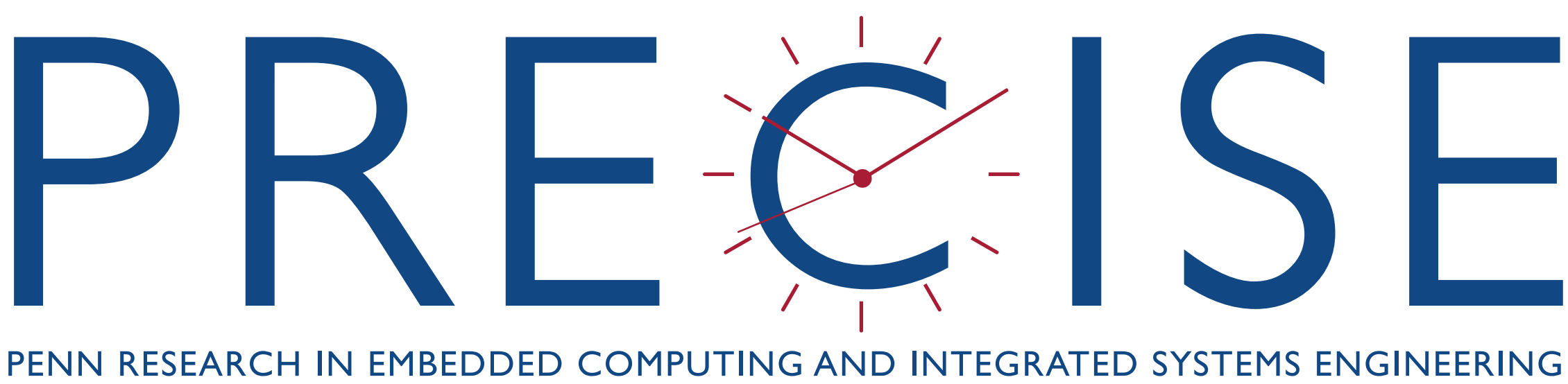# MIDdleware Assurance Substrate (MIDAS)

Andrew King, Vincent Wang, Oleg Sokolsky, Insup Lee
PRECISE Center, University of Pennsylvania

PRECISE
PENN RESEARCH IN EMBEDDED COMPUTING AND INTEGRATED SYSTEMS ENGINEERING

Motivation

The Problem

The notion of Cyber Physical Systems (CPS) recognizes that new and complex systems, which closely couple (digital) discrete, physical (continuous), and social elements have the potential to benefit many aspects of society. Many of these CPS, by their very nature will be integrated into the social unit which they serve and should be malleable according to the needs of that social unit. Additionally, many of these CPS will be safety critical; if the system behaves incorrectly humans may be injured or killed. Medical Device Plug and Play (MD PnP), described below, is an example of such a dynamic CPS that will be reconfigured and extended at runtime by the users.

Medical Device Plug and Play (MD PnP)

