# Towards a well-founded software component model for cyber-physical control systems

Jacques Malenfant

# Towards a well-founded software component model for cyber-physical control systems

Jacques Malenfant
Sorbonne Universités, UPMC Univ Paris 06, CNRS, UMR 7606 LIP6, 4 place Jussieu, F-75005, Paris, France
Email: Jacques.Malenfant@lip6.fr

***Abstract*—Cyber-physical control systems (CPCS), and their instantiation as autonomous robotic control architectures, are notoriously difficult to specify, implement, test, validate and verify. In this paper, we propose to integrate hybrid systems and their declension as hybrid automata and DEVS simulation models within a full-fledged and well-founded software component model tailored for CPCS. We present how the resulting comprehensive modeling tool can support the different phases of the software development to provide more reliable, more robust and more adaptable CPCS. The key concept is to provide components with a modeling and simulation capability that seamlessly support the software development process, from model-in-the-loop initial validations, until deployment time actual system verification.**

## I. Introduction

Robotic and cyber-physical control systems (CPCS) are very difficult to specify, implement, test, verify and validate correctly. CPCS bring computational systems and physical world phenomena tightly together, which they sense, control and actuate. CPCS and autonomous robotic have grown in complexity to a point where developing a correct control architecture now requires novel software engineering techniques and processes. Hence, computer scientists and roboticists are in urgent need for cooperation to develop specific robust software models and engineering processes.

Appopriately modeling CPCS requires behavioral models that capture both the discrete nature of computational systems and the continuous one of the physical world. Hybrid systems have been developed in the last decades to provide such a capability. However, this research has not yet reached software engineering for CPCS and robot systems development.

In this paper, we propose to integrate hybrid systems and software component models into a well-founded software model addressing many of the issues raised by CPCS and robot systems development: (1) behavioral specification, (2) model-in-the-loop simulation and validation, (3) unit and integration testing through software-in-the-loop simulation, (4) software verification and validation, deployment time system identification, control law synthesis and hardware-in-the-loop simulation for system validation and verification, (5) run time verification and (6) run time system self-adaptation.

In the rest of the paper, we first look at hybrid systems as introduced in mathematics. Then we introduce automata models of hybrid systems providing a modular modeling scheme. We next introduce our component model for CPCS. The paper ends with a short survey of the related work followed by a conclusion and a discussion of our perspectives.

## II. Hybrid systems

We now present hybrid systems [2], [3] as modeling tool for CPCS and robotic control architectures [4].

### A. Mathematical models of hybrid systems

Mathematicians have proposed several general models of controlled hybrid systems. Branicky's one [5] partition the hybrid state space $\mathcal{S} = \bigcup_{q \in Q} X_q \times \{q\}$ into a countable set of discrete states $\mathcal{Q} = \{q_0, q_1, ...\}$, each of them defining a continuous state space $X_q, q \in Q$. Jumps between discrete states occur upon discrete events, either a change in the value of discrete variable or a frontier condition met upon values of discrete and continuous variables. The continuous dynamics, controlled by control laws $U_q$ within each discrete state $q$, is defined by a set of equations $f_q$ that may be of different types, but often algebraic equations or differential ones such as:

$$\dot{\mathbf{x}}(t) = f_q(\mathbf{x}(t), u_q(t))$$

As many real systems are stochastic, stochastic extensions to hybrid system models have been proposed [6]. Randomness can show up in Branicky's model in different ways, such as: (1) stochastic continuous behaviours modeled as brownian motions and stochastic differential equations [7] or (2) stochastic jump transitions where the hybrid states after transitions follow density probability functions that depend upon the current hybrid state and the control signal.

### B. Case study

Our illustrative case study is an application where a portable computer exchanges large amounts of data with a server through WiFi [8]. When the WiFi bandwith is low, the system adapts itself by compressing the data. However, compression uses more PC processor, which consumes more battery. Hence, when the bandwith is high or the battery is too low, it stops compressing. The objective is to get the best possible data transfer rate, including the compression and decompression times, while maintaining the PC autonomy as long as possible. For the sake of simplicity, we consider only the components that are exchanging the data. Note that our goal is to illustrate the use of hybrid systems modeling for CPCS, not to provide a faithful complete model for this application.

