Software & Systems Modeling
https://doi.org/10.1007/s10270-017-0642-5

THEME SECTION PAPER

Rigorous design of cyber-physical systems

Linking physicality and computation

Simon Bliudze[1] · Sébastien Furic[2] · Joseph Sifakis[3] · Antoine Viel[4]

Received: 29 February 2016 / Revised: 13 February 2017 / Accepted: 10 November 2017

Abstract
Cyber-physical systems have developed into a very active research field, with a broad range of challenges and research directions going from requirements, to implementation and simulation, as well as validation and verification to guarantee essential properties. In this survey paper, we focus exclusively on the following fundamental issue: how to link physicality and computation, continuous time-space dynamics with discrete untimed ones? We consider that cyber-physical system design flow involves the following three main steps: (1) cyber-physical systems modeling; (2) discretization for executability; and (3) simulation and implementation. We review—and strive to provide insight into possible approaches for addressing—the key issues, for each of these three steps.

Keywords Cyber-physical systems design · Structural equational modeling · Modelica · Linear graphs · Bond graphs · Idealization · Abstraction · Hybrid dataflow networks · Discretization · Language embedding

1 Introduction

Over the past 8 years, cyber-physical systems have developed into a very active research field. There already exists a rich literature about research challenges and directions spanning all aspects of cyber-physical system design from requirements to implementation and simulation as well as validation and verification to guarantee essential properties [9,21,22,28,35,43,48,54]. Despite considerable effort of the involved research communities, it is important to recognize that we are still very far from reaching the desired degree of domain integration. Each involved research community privileges aspects and approaches they are familiar with. One can distinguish three main work directions.

The first direction, centered on the Modelica language, privileges a pragmatic and practically oriented approach focusing mainly on tools. The strength of Modelica over other approaches is that it allows freedom of expression, in particular differential algebraic equations (DAE), by keeping seamless all aspects related to execution e.g., causality of models, treatment of Zenoness. The second builds on dataflow languages including Matlab/Simulink and synchronous languages in general. It focuses on extending these languages to support directly the description of systems of ordinary differential equations (ODE) and event-driven mechanisms. The third takes as the basis model, hybrid automata that directly integrate event-driven mechanisms and ODE. This model has been thoroughly studied, especially theoretical aspects including semantics, analysis and synthesis techniques. It is closer to the world of event-driven systems e.g., programming languages and execution platforms.

Communicated by Prof. J. Sztipanovits, M. Broy, and H. Daembkes.

✉ Simon Bliudze
simon.bliudze@inria.fr
http://www.bliudze.me/simon

Sébastien Furic
sebastien.furic@inria.fr

Joseph Sifakis
joseph.sifakis@univ-grenoble-alpes.fr

Antoine Viel
antoine.viel@siemens.com

1 INRIA Lille – Nord Europe, Parc scientifique de la Haute Borne, 40 avenue Halley, 59650 Villeneuve d'Ascq, France

2 INRIA Centre de Paris, 2 rue Simone Iff, 75589 Paris, France

3 Verimag, Bâtiment IMAG, 700, avenue centrale, 38401 Saint Martin d'Hères, France

4 Siemens Industry Software SAS, 14 Boulevard de Valmy, 42300 Roanne, France

The objective of cyber-physical system modeling is twofold. Firstly, *simulation of such models provides means for validating the system design*. In particular, this is achieved by exhibiting behaviors that emerge from composing continuously evolving physical processes with discrete control sub-systems. The latter react to events generated by the physical processes (detecting zero-crossings), by changing their operational modes and resetting parameter values. The second objective of cyber-physical systems design is to *provide the means for the generation of executable code for the discrete control sub-system*.

To achieve these objectives, the paper advocates the integration of the three work directions described above in an ideal cyber-physical systems design flow involving three steps. Each work direction prevails in one of the considered steps. The focus is exclusively on the key issue: *how to link physicality and computation, continuous time-space dynamics with discrete untimed ones*? We do not address other important issues that are generic such as requirements specification, validation and verification, analysis and performance, architectures etc.

Linking the realms of physical systems and computing systems requires a better understanding of differences and points of contact between them. How is it possible to define models of computation encompassing quantities such as physical time and resources? Significant differences exist in the approaches and paradigms adopted by physical and computing systems engineering.

Cyber-physical systems design flows should consistently combine component-based frameworks for the description of both physical and cyber systems. The behavior of components for physical systems is described by equations, while cyber components are transition systems. Furthermore, connectors for physical systems are just constraints on flows, while for cyber systems they are synchronization events (interactions) with associated data transfer operations. Physical systems are inherently parallel, while computational models are built out of interacting components that are inherently sequential.

Physical systems engineering is *primarily* based on continuous mathematics, while computing is rooted in discrete non-invertible mathematics. It relies on the knowledge of laws governing the physical world as it is, while computing is rooted in a priori concepts. Physical laws are declarative by their nature. Physical systems are modeled by differential equations involving relations between physical quantities. They are governed by simple laws that are to a large extent, deterministic and predictable. Synthesis is the dominant paradigm in physical systems engineering. We know how to build artifacts meeting given requirements (e.g., bridges or circuits), by solving equations describing their behavior. By contrast, state equations of very simple computing systems, such as an RS flip-flop, do not admit linear representations in any finite field. Computing systems are described in executable formalisms such as programs and machines. Their behavior is intrinsically non-deterministic. For computing systems, synthesis is in general intractable. Correctness is usually ensured by a posteriori verification. Non-decidability of their essential properties implies poor predictability.

Despite these differences, both physical and computing systems engineering share a common objective which is the study of dynamic systems. We attempt below a comparison for a simplified notion of dynamic system.

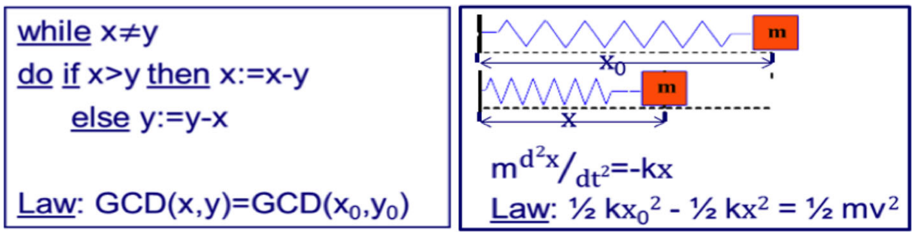
A dynamic system can be described by equations of the form $X' = f(X, Y)$ where X' is a "next state" variable, X is the current state and Y is the current input of the system. For physical systems, variables are functions of a single real-valued time parameter. For computing systems, variables range over discrete domains. The next state variable X' is typically $\mathrm{d}X/\mathrm{d}t$ for physical systems, while for computing systems it denotes system state in the next computation step.

Figure 1 shows a program computing the GCD of two integer variables and a mass-spring system. The operational semantics of the programming language associates with the program a next-state function, while the solution of the differential equation describes the movement of the mass. The set of reachable states of the program is characterized by the invariant $\mathrm{GCD}(x, y) = \mathrm{GCD}(x_0, y_0)$, where x_0 and y_0 are the initial values of x and y, respectively. This invariant can be used to prove that the program is correct if it terminates. In exactly the same manner, the law of conservation of energy $kx_0^2/2 - kx^2/2 = kv^2/2$ determines the movement of the mass as a function of its distance from the origin x, its initial position x_0, and its speed v.

This example illustrates remarkable similarities and also highlights some significant differences. Programs can be considered as scientific theories. Nonetheless, they are subject to specific laws (invariants) enforced by their designers and which are hard to discover. Finding program invariants is a well-known non-tractable problem. On the contrary, all physical systems, and electromechanical systems in particular, are subject to uniform laws governing their behavior.

Another important difference is that, for physical systems, variables are functions of a single time parameter. This drastically simplifies their specification and analysis. Operations on these variables are defined on streams of values, while, as a rule, operations on program variables depend only on current state.

In contrast to physical system models that inherently have time-space dynamics, models of computation, e.g., automata and Turing machines, do not have a built-in notion of time. For modeling/simulation purposes, *logical* time is represented by state variables (clocks) that must be increasing monotonically and synchronously. Nonetheless, clock synchronization can be achieved only to a certain degree of precision and can generate significant performance over-



```
while x≠y
do if x>y then x:=x-y
     else y:=y-x

Law: GCD(x,y)=GCD(x0,y0)
```

m
x_0
m
x
$m\,d^2x/dt^2=-kx$
Law: $\frac{1}{2} kx_0^2 - \frac{1}{2} kx^2 = \frac{1}{2} mv^2$

Fig. 1 Behavior and laws characterizing a GCD program and a spring-mass system

head. This notion of logical time as a state variable explicitly handled by execution mechanisms, significantly differs from physical time modeled as an ever increasing time parameter. In particular, logical time may be blocked or slowed-down.

We consider that cyber-physical system design flow involves the following three main steps:

1. *Cyber-physical systems modeling* Designers need methods and tools for faithful modeling of complex systems. A first difficulty to overcome is structural modeling of physical systems. If for simple electrical or mechanical systems, theory allows rigorous modeling, this is far from being the case for complex electromechanical systems. Furthermore, we badly need support for deciding whether the models are semantically sound, detecting Zeno situations and for simplifying models modulo some abstraction criterion. Last but not least, we need adequate languages supporting modularity of descriptions and allowing mixed coordination mechanisms e.g., dataflow and event driven.
2. *Discretization for executability* Given an equational model of a cyber-physical system e.g., in Modelica, we need discretization techniques to produce executable models. These should be supported by theory allowing decision at reasonable costs of model causalization. For cyber-physical system models that are amenable to execution, discretization techniques should be compositional, in particular to support heterogeneity of solvers.
3. *Execution and implementation techniques* For a discretized model, a network of components with a given dataflow relation and discrete synchronization events, we need techniques for efficient code generation and implementation. This is already a non-trivial problem for synchronous languages such as Lustre [17] or Simulink [20]. It is desirable that the generated code preserve the structure of the source model. Generating monolithic code is a much simpler problem but it precludes composition of sub-systems with existing systems and multi-site implementation. To ensure coherency between simulation and implementation the upstream code generation process should be common. It could diverge later to take into account specific requirements.

The paper assumes that the reader has some basic knowledge of Mechatronics (some notions of Mechanical Engineering or Electrical Engineering may suffice) and is familiar with the modeling techniques for hybrid and timed systems. It is structured as follows. In Sect. 2, we discuss issues related to cyber-physical systems modeling. In Sect. 3, we discuss discretization of cyber-physical system models and in particular aspects dealing with executability and quality. In Sect. 4, we discuss the principle for translating discretized models of cyber-physical systems into executable event-based languages using embeddings. These are structure-preserving homomorphisms that guarantee by-construction modularity. Section 5 concludes with a discussion pointing out future research directions.

2 Cyber and physical system modeling

In order to achieve the first objective mentioned in the introduction—that is allow faithful and efficient simulation of models exhibiting complex interactions between continuous physical and discrete control processes—semantics of cyber-physical systems models must account for the physicality of the underlying processes, most significantly, their continuous nature. Modeling languages should be adequately restricted and equipped to be amenable to analysis. In the following sub-sections, we discuss three aspects of cyber-physical systems modeling that are key to addressing these requirements.

2.1 Structural equational modeling of physical systems

Complex physical systems are typically described as networks of interconnected elements, whose conceptual limits depend on various assumptions made by the model designer. Derivation of equations from such networks by hand requires a lot of discipline from users: In particular, one has to make sure that the resulting system of equations captures invariants known as *laws of conservation* (like Kirchhoff laws for instance, which express conservation of energy and charge in the electrical domain). Corresponding equations actually depend on the interconnection of constitutive elements of models, and it is hard in general to find sets of *independent* such equations when models are complex. In the subsequent sections, we present domain-independent techniques based on dedicated structures, from which independent equations capturing the laws of conservation can be derived in a systematic way. The use of such structures characterizes the so-called *structural equational modeling* of physical systems.

