Co-simulation of embedded systems: a PVS-Simulink integrated environment

Cinzia Bernardeschi[1] Andrea Domenici[1]

Paolo Masci[2]

[1]Department of Information Engineering, University of Pisa

[2]INESC-TEC and Universidade do Minho, Portugal

Italian Workshop on Embedded Systems, 19-20 September 2016, Pisa

Embedded systems

- Embedded systems consist of a discrete-time component embedded in a continuous-time plant, and in most cases have safety requirements.
- Modelling and simulation formalisms for discrete systems and those for continuous systems are distinct
  - discrete systems: evolve through a set of states (e.g., Statecharts, Timed Automata)
  - continuous systems: described by a set of variables whose value changes continuously according to a set of laws, usually defined by differential equations (e.g., Matlab, ScicosLab)
- The two subsystems should be modeled in the appropriate formalism, also because digital control experts may not be familiar with plant modelling, and conversely

Besides being validated by simulation and testing, such systems should be formally verified

Contribution

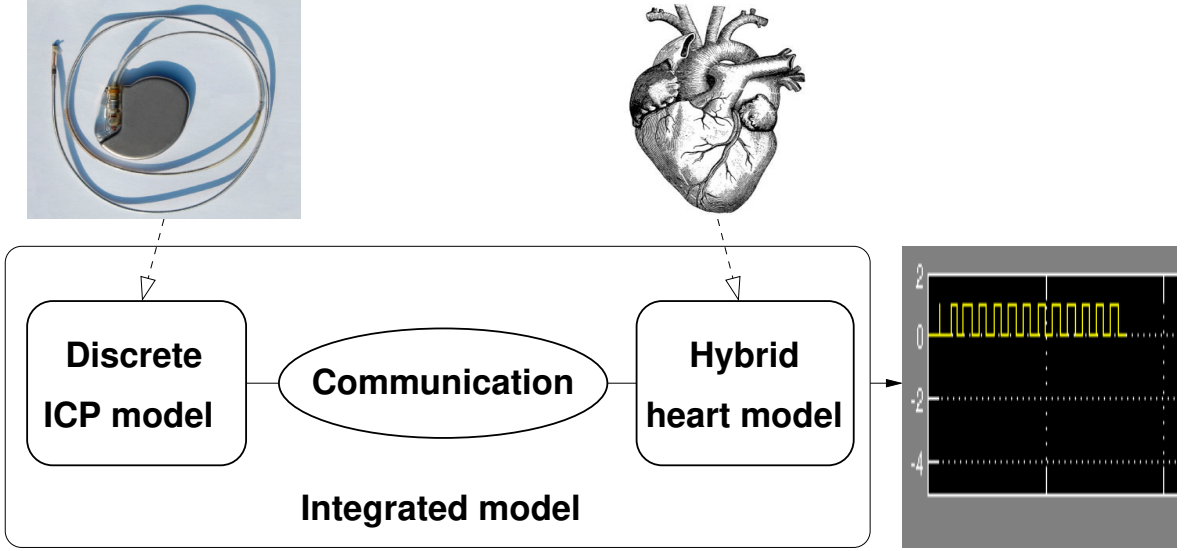
A framework, with supporting tools, where a logical model of a discrete system is interfaced to a block-based model of a continuous one

- discrete part formally specified with Timed Automata, translated into a set of logic theories
- continuous part described in Simulink

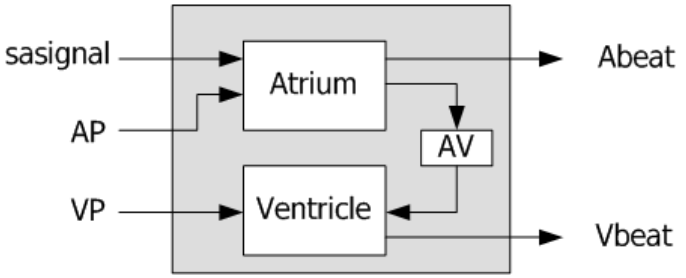
Main features of the framework

- A network of Timed Automata is created with a graphical editor;
- the network is automatically translated into logic theories according to patterns we have developed;
- the resulting logic specification is amenable both to verification and simulation with the PVS theorem prover and its PVSio extension;
- the logical specification can be co-simulated with a block- based model;
- two interface subsystems connect the respective models to a WebSocket communication framework. Their purpose is to catch simulation events generated by the two models, and forward them from one simulation environment to the other