Two variables are under the influence of the control:

- $d$: the data transfer rate in kbits/s.
- $b$: the battery level in mAh.

Around these variables, the model also defines:

- $p$ is the bandwith of the WiFi network in kbits/s.

- $\Delta B_{nc}$ (resp. $\Delta B_c$) is the battery draining rate when no compression (resp. compression) is used in mAh/s. They are assumed to be deterministic.
- $r_c$ is the rate of compression in kbits/s.
- $r_u$ is the rate of uncompression in kbits/s.
- $\tau_c$ is the compression factor, $0 < \tau_c < 1$.

The dynamics of the system exhibits two important behaviours. First, the evolution of the remaining level of energy in the PC battery, which has two modes depending on whether compression (c) is used or not (nc):

$$\dot{b}(t) = -\Delta B_c \qquad \dot{b}(t) = -\Delta B_{nc}$$

Second, the bandwith is assumed to be stochastic and, for the sake of simplicity, to follow a brownian motion expressed by a stochastic differential equation of the form:

$$\dot{p}(t) = \mu(p(t))dt + \sigma(p(t))dB(t)$$

where $\mu(p(t))dt$ represents a deterministic trajectory and $\sigma(p(t))dB(t)$ a stochastic perturbation.

We know intuitively that a threshold bandwith exists over which the data transfer rate is higher without compression and under which it is higher with compression. From the data transfer rates equations in the two cases, simple algebraic manipulations show that this threshold $p_s$ is:

$$p_s = \frac{(1-\tau_c)r_c r_u}{r_d + \tau_c r_u}$$

For example, for $r_c = 50$ kbits/s, $\tau_c = 0.4$, $r_u = 75$ kbits/s, $p_s \approx 23.68$ kbits/s. A simple control law with hysteresis can be adopted for the decision on the compression mode:

- If $p \geq P_{sup} > p_s$, go to the non compression mode;
- If $p \leq P_{inf} < p_s$, go to the compression mode.

For this example, we may use $P_{sup} = 25$ kbits/s and $P_{inf} = 21$ kbits/s. To get a longer autonomy on battery, we use a simple control law with a threshold $B$:

- If $b \geq B$, authorise the compression.
- If $b < B$, do not authorise the compression.

For example, if the battery has a capacity of $B_{max} = 5000$ mAh, the threshold can be set to $B = 2000$ mAh.

The figure 1 presents a hybrid systems model for this use case. The notation uses an automaton representation exhibiting different modes of continuous behaviours with differential equations and discrete transitions triggered by conditions on the variables. Discrete states are identified by the values of the discrete variable $q$, which can take three values: with compression ($C$), no compression ($NC$) and low battery ($LB$). When a transition occurs, variables can be set to initial values in the new discrete state. Hence, a transition has a trigger part and a variable assignment part, separated by "/". As states can also express assertions on their variables, the symbol "$\leftarrow$" means assignment while "=" is used in assertions. When resetting variables at time $t$, the notations $t^-$ and $t^+$ mean the value right before and right after the transition, respectively.

## III. Hybrid automata

Hybrid automata models allow to break complex models into smaller ones and compose them, providing modularity.

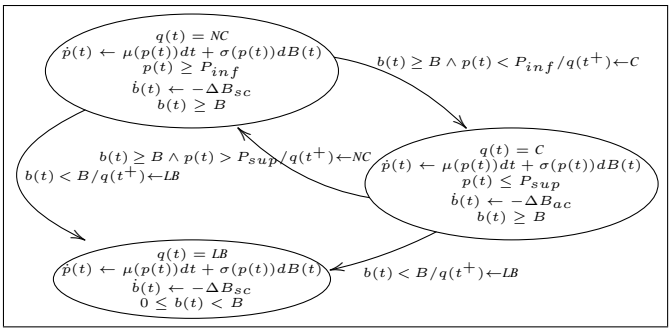


Fig. 1. Hybrid system baseline model for the data transfer use case.