Beyond the problem of completing systems of equations by means of independent additional constraints, another important aspect of modeling, for which structural equational modeling of physical systems is of great help is the debugging of physical models. When a modeling language does not offer syntactic support for structural equational modeling, one can only resort to lower-level constructs (such as equations) to build correct models. This is error-prone, and possible resulting errors are left undetected by modeling language compilers. This makes debugging of complex models extremely cumbersome [25,26]. Notice that such errors appear precisely in complex models. On the other hand, with syntactic support for structural equational modeling, many programming errors are simply impossible to commit—because users no longer have to directly, explicitly *program* conservation laws—and, when an error is detected, its cause can be explained at a higher level of abstraction [25,26]: it is clearly more informative for a user to be told that, for instance, two designated high-level model elements are in conflict, than to be given their associated equations with a message saying that these equations constitute a singular sub-system.

The problem of deriving equational models from physical systems has led to various approaches based on physical analogies. In Sects. 2.1.3 and 2.1.4, we describe two such approaches among the most popular ones, namely the *linear* and *bond graphs*. But before describing these approaches, we quickly recall some elements of the history of mechanical–electrical analogies.

2.1.1 Understanding physical phenomena with the help of analogies

Mechanical–electrical analogies can be used to explain mechanical phenomena from electrical ones and vice-versa by identifying suitable *conjugate variables* whose product has the dimension of a power (i.e., energy per unit of time)[1] and from which it is possible to derive similar equations in both domains (these equations may even have the same numerical solutions under suitable normalization).

In the force–voltage analogy (also known as Maxwell's analogy), conjugate variables are called *effort variables* (representing voltage in the electrical domain and force in the mechanical domain) and *flow variable* (representing current in the electrical domain and velocity in the mechanical domain), respectively.

In the force–current analogy (also known as Firestone's analogy), conjugate variables are called *across variable* (representing voltage in the electrical domain and velocity in the mechanical domain) and *through variable* (representing current in the electrical domain and force in the mechanical domain) respectively.

Moreover, mechanical–electrical analogies can be generalized to other physical domains, through the identification of suitable conjugate variables. This leads to uniform treatment of physical systems and allows general formalisms to be devised to treat a wide variety of physical modeling problems.

2.1.2 A preliminary remark regarding causality

We typically assume that physical models can be interpreted as "effects following causes". This is important if one needs to explain or even predict phenomena. This "effects following causes" property is often referred to as *physical causality*.

It could then be tempting to consider physical causality as a fundamental concept in the design of cyber-physical system tools. However, we will prefer another notion of causality in practice, called *computational causality* which, in addition to physical causality, also captures some computational aspects. In particular, we need validation procedures which ensure the existence of a processing order of elementary operations required to produce simulation results.

This is why "causality" typically refers to computational causality in the context of the modeling approaches presented in the following sections. In particular, the bond graph approach traditionally requires *causality analysis* [31] to be performed over models, both as a validation step and as an ordering of operations required to compute simulation results.

2.1.3 The linear graph approach to physical system modeling

We now introduce linear graphs—the first of the two approaches, mentioned in the opening of this section. In the linear graph approach [50], basic elements representing elementary physical phenomena (storage and dissipation, to which one can add *sources* to model boundary conditions, as well as *transducers* to model energy transduction)[2] are associated with arcs of an oriented graph whose nodes represent in some sense "boundaries" between identified phenomena. This approach makes use of Firestone's analogy to identify conjugate variables which appear in *constitutive equations* characterizing each elementary phenomenon. In Table 1, we

[1] For this reason, approaches to physical system modeling based on these analogies are often termed *energetic approaches*.

[2] In this short introduction to linear graphs, we will not consider transducers for the sake of simplicity. The interested reader can refer to any decent book about linear graphs for more information about transducers.

Table 1 Examples of conjugate quantities in linear graphs

Physical domain	Across quantity (unit)	Through quantity (unit)
Electrical	Voltage drop (V)	Current (A)
Mechanical (translational)	Velocity difference (m/s)	Force (N)
Fluid	Pressure difference (P)	Volumetric flow rate (m^3/s)

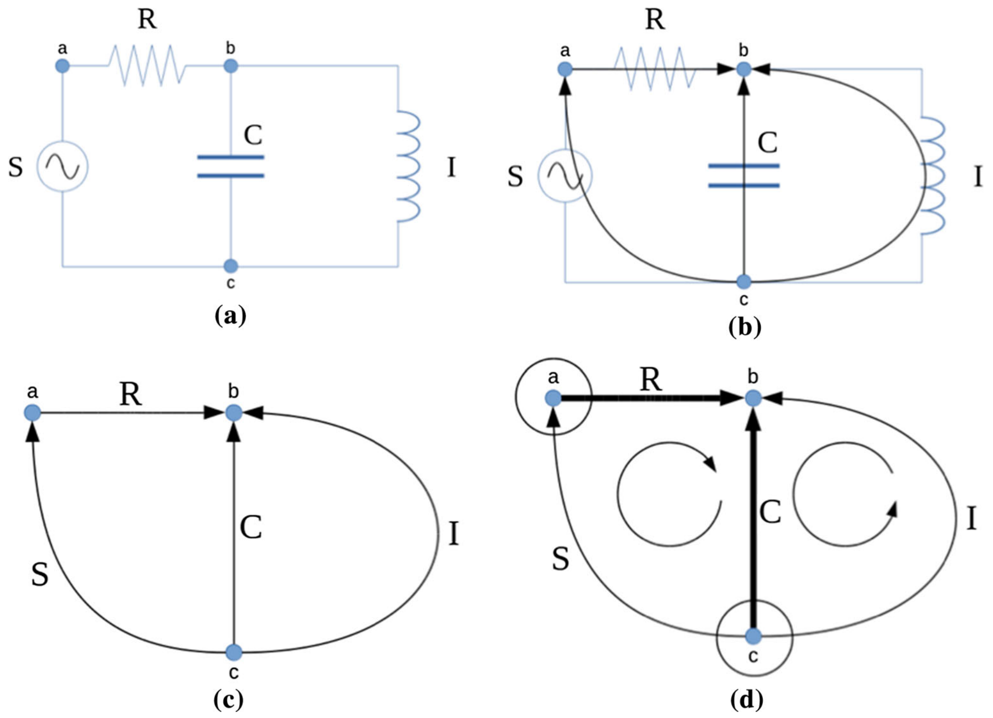


Fig. 2 Linear graph modeling example. **a** A simple electrical system, **b** linear graph construction, **c** linear graph of the system, **d** orientation of cycles

give a few examples of *conjugate quantities* over which conjugate variables range in the corresponding physical domain.

As a direct consequence of Tellegen's theorem [49], linear graphs ensure global energy balance of models: this is of considerable help in practice to enforce correctness of models.

To illustrate the use of linear graphs in physical system modeling, consider the simple electrical circuit of Fig. 2a. Its linear graph representation can be constructed by identifying nodes and elementary dipoles in the model (which are given a conventional orientation). Figure 2b illustrates the construction of the graph and Fig. 2c shows the final result.

A system of differential and algebraic equations can be derived from this model by merging:

- constitutive equations attached to interconnected elements (e.g., $v_R = Ri_R$ for the resistor), and
- connection equations associated with the graph structure (in our example, these correspond to applications of Kirchhoff laws, since we are in the electrical domain).

Connection equations can be obtained by considering fundamental circuits and fundamental cut-sets in the graph (defined with respect to a reference spanning tree chosen arbitrarily). Figure 2d shows a particular spanning tree of the linear graph of Fig. 2c (the root is b and branches correspond to edges R and C, in bold on the picture). From this tree, two fundamental circuits can be identified (curved arrows indicate their orientation, chosen arbitrarily), which yield two across equations (taking circuit orientation in account):

$$v_S + v_R = v_C,$$
$$v_I = v_C.$$

Similarly, two through equations can be derived from the cut-sets incident on a and c, respectively (appearance on one side of the equal sign depends on the direction of the incident arc with respect to the node under consideration):

$$i_R = i_S,$$
$$i_S + i_C + i_I = 0.$$

The equational model associated with our example is obtained by merging the sets of across equations, through equations, and constitutive equations:

$$
\begin{aligned}
&v_S = V_0 \sin\ (\omega t + \varphi),\\
&v_R = R i_R,\\
&\dot{q_C} = i_C,\\
&q_C = C v_C,\\
&\dot{p_I} = v_I,\\
&p_I = L i_I,\\
&v_S + v_R = v_C,\\
&v_I = v_C,\\
&i_R = i_S,\\
&i_S + i_C + i_I = 0.
\end{aligned}
$$

From this equational model, it is possible to derive an executable model as follows. Notice first that only q_C and p_I are defined by means of differential equations: they hold the state of the system. From their current value, we expect that it is possible to approximate the evolution of the system by integrating the differential equations numerically.[3] This requires the original equational model to be transformed into a sequence of elementary computational steps following a certain discretization technique. In this example, since all equations can be rewritten to turn them into explicit functions of some known variables,[4] it is possible to derive the following simple sequence of assignments (inputs are the current values of q_C and p_I, and outputs are their "next" values, i.e., their approximated values after ϵ units of time):

$$
\begin{aligned}
&v_C := q_C / C,\\
&v_I := v_C,\\
&v_R := v_C - V_0 \sin\ (\omega t + \varphi),\\
&i_R := v_R / R,\\
&i_S := i_R,\\
&i_I := p_I / L,\\
&i_C := -i_S - i_I,\\
&q_C^{\text{next}} := q_C + \epsilon i_C,\\
&p_I^{\text{next}} := p_I + \epsilon v_I.
\end{aligned}
$$

Well-known techniques based on variants of matching algorithms in suitable bipartite graphs can be used to obtain assignment sequences from equational models in an efficient way. We advise the interested reader to refer to, e.g., Fritzson [23] for a detailed overview of this.

2.1.4 The bond graph approach to physical system modeling

Bond graphs [31]—the second approach mentioned in the opening of this section—constitute another means to yield equational models of physical systems. This approach is also energetic: in bond graphs, power exchanges are materialized by *bonds* (i.e., "arcs" of bond graphs, graphically represented by half-arrows) connecting elements. This approach makes use of Maxwell's analogy to identify conjugate variables appearing in constitutive equations. Table 2 gives some examples of conjugate quantities following the bond graph convention (notice the difference with linear graphs regarding the mechanical convention).

In bond graphs, *terminal elements* representing physical phenomena of interest in a model (e.g., power supply, dissipation and storage) have to interact through a *junction structure* composed of interconnected *zero-* and *one-junctions*, and *transducers*.[5] Zero-junctions are n-port elements whose purpose is to impose, on each connected bond, the same effort and the zero algebraic sum of flows (the sign of each flow in the sum depending on the orientation of the corresponding bond). Similarly, one-junctions impose, on each connected bond, the same flow and the zero algebraic sum of efforts.

To illustrate the idea with a concrete example, we will again consider the example of Fig. 2a. The construction of the junction structure and its ramifications toward terminal elements is illustrated in Fig. 3a (no orientation is chosen for the moment).

The idea is the following. Considering the fundamental circuits associated with an arbitrary spanning tree of the underlying graph structure (as in the case of linear graphs), we associate a one-junction with each fundamental circuit. Notice that, by construction, there is exactly one edge per fundamental circuit that does not belong to the spanning tree and, moreover, this edge belongs to a unique fundamental circuit. Consequently, we choose this edge as the *reference edge* of the associated fundamental circuit. Connections are then introduced as follows:

– terminal elements associated with reference edges (the source and the inductor in our example) are directly connected to their associated one-junction, and
– each remaining terminal element (the resistor and the capacitor in our example) is connected to an interme-

[3] Analytical integration is generally not practicable in real-world applications, for performance reasons and also because tools cannot often be given a symbolic version of equations to be solved.