Example: heart-pacemaker co-simulation



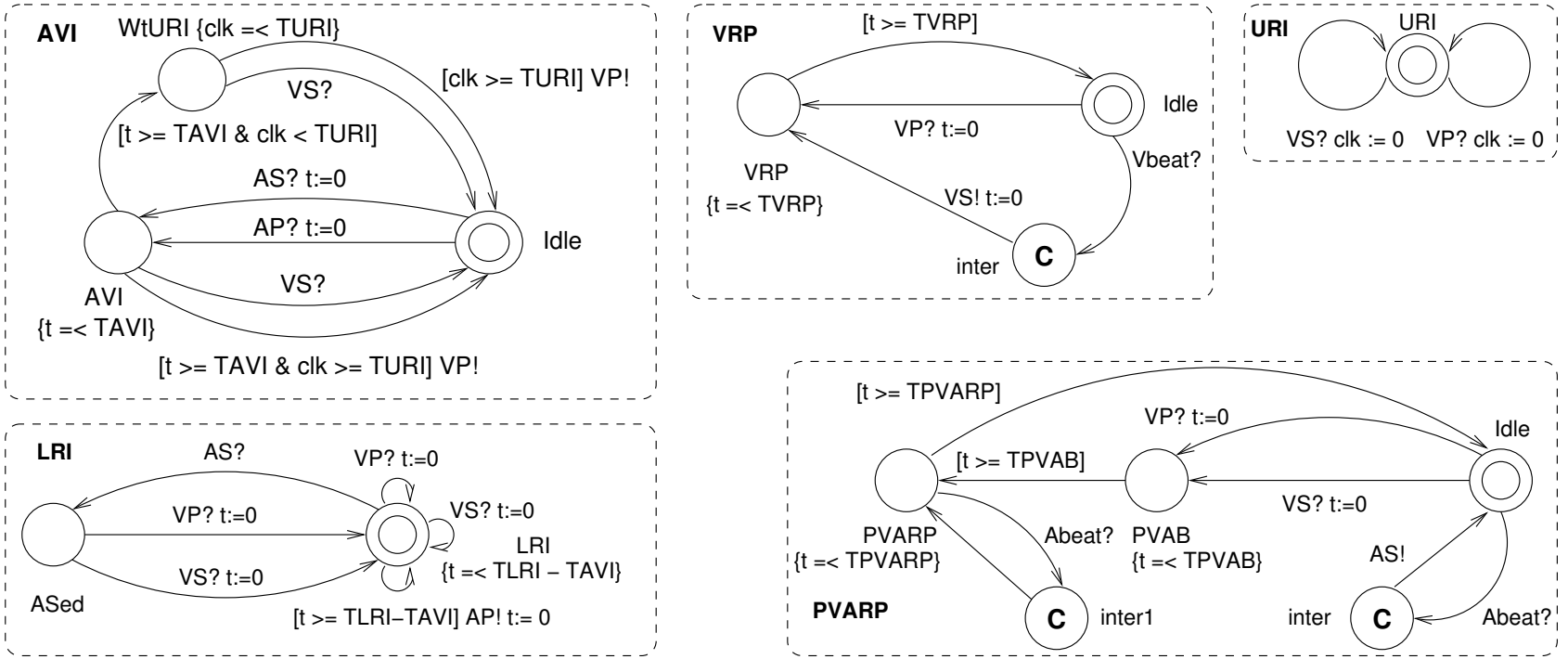
Architecture of the heart model (Chen et al., 2014)



(Chen et al., 2014) T. Chen, M. Diciolla, M. Kwiatkowska, and A. Mereacre, Quantitative verification of implantable cardiac pacemakers over hybrid heart models, Information and Computation, vol. 236, n. 0, 2014.

The pacemaker model

A TA network modeling the behavior of a pacemaker (Jiang et al., 2012)



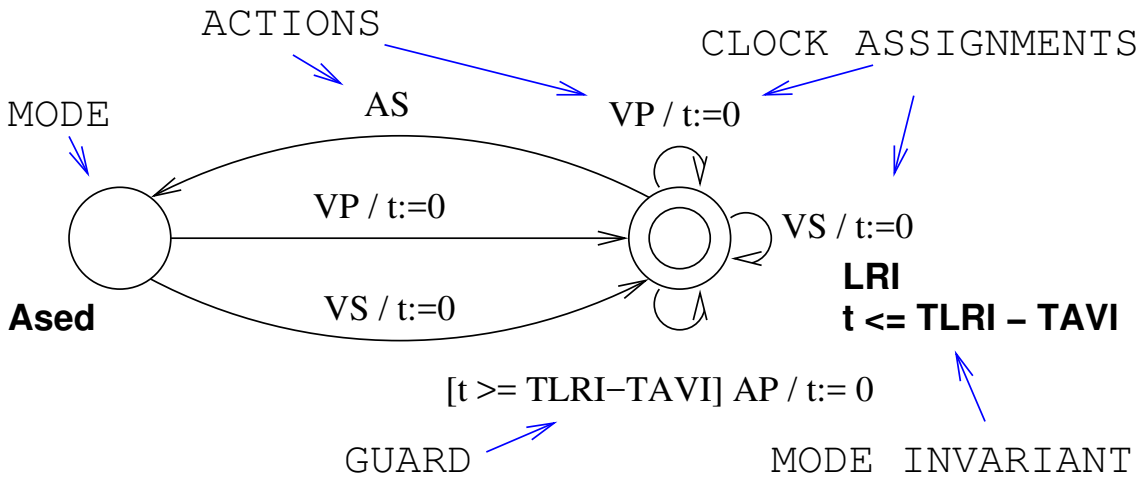
(Jiang et al., 2012) Z. Jiang, M. Pajic, S. Moarref, R. Alur, and R. Mangharam, Modeling and verification of a dual chamber implantable pacemaker, in Tools and Algorithms for the Construction and Analysis of Systems, LNCS vol. 7214, 2012.