### A. HIOA and TIOA

Henzinger [9], [10] as well as Lynch and her team [11], [12] were pionneers in the definition and study of *hybrid automata*. Both lines of works have proposed slightly different models and studied their properties, such as expressing progressive behaviours *e.g.*, that show no indefinite holds in the progression of time. Lynch's team's work provides a little more of the modularisation features we are looking for. First they define a hybrid automaton as follows [11]:

**Definition.** A hybrid automaton (HA) $\mathcal{H}$ is a tuple $(W, X, Q, \Theta, E, H, D, \mathcal{T})$ such that:

- A set $W$ of external variables and a set $X$ of internal variables, disjoint from each other. Define $V \triangleq W \cup X$.
- A set $Q \subseteq val(X)$ of states.
- A nonempty set $\Theta \subseteq Q$ of start states.
- A set $E$ of external actions and a set $H$ of internal actions, disjoint from each other. Define $A \triangleq E \cup H$.
- A set $D \subseteq Q \times A \times Q$ of discrete transitions.
- A set $\mathcal{T}$ of trajectories $\tau(t)$ for $V$ such that the values of the variables in $X$ remain in $Q$ for all $t \in dom(\tau)$. □

Lynch *et al.* [11] distinguish continuous variables from actions, assumed discrete. For modeling purposes, we adopt a model of HA where actions are expressed as modifications of discrete variables triggering transitions. Lynch *et al.* define the composition $\mathcal{H}_1 \parallel \mathcal{H}_2$, (and for this the compatibility of $\mathcal{H}_1$ and $\mathcal{H}_2$) by merging their variables, transitions and trajectories, if they respect rules such as not sharing internal variables.

With the above definition, composing two HA imposes that their external variables defined with the same name represent in fact the same variable, which implies that the two HA define the same trajectory for it. To avoid this complexity, Lynch *et al.* introduce hybrid I/O automata, which further distinguish among their external variables imported and exported ones, and then impose a unique producer for each of them.

**Definition.** A hybrid I/O automaton (HIOA) $\mathcal{A}$ is a tuple $(\mathcal{H}, U, Y, I, O)$ where:

- $\mathcal{H} = (W, X, Q, \Theta, E, H, D, \mathcal{T})$ is a hybrid automaton.
- $U$ and $Y$ partition $W$ into input and output variables, respectively.
- $I$ and $O$ partition $E$ into input and output actions, respectively. □

The next step is to define the composition of two HIOA:

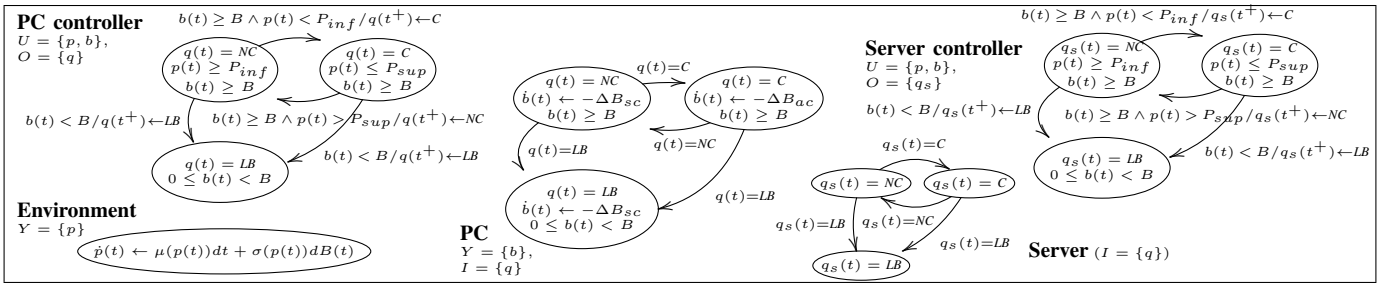


Fig. 2. Factorising the baseline model into HIOA, introducing the environment as a proper HIOA ($U$: imported continuous variables, $Y$: exported continuous variables, $X$: internal continuous variables; $I$: imported discrete variables, $O$: exported discrete variables, $H$: internal discrete variables).