[4] This is of course not possible for any equation in general. More elaborated techniques (including some fix point determination) will be needed in case inversion is not possible by means of simple symbolic manipulations.

[5] As in the case of linear graphs, we will not consider transducers in this short presentation of bond graphs. Again, we advise the interested reader to refer to any decent book about bond graphs for more information.

Table 2 Examples of conjugate quantities in bond graphs

Physical domain	Effort quantity (unit)	Flow quantity (unit)
Electrical	Voltage (V)	Current (A)
Mechanical (translational)	Force (N)	Velocity (m/s)
Fluid	Pressure (P)	Volumetric flow rate (m^3/s)

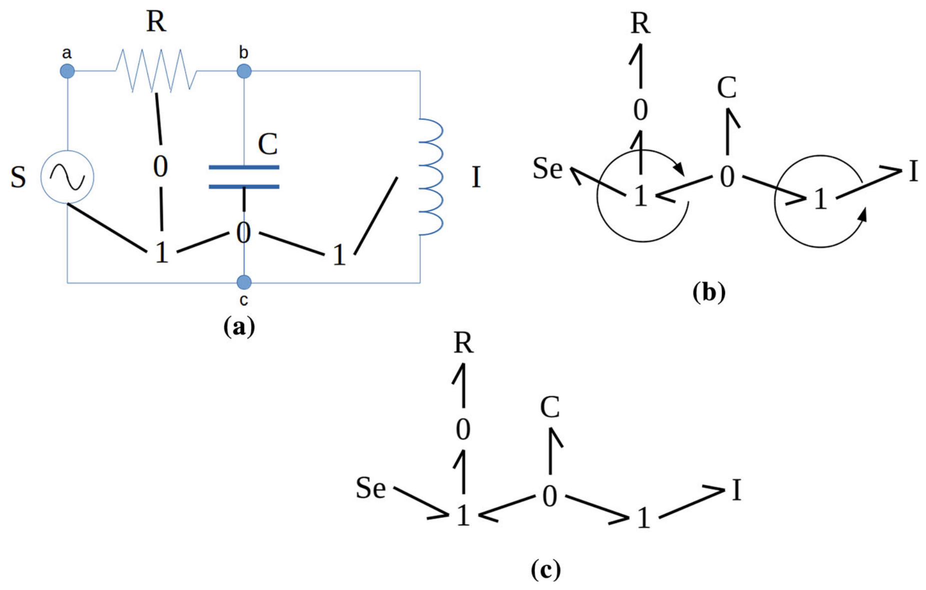


Fig. 3 Bond graph modeling example. **a** Bond graph construction, **b** bond graph preorientation, **c** conventionally oriented bond graph

diate zero-junction, which in turn is connected to the one-junctions corresponding to fundamental cycles containing the terminal element (actually the zero-junction plays the role of a proxy for the associated terminal element as seen from the one-junctions).

In order to obtain connection equations, we need to fix an orientation for each bond of the graph. These orientations have to be consistent with the corresponding linear graph model defined before. To this end, we choose an equivalent orientation (curved arrows of Fig. 3b correspond to orientation of fundamental cycles of Fig. 2d). We can proceed as follows. First, we give the unoriented bond graph a globally consistent preorientation:

– bonds connecting terminal elements to junctions are oriented toward the terminal element,
– remaining bonds (connecting zero-junctions to one-junctions) are oriented toward the one-junction if the edge whose zero-junction is a proxy of is negatively oriented with respect to the reference orientation, and toward the zero-junction otherwise.

The obtained bond graph, depicted in Fig. 3b, is however not correctly oriented according to the usual bond graph convention, which requires external power exchanges (i.e., associated with source elements) to have opposite signs with respect to their linear graph counterparts. The following correction step has then to be applied:

– direction of bonds incident to zero- and one-junctions, which are directly connected to, respectively, an effort and a flow source have to be inverted, and
– direction of bonds connecting zero- and one-junctions to, respectively, flow and effort sources have to be inverted.

The final, conventionally oriented bond graph is depicted in Fig. 3c.

As in the case of linear graphs, connection equations can be obtained from bond graphs by exploiting their structure. Remember that we carefully constructed our bond graph according to a reference spanning tree and corresponding cycle basis of the associated linear graph (Figs. 2d, 3b). Elementary reasoning about properties of this transformation reveals a one-to-one mapping between, respectively:

– fundamental cycles of the linear graph and one-junctions of the bond graph, and
– nodes of the spanning tree of the linear graph, except the root, and zero-junctions of the bond graph.

Remember also that, by definition, zero- and one-junctions impose algebraic sum of, respectively, flows and efforts to be zero. We can then exploit the aforementioned mappings and

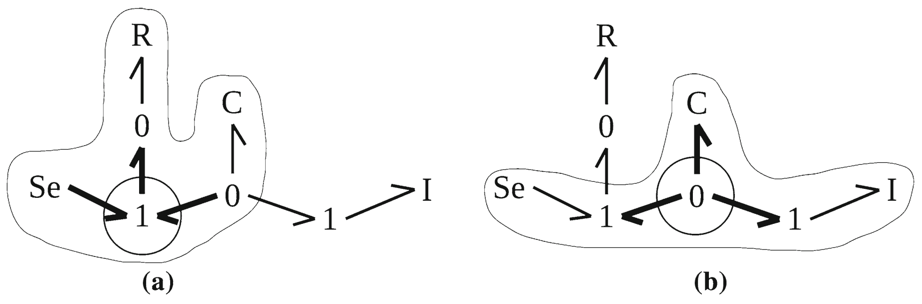


Fig. 4 Derivation of effort and flow equations from a bond graph. **a** Derivation of an effort equation, **b** derivation of a flow equation

these properties of junctions to directly obtain the desired *effort* and *flow equations* from our bond graph. Figure 4a shows how to derive the effort equation associated with the leftmost one-junction. The neighbors of this junction are the source and the two zero-junctions serving as proxies for the resistor and the capacitor. According to orientation of bonds incident to the junction (in bold), we obtain the following effort equation:

$$v_R = v_S + v_C.$$

Similarly, Fig. 4b shows how to derive a flow equation from a zero-junction: we simply have to exchange the roles of zero- and one-junctions in previous recipe.[6] We obtain the following flow equation:

$$i_S + i_C + i_I = 0.$$

Gathering all connection equations with constitutive equations of terminal elements finally leads to the following equational model:

$$\begin{aligned}
v_S &= -V_0 \sin\ (\omega t + \varphi),\\
v_R &= R i_R,\\
\dot{q_C} &= i_C,\\
q_C &= C v_C,\\
\dot{p_I} &= v_I,\\
p_I &= L i_I,\\
v_R &= v_S + v_C,\\
v_I &= v_C,\\
i_S + i_C + i_I &= 0,\\
i_R &= i_S.
\end{aligned}$$

Notice that the sign of v_S differs from the sign of v_S in the equational model obtained previously from the linear graph. This is because of the bond graph's sign convention having for sole consequence a difference of *interpretation* of signs with respect to its linear graph counterpart.

As in the case of linear graphs, executable models can be obtained from bond graphs by means of matching algorithms [23].[7]

2.1.5 A comparison of linear graph and bond graph approaches

Both approaches have their own merits and drawbacks that we discuss below.

Linear graphs benefit from strong mathematical foundations: graph theory provides many useful results to prove the correctness of algorithms processing graph structures, whereas Tellegen's theorem [49] has played a central role in the use of linear graphs for physical system modeling. Moreover, linear graphs naturally lead to compositional approaches in the design of physical models. Indeed, nodes of linear graphs represent boundaries between identified physical phenomena in models: as such, they constitute natural connection points with a topological interpretation.[8] As a consequence, there is typically a nice mapping between the "technological representation" of a model and its linear graph representation (see the electrical example of Fig. 2b, where both representations are superimposed).

On the other hand, as noticed by some proponents of the bond graph approach [30], linear graphs impose a certain choice for across and through quantities that does not always correspond to a natural choice as far as the physical nature of quantities is considered. Recall Table 1 in Sect. 2.1.3: in order to preserve the mapping between technological representations of mechanical models and their linear graph representations, velocity needs to be an across quantity. But from a physical point of view, velocity is associated with the

[6] The noteworthy symmetry between both recipes reflects the *duality* property of efforts and flows in the underlying physical model [30]. In contrast, linear graphs do not enjoy such a symmetry.

[7] Causality analysis [31] can also be used to obtain executable models directly from the bond graph structure. However, this technique has been superseded in most industrial tools by the more modern matching approaches which are more general (they don't require the initial model to be a bond graph) and more efficient (they achieve polynomial time performance in the worst case).

[8] They are actually "virtual measurement points" according to the common interpretation of linear graphs following Trent [50].

motion of particles, like electrical current. From this point of view, velocity and current should be considered similar. However, linear graphs treat them as of different nature (the same remark can be formulated for force and voltage drop).

Compared to linear graphs, bond graphs clearly put more emphasis on the physical nature of models. As such, they constitute a tool of choice for practitioners looking for insight into characterization of fundamental physical phenomena in physical systems. The apparent cryptic aspect of bond graph notation (wrongly criticized in our opinion) reveals the profound meaning of lumped parameter physical models.

On the other hand, bond graphs do not enjoy compositionality properties of linear graphs. In particular, a bond graph representing a network of interconnected elements cannot be directly obtained by composing the bond graphs representing its sub-networks. This makes their direct application to the design of physical modeling languages problematic: in practice, transition from technological representations to bond graph representations requires substantial analysis of models. Furthermore, bond graphs lack the ability to be reduced to a *normal form* (the profound cause lies in the ambiguity between absolute and relative potentials in this formalism). Finally, bond graphs lack a validation procedure that ensures soundness of models in the general case[9]: causality analysis of bond graphs [31] can be shown to be neither necessary, nor sufficient for this purpose as soon as parameters of models are "taken to the limit" (e.g., a resistance becomes infinite). However, this limitation of bond graphs exists mainly for historical reasons, since results from graph theory [34,42] show that, under reasonable restrictions, bond graphs and linear graphs are equally safe from this point of view.[10]

Ongoing work by Furic [24] suggests that it is possible to merge bond graphs and linear graphs into a new mathematical structure that enjoys the compositionality features of linear graphs as well as the regularity of bond graphs regarding the physical nature of the underlying phenomena.

2.2 Semantic issues

To allow separate and independent compilation of model subsystems, cyber-physical systems semantics must be modular. It must be independent of parameters, such as simulation step size and precision, which are specific to simulators and code generators, but not to the underlying physical processes. Furthermore, to allow faithful, but efficient simulation, cyber-physical systems semantics must be amenable to model simplifications by abstraction of low-level details of the physical processes. Ensuring correct abstraction and modularity is not trivial already for purely software systems—it becomes a major challenge in the cyber-physical systems context, where continuous evolution of physical processes is combined with discrete events, e.g., generated by digital controllers or user actions.

2.2.1 Abstraction, idealization and non-determinism

There exist two sources of non-determinism: (1) non-determinism in the system being modeled and (2) non-determinism resulting from the abstraction of low-level detail of the underlying physical processes. For an example of system non-determinism, consider an object thrown vertically and suppose that a discrete event is triggered when (and if) the object attains a given threshold altitude. A slight variation in the initial conditions can affect whether the event is triggered, potentially leading to radically different behaviors of the composed system. Similarly, a physical system can behave differently depending on the timing of a particular control decision taken by a digital controller. Such timings may depending on unpredictable variables, such as the state of a cache of the processor performing the computations of the controller. Thus, the absence of determinism in the initial conditions of a physical process or timing of control decisions gives rise to non-determinism intrinsic to the system being modeled.