MD PnP envisions a future where commodity medical devices are implemented with support for a widely accepted interoperability standard. These medical devices would expose a logical interface which would allow applications to access the sensor data these devices are collecting (e.g. blood pressure, respiratory rate) as well as reconfigure any actuators present in the device (e.g. the infusion rate of an infusion pump, stroke of a ventilator, etc.) The MD PnP platform would then let clinicians launch applications for specific clinical scenarios (e.g. multi-device closed loop control, smart alarms, etc.) if application compatible devices are connected, essentially creating a new, complex, medical device which we call a Virtual Medical Device (VMD). Unlike normal medical devices, the **systems integration stage of the device's lifecycle is performed by the hospital (point of care)**. How can we ensure that the newly created VMD is safe?
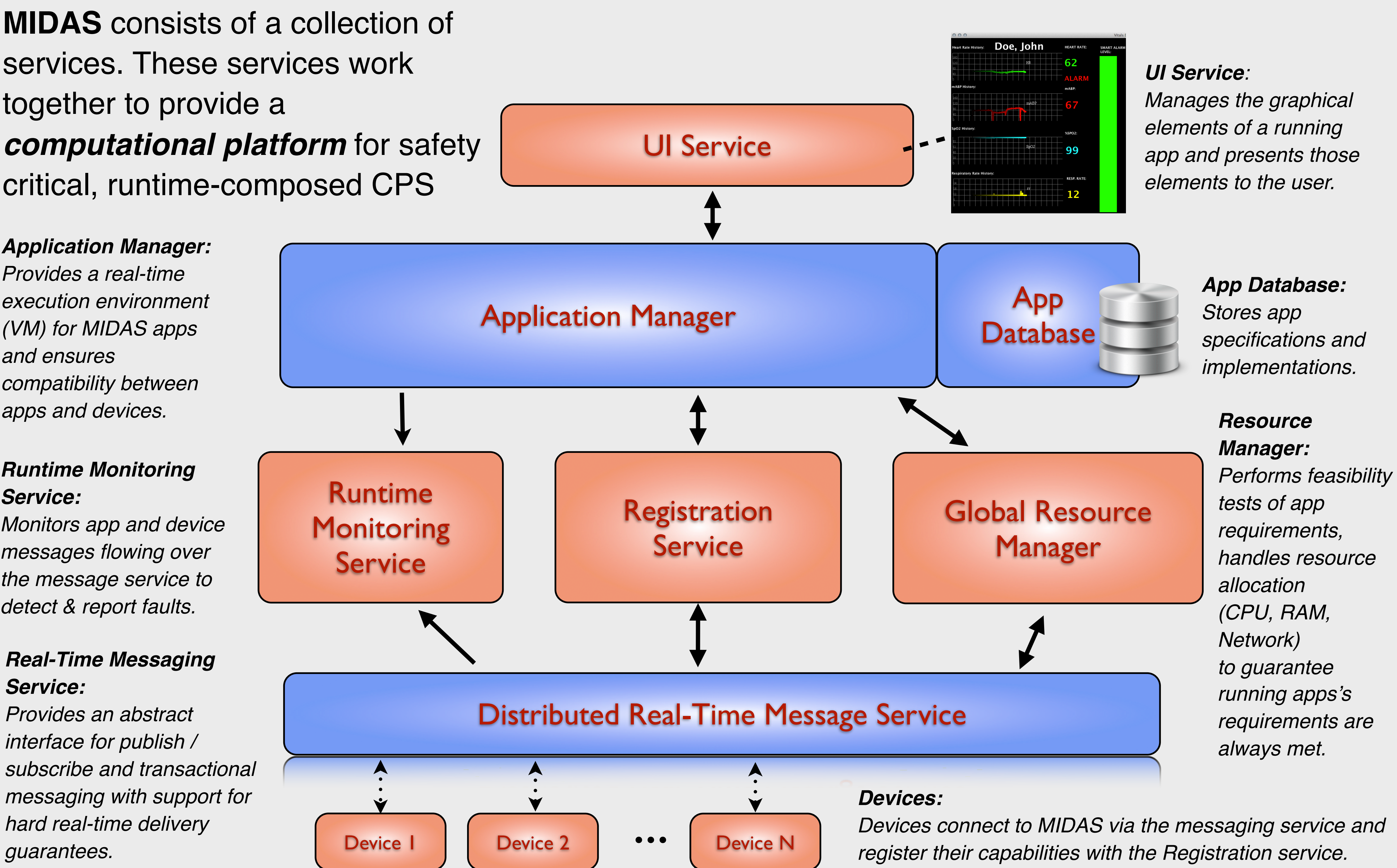
Our Approach