**Definition.** Let $\mathcal{A}_1 = (\mathcal{H}_1, U_1, Y_1, I_1, O_1)$ and $\mathcal{A}_2 = (\mathcal{H}_2, U_2, Y_2, I_2, O_2)$, be two hybrid I/O automata, they are compatible if $\mathcal{H}_1$ and $\mathcal{H}_2$ are compatible and if $Y_1 \cap Y_2 = O_1 \cap O_2 = \emptyset$. If $\mathcal{A}_1$ and $\mathcal{A}_2$ are compatible, their composition $\mathcal{A}_1 \parallel \mathcal{A}_2$ is the tuple $\mathcal{A} = (\mathcal{H}, U, Y, I, O)$ where $\mathcal{H} = \mathcal{H}_1 \parallel \mathcal{H}_2$ and:

- $Y = Y_1 \cup Y_2$,
- $O = O_1 \cup O_2$, and
- $U = (U_1 \cup U_2) \cap Y$,
- $I = (I_1 \cup I_2) \cap O$. □

HIOA do not solve all the modularity concerns to model software architectures. Indeed, sharing continuous variables does not account for digital network communication, where only discretised values can be punctually exchanged. To cater for this restriction, Lynch's team introduces timed I/O automaton [12] *i.e.*, HIOA with no external continuous variables:

**Definition.** A timed I/O automaton (TIOA) is a hybrid I/O automaton $\mathcal{A} = (\mathcal{H}, U, Y, I, O)$ where $U = \emptyset$ and $Y = \emptyset$. □

While HIOA express tightly coupled models of centralised artefacts, TIOA express models of decentralised artefacts.

### B. Case study: continued

With HIOA, we can decompose our model to reveal actual components (Figure 2): the PC and server components, their controllers (separated, to better align with a distributed architecture) and the environment that models the WiFi bandwith. The five HIOA produce and consume external variables from each others by composition. However, this composition shares continuous variables between the HIOA, which makes the whole model more tightly coupled than the expected distributed architecture. To introduce more decoupling, HIOA must be turned to TIOA by replacing the sharing of continuous variables by discrete ones. This can be done by explicitly modeling sensors that convert a continuous signal into a discrete one, and the network by imposing a copy delay of discretised values (included as HIOA in Figure 3).

## IV. Cyber-physical software component models

In robotics, physical modelers can be seen as large sets of reusable models. Computer scientists can learn from this experience and use simulation to test and verify CPCS.

### A. From hybrid systems to simulation models

From a semantics point of view, hybrid systems are expressed in a declarative way: they say *what* are the models but not *how* to use them, aiming at formally proving properties. As hybrid systems are difficult to assess formally, another use is to simulate them *i.e.*, to compute trajectories of state variables as examples of realistic runs of the system. Simulation models express an operational semantics *i.e.*, *how* to perform these computations by employing simulation engines and defining simulation models that these engines can execute directly.

In simulation, DEVS [13] offers modular models capabilities. DEVS is based on discrete event simulation, splitting models into (1) *atomic models* representing monolithic models that input and output events and (2) *couped models* composing models by connecting outputs of ones to inputs of others.

DEVS defines a unique *simulation protocol* through which models are coordinated and activated repeatedly to perform their next transition while ensuring the order of the transitions and the communication of the external events. DEVS also proposes different ways to coordinate models:

- for MIL, an explicitly synchronised protocol where coordinators enforce a global simulation clock by distributing events and activating model transitions in strict order;
- for SIL and HIL, an implicitly synchronised one forcing the model local clocks to strictly follow the real time [14], models exchanging events directly as messages.