An example of non-determinism originating from behavior abstraction is provided by Bliudze and Furic [10]. It comprises a ramp voltage source and two fuses with *different* rated currents (Fig. 5). A fuse behaves like an electrical switch that is closed by default but that can eventually become open if the branch current exceeds a limit. Consider first a model, where only the fuse behavior is *idealized*, i.e., the voltage generated by the source is initially zero, then linearly augments until reaching the maximum value $V_{\max}$, whereas the fuses melt *instantaneously*, when the current exceeds the corresponding ratings. The only behavior of this system is the following: both the voltage and the current raise continuously until the first fuse melts, following which the current instantaneously falls and thereafter stays constant at zero; the remaining fuse does not melt. Consider now another variant of this model, where the behavior of the source is also idealized: initially, the generated voltage is zero until a certain moment, when it instantaneously becomes $V_{\max}$. The behavior of such model is non-deterministic: depending on which of the instantaneous actions is "faster", either one or both fuses would melt.

When non-determinism occurs as a consequence of abstraction and has significant influence on the behavior of the model, this means that important information has been discarded, e.g., relative speed of ramping up the voltage generated by the source and of fuse melting in the above example. When the model is used for simulation, the objective is, precisely, to identify the parameters that have strong influence on

[9] This is a necessary condition for executability.

[10] In Sect. 2.1.4, we used these results to build the model based on bond graphs.

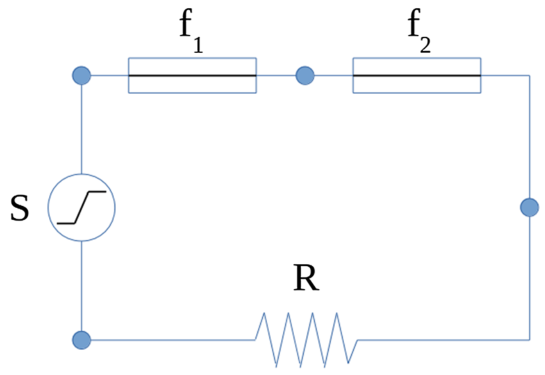


Fig. 5 An electrical circuit with a ramp voltage source and two fuses

the system behavior. One cannot rely on (1) running the simulation a sufficient number of times for all possible behaviors to manifest themselves and (2) the engineers being capable of identifying the source of non-determinism, when it does manifest. Thus, to be useful for simulation, model semantics has to be sufficiently rich to allow simulators to identify and report sources of non-determinism.

The fundamental issue, central to the ability of the model semantics to capture the kind of phenomena described above, is the model of time. Physical processes evolve in continuous time, which is most intuitively modeled by positive real numbers. Given a model of sufficiently high fidelity, one could argue that any change (typically, modifying the parameters of physical processes) triggered by a discrete event has non-zero duration. This would allow describing the evolution of such systems within the same model of time, based on real numbers.

However, high-fidelity models are not practical for at least two reasons. Fine details of physical processes are usually not known, due to imperfections of materials, environment noise and other types of uncertainty. Therefore, one usually employs idealized models often resorting, as in the above example, to instantaneously resetting a signal to a known value reached at the end of the transition phase. Furthermore, when used for simulation, high-fidelity models—especially, when representing the mode-changing dynamics—are computationally very expensive.

Idealized models can exhibit sequences of cascading instantaneous value changes (i.e., an ordered sequence of discrete events occurring at the same real time). Models based on the standard real-number representation of time cannot capture such situations. Two alternative models have been proposed: *superdense time* [36,37] and *non-standard time* [7,11,45]. The former takes the model of time to be the lexicographically ordered Cartesian product $\mathbb{R} \times \mathbb{N}$, where the moments (t, n) and (t, m) are considered simultaneous. The latter relies on non-standard analysis [39,44] and takes the model of time to be the field of non-standard real numbers that, in addition to usual, *standard reals*, contains *infinitesimals*—numbers, whereof absolute values are strictly positive, but less than any positive standard real—and the *infinitely great*, which are the inverses of infinitesimals. Every finite non-standard real can be uniquely represented as the sum of a standard real—its standard part—and an infinitesimal. The projection of finite non-standard reals onto the set of standard reals, which discards the infinitesimal part of this decomposition, is called *standardization*. Thus, both the superdense and the non-standard models of time allow arbitrary numbers of distinct, but simultaneous time moments.

Another consequence of idealization is the emergence of so-called *Zeno behaviors*, when an infinite number of events happen within a finite time span. The classical example models a ball falling from a height h_0 and losing a fraction $p \in [0, 1]$ of its speed due to non-elastic shock of bouncing off the ground. In an idealized model, the speed of the ball is instantaneously reset from its current value v to $(p - 1)v$. It is easy to see that the infinite sequence of moments when the ball bounces of the ground converges to a finite moment, called the *Zeno point*.

Notice that the behavior of the bouncing ball model described above is not defined beyond the Zeno point. Thus, a faithful model has to comprise two *modes*: before and after the Zeno point. However, the problem remains of detecting the Zeno behavior and deciding when the transition between the modes can be taken without the risk of considerable deviation from ideal behavior. Although this problem has been studied by a number of authors [32,40,56], most existing tools leave to the model designer the responsibility of providing, in the model, additional information, such as patterns of energy dissipation, which allows deciding when the transition is to be taken. Although for simple examples, such as the bouncing ball, providing such information is relatively easy, doing so for complex realistic models is not practical. When the model is used for simulation, this approach defeats the purpose, which consists, precisely, in discovering such information. We conjecture that a sufficiently rich semantic model, for instance based on the non-standard model of time, might be the key to addressing this problem.

2.2.2 Modularity

Physical system engineers rarely start from scratch when designing new models. Most of the time, they reuse already existing models and libraries. As in general-purpose programming, modifying existing models (e.g., by replacing given parts by "compatible" ones) or building models hierarchically requires means to incrementally assemble parts and, ideally, the possibility to type-check the model and compile its components separately.

Modularity and typing of physical models are, from our point of view, in their very early days. Indeed, while a certain degree of modularity is supported in some languages, such as VHDL-AMS (this language offers syntactic constructs to build linear graph elements explicitly), users still experience unexpected simulation errors due to the lack of expressive power of the equation layer. A well-known example is the ideal electrical switch model (which imposes a null current when open and a null voltage when closed). This model is perfectly legal in VHDL-AMS; however, there should not exist any cut-set or any circuit only made of such switches in the linear graph structure of the bigger model they belong to. Indeed, if a cut-set of the graph contains only switches and if those switches happen to be open at the same time during simulation, then the system's Jacobian matrix becomes singular (its rank is no longer maximal). Similarly, if a circuit of the graph contains only switches, the same symptom occurs when the switches happen to be closed at the same time. Clearly, some improvements are necessary to enhance expressiveness of the equation layer: in our example with switches, it is possible to correct the model by hand, by combining "conflicting" switches into one (by or-ing or and-ing their switch conditions). However, this operation cannot be performed automatically by the compiler. A possibility is to resort to non-standard semantics: indeed, a switch can actually be modeled by a linear (although modulated) resistor capable of taking infinite or infinitesimal resistance depending on the switch condition. Since ideal switches become ordinary linear models under this semantics, they can be (for instance) automatically combined by means of appropriate rewrite steps (of the same kind as those already implemented in most physical modeling languages).

Modularity deserves an additional comment in the context of languages with no support for structural equational modeling (such as Modelica). Let us consider the well-known 3-pin ideal operational amplifier, whereof the Modelica code is shown in Listing 1.

Listing 1 Modelica model of a 3-pin ideal operational amplifier

```
model IdealOpAmp3Pin
  "Ideal operational amplifier (norator-nullator
    pair), but 3 pins"
  Interfaces.PositivePin in_p "Positive pin of
    the input port";
  Interfaces.NegativePin in_n "Negative pin of
    the input port";
  Interfaces.PositivePin out "Output pin";
equation
  in_p.v = in_n.v;
  in_p.i = 0;
  in_n.i = 0;
end IdealOpAmp3Pin;
```

It should be noted that this model is only valid if the out pin is connected from the outside (indeed, no equation involves any of the connection variables of this pin inside the model). Moreover, in order to deliver meaningful results, this model requires additional collaboration from its environment: either a positive feedback loop or a negative feedback loop should exist in the final model involving this operational amplifier. This is due to the fact that the first equation actually results from a simplification of the model under this assumption. As a consequence, it is easy to build wrong models (i.e., having no physical interpretation) by means of this ideal operational amplifier: many modeling assumptions are simply not enforced by any language construct. More precisely, this model does not correspond to a valid piece of a linear graph. Indeed, deriving the equations for such an ideal operational amplifier using the linear graph approach, would require (the effect of) a modulated voltage source with infinite gain driven by a voltage sensor with infinite impedance: as in the case of ideal switches, we see again here the potential of the coupling of linear graphs with an adequate non-standard-based semantics (allowing infinite quantities to be explicitly represented).

Type checking of physical models reflects the current situation with modularity: to the best of our knowledge, no type system has been proposed so far in the field of physical modeling languages that would be strong enough to guarantee modularity. The state of the art today mostly consists in type-checking expressions involving physical signals (including physical connections) in programs. This is far from sufficient to really enable modeling languages to protect their own abstractions—if we suppose that abstractions are physical sub-models—as exemplified above. Consequently, desirable properties such as checking of sub-model physical compatibility are simply not possible today at the type level: in many cases, such incompatibilities are discovered at execution time, if ever.

2.3 Hybrid models for cyber-physical systems

Hybrid models combine discrete event and continuous dynamics. In equational models, there may be hidden discrete events when differential equations are associated with constraints on continuous variables that specify regions of validity. These discrete events characterize crossing of regions with different dynamics without there being a jump in the values of the continuous variables. As a rule, hybrid models encompass continuous and discrete change by allowing, in particular, jumps of values and non-determinism.

Equational models are declarative by their nature. It is not always possible to find a partial order of evaluation specifying which unknown is determined by which equation. Translation of equational models into causal models is a step toward discretization discussed in the next section. There are two possible avenues for the definition of hybrid models.

2.3.1 Hybrid automata

Hybrid automata, introduced by Henzinger [29], can be obtained by associating systems of differential equations

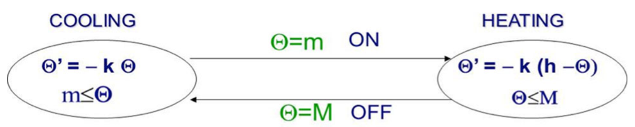


Fig. 6 A hybrid automaton modeling a thermostat

to the states of a discrete model. In that case, states can be interpreted as modes where time may progress until the state is left by executing some transition. Figure 6 shows a model of a thermostat consisting of two modes `COOLING` and `HEATING`, with associated differential equations modeling the evolution of temperature Θ. The discrete events `ON` and `OFF` are triggered when the minimal temperature m, respectively, the maximal temperature M is reached. The constraints $m \leq \Theta$ and $\Theta \leq M$ determine the domain of application of the differential equations.

The operational semantics of hybrid automata is defined by means of relations involving two types of transitions: *timed transitions*, labeled by positive real numbers, represent the time elapsed at a certain mode when the corresponding validity constraint continuously holds, and *discrete event transitions* that may involve jumps of continuous values. An important semantic issue is the urgency [14] of discrete events in the presence of time non-determinism. For instance consider in the Thermostat model that the guard $\Theta = M$ is replaced by an interval guard $M_l \leq \Theta \leq M_u$ and the constraint associated to state `HEATING` by $\Theta \leq M_u$. Such a condition is practically more realistic because it replaces a sharp event, impossible to implement, by another that must happen between given bounds. This introduces non-deterministic choice in the model that should be resolved by respecting the constraint $\Theta \leq M_u$. Logically in this case, the event `OFF` can occur whenever the guard is respected without violating the upper bound M_u.