A CPS designed in an ad-hoc fashion (e.g. current health IT systems and the national power grid) will suffer from both complexity and "design-rot." This complexity and design-rot will make it difficult or impossible to perform relevant verification and validation or ensure that the system operates as intended. A computational platform that 1) mediates the interaction between the different sub-systems of the CPS, 2) provides a form of global resource management for QoS control, 3) provides a automatic compatibility check which will ensure that only components with the correct capability are composed into the CPS and 4) provides useful abstractions for design and programming will help manage this complexity and provide a way for engineers and tools to extract checkable models of the CPS.
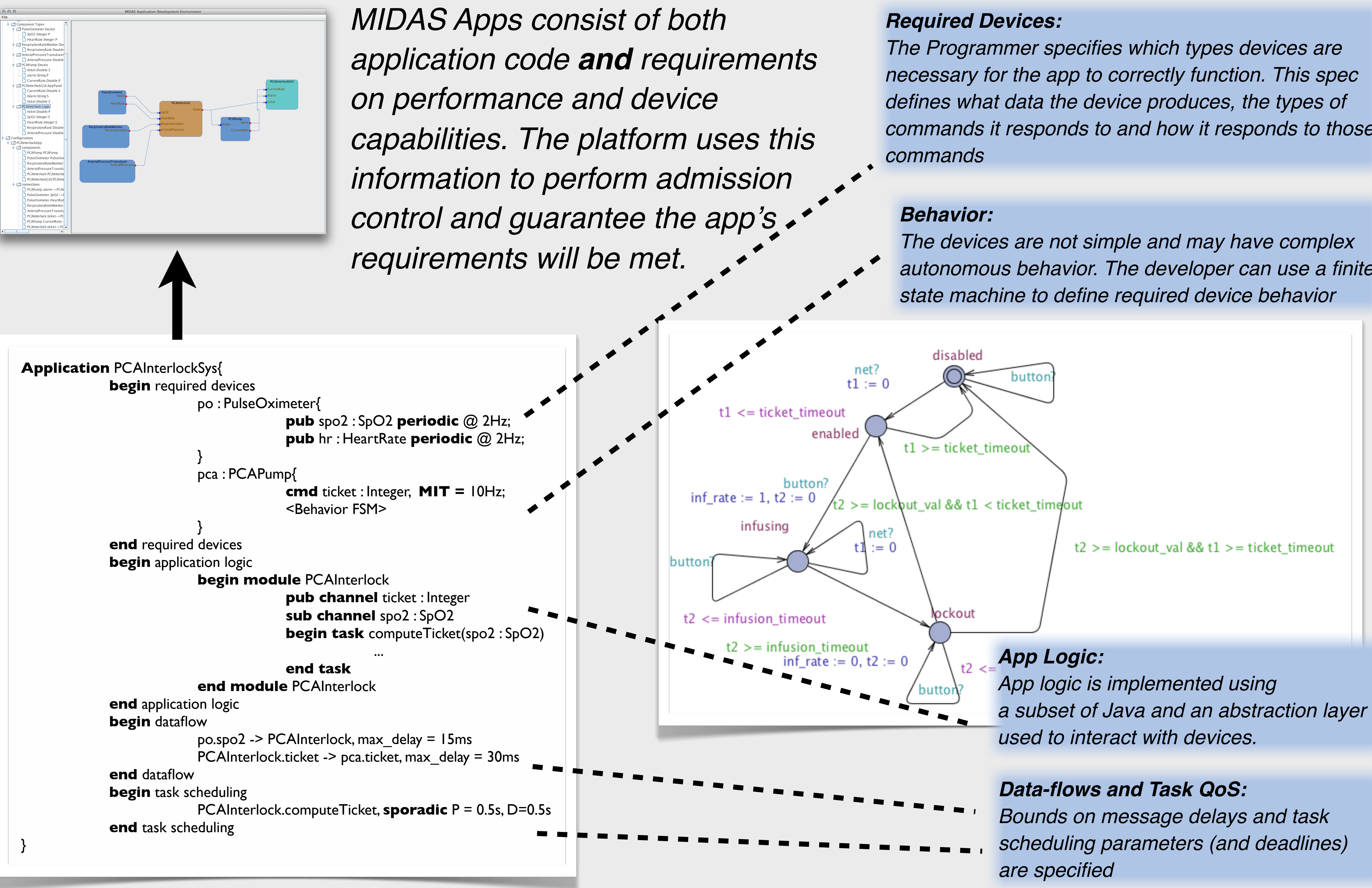
Relation to MDCF

