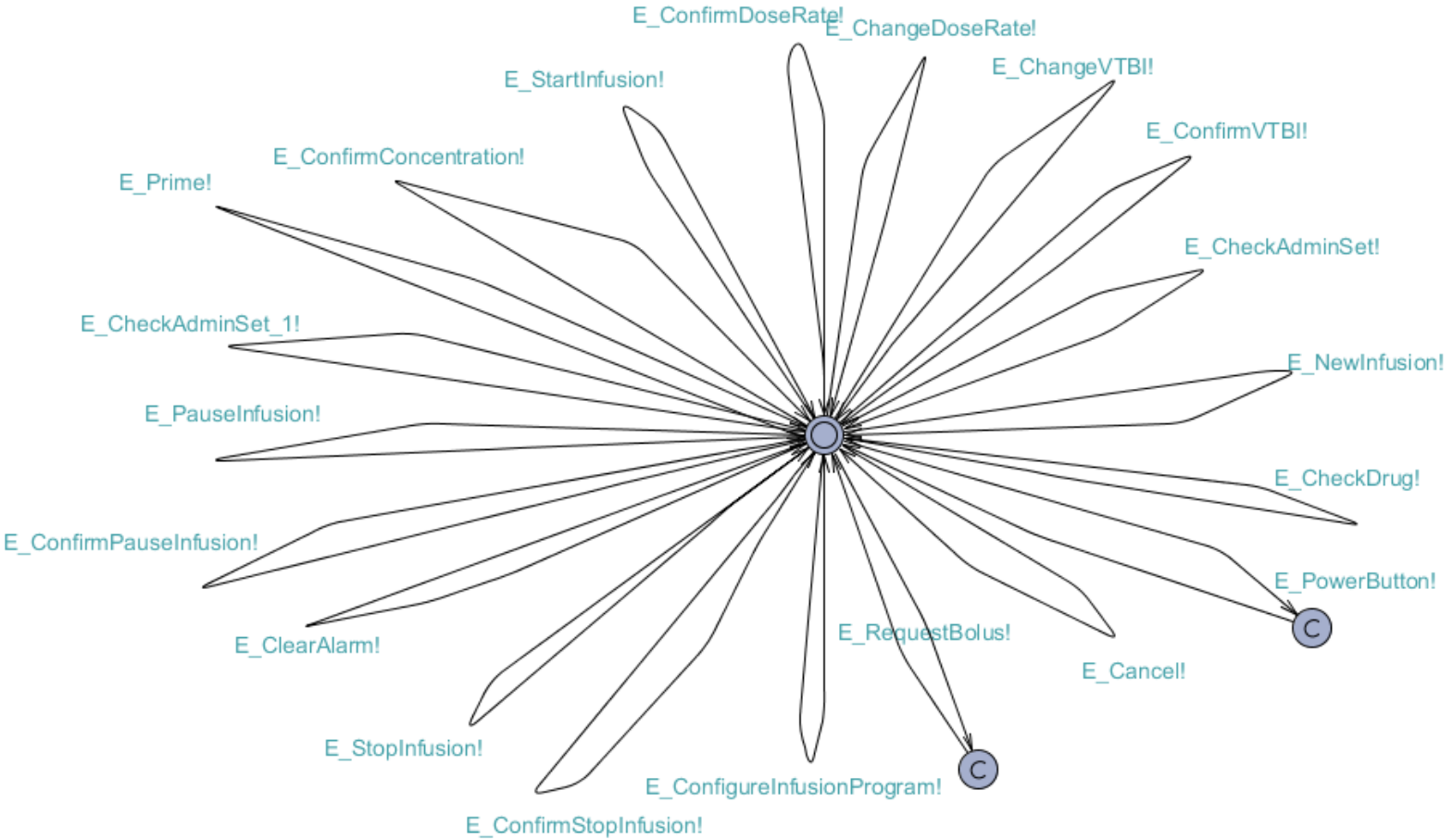




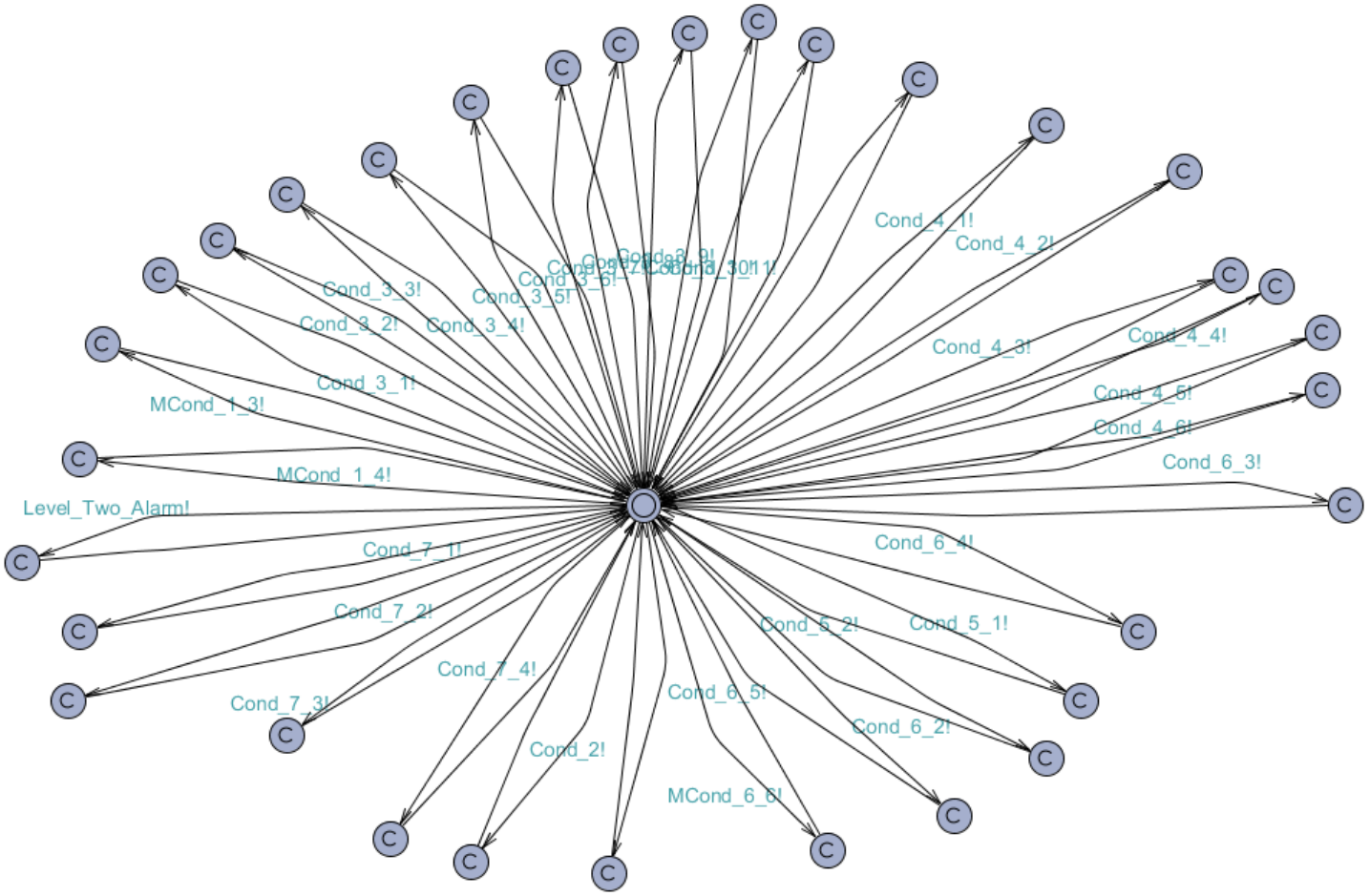
PowerOff
ConfirmPowerOff
E_Confirm
E_PowerButton
POST
Cond_1_1
Cond_1_2
POSTDone
E_Cancel
E_PowerButton
E_CheckAdminSet
Infusion Session
Sub Machine
Check Drug
Routine
E_NewInfusion
Infusion Configuration
Routine
E_ConfigureInfusionProgram
Alrm_LevelOneHardwareFaiure
Level_One_Alarm

The GPCA State Controller

Environment: User Actions



Environment : Hardware Conditions



Cond-6-3 implies “An infusion error *Empty Reservoir* is detected during the ongoing infusion process”

Formalization of the Safety Requirements

- Not all safety requirements can be translated into temporal logic formula.
- Categorization of the safety requirements.

  Category 1) A safety requirement can be formalized and verified in the UPPAAL model. (~20 out of 97 requirements)
  - No bolus dose shall be possible during the POST
  - The pump shall issue an alert if paused for more than t minutes

  Category 2) A safety requirement can be formalized, but the GPCA model needs additional information to verify it. (~23 out of 97 requirements)
  - If the suspend occurs due to a fault condition, the pump shall be stopped immediately without completing the current pump stroke.

Formalization of the Safety Requirements

- Not all safety requirements can be translated into temporal logic formula.
- Categorization of the safety requirements.