Timed Automata

This TA (LRI) models the part of the ICP correcting bradycardia by keeping the heart rate above a minimum level.

AS: atrial sense AP: atrial pulse

VS: ventricular sense VP: ventricular pulse



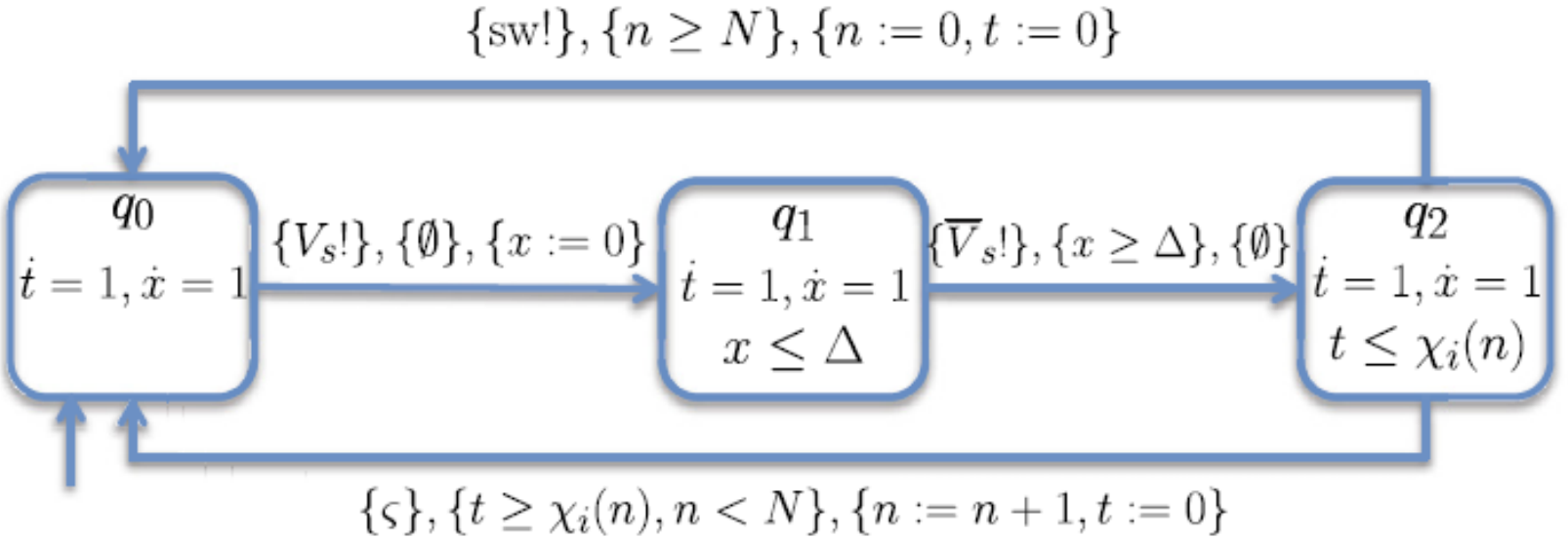
TLRI: lower rate interval TAVI: atrio-ventricular interval

MathWorks Models

MathWorks® Simulink is a widely used tool to model complex systems.

Heart models have been developed in Simulink with the *hybrid automaton* paradigm.