### B. From simulation models to CPS software components

Control engineering and robotics tell us that three major types of simulations are useful in CPCS development: (1) model-in-the-loop (MIL) simulations using models only to assess formal correctness; (2) software-in-the-loop (SIL) simulations using models with the software to validate it; and (3) hardware-in-the-loop (HIL) simulations using models for the environment only to verify the full system.

Our objective is to support these three kinds of simulations to cater for a *seamless* process going from a MIL simulation at the earlier stages to validate the specification of the system, to SIL simulations to perform unit testing, and then integration testing of the software and to HIL simulations, both at design and deployment time, to verify the system. To achieve this goal, we propose to build the simulation capability into the software architecture by making components able to hold and execute DEVS models packaged as plug-ins that can be easily switched on and off. They implement the DEVS models and simulation engines and automatise the deployment of the simulation architecture and the execution of runs.

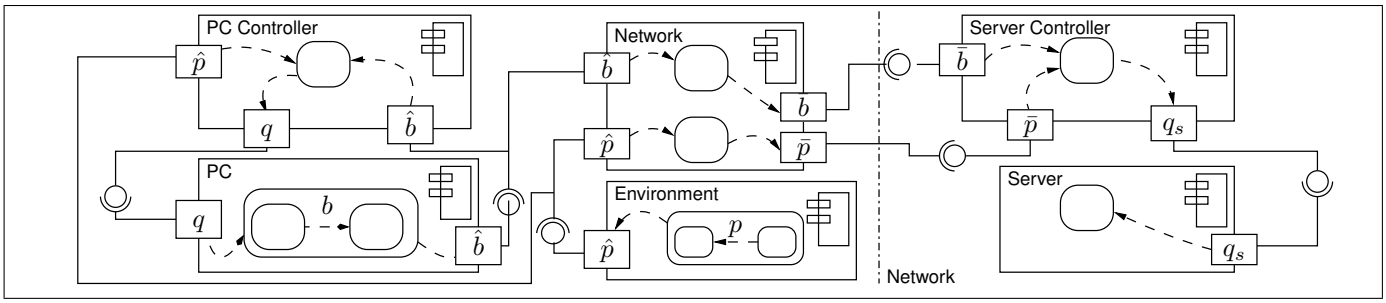


Fig. 3. Component model for the use case, with the flow of DEVS simulation models variables ($\hat{v}$: discretised variable; $\bar{v}$: delayed variable).

### C. Case study: continued

The figure 3 illustrates how HIOA/TIOA models are turned into DEVS models implemented on their corresponding software components. The four major components are implementing the PC side and server side data exchangers and their respective controllers. The figure shows the TIOA of Fig. 2 represented as round corner rectangles, with the ports presenting discrete variable values exchanged among the simulators. Round corner rectangles also represent TIOA which are composed of HIOA with internal round corner rectangles with their own internal flow of variable values.

For unit and integration testing through SIL simulations, the environment model capturing the bandwith evolution is implemented as a simulation model in the Environment component. Also, the Network component is introduced only for the purpose of MIL and SIL simulations to ease the testing. Indeed, for HIL, the network transmissions would pass through the actual network and exhibit this network real delays.

## V. Related work

Masaccio [10] proposes a component-oriented reformulation of hybrid automata but where "components" are modular models. Co-simulation, understood as the joint execution of a simulation with a software system, has been proposed to provide a form of SIL [14]–[16]. Robotics has produced an extensive literature on the joint use of software/hardware and simulation to test robots too large to cite here. None of these works consider the integration of software components and modular simulation models, but rather keeps them separated.

Most of the related works on the simulation of CPCS targets MIL for verification, yet some of them also consider SIL for software testing. However, none tries to define a full-fledged testing process. The most comprehensive work on this subject we know of is the Zohaib Iqbal's *et al.* [17], though they only consider discrete systems and keep software and simulators separated. De Roo *et al.* [18] propose the only work that we found addressing the integration of software and simulation, but they only use the continuous part of the modeling language and do not tackle the composability of models.