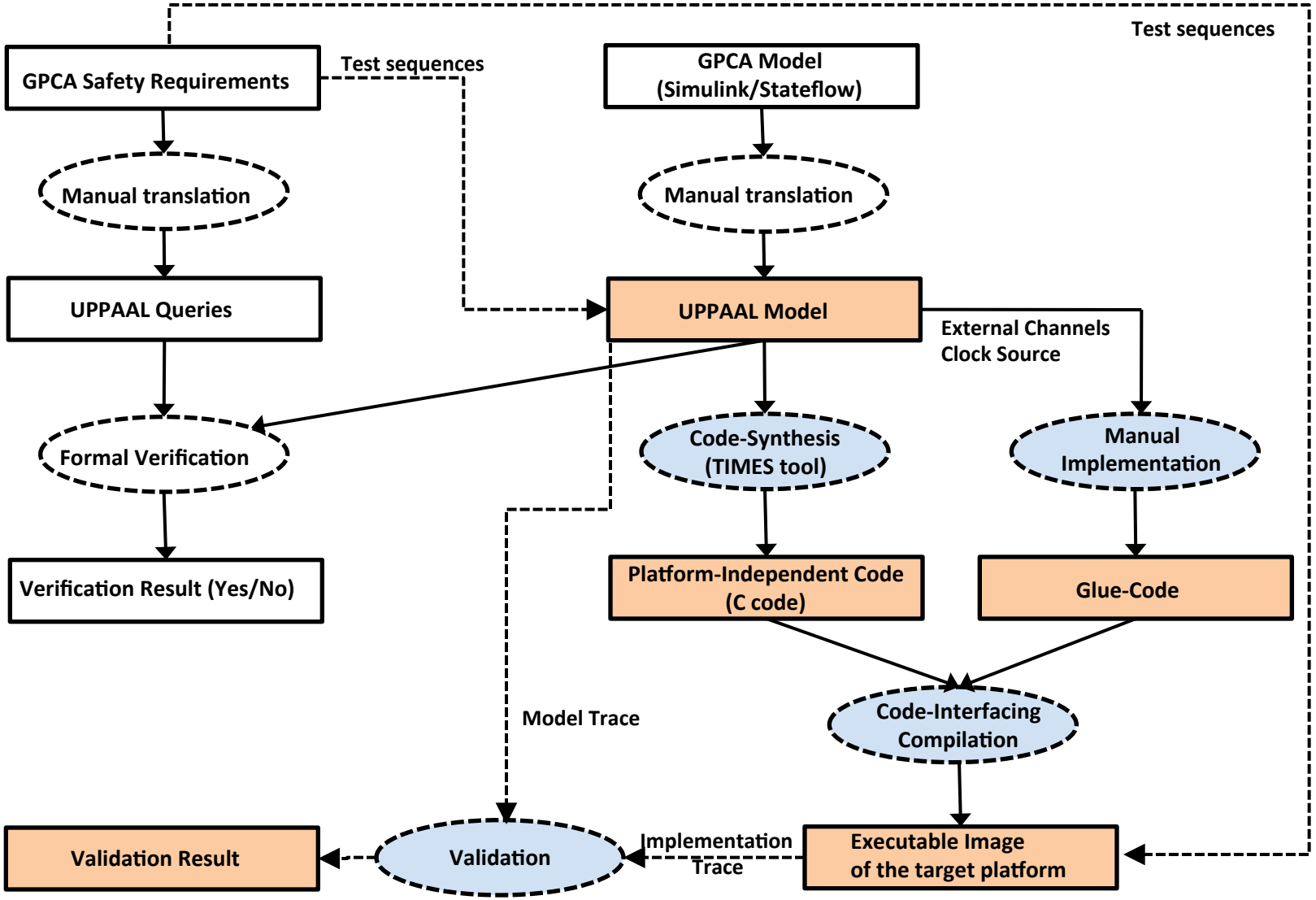
  Category 1) A safety requirement can be formalized and verified in the UPPAAL model. (~20 out of 97 requirements)
  - No bolus dose shall be possible during the POST
  - The pump shall issue an alert if paused for more than t minutes

  Category 2) A safety requirement can be formalized, but the GPCA model needs additional information to verify it. (~23 out of 97 requirements)
  - If the suspend occurs due to a fault condition, the pump shall be stopped immediately without completing the current pump stroke.

  Category 3) A safety requirement cannot be formalized, but can be validated at the implementation level. (~31 out of 97 requirements)
  - The flow rate for the bolus dose shall be programmable.

  Category 4) A safety requirement cannot be formalized because the statement is too vague or related to the environment of the GPCA model. (~23 out of 97 requirements)
  - Flow discontinuity at low flows should be minimal ("minimal" is not clear).
  - A key that is depressed shall not be identified as a distinct key press for a period of t seconds (related to UI).

Phase 2: Implementation



Code Synthesis

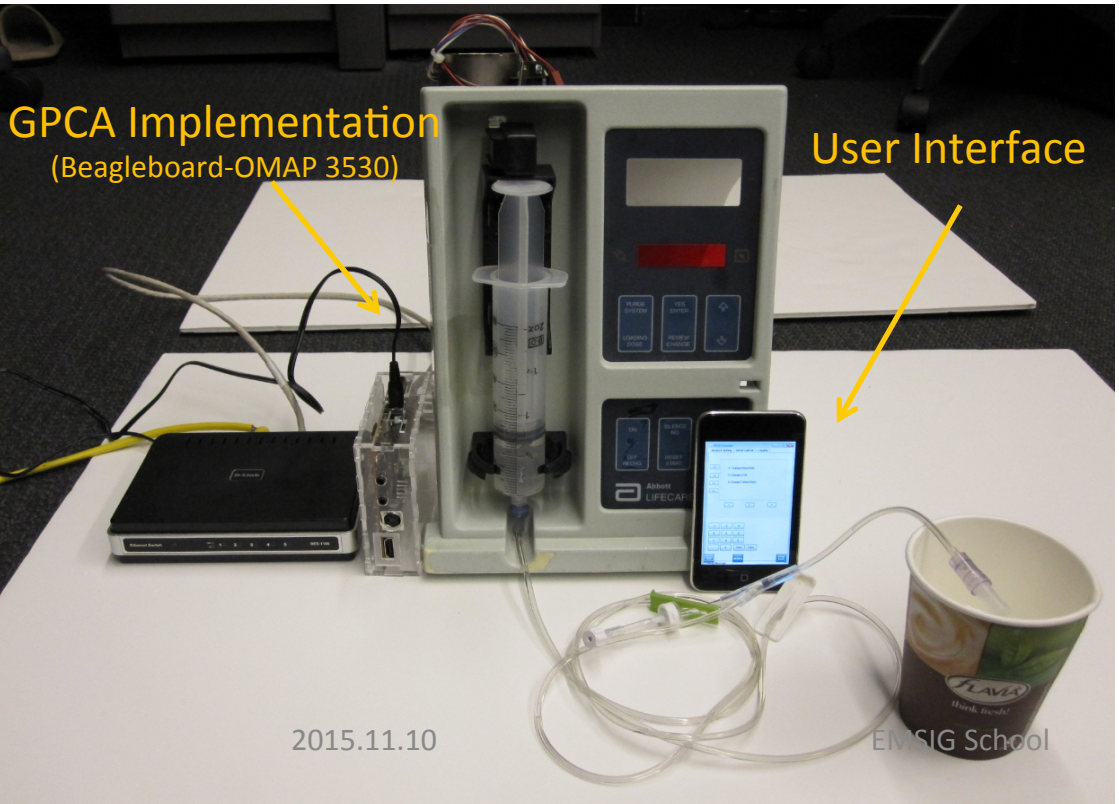
- Advantages of automated implementation
  - An automated implementation improves the quality of embedded software by preserving the properties of model verification.
- Practical obstacles in automated implementation
  - There is a gap between abstract model and implementation

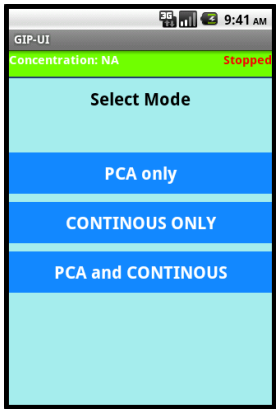
Types of the GPCA Pump Source Code

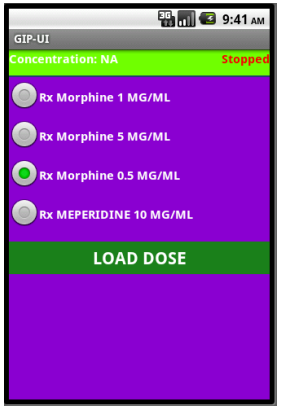
1. GPCA model code (Platform-independent)
   - GPCA model is synthesized into C-code using TIMES tool.
   - This code implements control-flow of the GPCA model depending on user-action and hardware conditions.
2. Glue code to interface to the target platform (Platform-dependent)
   - Clock implementation using the target platform APIs.
   - Environmental interface (for user and GPCA hardware).
3. Code for abstracted functionalities
   - Pump-motor driving code on transition to Infusion-Normal-Operation to inject drug to patient (e.g., providing electrical signal to the pump motor)
   - Code for updating dose rate on ChangeDoseRate state (e.g., maintaining variables for dose rate that is updated by user request)

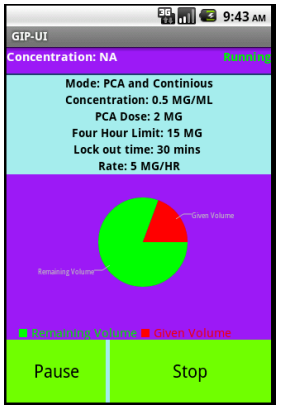
GPCA Project

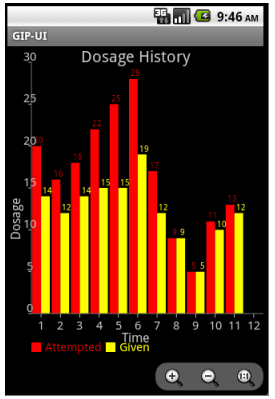
- Open platform for medical device research
- Support a variety of pump hardware