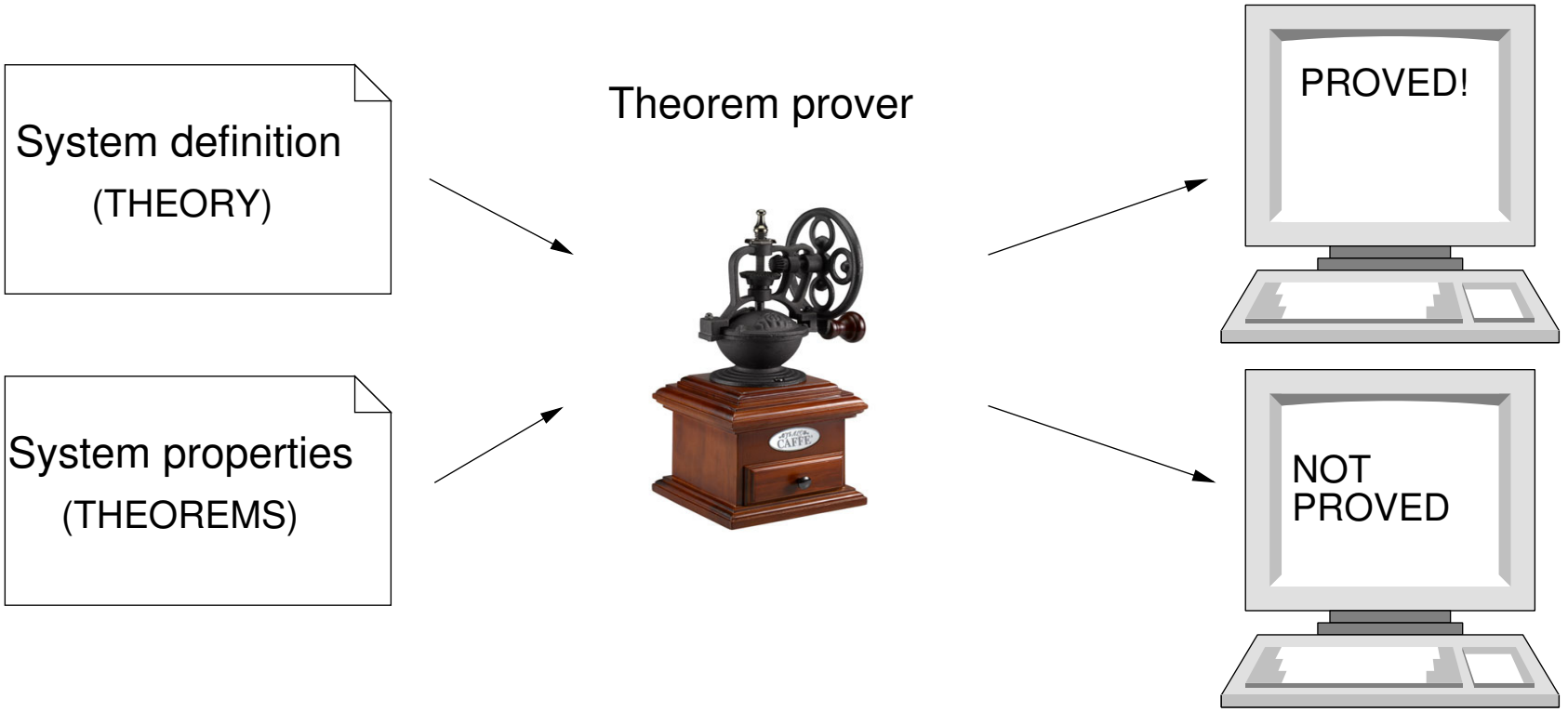
E.g., a *small* part of a heart model, the *SA stimulation automaton* (Chen et al., 2014):



Logic Specifications

Logic specifications model a system by stating its properties in a formal language.

Logic specifications are used for the formal verification of systems, using **automatic theorem proving**.



The PVS Theorem Proving Environment

The *Prototype Verification System* is an interactive theorem prover developed at SRI International by S. Owre, N. Shankar, J. Rushby, and others.

PVS has a rich **specification language** to define theories.

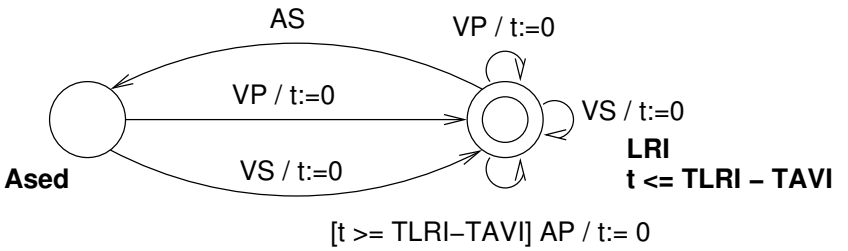
PVS has many powerful **inference rules** to prove theorems **interactively**.

- A user submits a theorem, then chooses inference rules until the proof ends successfully, or gets stuck.
- Often a single step is sufficient.

PVS has a **simulation extension** (*PVSio*). PVSio generates for each function in the declarative PVS language a procedure to compute its value (Lisp). PVSio can also compute functions with side effects, such as producing outputs.

From Timed automata to PVS

Modelling patterns have been defined to represent TAs in PVS executable specifications.



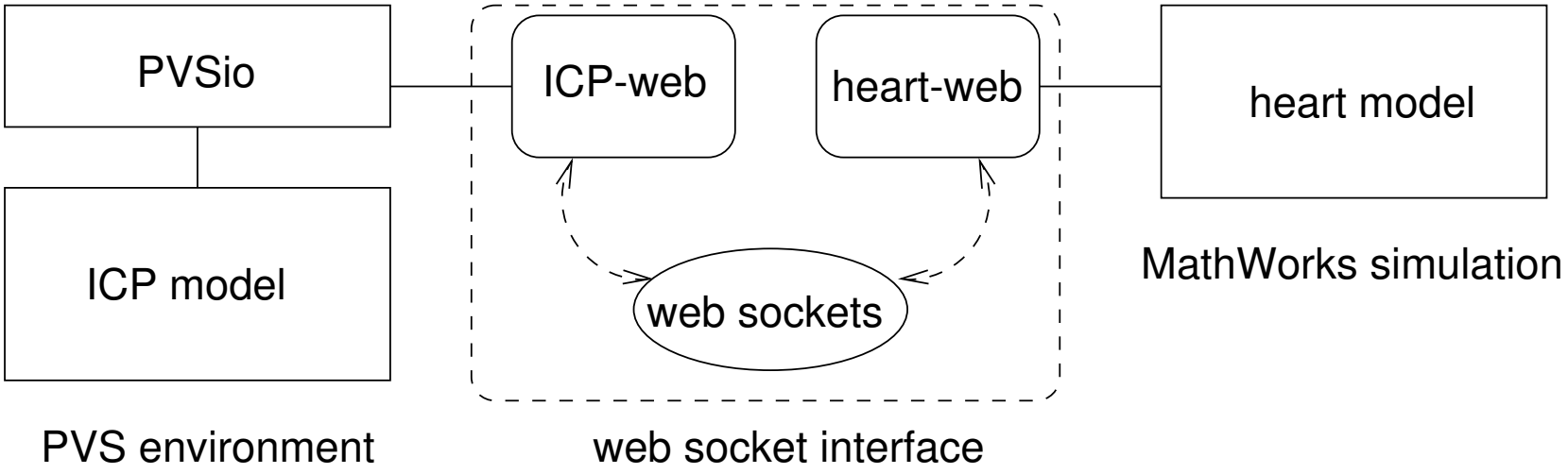

```
LRI: THEORY  BEGIN
Mode: TYPE = {LRI, ASed}
state: TYPE = [#
    time: real,
    loc: Mode #]

init_LRI: state = (# time := 0, loc := LRI #)

en_APout(st: state): boolean =    % enabling
    loc(st) = LRI AND time(st) >= TLRI-TAVI

APout(st: (en_APout)): state = (# time := 0, loc := LRI #)
..............
en_tau(st: state): bool = false % time-checking predicate
tau(st: state): state = st % timing function
END LRI
```