Composition of hybrid automata is event driven. It is defined for hybrid automata having disjoint continuous variables. At semantic level, it boils down to building a product automaton by synchronizing as required the discrete events. A product state is a mode whose evolution is driven be the union of the equations of constituent modes and the composition of the corresponding constraints. The latter defines different ways for dealing with urgency [14].

2.3.2 Hybrid dataflow networks

Hybrid dataflow networks can be obtained by enriching continuous dataflow networks with discrete events that can in particular stop and start integration processes. These events can be either generated from continuous values when some condition is met or be provided by the external environment.

Given a system of explicit differential equations that represents a network of physical components, it is always possible to obtain a continuous dataflow network (continuous block diagram) involving primitive operators. The translation is compositional and systematic as illustrated by Fig. 7 that shows the network corresponding to the system of differential equations

$$v_1' = f_1(x, v_1, v_2) = ax + bv_1 + cv_2,$$
$$v_2' = f_2(x, v_1, v_2) = dx + ev_1 + fv_2$$

and its refinement, replacing the blocks that compute functions f_1 and f_2 by equivalent sub-networks that use primitive operators. In Fig. 7, v_1 and v_2 are state variables, x is an input variable.

Each equation is implemented by a loop involving an integrator. In addition to dataflow inputs and outputs, integrator nodes have a discrete event input `start` that initiates the integration with a given initial value.

The semantics of such networks is well understood: each node continuously computes a function from input streams of values to an output stream. Computation is synchronously parallel. This model puts emphasis not on the physical components and the way they are interconnected, but rather on the mathematical operators and their causal dependency. Such a translation proves to be very useful as it can provide a basis for discretization: when integrators are replaced by iterative integration programs (solvers), the resulting model is discrete dataflow which is the basic model for languages such as Lustre [17] or Simulink [20].

We can extend the continuous dataflow model to encompass discrete events following an approach similar to Simulink/Stateflow, where a discrete Stateflow controller enables and disables Simulink blocks. To this end, we introduce a `when` operator that receives as an input a continuous signal y and is parametrized by a guard G and a function H on y. The `when` operator is a trigger that simultaneously produces two events `stop` and `start` when the guard $G(y) =$ `true`. The parameter of start is the value $H(y)$. Using `when` operators, one can model the effect of a transition of a hybrid automaton (see Fig. 8). This raises similar issues regarding urgency. Nonetheless, for hybrid automata the modes are mutually exclusive which means that only a single integration process is running at a time.

Although hybrid automata and hybrid dataflow networks have the same expressive power, the underlying composition mechanisms are very different. Hybrid automata privilege event-driven composition while, for dataflow networks, composition is by giving constraints on flow variables (very often equations). The latter holds any time, while the former defines instants in the system execution where composed components can interact. As a rule these instants are determined dynamically over system execution.

The advantage of hybrid dataflow networks over hybrid automata is that they are intrinsically parallel; parallelism

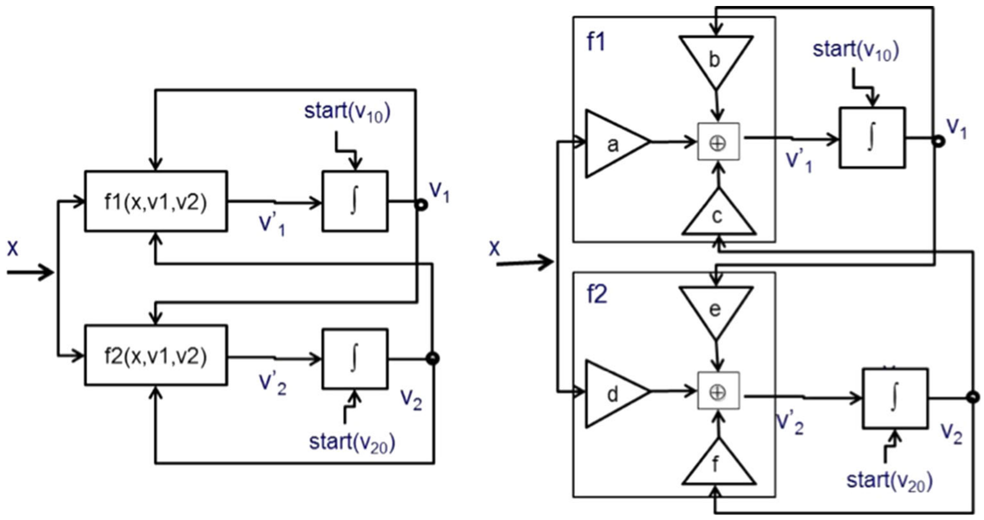


Fig. 7 Continuous data flow networks

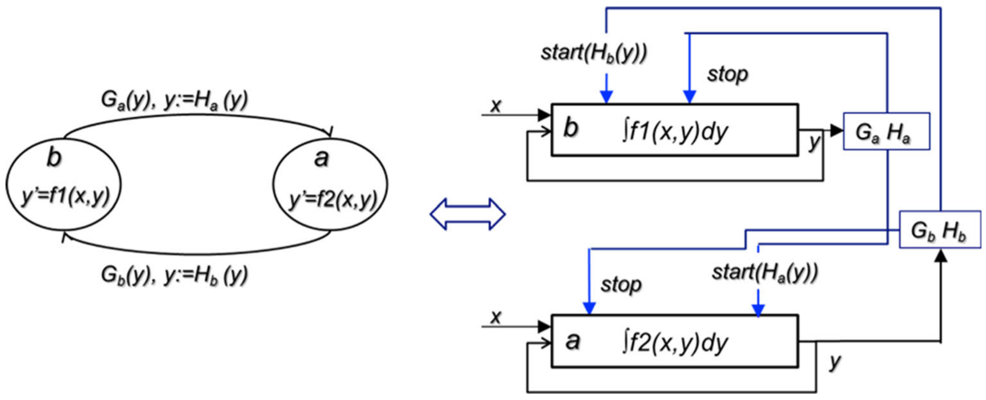


Fig. 8 A hybrid automaton and the corresponding hybrid dataflow network

may be reduced using `when` operators to control the activity of integrators. On the contrary hybrid automata are intrinsically sequential and to avoid state (mode) explosion for complex systems, it is essential to find adequate decompositions.

The literature on cyber-physical systems modeling is very rich, dealing with various aspects regarding the combination of acausal (equational) and causal models as well as event-driven models. An overview of approaches and results can be found in Alur [1] and Derler et al. [21]. This paper focuses on an approach for relating basic models used in a design flow.

3 Discretization techniques for executability

3.1 Discretization algorithms and the problem of algebraic equations

Most of the existing system-level simulation software tools rely on the same mathematical model, namely differential equations possibly accompanied with reset equations. These equations drive the dynamics of signals which, as a consequence of possible resets, exhibit a piecewise continuous behavior. Thus, a numerical solver must solve a sequence of *initial value problems* (IVPs), defined as a combination of a system of ordinary differential equations together with the *initial condition*, i.e., the set of initial values for the variables of system. This process boils down to cyclically repeating the following two phases: (1) the (*re*)*initialization phase*, which consists in determining the new initial condition; (2) the *continuous integration phase*, which consists in applying discretization algorithms to approximate the system dynamics, while detecting discrete events, such as zero-crossings, that cause the resets.

There are numerous techniques for solving ODEs in a reasonably fast and robust way. Numerical solvers usually combine different discretization algorithms and heuristics, including event detection algorithms (which most of the time simply consist in monitoring sign changes between solver steps). The interested reader can refer to Cellier and Kofman

[18] for a comprehensive overview of this aspect of cyber-physical system simulation. Different approaches are also used to analyze models and obtain equational descriptions to be submitted to solvers. For instance, one should mention variants of nodal analysis for SPICE [52] and block-lower-triangular transformation for Modelica tools [53].

Although in some cases modeling tools manage to generate ODEs, in many situations, the resulting systems of equations are DAEs. Indeed, in order to obtain an ODE model, explicit formulations of derivatives as functions of state variables have to be found. However, in general, it is not possible to perform this transformation automatically. Hence, simulation tools heavily rely on DAE solvers, which allow implicit formulations to be directly processed. It is well known, however, that solving DAEs poses hard problems: there is no general criterion for deciding well-posedness of an IVP whose dynamics are driven in part by means of algebraic equations (general algebraic equations may have zero, one or many solutions). Moreover, additional issues arise when so-called *high-index*[11] problems have to be solved. In these problems, algebraic equations impose constraints on state variables, restricting trajectories of the associated signals. As a simple illustration, consider Listing 2, which shows a variation of an example from Mattsson et al. [41]. Notice in particular the algebraic constraint (Eq. 5), which forces the trajectory of the moving end of the pendulum's rod to belong to a circle centered at the origin: x cannot evolve independently of y.[12]

Initialization of this model is a difficult problem, because for almost any value of x (except 1 and -1), there are two distinct solutions for y—a positive one and a negative one—satisfying the algebraic constraint. So an unambiguous initialization of the model seems to require both coordinates to be given, but this is likely to contradict the algebraic constraint. In practice, one will have to disambiguate the problem by selecting the correct value of, say, y given the value of x. Because of nonlinearities, initialization will typically require, in addition to known values, "guess values" for unknowns with the hope that the numerical solver called to the rescue will converge to a suitable solution (one generally expects the returned values to be "close" to the initial guesses). In the most favorable situation, simulation eventually starts, typically after some interaction with the user who contributed to the effort by refining guess values if needed, and by confirming at some point that the returned solutions were satisfactory. Alas, as specified by the "when" clause in the program, the pendulum has to be reset as soon as time reaches 1: we are in the same situation as during initialization but now we would like the solver to decide by itself how to proceed with calculations! Notice that in our example, the numerical solver is even presented with a singular problem at reset time.[13] Indeed, as in the case of initialization, one needs to specify two coordinates to disambiguate the reset position, while the algebraic constraint still holds. A possibility to solve this problem would be for instance to allow only the sign of either x or y to be specified, instead of the actual value, but this solution is specific to our problem (we know a priori that the algebraic equation has two solutions with opposite signs). This very simple example illustrates the difficulties of simulating models using unrestricted DAEs.

Listing 2 Modelica model of a planar pendulum using Cartesian coordinates

```
model Pendulum "A simple planar pendulum of fixed
   length 1, with reset"

  parameter Real m(fixed=true) = 1;
  constant Real g = 9.81;
  Real x(start=0.6, fixed=true), vx(start=0.0,
    fixed=false);
  Real y(start = 0.8, fixed=true), vy(start=0.0,
    fixed=false);
  Real F(start=0.0, fixed=false);

equation

  der(x) = vx                    "eq 1";
  der(vx) = -x * F               "eq 2";
  der(y) = vy                    "eq 3";
  der(vy) = -y * F - m * g       "eq 4";
  x * x + y * y = 1.0            "eq 5";

  when time >= 1 then
    reinit(x, 0.6)               "eq 6";
    reinit(y, 0.8)               "eq 7";
  end when;

end Pendulum;
```

On the other hand, IVPs involving ODEs pose no problem regarding executability: given an initial state (and possible reset values), an ODE specifies an explicit calculation of the future given the current state, in a constant dimensional state space. An interesting (and still open) question concerns the existence of mathematical models enjoying this nice property of ODEs while still allowing discretization algorithms to be devised that would offer the same efficiency as current DAE solving algorithms. Consider Listing 3 which shows a variation of the previous pendulum model using the stabilization method by Baumgarte [5]. This method consists in replacing the original algebraic equations with a suitable combination of its derivatives in order to get an ODE in place of the original high-index DAE. This immediately solves the initialization and reinitialization problems. However, the

[11] Although several distinct definitions of the notion of *index* exist in the literature, they all reflect the "distance" between a system of DAEs and the corresponding system of ODEs.