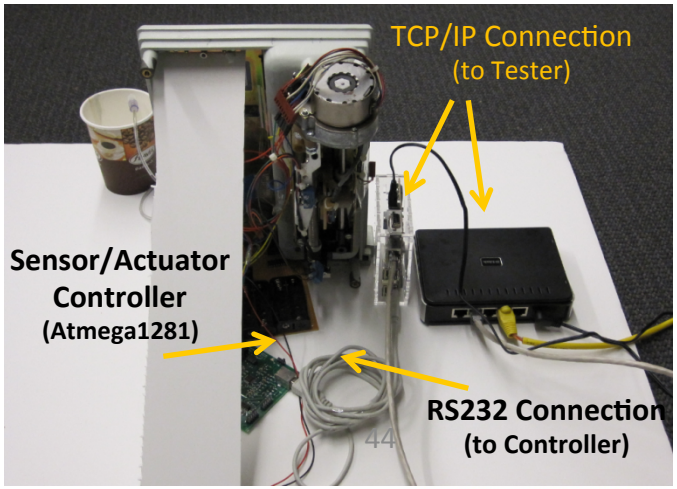




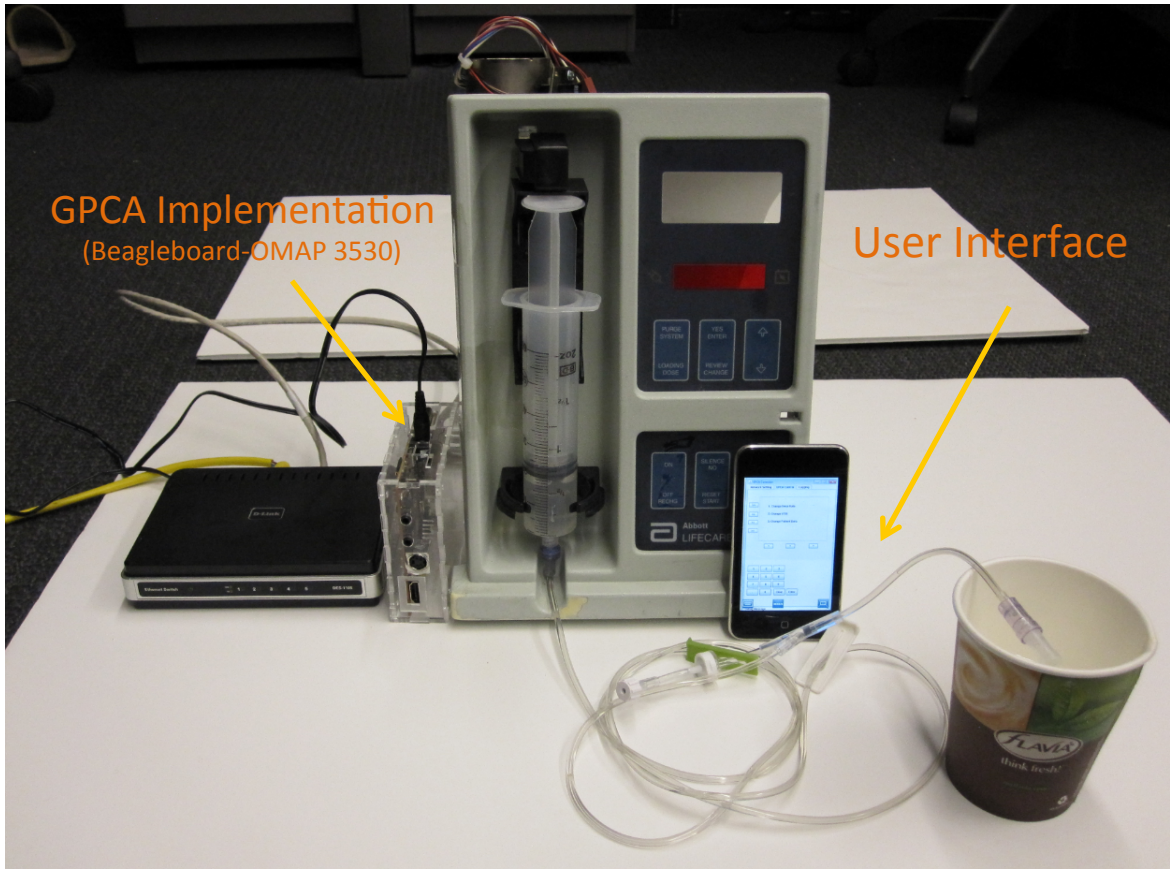


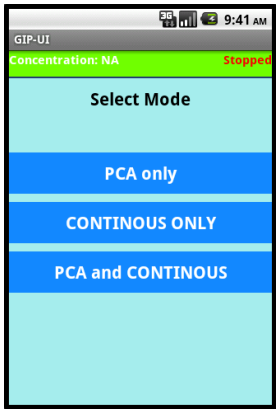


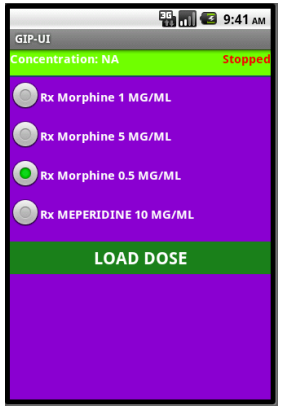


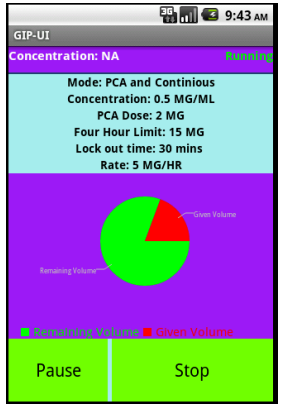


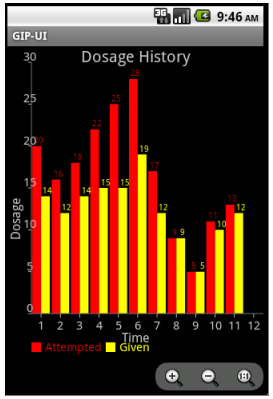
GPCA Implementation Testbed

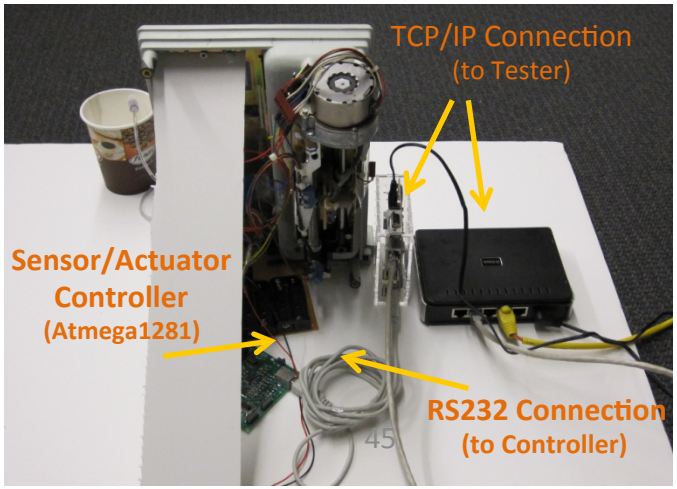












•We note that the Android UI design is motivated from CADD –Solis Ambulatory Infusion System. The functionalities are instantiated from the GPCA model.

The Cross-Platform Software Modeling: PCA Infusion Pump Systems

- PCA Infusion Pump Systems
  - Inject drugs by pressing the bolus request button.
  - Used for the pain-treatment.
- The cross-platform model
  - A model captures the common behavior for infusion operations.

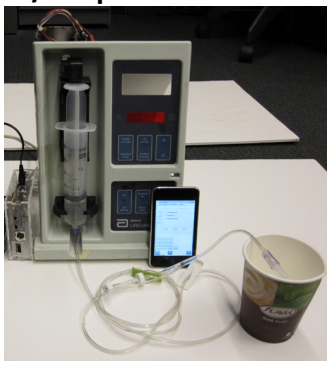
** Safety-Assured Development of the GPCA Infusion Pump Software, Kim et al., EMSOFT2011*

- Heterogeneous Target Platforms
  - Different platforms may have different way of implementing the abstracted behavior.

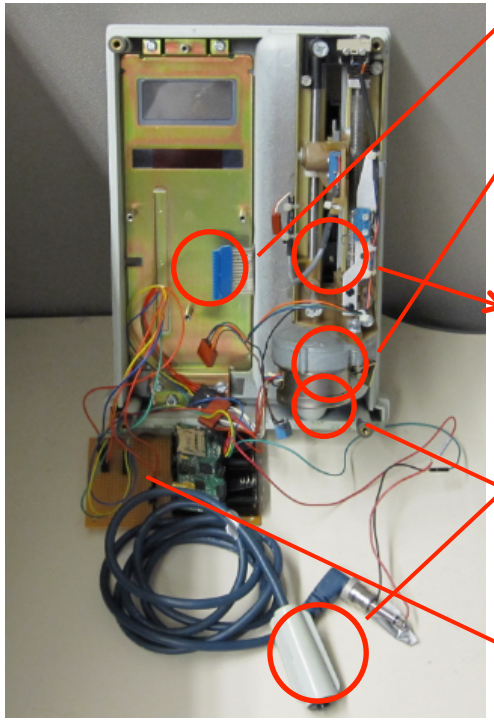
The Cross-Platform MBD

- Comparison of Infusion Pump Platforms

Abbott/Hospira Lifecare 4100 PCA PLUS II



Baxter PCA II Syringe Pump



(1) UI Data Port

(2) Pump motor (Stepper or DC)

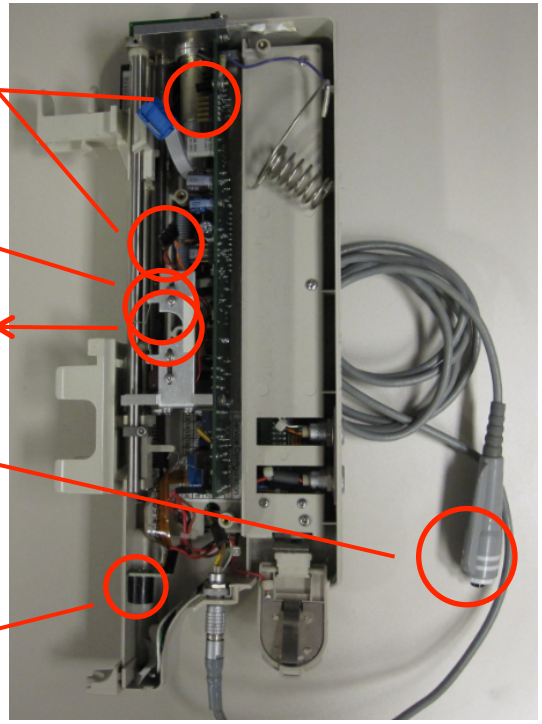
(3) Low Reservoir Detection switch

(4) Empty Reservoir Detection switch

(5) Patient Pendant

(6) Optical switch (for precise flow-control)