Integration: Architecture



The ICP and heart model communicate through the **WebSocket** protocol.

The two models can be executed on the same computer, or be **distributed** on different machines.

Co-simulation

The PVSio-web co-simulation environment integrates the two models:

- The Heart model simulation is started in the Simulink environment
- the PVSio environment is loaded with the PVS theory describing the Pacemaker
- PVSio-web requests the heart model to return heartbeat signals
- PVSio-web sends PVSio a function call with heartbeat signals as arguments
- PVSio computes the pacemaker model response and returns it to PVSio-web
- PVSio-web sends the pacemaker response to the heart model

Models' synchronization

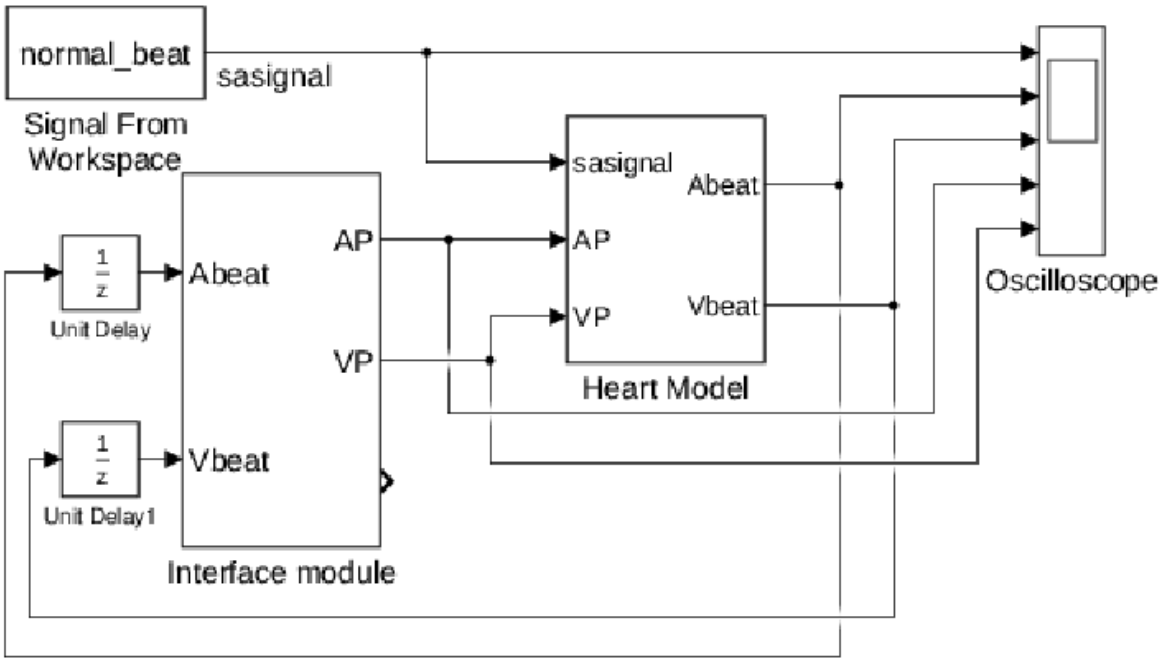
Synchronization between the two models is achieved by letting the heart model control the simulated time.

- The sampling time interval of the heart model is taken as the time unit.
- At each sampling time, a Simulink interface block (an S-function) is executed, it waits on the websocket for the PVSio-web request for the heartbeat signals, provides the signals and waits for the response, then it returns control to Simulink with the AP and VP signals.
- On the pacemaker side, time is updated by one unit.