## VI. Conclusions and perspectives

We have discussed and illustrated how hybrid systems, hybrid automata and DEVS simulation models can be leveraged to propose a software component model tailored for CPCS and we illustrated it through a real-world case study. Such a model can support a full-fledged software engineering for CPS addressing many crucial issues for practioners. We are currently developing this component model for large-scale distributed CPCS in Java with integrated HIOA/TIOA and DEVS modeling and simulation capabilities. Afterwards, we will attack the software engineering processes *per se*.

## References


[1] D. Hristu-Varsakelis and W. S. Levine, Eds., *Handbook of Networked and Embedded Control Systems*. Birkhäuser, 2005.

[2] W. Heemels, D. Lehmann, J. Lunze, and B. D. Schutter, *Introduction to hybrid systems*. Cambridge University Press, 2009, pp. 3–30.

[3] J. Lunze and F. Lamnabhi-Lagarrigue, Eds., *Handbook of Hybrid Systems Control*. Cambridge University Press, 2009.

[4] M. Egerstedt, "Behavior Based Robotics Using Hybrid Automata," in *HSCC'00*, Springer-Verlag LNCS, vol. 1790. 2000, pp. 103–116.

[5] M. S. Branicky, "Studies in Hybrid Systems: Modeling, Analysis, and Control," Ph.D. dissertation, MIT, June 1995.

[6] J. Lygeros and M. Prandini, *Stochastic hybrid systems*. Cambridge University Press, 2009, pp. 249–276.

[7] B. Øksendal, *Stochastic Differential Equations*, 6th ed. Springer, 2013.

[8] J. Malenfant, M.-T. Segarra, and F. André, "Dynamic Adaptability: the MolèNE Experiment," in *Third Int. Conf. on Metalevel Architectures and Separation of Crosscutting Concerns, Reflection 2001*, Springer-Verlag LNCS vol. 2192. 2001, pp. 110–117.

[9] T. A. Henzinger, "The Theory of Hybrid Automata," Electrical Engineering and Computer Sciences, University of California at Berkeley, Tech. Rep., 1996, updated version of a paper published by the same author at LICS 1996, pp. 278–292.

[10] ——, "Masaccio: A Formal Model for Embedded Components," in *IFIP TCS 2000*, Springer-Verlag LNCS, vol. 1872. 2000, pp. 549–563.

[11] N. Lynch, R. Segala, and F. Vaandrager, "Hybrid I/O Automata," *Information and Computation*, no. 185, pp. 105–157, 2003.

[12] D. K. Kaynar, N. Lynch, R. Segala, and F. Vaandrager, "Timed I/O Automata: A Mathematical Framework for Modeling and Analyzing Real-Time Systems," in *24th IEEE Int. Real-Time Systems Symposium*, ser. RTSS'03, 2003, pp. 166–177.

[13] B. P. Zeigler and H. S. Sarjoughian, *Guide to Modeling and Simulation of Systems of Systems*. Springer, 2013.

[14] H. S. Sarjoughian, S. Gholami, and T. Jackson, "Interacting Real-Time Simulation Models and Reactive Computational-Physical Systems," in *2013 Winter Simulation Conference*, 2013, pp. 1120–1131.

[15] A. T. Al-Hammouri, M. S. Branicky, and V. Liberatore, "Co-simulation Tools for Networked Control Systems," in *HSCC'08*, Springer-Verlag LNCS, vol. 4981. 2008, pp. 16–29.

[16] Z. Zhang, E. Eyisi, X. Koutsoukos, J. Porter, G. Karsai, and J. Sztipanovits, "Co-Simulation Framework for the Design of Time-Triggered Cyber Physical Systems," in *ACM ICCPS'13*, 2013, pp. 119–128.

[17] M. Z. Iqbal, A. Arcuri, and L. Briand, "Environment modeling and simulation for automated testing of soft real-time embedded software," *SoSyM Journal*, vol. 14, pp. 483–524, 2015.

[18] A. de Roo, H. Sözer, and M. Aksit, "Composing domain-specific physical models with general purpose software modules in embedded control software," *SoSyM Journal*, vol. 13, pp. 55–81, 2014.