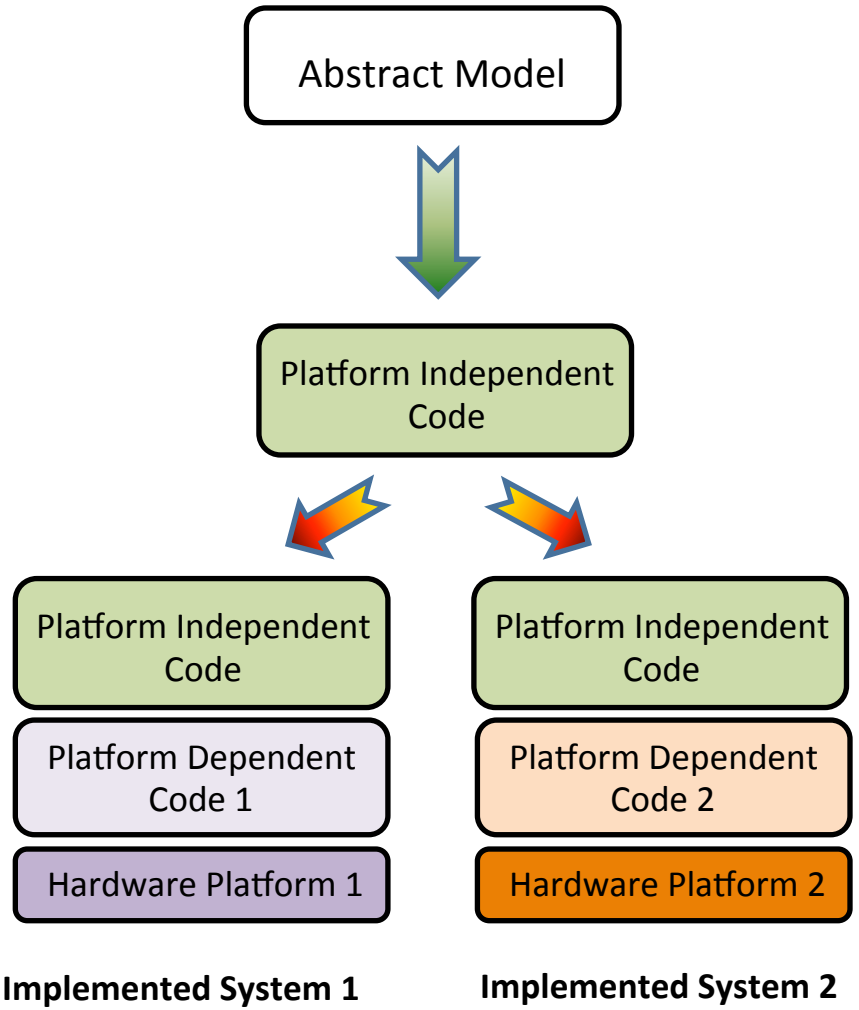
(7) Buzzer (for alarm)



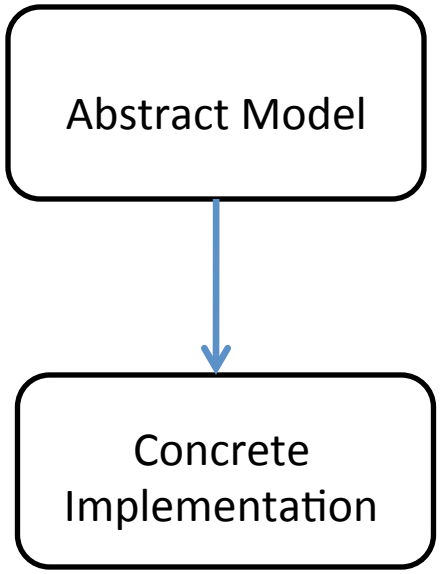
The Cross-Platform Software Modeling

- A model encodes the platform-independent behavior of the system
  - Independent from a particular platform.

- The implemented systems consist of two types of code
  1. Platform Independent code
  2. Platform Dependent code

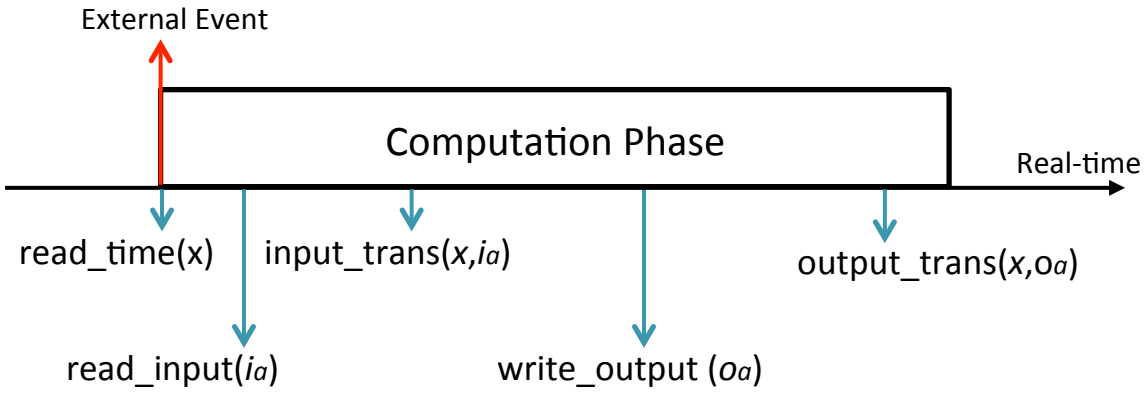


Gap: Synchrony Assumption in Modeling



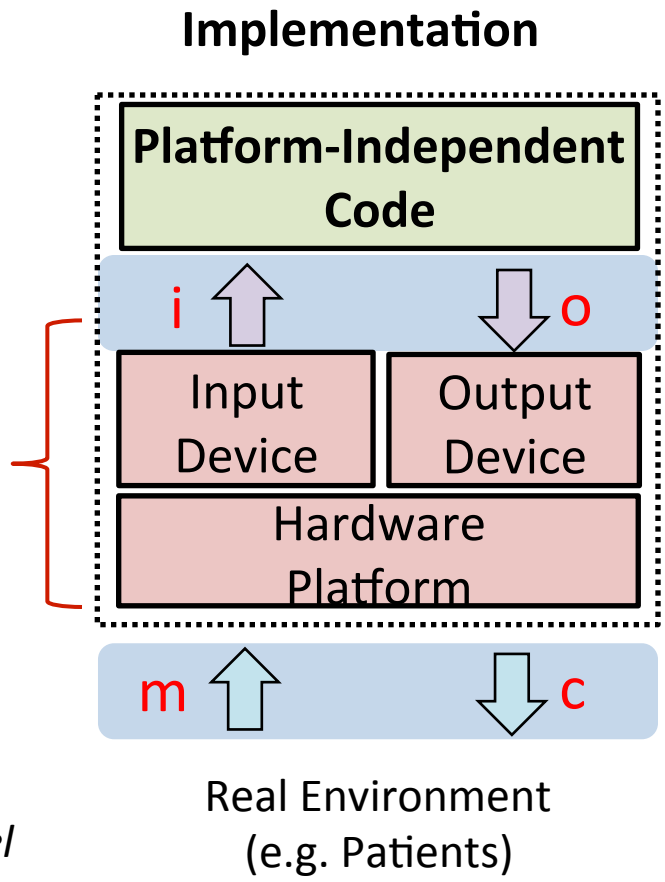
- Synchrony Assumption
  - The program reacts to external events instantaneously.
  - Pros: greatly simplifies formal analysis of real-time systems.
  - Cons: real systems cannot guarantee the assumption due to computation delay.

1. Read Time
2. Read Input
3. Input-Transition
4. Write Output
5. Output-Transition



Platform and System Boundaries

- A *platform* is a collection of hardware and software components that are *heterogeneous* across different systems
- *MC*-boundary
  - **M**onitored variable
    - (e.g.) Bolus button (sensor) status
  - **C**ontrolled variable
    - (e.g.) Pump motor (actuator) status
- *IO*-boundary
  - **I**nput variable
    - (e.g.) Boolean value abstracting sensor input
  - **O**utput variable
    - (e.g.) Integer value abstracting actuator output

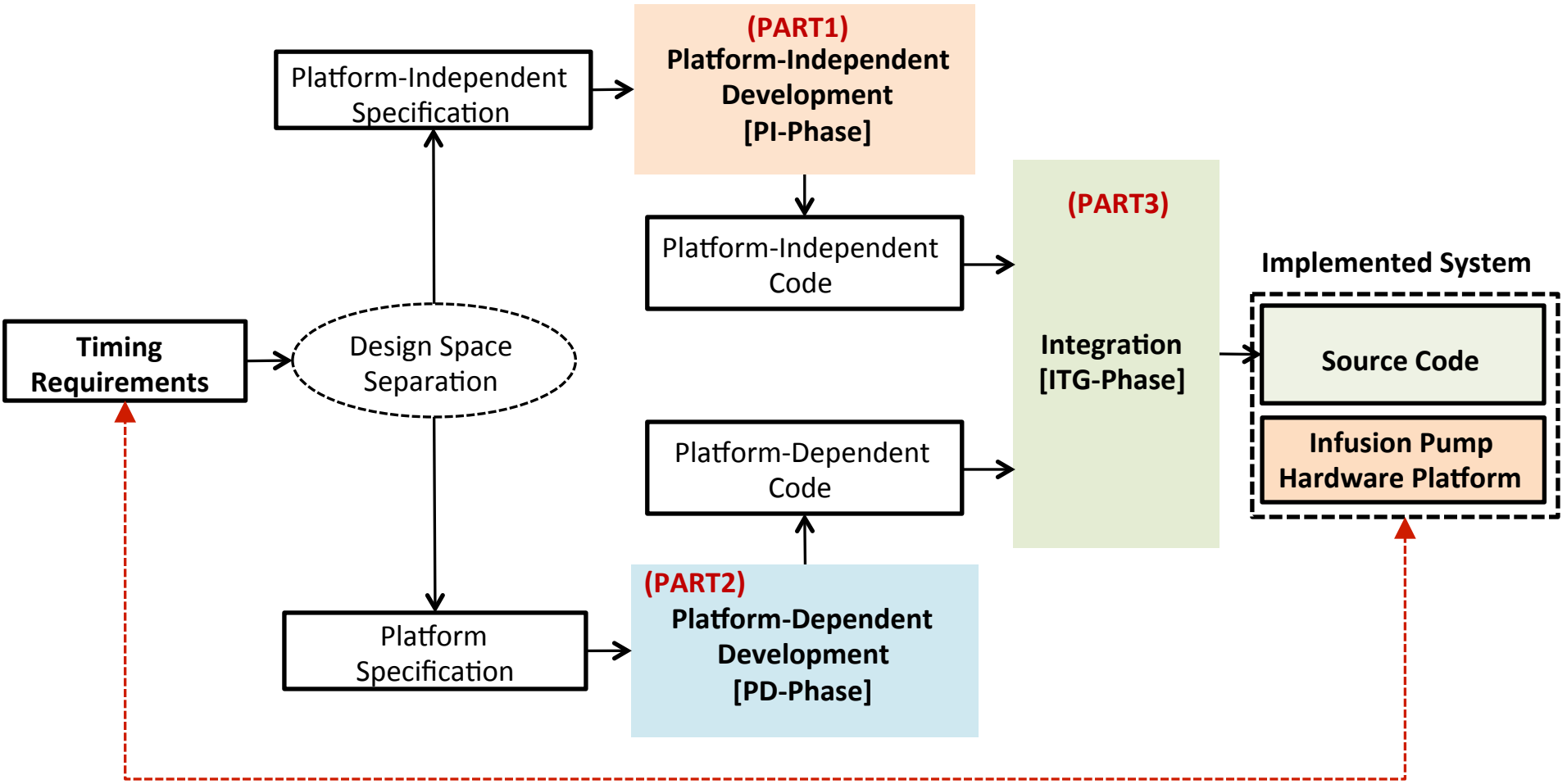


**m, i, o, c are motivated from Parnas' four-variable model*

Our Revised Approach

- Modeling and Analysis
  - Formalize I/O timing mapping between a model and implementations using Parnas' four variables
- Code generation techniques
  - Platform-independent code generation method based on timed automata
  - Platform-dependent code generation method using *code snippet repositories* and AADL models
- Integration techniques
  - *Formal verification and testing methods* that can check how well the platform-independent code is composed with a platform
  - *Timing parameter adjustment method* using *integer-linear programming* for the platform-integration