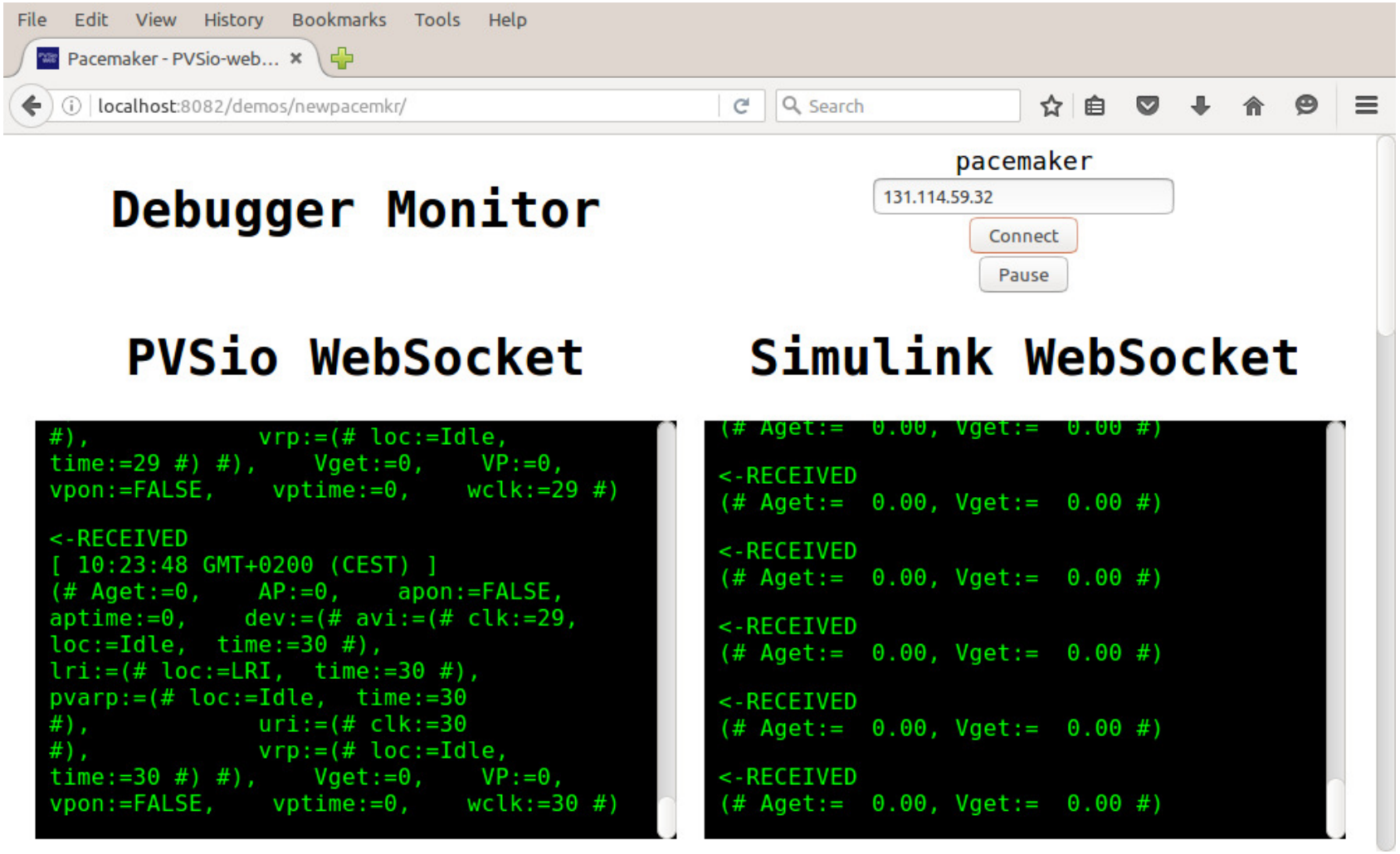
So the two models execute in lockstep with a blocking communications paradigm.