[12] It can be shown that this is also the case for vx with respect to vy, see Mattsson et al. [41] for a complete discussion.

[13] Undetected at compilation time according to Modelica semantics which only impose restrictions over the number of independent equations (determined based on syntax considerations). Here, the model is found to have two degrees of freedom, we should then be able to reinitialize two state variables on discrete event instants.

model now depends on a new parameter e, which must be carefully tuned to avoid important numerical issues [2].[14]

Listing 3 Modelica model of a planar pendulum using Cartesian coordinates, with Baumgarte's stabilization

```
model StabilizedPendulum "A stabilized version of
   the simple pendulum"

  parameter Real e = 1e-3 "stabilization
     parameter";

  parameter Real m(fixed=true) = 1;
  constant Real g = 9.81;
  Real x(start=0.6, fixed=true), vx(start=0.0,
     fixed=false);
  Real y(start = 0.8, fixed=true), vy(start=0.0,
     fixed=false);
  Real F(start=0.0, fixed=false);

equation

  der(x) = vx                  "eq 1";
  der(vx) = -x * F             "eq 2";
  der(y) = vy                  "eq 3";
  der(vy) = -y * F - m * g     "eq 4";

  -8 * e * e * e * (x * vx + y * vy) * F
  - 2 * e * e * e * (x * x + y * y) * der(F)
  - 6 * e * e * e * m * g * vy
  + 6 * e * e * (vx * vx + vy * vy)
  - 6 * e * e * (x * x + y * y) * F
  - 6 * e * e * m * g * y
  + 6 * e * (x * vx + y * vy)
  + x * x + y * y = 1.0;       "eq 5";

  when time >= 1 then
    reinit(x, 0.6)             "eq 6";
    reinit(y, 0.8)             "eq 7";
    reinit(vx, 0)              "eq 8";
    reinit(vy, 0)              "eq 9";
    reinit(F, 0)               "eq 10";
  end when;

end StabilizedPendulum;
```

Another interesting open question concerns the link with structural equational modeling: would it be possible to automatically derive stable systems of differential equations like in Listing 3 without resorting to manual encoding and parameter tuning (e in this example)? We hope that this could be achieved by coupling mathematical models capable of capturing idealized behavior [10] with structural modeling approaches presented in Sect. 2.1, since the latter allow users to precisely identify fundamental phenomena driving the dynamics of cyber-physical systems.

3.2 Discrete dataflow models

Given an executable continuous dataflow model the principle of its discretization is very simple. It consists in replacing each component computing a function F by a synchronous iterative program. In this translation, that fully preserves the structure of the network, the iterative program for integrators can be obtained by application of well-known integration techniques. For instance, we can apply Euler's method to approximate the solution of $y'(t) = f\big(t, y(t)\big)$ with $y(t_0) = y_0$ by an iterative computation involving a sequence of steps t_n with the step size h: for every step $t_{n+1} = t_n + h$ of the sequence, the computed approximation is $y_{n+1} = y_n + hf(t_n, y_n)$. Figure 9 illustrates this transformation for integrators.

The transformation preserves the dataflow links for discretized flows. The unit delay component in the translation is needed to store the value of y produced in one cycle, to be reused in the next cycle. The unit delay in addition to the `start` event, receives another discrete event `act` that is used to trigger the beginning of an iteration cycle.

Discrete dataflow models are at the basis of synchronous languages. These are networks of functional components characterized by a function F with input and outputs and a particular discrete event `act`. Components cyclically perform the computation of F, triggered by the event `act` which plays the role of logical clock. For the component of Fig. 10, at each instant t the inputs x and y are updated and an output z is produced: $z(t) = F(x(t), y(t))$.

In discrete dataflow networks, data output ports of a component may be connected to data input ports of other components. This defines the dataflow relation. Events `act` can be either external inputs or generated by using specific functions that generate events from data streams. These events are subject to strong synchronization constraints as they trigger the production of the data values by components.

A key issue is the efficient compilation of discrete dataflow models, so that the activation constraints are met. The signal `act` can be defined in many possible ways. The simplest is to admit a finest common granularity of computation with the same integration step.

3.2.1 The synchronous execution assumption

The translation from continuous dataflow networks to discrete dataflow networks makes a very strong implicit assumption regarding the speed of the discretized system with respect to its environment. This assumption known as the *synchronous execution assumption*, says that the input x (external environment) does not change or does not change significantly between two successive `act` signals. Such an assumption adopted by all synchronous reactive languages [6] must be respected for the translation to be faithful.

The need for the synchronous execution assumption can be understood when we try to find an automaton that is behaviorally equivalent to a function as simple as a unit delay. A unit delay is specified by the equation $y(t) = x(t-1)$. For the sake of simplicity, we consider that x and y are binary variables, functions of time t. The behavior of a unit delay can

[14] A careful reader may have noticed that the actual value of e has no influence on the numerical solution, at least theoretically. In practice, however, numerical conditioning issues arise as a consequence of finite precision of computer arithmetic.

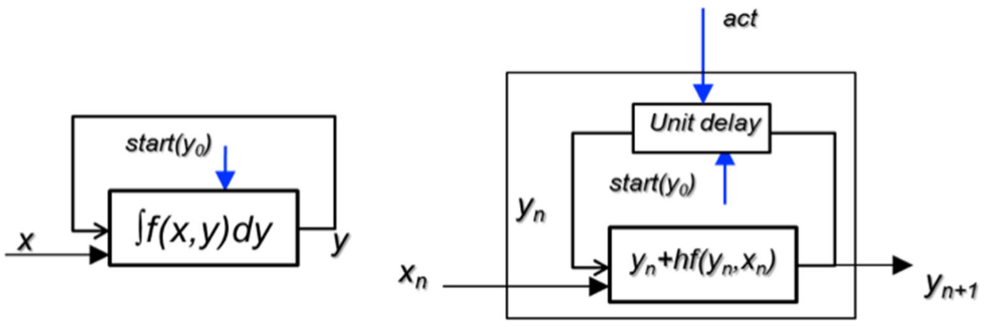
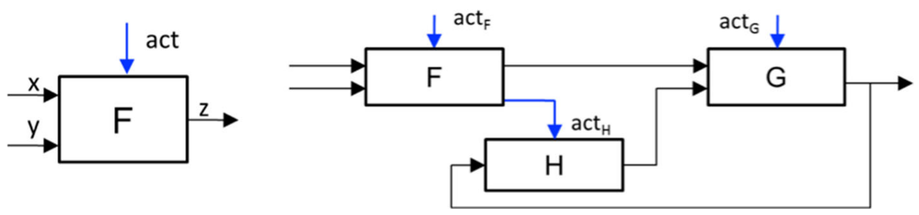


Fig. 9 Continuous dataflow model and the corresponding discrete dataflow model

Fig. 10 Discrete dataflow component and discrete dataflow network



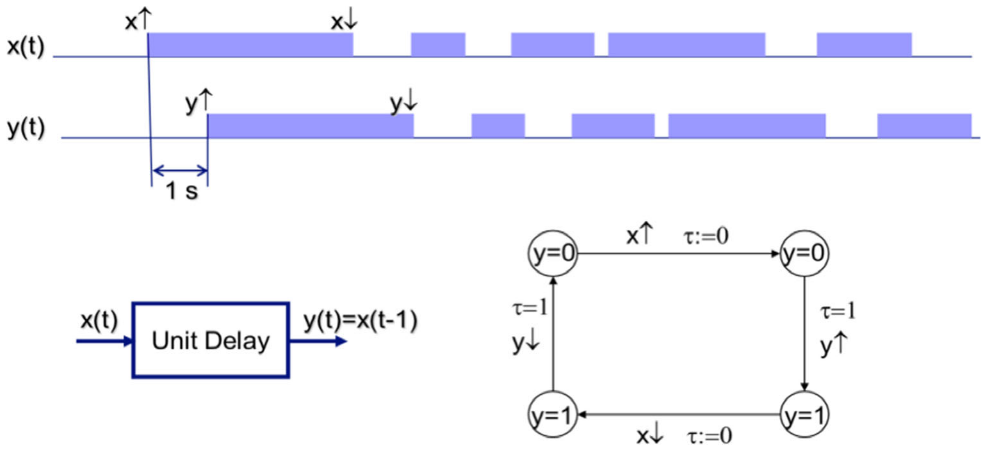


Fig. 11 Timed automaton representing a unit delay $y(t) = x(t-1)$

be represented by the timed automaton in Fig. 11 with four states, provided that there is at most one change of x in one time unit. The automaton detects for the input x, rising edge ($x\uparrow$) and falling edge ($x\downarrow$) events and produces corresponding outputs in one time unit. Reaction times are enforced by using a clock τ. Notice that the number of states and clocks needed to represent a unit delay, increases linearly with the maximum number of changes allowed for x in one time unit. So, there is no finite state computational model equivalent to a unit delay if we do not make an assumption on the upper bound of input changes over one time unit! If the synchronous execution assumption holds then the provided automaton is behaviorally equivalent to the unit delay function.

3.3 Discrete event dataflow models

Discrete event dataflow models are obtained by extending discrete dataflow models with a `when` operator. They can also be considered as the model obtained after discretization of hybrid dataflow models. Thus, their components can be triggered by three different types of events:

1. `act` events that mark the beginning of a computation cycle (step) of the component,
2. `stop` events that switch off the activity of a component,
3. `start` events that resume the activity of a component—these are parametrized by the initial state of the component.

A key issue is defining appropriate operational semantics for such models. Following the synchrony assumption, computation steps should run to completion—that is, they should not be interrupted by `stop` (change of mode) events.

Another issue is how much the assumption about strong synchronization of `act` signals can be relaxed without affecting the overall behavior or essential properties. The activation

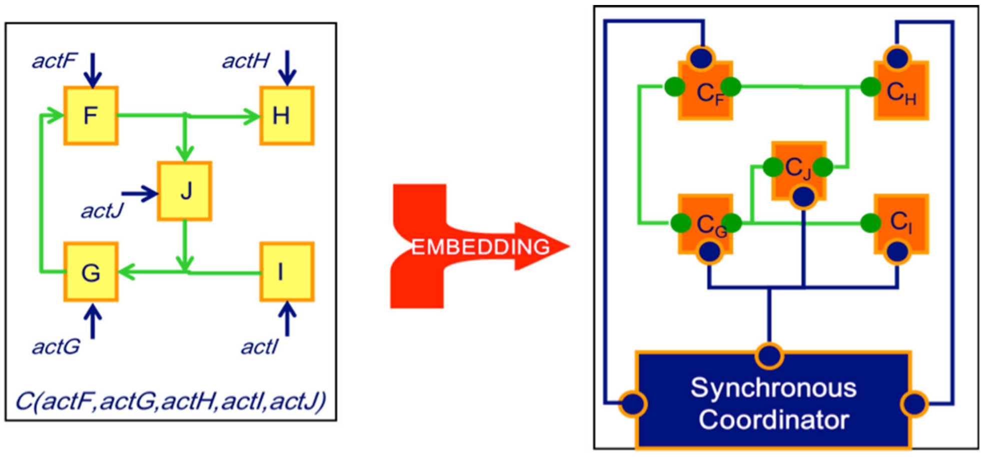


Fig. 12 Embedding discrete dataflow models into event-driven models

policy for a component should also determine the integration step. For variable integration step, each time a component receives an `act` signal, it should also receive the amount of time by which time can progress.