Our Revised Code Synthesis



Platform-Independent Timing Aspects

- Every pump shall meet common input/output timing constraints for its safe operation in the *environmental context*

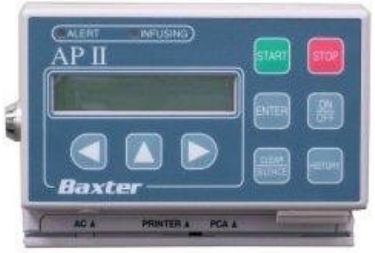
If a patient requests a drug, a pump shall start infusion within X sec

If an occlusion condition occurs, a pump shall raise an audible alarm within Y ms

Common Aspects

(Pump 1)

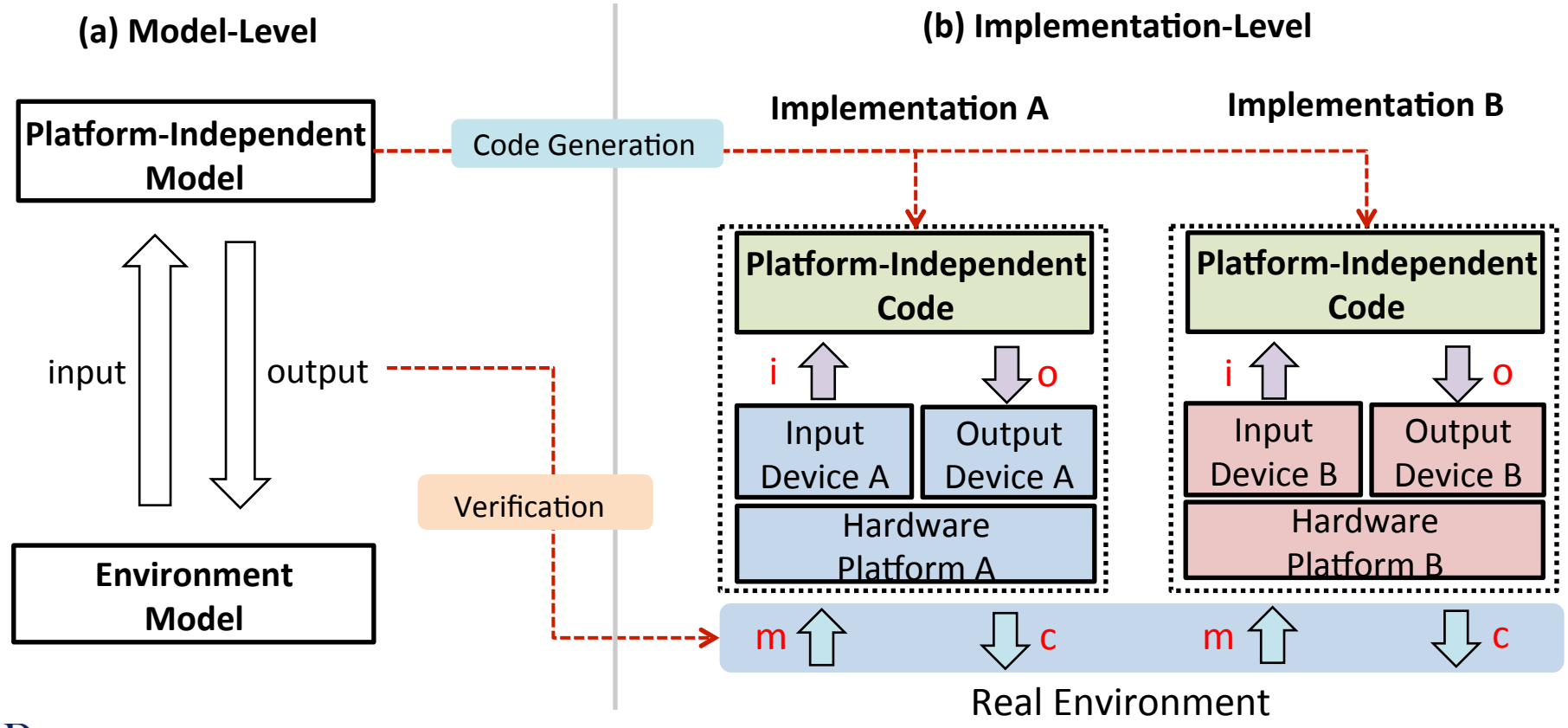
(Pump 2)



(Pump 3)

Platform-Independent Model

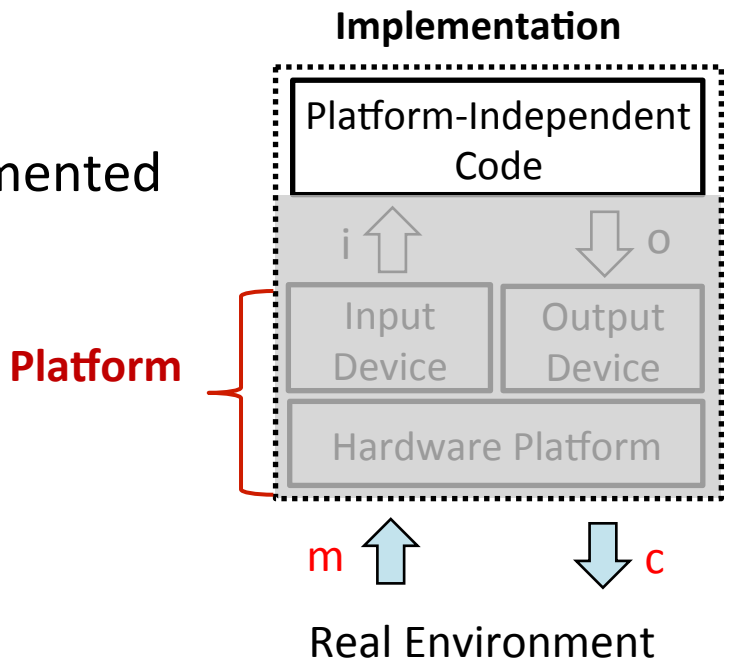
- A platform-independent model abstracts I/O timed behavior *at the mc-boundary*



* *m, i, o, c are motivated from Parnas' four-variable model*

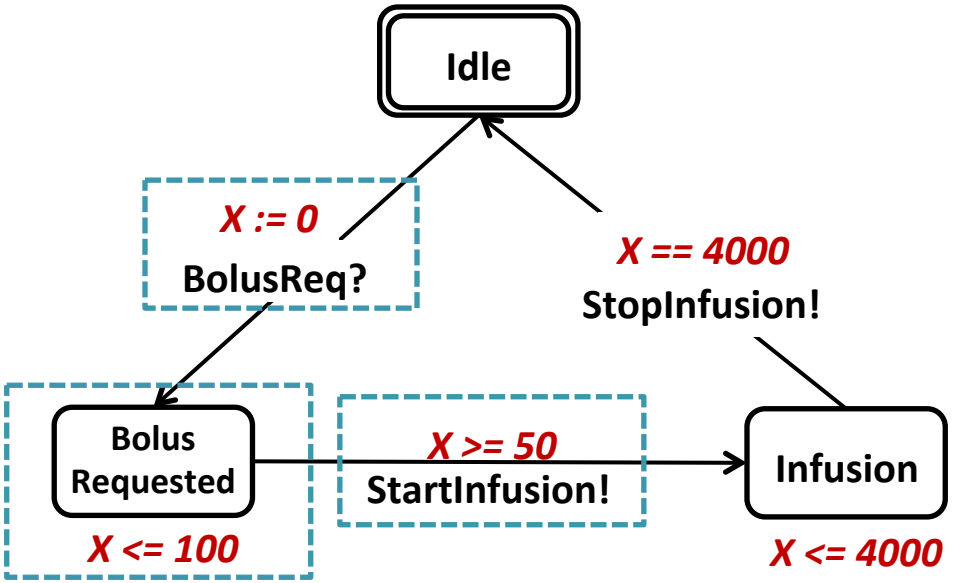
Implication of Platform-Independent Model

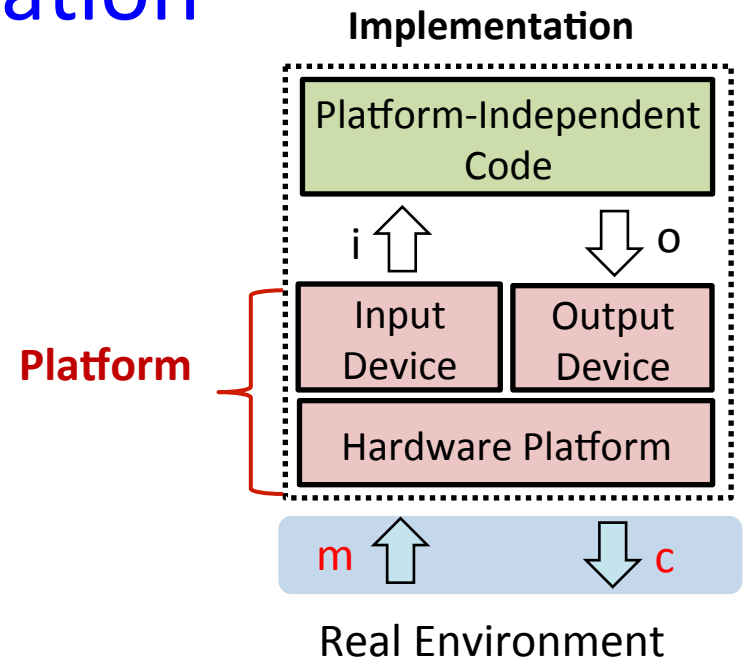
- What it expresses...
  - *External* I/O *interaction timing*
  - I/O *dependency* commonly implemented across different platforms
- What it hides...
  - *Internal* I/O *interaction timing*
  - Platform-specific I/O processing mechanism