The **Medical Device Coordination Framework (MDCF)** is a joint project between the University of Pennsylvania and Kansas State University to implement software infrastructure for medical device plug and play. **MIDAS** is an extension of the MDCF to support a more general notion of runtime composable CPS. MIDAS removes medical device specific elements of the MDCF (in order to generalize the concepts) but adds support for real time communications and tasks, a CPS-oriented notion of component compatibility checks, and a runtime monitoring system that enables runtime verification (RV) of executing applications. Eventually, some technologies prototyped with MIDAS (global resource management, real-time support and RV) will get back ported into the MDCF infrastructure as the broader MDCF development team sees fit.
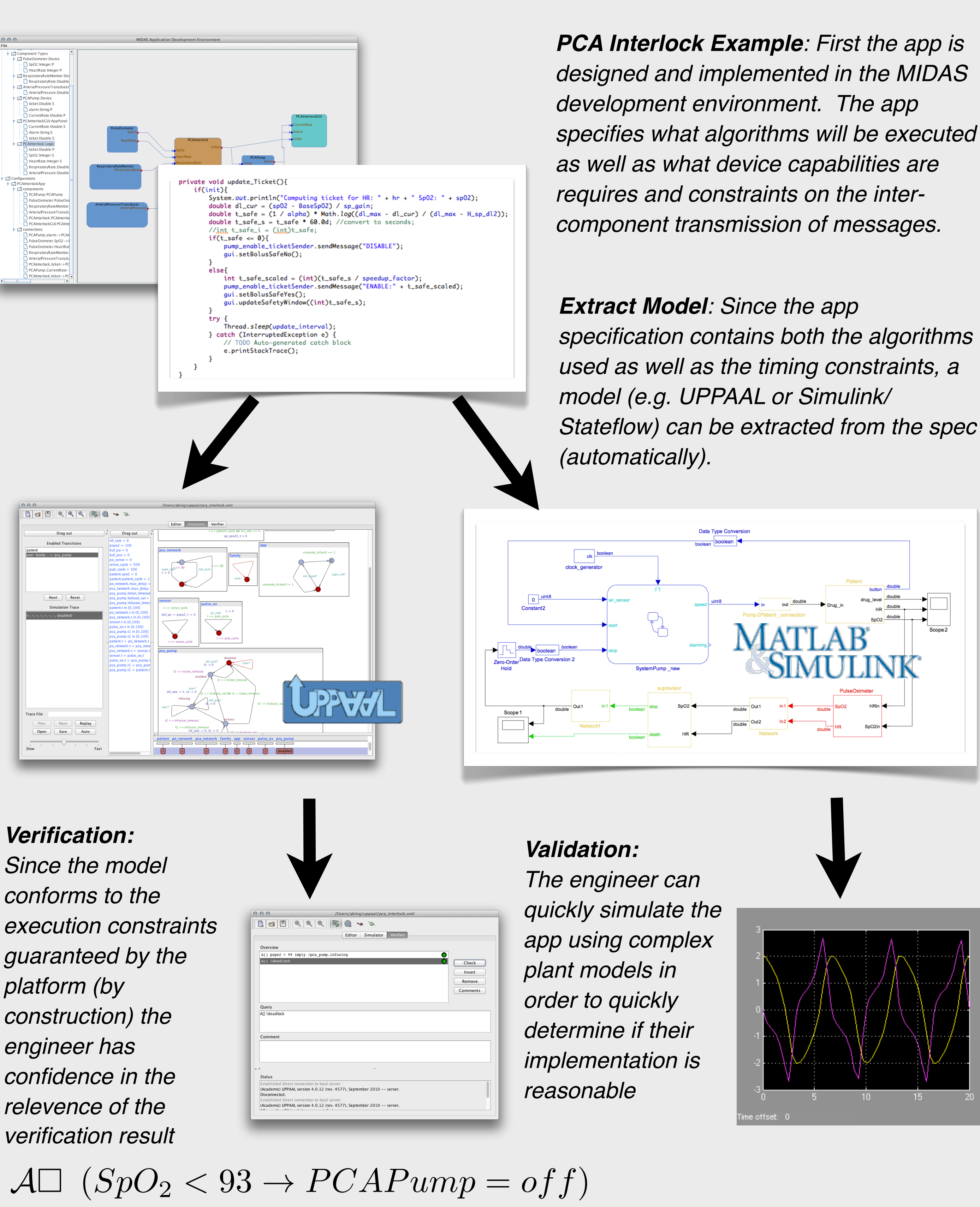
Computational Platform Services / Architecture

MIDAS consists of a collection of services. These services work together to provide a **computational platform** for safety critical, runtime-composed CPS

UI Service:
Manages the graphical elements of a running app and presents those elements to the user.