In the next section, we discuss the problem of translating these models into event-based systems.

4 Execution and implementation techniques

To achieve the second objective of cyber-physical systems modeling mentioned in the introduction—that is to enable automatic code generation for the discrete components of a cyber-physical systems model—code generation must preserve the structure of the model. This allows the reuse of the same code for both the simulation and the implementation of control sub-systems. Furthermore, the ability to preserve model structure at code generation is key to enabling separate compilation, multi-site implementation and co-simulation. These issues are addressed to some extent by the FMI standardization initiative [13]. We discuss below the most relevant and urgent research challenges.

4.1 Modular code generation

Given hybrid dataflow models, we discuss the problem of their translation into an executable model that explicitly implements the relationships between the signals `act`, `start` and `stop`. We show how techniques for embedding synchronous languages can be adapted to discrete event dataflow languages.

Compilers of most synchronous languages generate monolithic code. This is clearly a limitation for multi-site implementation and linking the generated code with existing legacy one. We show how we can generate code that preserves the structure of the source model using embeddings [47], based on the results of Bozga et al. [15] and Sfyrla et al. [46] on the translation of Lustre and Simulink, respectively, into the BIP component framework [3,4,12]. [15]

In this section, we discuss only the principle of a modular translation of hybrid dataflow models into BIP, which follows the approach presented by Sfyrla et al. [46], and we skip technical results, in particular those for checking whether modular generation is possible using modular flow graphs. The principle of the translation is illustrated in Fig. 12. On the left, the discrete dataflow model in the source language L consists of a set of functional components characterized by the function they compute and their input and output data ports. It is a network defined by the dataflow relation connecting outputs to inputs. Each component has an `act` event, triggering computation cycles. These events are subject to constraints enforcing relationships between the execution speeds of the components.

The structured operational semantics of L defines an execution engine that coordinates the execution of components as specified by synchronization constraints. The resulting model in the host language is obtained by replacing each function F of the source model by a component C_F iteratively computing F. Each dataflow link in the source model is replaced by a connector involving strong synchronization. Furthermore, the execution engine for L is a synchronous coordinator that orchestrates the triggering events `act`.

This construction involves separate translation of components and the dataflow connections, explicitly defined by the programmer in language L. The triggering events and their relations are derived automatically, based on the operational semantics of L.

The translation of a purely functional dataflow component computing a function F is illustrated in Fig. 13 using the BIP

[15] http://www-verimag.imag.fr/Rigorous-Design-of-Component-Based.html.

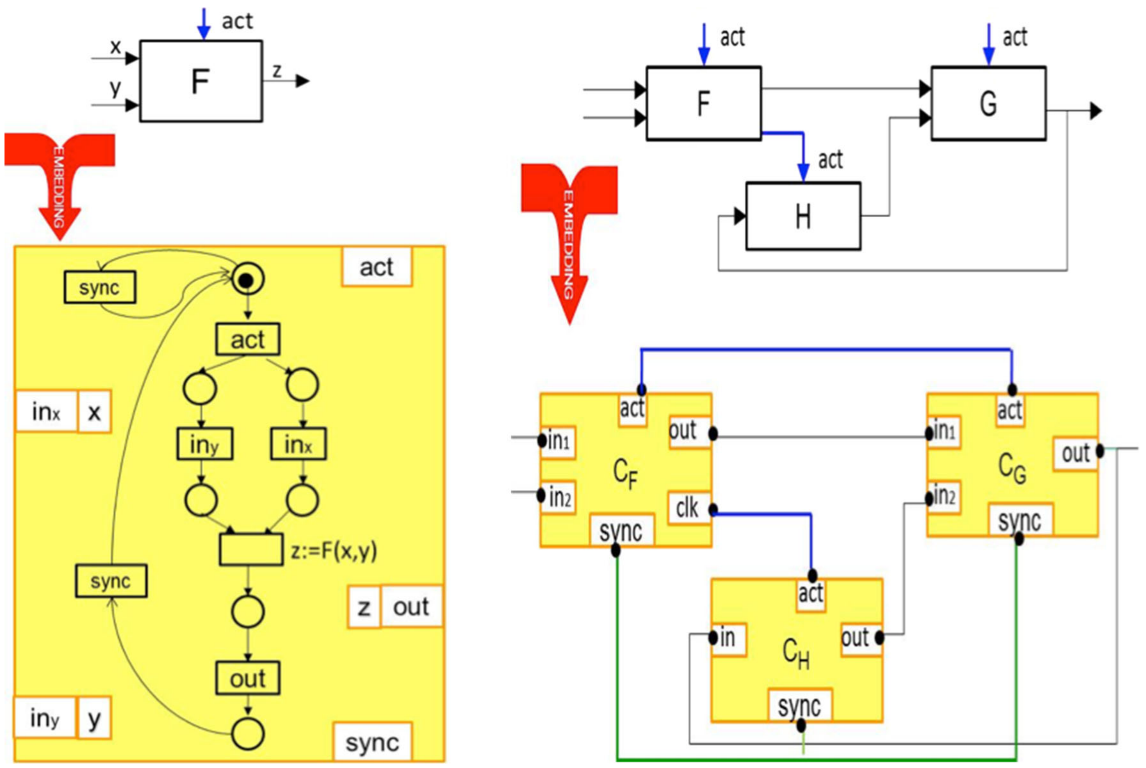


Fig. 13 Translation of functional components and embedding

notation. The event-driven component has input and output data ports corresponding to the input and output leads of the functional component. In addition, it has an `act` action with the corresponding port that initiates the computation cycle. The latter terminates with a `sync` event. Reading the inputs is followed by the computation of F and posting the produced result.

The embedding consists in replacing dataflow connections by BIP connectors (rendezvous). The free `act` events (that are not generated from signals) define a basic clock and must be strongly synchronized. The `sync` event marks the end of a cycle and must be strongly synchronized (green connector).

Embedding hybrid dataflow models involves an additional difficulty: handling not only `act` events but also `start` and `stop` events that are needed to transition between modes. We are currently studying the principle of the implementation of such an embedding illustrated in Fig. 14.

A functional node F of hybrid dataflow network is specified by its input and output ports and the events `start`, `stop` and `act`. The former two events are generated by `when` operators parametrized by guards G and actions H. For instance, $H_{\texttt{startF}}$ is used to compute the initial state of F each time $G_{\texttt{startF}}$ becomes true. The event `stopF` may be generated by a `when` operator that starts some other node G.

The translation is compositional. It generates the event triggered component C_F corresponding to the functional node F controlled by another switching component that determines whether C_F can be activated or not (action `onF`). A change of mode can happen only upon completion of an execution cycle. The `syncF` event is a triggering event controlling the synchronization between ports.

This solution requires the `stopA` action to have higher priority than the `onA` action.

4.2 Co-simulation techniques

A recent trend in system-level simulation deals with the coupling of heterogeneous simulation models. This is usually needed in the context of industrial system simulations, where large systems may be composed of many sub-systems designed by different teams, departments, companies, etc. System engineers use large interconnected models to simulate the behavior of these systems in order to anticipate potential inter-system issues or design control algorithms for the whole system (as opposed to local control loops that are part of the sub-systems). Standards for model coupling thus emerged in the simulation software industry, either ad hoc, like the Simulink *S-functions* [20], or designed by industrial consortia, like the recent *Functional Mock-up Interface* initiative [13]. Model coupling currently comes in two flavors: *model exchange* and *co-simulation*.

Coupling continuous system-level simulation models through model exchange means that some models are cre-

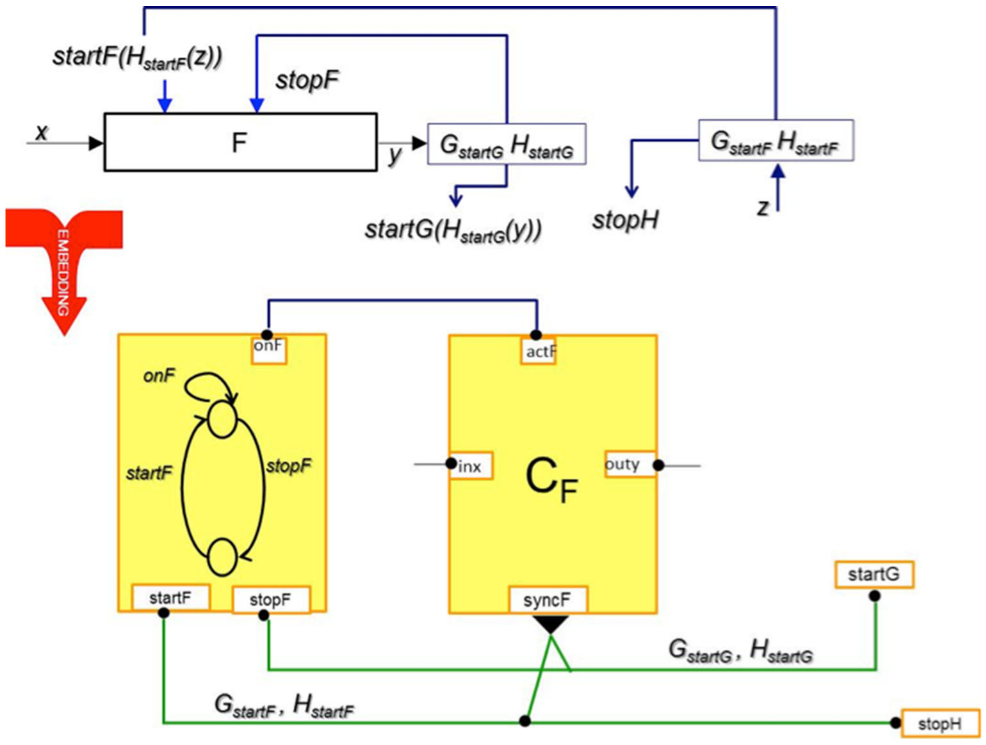


Fig. 14 Embedding discrete event dataflow models into event-driven models

ated with different modeling tools and are exported toward an importing environment. This environment provides a unique numerical solver that is able to simulate the new system model which results from the interconnection of the imported models. To achieve this, it is assumed that the exporting tools share a common model semantics with the importing environment. The most common semantics for continuous system-level simulation model is the continuous data flow network that is represented using a block diagram defining the right-hand side of a set of ordinary differential equations (ODEs).

Each exported model thus defines an ODE system:

$$\dot{x_i} = f_i(x_i, u_i),$$
$$y_i = g_i(x_i, u_i),$$

where x_i is the state vector, u_i is the vector of the exogenous inputs, and y_i is the vector of the system outputs.

In the importing environment, the system designer—using the model exchange coupling technique—specifies interconnections of sub-models using block diagrams. This is equivalent to specifying a connection $\{0, 1\}$-matrix K such that:

$$u = Ky,$$

where $u = (u_i)$ and $y = (y_i)$ are the vectors of *all* inputs and outputs, respectively.

The importing environment handles the simulation problem by constructing and solving a semi-explicit differential algebraic equation (DAE) representation of the data flow network:

$$\dot{x} = f(x, Ky),$$
$$y = g(x, Ky).$$

Model coupling through model exchange is a well-posed problem as long as (1) sub-models are, indeed, continuous, and (2) consistent initial conditions for state and algebraic variables have been provided. These conditions are not, in general, satisfied in the context of cyber-physical systems, which mix continuous-time differential equations with discrete-time equations. As illustrated by Benveniste et al. [8], system-level simulators typically resort to ad hoc semantics to handle the discrete events in such models. This makes composition of sub-models into a common, well-defined model a complex process, heavily relying on the designers' insight.