Models' synchronization

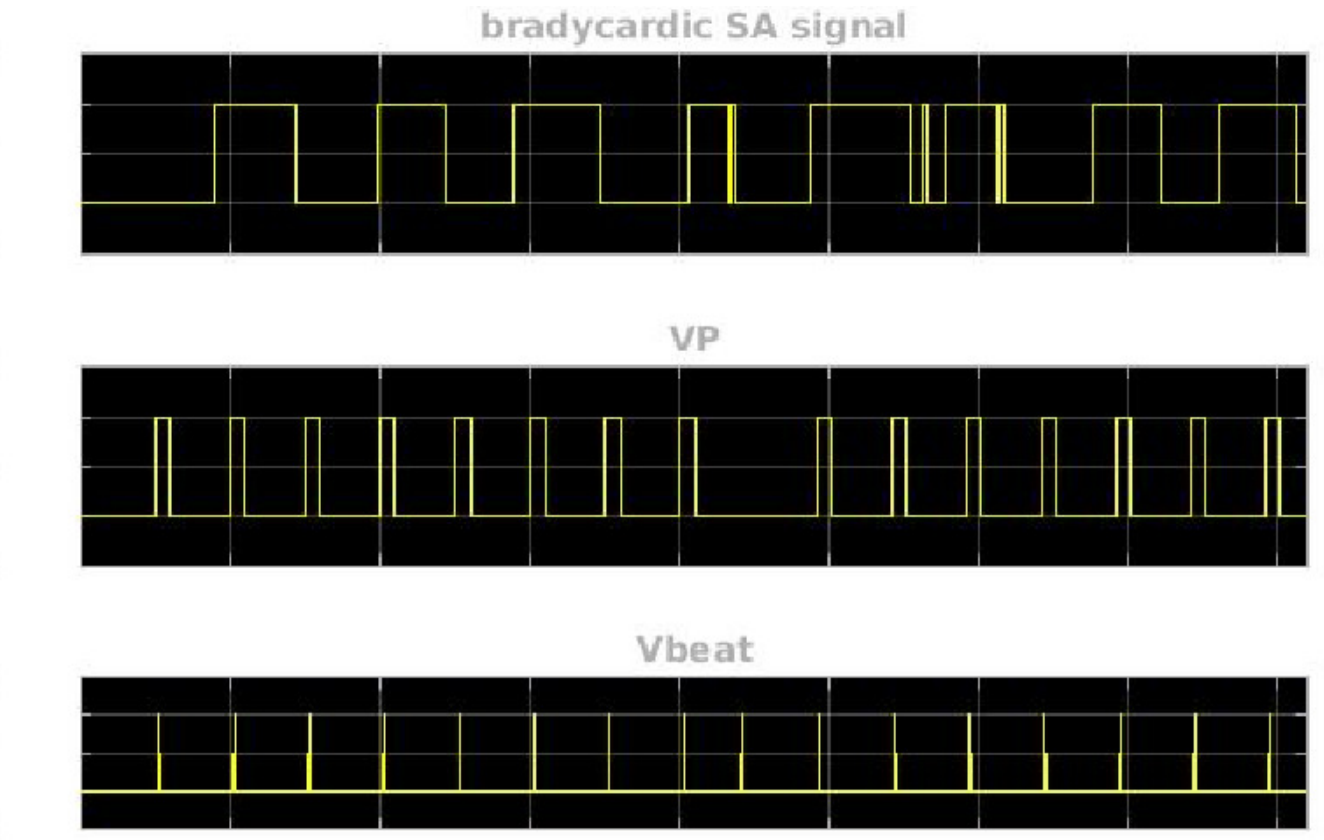
Simulink diagram of the extended heart model



The Co-simulation framework



Simulation output for bradycardia



Formal Verification

Formal verification is an important complement to simulation.

E.g., suppose we want to verify that the previously shown ICP system satisfies this property:

It is always the case that module LRI is in state LRI and its clock is reset when transition AP is executed.

```
lri_ap: LEMMA
  FORALL (s0, s1: State):
    per_APout(lri(s0)) AND s1 = APout(s0) IMPLIES
    mode(lri(s1)) = LRI AND time(lri(s1)) = 0
```

A single application of the *grind* rule (multiple simplifications) is sufficient.

```
Rule? (grind)
...
Q.E.D.
Run time  = 0.17 secs.
```

Formal Verification

More interesting properties require complex proofs.

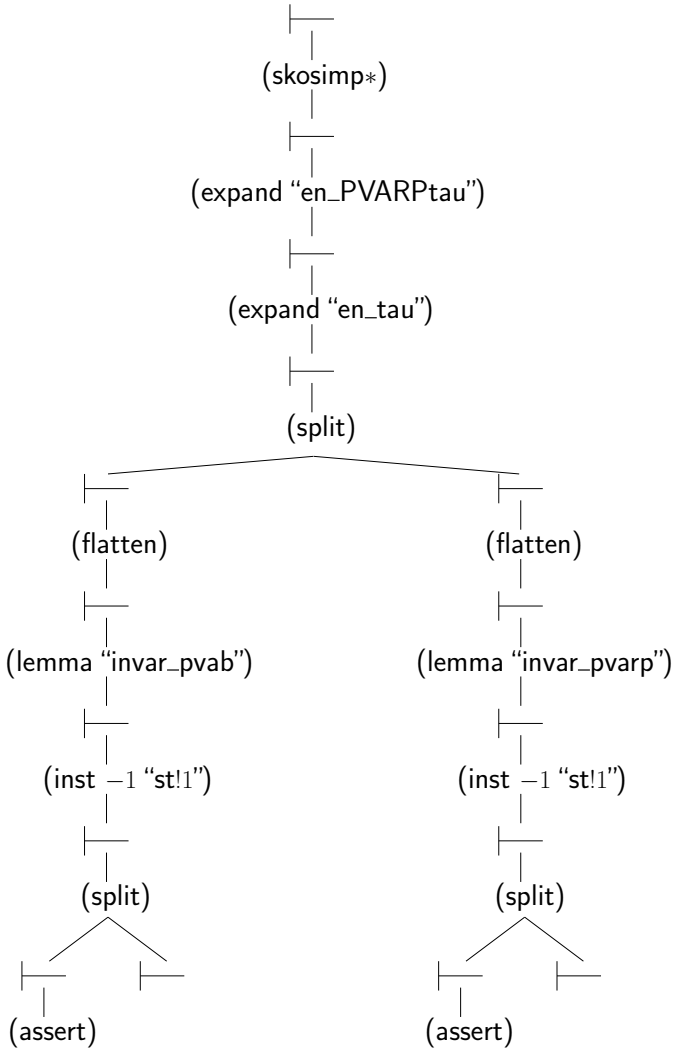
THEOREM: *All actions are mutually exclusive*.