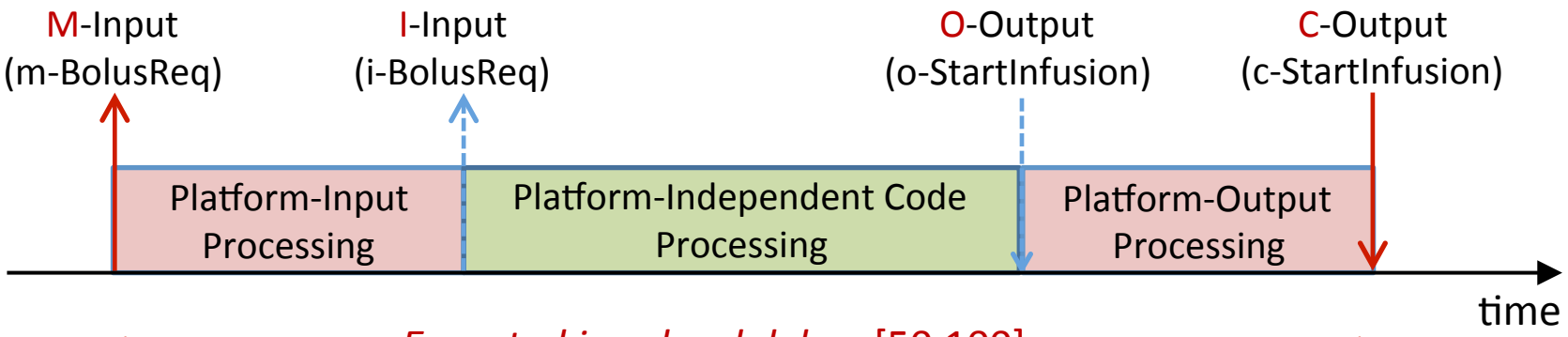
- The timing abstraction
  - Allows the code to be composed with *multi-platforms*
  - Provides *correctness criteria* of many different compositions
    - e.g., an implementation conforms to the timing requirements verified in the platform-independent model

Timing Semantics Mapping in Implementation





[Implementation-Level Timed Behavior]



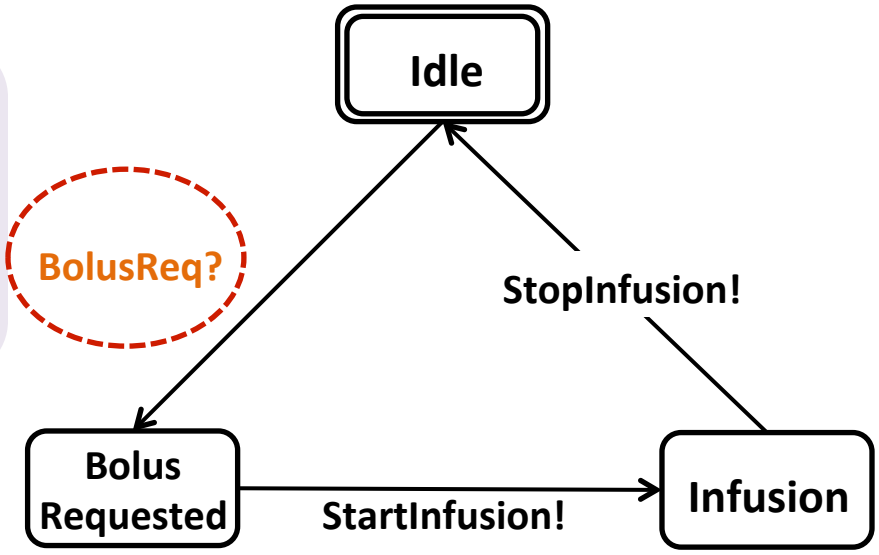
Expected imp-level delay: [50,100]

Platform-Dependent Timing Aspect 1

- Each platform has *different architectural option* to implement platform-independent timing semantics

[Platform A]
- ***A periodic thread*** that samples the electrical signal changes in the bolus request button (e.g., polling)

[Platform B]
- ***An aperiodic thread*** that is invoked upon when a change in the signal is detected (e.g., interrupts)



<Platform-Independent Model>

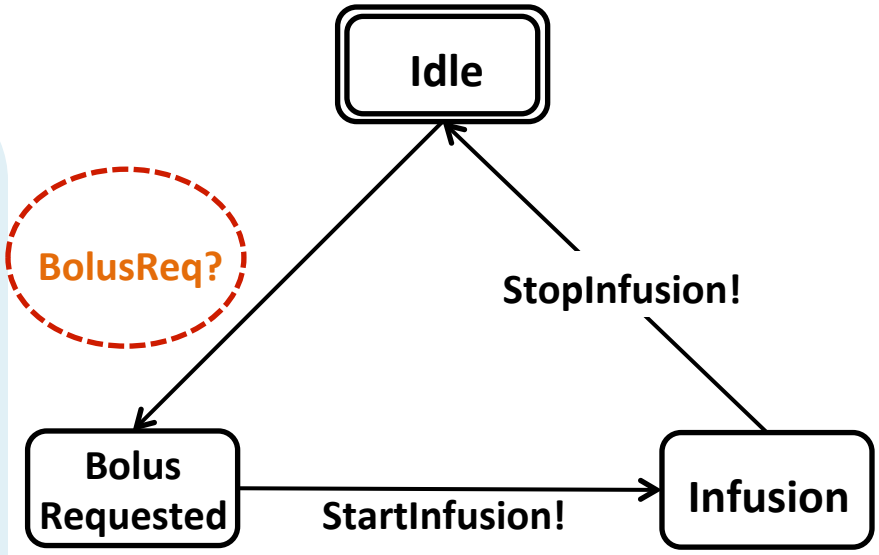
Platform-Dependent Timing Aspect 2

- Each platform has *different programming interface* to implement a particular architectural option

<Example: FreeRTOS Programming Interface>

[Task callback function]

```
void cbBolusReq (void* pvParameters){
   portTickType xLastWakeTime;
   xLastWakeTime = xTaskGetTickCount();
   for(;;){
      //Wait for the next cycle
      vTaskDelayUntil(&xLastWakeTime, periodEmptyRsv );
      //Perform action here
      //(1)Read
      //(2)Compute
      //(3)Write
   }
}
```



<Platform-Independent Model>

Other platforms have different code patterns and APIs

Platform-Dependent Timing Aspects

Each pump has a different way of implementing the platform-independent timing aspects

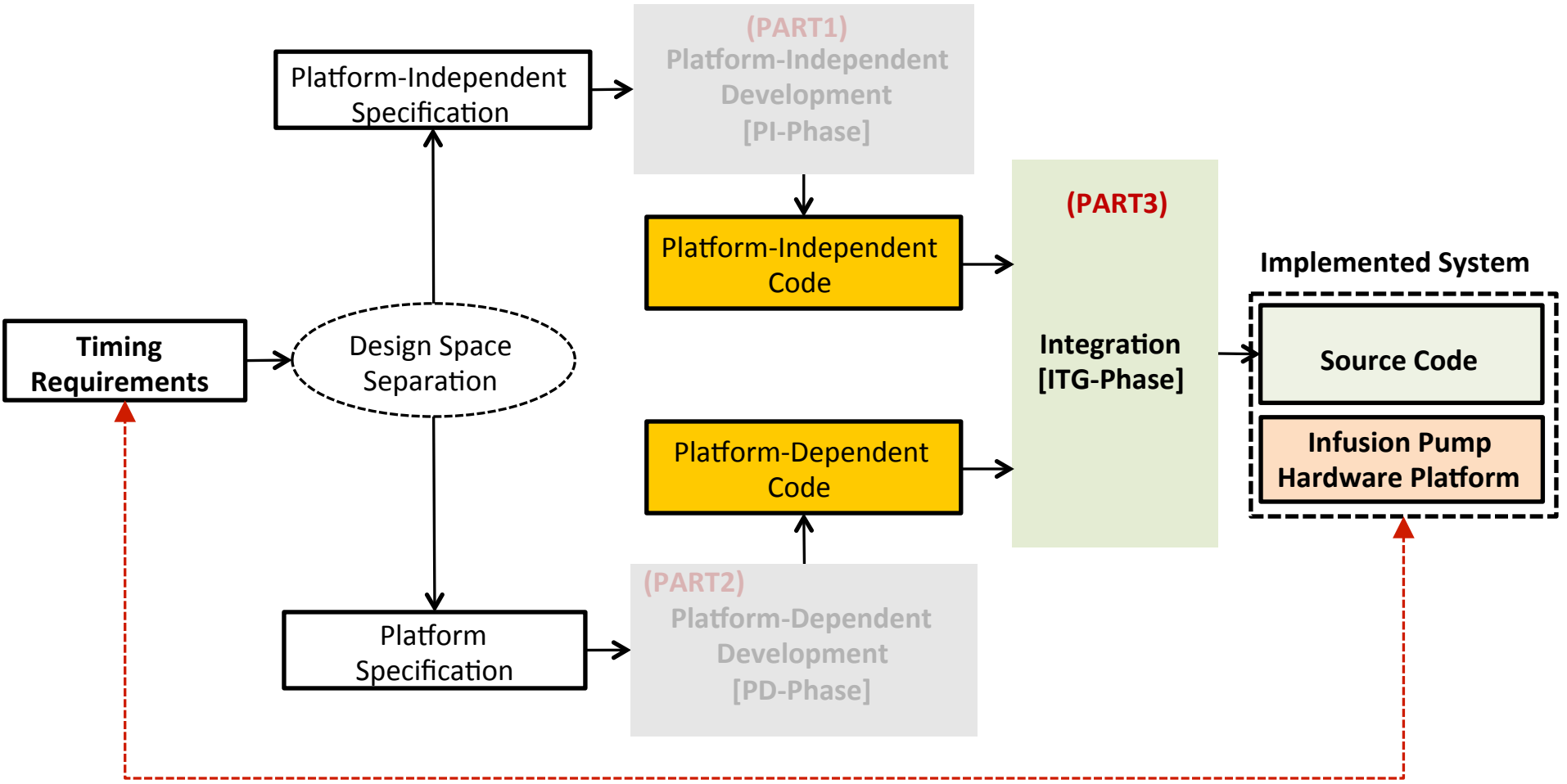
- (E.g.) Different interaction mechanisms with sensors/actuators
  - Polling or interrupt-based mechanisms
- (E.g.) Different timing overhead for I/O device drivers
- (E.g.) Different scheduling mechanisms
  - Periodic or aperiodic scheduling