Application Manager:
Provides a real-time execution environment (VM) for MIDAS apps and ensures compatibility between apps and devices.

App Database:
Stores app specifications and implementations.

Runtime Monitoring Service:
Monitors app and device messages flowing over the message service to detect & report faults.

Resource Manager:
Performs feasibility tests of app requirements, handles resource allocation (CPU, RAM, Network) to guarantee running apps's requirements are always met.

Real-Time Messaging Service:
Provides an abstract interface for publish / subscribe and transactional messaging with support for hard real-time delivery guarantees.

Devices:
Devices connect to MIDAS via the messaging service and register their capabilities with the Registration service.

App Design & Development

MIDAS Apps consist of both application code **and** requirements on performance and device capabilities. The platform uses this information to perform admission control and guarantee the app's requirements will be met.

Required Devices:
The Programmer specifies which types devices are necessary for the app to correctly function. This spec defines what data the device produces, the types of commands it responds to and how it responds to those commands

Behavior:
The devices are not simple and may have complex autonomous behavior. The developer can use a finite state machine to define required device behavior

App Logic:
App logic is implemented using a subset of Java and an abstraction layer used to interact with devices.

Data-flows and Task QoS:
Bounds on message delays and task scheduling parameters (and deadlines) are specified

```
Application PCAInterlockSys{
    begin required devices
        po : PulseOximeter{
            pub spo2 : SpO2 periodic @ 2Hz;
            pub hr : HeartRate periodic @ 2Hz;
        }
        pca : PCAPump{
            cmd ticket : Integer, MIT = 10Hz;
            <Behavior FSM>
        }
    end required devices
    begin application logic
        begin module PCAInterlock
            pub channel ticket : Integer
            sub channel spo2 : SpO2
            begin task computeTicket(spo2 : SpO2)
                ...
            end task
        end module PCAInterlock
    end application logic
    begin dataflow
        po.spo2 -> PCAInterlock, max_delay = 15ms
        PCAInterlock.ticket -> pca.ticket, max_delay = 30ms
    end dataflow
    begin task scheduling
        PCAInterlock.computeTicket, sporadic P = 0.5s, D=0.5s
    end task scheduling
}
```


App Verification and Validation

PCA Interlock Example: First the app is designed and implemented in the MIDAS development environment. The app specifies what algorithms will be executed as well as what device capabilities are requires and constraints on the inter-component transmission of messages.

Extract Model: Since the app specification contains both the algorithms used as well as the timing constraints, a model (e.g. UPPAAL or Simulink/ Stateflow) can be extracted from the spec (automatically).

Verification:
Since the model conforms to the execution constraints guaranteed by the platform (by construction) the engineer has confidence in the relevence of the verification result

Validation:
The engineer can quickly simulate the app using complex plant models in order to quickly determine if their implementation is reasonable

$$\mathcal{A}\square \ (SpO_2 < 93 \rightarrow PCAPump = off)$$

User's Workflow

1) Choose Devices
First the user determines what devices they would like to bind with their app of choice. The user plugs in the necessary devices to the network.

2) Device Registration
The devices register their capabilities with MIDAS's registration service. The registration service actively tracks the network connectivity of the device.

3) User binds devices and initiates app
The user uses the MIDAS UI to select which specific registered devices to bind to a new app instance.

4) App Manager asks Resource Manager for app resources
The resource manager performs a feasibility check to determine if the platform can guarantee the QoS requirements of the app at the time. If not, app initialization is halted.

5) App Manager performs a device compatibility check
The selected device capabilities are checked against the apps device requirements. If the any device is incompatible then app initialization is halted and the user is informed.

6) App Manager initiates App
The app manager loads app logic into memory, uses the resource manager to lock the necessary resources, and then schedules and releases the app tasks.