The following Lemma states that the internal actions of PVARP are not enabled when AP is enabled.

```
ap_en_pvarptau: LEMMA
    forall (st: State):
        en_APevent(st)
        IMPLIES
        NOT en_PVARPtau(t)
```

Axioms that represents invariants of locations or expressing the semantics of TA committed locations are defined.

Proof tree: LEMMA ap_en_pvarptau



Conclusions

A method and its supporting tools have been developed to integrate discrete-time and continuous-time models built on different modeling paradigms.

This enables developers of discrete-time systems to model each component with the most appropriate tools and languages.

The resulting simulation environment can be executed on a laptop or on a distributed system. This affords more computing power when needed, and the convenience of switching easily between different models.

Using the PVS environment for the discrete-time model makes it possible to couple simulation to formal verification.

The method has been used to integrate ICP and heart models described in the literature, developed independently.

Additional work

PVS is currently used to specify, simulate and verify the software embedded in medical devices (e.g, infusion pumps) and requirements of integrated clinical environments.

PVS includes theories on mathematical analysis that can be used to model continuous systems. For example, we have used it to find safe range of operations for a simple non-linear control of a water tank (simple ordinary differential equation).

C. Bernardeschi, A. Domenici, P. Masci: Modeling communication network requirements for an integrated clinical environment in the Prototype Verification System. ISCC 2016: 135-140 2015 C.

Bernardeschi, A. Domenici:Verifying safety properties of a nonlinear control by interactive theorem proving with the Prototype Verification System. Inf. Process. Lett. 116(6): 409-415 (2016)

Sequent calculus

The sequent calculus works on formulae of a special form, called *sequents*, such as:

$$A_1, A_2, \ldots, A_n \vdash B_1, B_2, \ldots, B_m$$

where the A_i's are the *antecedents* and the B_i's are the *consequents*.

Each antecedent or consequent, in turn, is a formula of any form (it may contain subformulae with quantifiers and connectives, but not "sub-sequents").

The symbol in the middle ($\vdash$) is called a *turnstile* and may be read as "*yields*".

Informally, a sequent can be seen as another notation for

$$A_1 \wedge A_2 \wedge \ldots \wedge A_n \Rightarrow B_1 \vee B_2 \vee \ldots \vee B_m$$

Sequent calculus

A sequent is true if:

- Any formula occurs both as an antecedent and as a consequent; or
- any antecedent is false; or
- any consequent is true.

In the PVS prover interface, a sequent is represented as:

```
{-1}   A1
 ...
{-n}   An
  |-------
{1}    B1
 ...
{m}    Bm
```

Sequent calculus

The Sequent calculus has one axiom: $\Gamma, A \vdash A, \Delta$ where Γ and Δ are (multi)sets of formulae.

Inference rules:

$$\frac{}{\Gamma, A \vdash A, \Delta}\,\mathrm{axm} \qquad \frac{\Gamma \vdash \Delta, A \;\; A, \Gamma \vdash \Delta}{\Gamma \vdash \Delta}\,\mathrm{cut} \qquad \frac{A, A, \Gamma \vdash \Delta}{A, \Gamma \vdash \Delta}\,\mathrm{ctr\ L} \qquad \frac{\Gamma \vdash \Delta, A, A}{\Gamma \vdash \Delta, A}\,\mathrm{ctr\ R}$$

$$\frac{\Gamma \vdash \Delta, A}{\neg A, \Gamma \vdash \Delta}\neg L \qquad \frac{A, \Gamma \vdash \Delta}{\Gamma \vdash \Delta, \neg A}\neg R \qquad \frac{A, B, \Gamma \vdash \Delta}{A \wedge B, \Gamma \vdash \Delta}\wedge L \qquad \frac{\Gamma \vdash \Delta, A \;\; \Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \wedge B}\wedge R$$

$$\frac{A, \Gamma \vdash \Delta \;\; B, \Gamma \vdash \Delta}{A \vee B, \Gamma \vdash \Delta}\vee L \qquad \frac{\Gamma \vdash \Delta, A, B}{\Gamma \vdash \Delta, A \vee B}\vee R \qquad \frac{\Gamma \vdash \Delta, A \;\; B, \Gamma \vdash \Delta}{A \Rightarrow B, \Gamma \vdash \Delta}\Rightarrow L \qquad \frac{A, \Gamma \vdash \Delta, B}{\Gamma \vdash A \Rightarrow B, \Delta}\Rightarrow R$$

$$\frac{A[x \leftarrow t], \Gamma \vdash \Delta}{\forall x.A, \Gamma \vdash \Delta}\forall L \qquad \frac{\Gamma \vdash \Delta, A[x \leftarrow y]}{\Gamma \vdash \forall x.A, \Delta}\forall R \qquad \frac{A[x \leftarrow y], \Gamma \vdash \Delta}{\exists x.A, \Gamma \vdash \Delta}\exists L \qquad \frac{\Gamma \vdash \Delta, A[x \leftarrow t]}{\Gamma \vdash \exists x.A, \Delta}\exists R$$

axm: the axiom
cut: the cut rule
ctr: the contraction rules

The quantifier rules have caveats on the quantified variable.