Different Aspects

Different Aspects

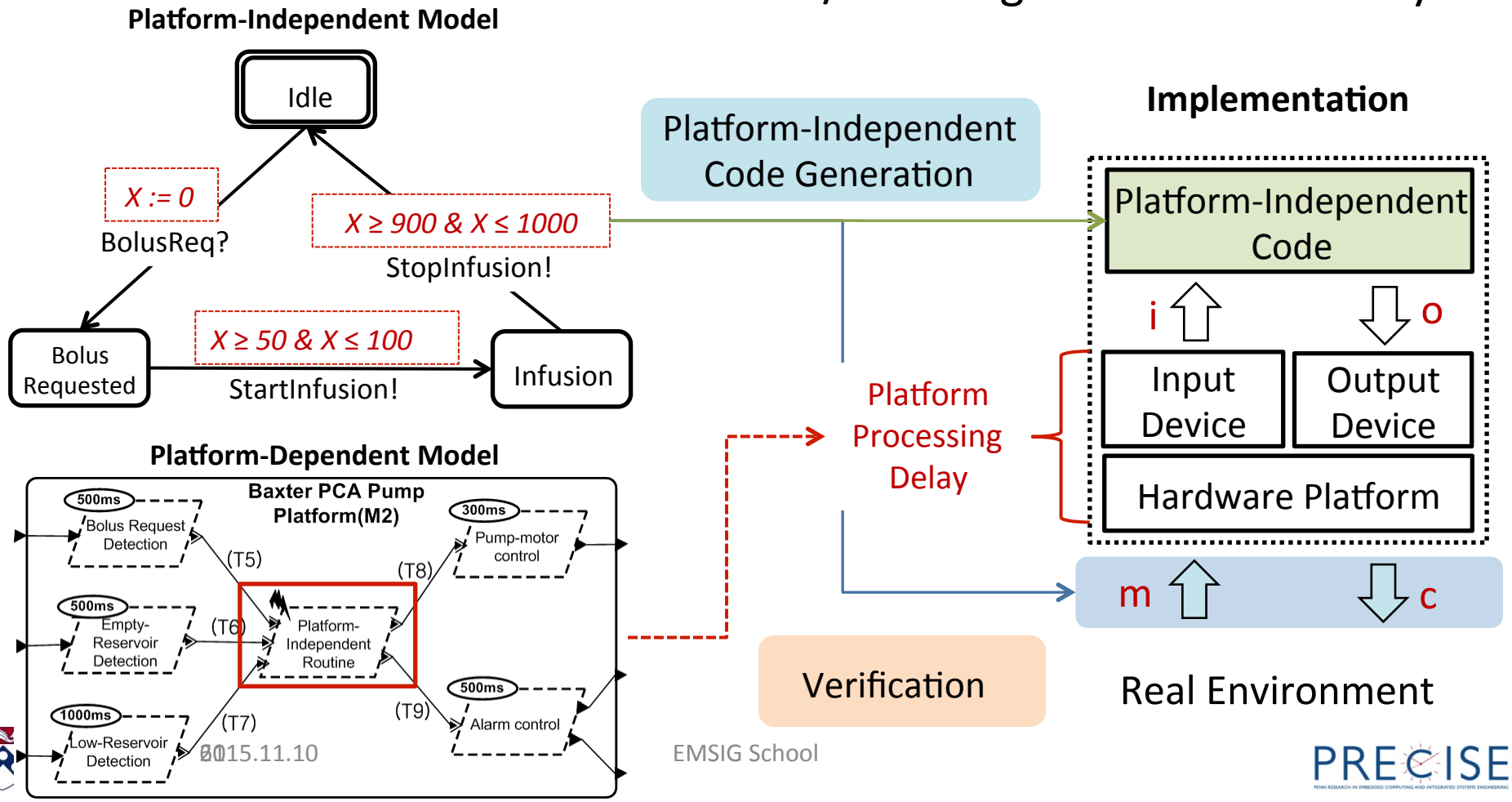
(Pump 1) (Pump 2) (Pump 3)

ITG-Phase


Platform-Independent Specification
(PART1)
Platform-Independent Development
[PI-Phase]
Platform-Independent Code
(PART3)
Integration
[ITG-Phase]
Implemented System
Source Code
Infusion Pump Hardware Platform
Timing Requirements
Design Space Separation
Platform-Dependent Code
Platform Specification
(PART2)
Platform-Dependent Development
[PD-Phase]

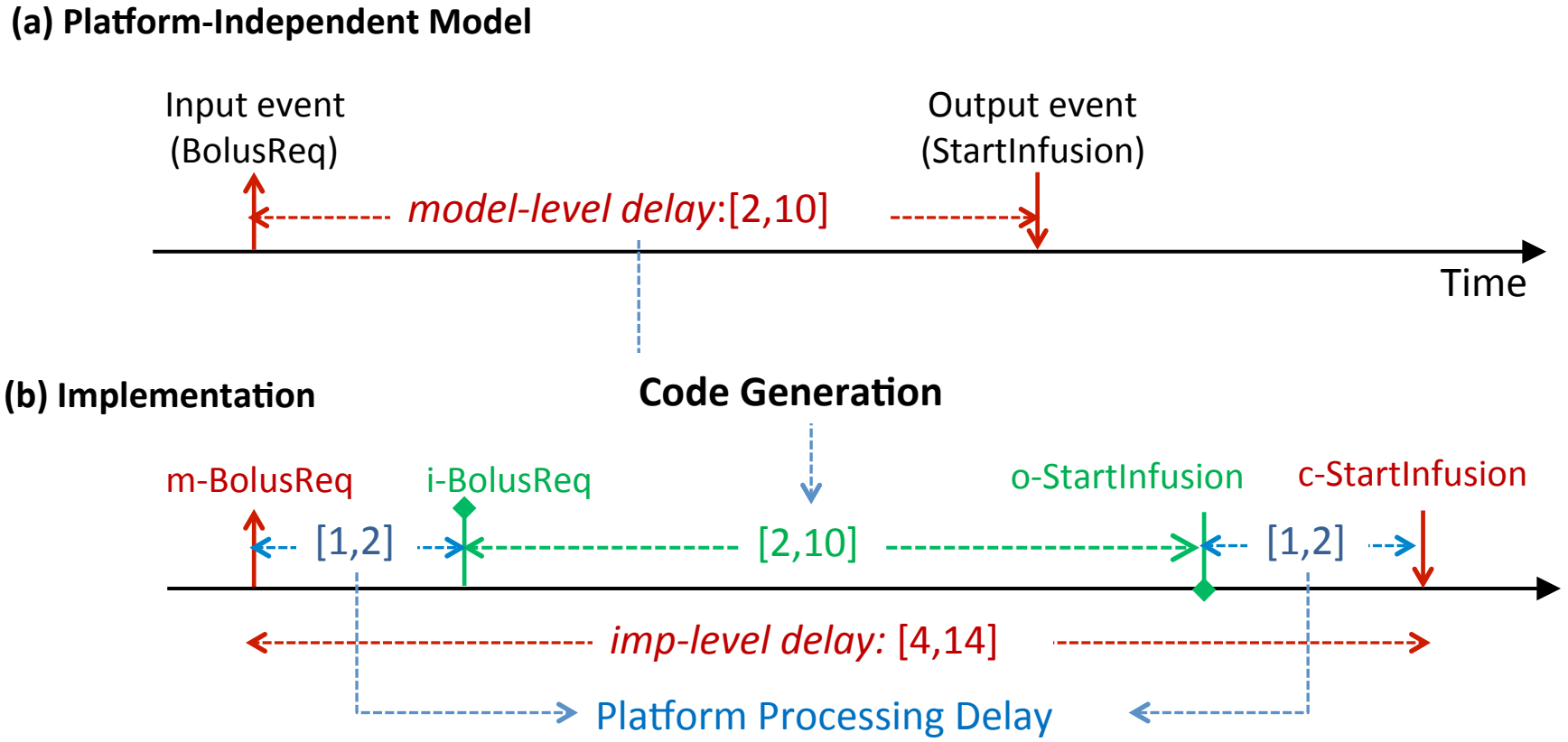
Integration Issue (1/2)

- Usage of *timing parameters* of PIM
  - **Verification**: determine I/O timing at the *mc*-boundary
  - **Code Generation**: determine I/O timing at the *io*-boundary



Integration Issue (2/2)

- *The platform processing delay is added in the code-level delay*



Proposed Approaches

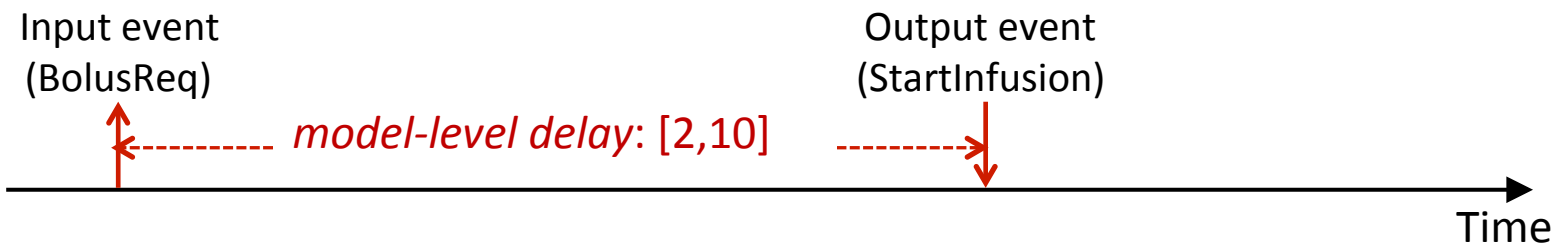
- Taking into account platform processing delays

- Testing approach [DATE 2014]
  - Timing testing framework to check timing requirement conformance and to measure the timing deviation due to the platform-processing delays
- Model-checking approach [DATE 2015]
  - Systematic construction of the platform-specific models (*PSM*) that better characterizes the implementation-level timed behavior

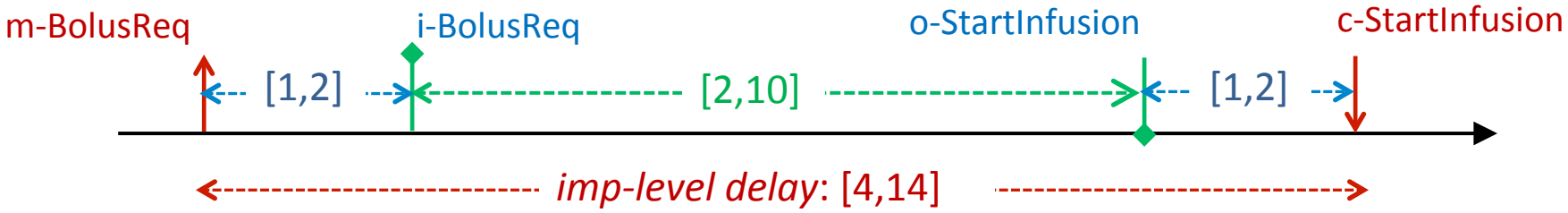
- **Timing parameter adjustment approach [RTSS 2015]**
  - Adjusting timing parameters of the platform-independent code to compensate the platform-processing delays

Illustration of the Parameter Adjustment

(a) Platform-Independent Model

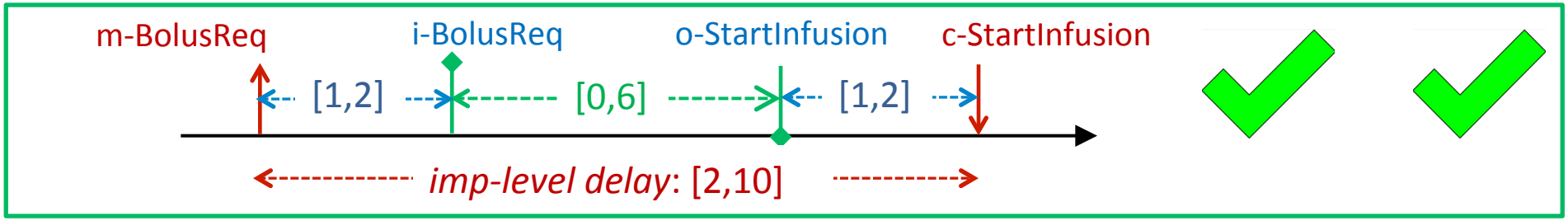


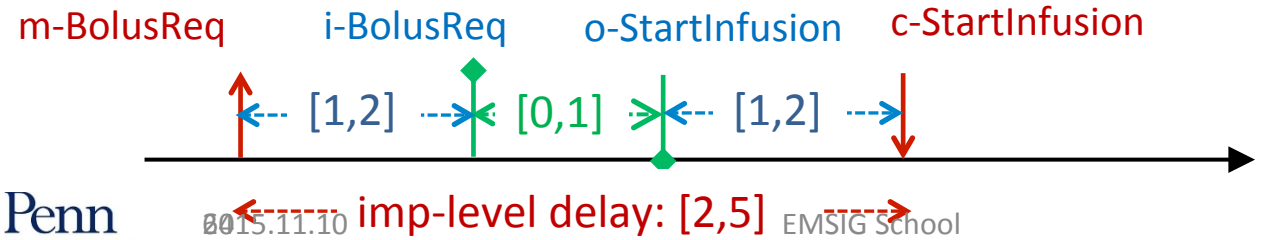
(b) Implementation (*before* adjustment)



(c) Implementation (*after* adjustment)

Delay-Bound Inclusion **Optimal Adjustment**





Summary of ITG-Phase

- Case Study
  - The experimental result shows that the implementation with parameter adjustment better preserves the timing requirements
    - (1) PCA pump implementation *without* parameter adjustment
    - (2) PCA pump implementation *with* parameter adjustment
- Benefits of the parameter adjustment method
  - Checking *composability* in terms of timing requirement conformance
  - Finding min/max timing parameters of the platform-independent code for the composition
- Complement to other approaches
  - Testing approach [DATE 2014]
  - Model-checking approach [DATE 2015]

Related Work

- Software development process
  - Parnas' work: Functional documents for computer systems (SCP 1995)
  - Jeffrey's work: Specification based prototyping for embedded systems (ESEC/FSE 1999)
  - Hassan's work: Designing software product lines with UML: From use cases to pattern-based software architectures (2004)
- Modeling/Verification
  - K. Altisen's work: Implementation of timed automata : an issue of semantics or modeling (FORMAT 2005)
  - Martin Wulf's work: Almost ASAP semantics : From timed model to timed implementations (HSCC 2004)
  - Tesnim Abdellatif's work: Model-Based Implementation of Real-Time Applications (EMSOFT 2010)
- Code Generation
  - Gilles' work: An environment for AADL models analysis and automatic code generation for high integrity applications (2009)
  - Thomas' work: The embedded machine: Predictable, portable real-time code (2007)
- Testing
  - Gregory's work: Steering model-based oracles to admit real program behaviors (ICSE 2014)
  - Larsen's work: Online testing of real-time systems using UPPAAL (2005)

Publications

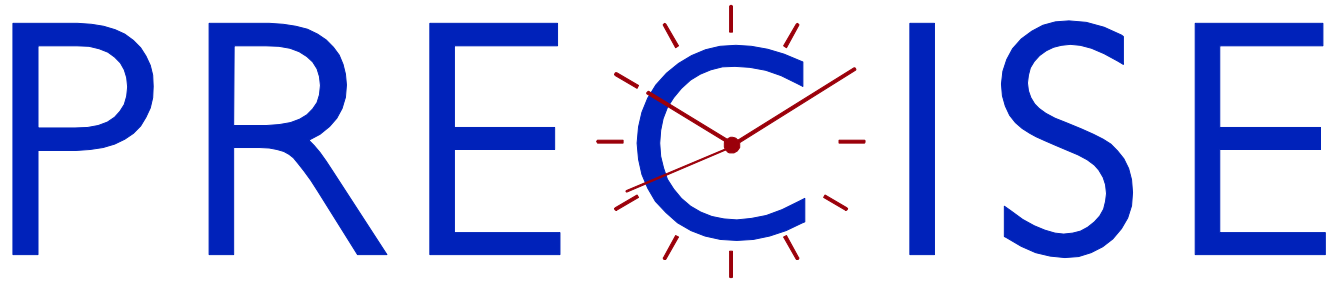
[***Work appeared in Dissertation***]

- Platform-Specific Code Generation from Platform-Independent Timed Models, BaekGyu Kim, Lu Feng, Oleg Sokolsky and Insup Lee (*RTSS 2015*). Dec 2015
- Platform-Specific Timing Verification Framework in Model-Based Implementation, BaekGyu Kim, Lu Feng, Linh T.X. Phan, Oleg Sokolsky and Insup Lee, *Design, Automation and Test in Europe (DATE 2015)*. Grenoble, France, Mar 2015
- A Layered Approach for Timing Testing in the Model-Based Implementation, BaekGyu Kim, Hyeon I Hwang, Taejoon Park, Sanghyuk Son, Insup Lee. *Design, Automation and Test in Europe (DATE 2014)*. Dresden, Germany, Mar 2014
- Platform-Dependent Code Generation for Embedded Real-Time Software, BaekGyu Kim, Linh T.X. Phan, Insup Lee, and Oleg Sokolsky. *International Conference on Compilers, Architectures and Synthesis of Embedded Systems (CASES 2013)*. Montreal, Canada, October 2013
- A Model-Based I/O Interface Synthesis Framework for the Cross-Platform Software Modeling. BaekGyu Kim, Linh T.X. Phan, Insup Lee, and Oleg Sokolsky. In *IEEE International Symposium on Rapid System Prototyping (RSP 2012)*. Tampere, Finland, October 2012
- Safety-Assured Development of the GPCA Infusion Pump Software. BaekGyu Kim, Anaheed Ayoub, Oleg Sokolsky, Insup Lee, Paul Jones, Yi Zhang, and Raoul Jetley. *International Conference on Embedded Software (EMSOFT 2011)*. Taipei, Taiwan, October 2011

[***Additional Work***]

- From Requirements to Code: Model Based Development of a Medical Cyber Physical System, Anitha Murugesan, Michael Whalen, Sanjai Rayadurgam, John Komp, Lian Duan, Mats Heimdahl, Baek-Gyu Kim, Oleg Sokolsky and Insup Lee, *FHIES/SEHC 2014*. Washington, D.C., USA, July 2014
- A Causality Analysis Framework for Component-based Real-time Systems, Shaohui Wang, Anaheed Ayoub, BaekGyu Kim, Gregor G ossler, Oleg Sokolsky, and Insup Lee, *Runtime Verification (RV2013)*, INRIA Rennes, France
- A Systematic Approach to Justifying Sufficient Confidence in Software Safety Arguments. Anaheed Ayoub, BaekGyu Kim, Insup Lee, and Oleg Sokolsky. In *31st International Conference on Computer Safety, Reliability and Security (SAFECOMP 2012)*, Magdeburg, Germany, September 2012
- A Safety Case Pattern for Model-Based Development Approach. Anaheed Ayoub, Baek-Gyu Kim, Insup Lee and Oleg Sokolsky. In *NASA Formal Methods Symposium (NFM)*. Norfolk, VA, April 2012
- Challenges and Research Directions in Medical Cyber-Physical Systems. Insup Lee, Oleg Sokolsky, Sanjian Chen, John Hatcliff, Eunkyoung Jee, BaekGyu Kim, Andrew King, Margaret Mullen-Fortino, Soojin Park, Alexander Roederer, and Krishna Venkatasubramanian. *In Special Issue on Cyber-Physical Systems, Proceedings of the IEEE*, Volume 100, Issue 1, pp.75-90, January 2012
- The Medical Device Dongle: An Open-Source Standards-Based Platform for Interoperable Medical Device Connectivity. Philip Asare, Danyang Cong, Santosh Vattam, Baek-Gyu Kim, Shan Lin, Oleg Sokolsky, Margaret Mullen-Fortino and Insup Lee. *In Proceedings of the 2nd ACM SIGHIT International Health Informatics Symposium (IHI 2012)*. Miami, FL, January 2012

Thank You!
Questions?



http://precise.seas.upenn.edu