Coupling of simple piecewise continuous models (i.e., ODE with reset equations) can produce DAEs in the resulting composed model. In such cases, restart conditions to be applied following a discontinuity become difficult to specify, due to the algebraic constraints. This requires system designers to have a global understanding of the interconnected systems, which is usually not the case. In absence of well-specified initial or restart conditions, the importing

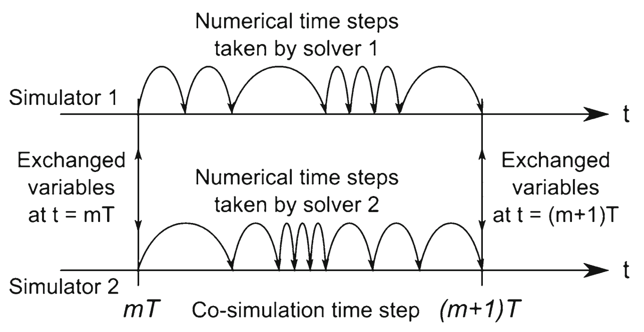

Numerical time steps taken by solver 1
Simulator 1
t
Exchanged variables at t = mT
Numerical time steps taken by solver 2
Exchanged variables at t = (m+1)T
Simulator 2
t
mT
Co-simulation time step
(m+1)T

Fig. 15 Basic explicit co-simulation scheme involving two simulators

environment has to rely on the algebraic solving methods described in Sect. 3.1, for which convergence is not guaranteed.

Finally, even if constituent models do not have direct feedthroughs, the resulting set of ODEs that are integrated in the importing environment may exhibit unexpected computational performance degradation, since the numerical solver must adapt its step size to the dynamics of the fastest subsystem. This is also a practical issue from the point of view of the system designers, who build constituent models that display good computational performance individually, but perform poorly when the system, coupled through model exchange, is simulated.

Co-simulation was introduced about thirty years ago by Gear [27] as the multi-rate integration method, and Lelarasmee et al. [38] as the waveform relaxation method. It was first used as a way to speed up electronic circuit simulations, before being applied more recently to mechatronics systems by Kübler and Schiehlen [33]. It consists in coupling the simulators themselves, each sub-system model being equipped with its own (and a priori best fitted) numerical solver. A limited set of signals are exchanged at predefined macro-time steps (as opposed to numerical micro-time steps taken by the numerical solvers involved in the coupled system). A common co-simulation scheme is depicted in Fig. 15 in the case of two coupled sub-systems. This basic explicit scheme consists in sampling the output signals of each sub-system along a fixed sample rate T. The input signals are thus held constant throughout the duration of a macro-step. The numerical solver of each sub-system is used to simulate the local models up to the end of the macro-step. Variants of this scheme were proposed, but seldom implemented in practice, using variable macro-step size to control the extrapolation error, through multistep higher-order extrapolation schemes.

Since co-simulation explicitly discretizes the coupling signals it aims at decoupling the continuous dynamics of the sub-systems wherever possible. This apparent increase in robustness and practical simplicity of the method is, however, counterbalanced by possible loss of numerical stability resulting from the discretization of high dynamics that reside in the coupling itself, even if the numerical solvers taken alone work in their stability domain.

A large body of work (e.g., [16,51]) is dedicated to the semantics and the design of master algorithms for co-simulation standards, notably FMI [13]. Such standards pave the way to a wider adoption of co-simulation. However, they fall short of addressing the fundamental problem of numerical stability, which still requires further research.

4.3 Distributed modular simulation

As discussed in Sect. 4.2, simulators of physical systems relying on co-simulation require models to be compiled into a set of distinct sub-models to be executed separately (possibly in different processes), with adequate coordination between them. In current co-simulation frameworks, including the FMI standard [13], the importing environment is responsible for the global coordination of involved simulators. Moreover, the coordination scheme requires each simulator to provide the value of its locally managed signals at some predetermined dates.

We exposed above some performance issues raised by current co-simulation techniques. We want to discuss here some challenges regarding *efficient* distributed modular simulation of physical systems. We illustrate the problem by an example. Consider the circuit of Fig. 16 involving components R, I and C. To simulate its behavior, the usual method consists in solving the system of equations describing the dynamics of the components and the constraints on currents and voltages induced by the connectors (represented by bullets). An alternative approach that would avoid the construction of a global model, is to run simulation programs P_R, P_I and P_C for R, I and C, respectively, separately. The needed coordination can be in principle achieved by communication protocols including proper enforcement of Kirchhoff's laws (represented by the Σ symbols), the interconnection topology being statically computed by the compiler based on the physical semantics discussed in Sect. 2.1.

The problem of coordination of sub-models in distributed simulations involving signals with *discrete evolution* is well understood [55]. In contrast, efficient coordination of *continuous and hybrid* sub-models constitutes an open challenge and an active research topic with encouraging results [19]. However, idealization poses additional problems among which correct composition, as discussed in Sect. 2. We would like composition not to compromise modularity by requiring involved parts of models to be combined as a whole. However, composition of ideal models typically results in DAEs whose algebraic parts require global fix point calculations: this seems to go against the idea of modularity. Finally, an important challenge for the distributed simulation of continuous systems is to design adequate higher-order solvers capable of efficiently handling stiff problems.

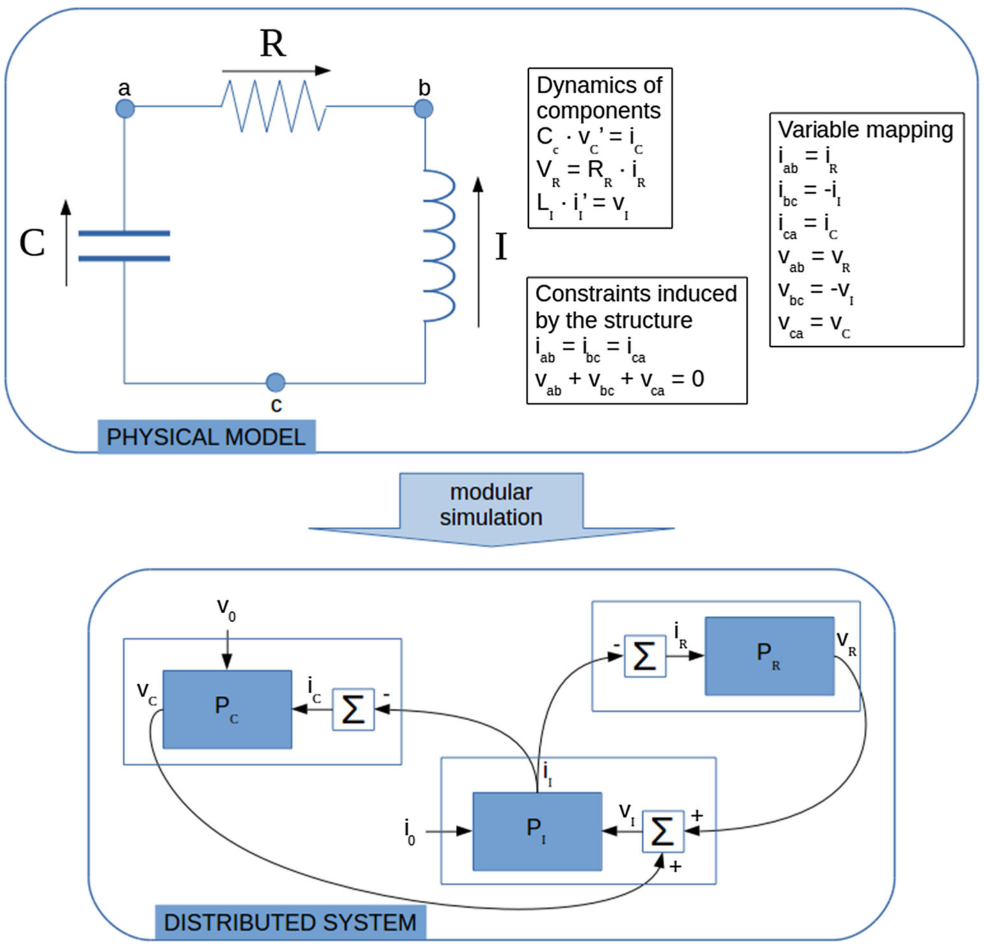


Fig. 16 Distributed modular simulation

The interesting open question is whether distributed modular simulation of cyber-physical systems is at all possible, moreover with decent performance, given these constraints.

5 Discussion

This paper attempts to perform a fair assessment of the state of the art and of the state of the practice in cyber-physical system design. Though its findings and conclusions contrast with the optimism of some other surveys, the main message is that we are still very far from reaching the vision. There exist some very basic theoretical difficulties to be overcome by proposing methodologies adequately combining tool automation and designer ingenuity.

The problem of writing faithful and consistent models from networks of physical components is still open. We have discussed various issues related to semantics, in particular the adequate treatment of Zenoness, modularity and determinacy of models. We need languages for equational modeling such as Modelica, allowing the right level of abstraction and supporting structured approaches.

For discretization, we need effective methods for deciding model executability at low cost. We also need theory for assessing the quality and safety of integration techniques. Discrete dataflow languages should be adequately enriched to support both clocks and events that force change of mode without jeopardizing overall synchrony of computation.

Efficient and rigorous code generation still remains a distant goal for both simulation and real-time control, in centralized and distributed contexts. To enforce consistency, it is desirable that the code generation process is as common as possible, differing only in order to take into account specific needs.

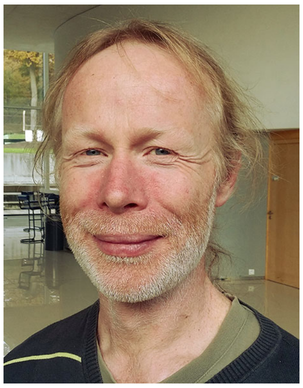
Finally, it is essential that theory integration is accompanied by its application in design flows that support consistent integration of tools [21,22,48]. Work in that direction should go hand in hand with developing theoretical foundations for elaborating sound principles for compositionality and componentization, and defining sufficiently abstract component interfaces ensuring independence from modeling tools selected by developers.

References

1. Alur, R.: Principles of Cyber-Physical Systems. MIT Press, Cambridge (2015)
2. Ascher, U.M., Chin, H., Petzold, L.R., Reich, S.: Stabilization of constrained mechanical systems with DAEs and invariant manifolds. Mech. Struct. Mach. **23**(2), 135–157 (1995). https://doi.org/10.1080/08905459508905232
3. Basu, A., Bozga, M., Sifakis, J.: Modeling heterogeneous real-time components in BIP. In: 4th IEEE International Conference on Software Engineering and Formal Methods (SEFM06), invited talk, pp. 3–12 (2006). https://doi.org/10.1109/SEFM.2006.27
4. Basu, A., Bensalem, S., Bozga, M., Combaz, J., Jaber, M., Nguyen, T.H., Sifakis, J.: Rigorous component-based system design using the BIP framework. IEEE Softw. **28**(3), 41–48 (2011). https://doi.org/10.1109/MS.2011.27
5. Baumgarte, J.: Stabilization of constraints and integrals of motion in dynamical systems. Comput. Methods Appl. Mech. Eng. **1**, 1–16 (1972). https://doi.org/10.1016/0045-7825(72)90018-7
6. Benveniste, A., Caspi, P., Edwards, S.A., Halbwachs, N., Guernic, P.L., de Simone, R.: The synchronous languages twelve years later. Proc. IEEE, Spec. Issue Embed. Syst. **91**(1), 64–83 (2003)
7. Benveniste, A., Bourke, T., Caillaud, B., Pouzet, M.: Non-standard semantics of hybrid systems modelers. J. Comput. Syst. Sci. **78**, 877–910 (2012). https://doi.org/10.1016/j.jcss.2011.08.009
8. Benveniste, A., Bourke, T., Caillaud, B., Pouzet, M.: Hybrid systems modeling challenges caused by cyber-physical systems. In: Baras, J., Srinivasan, V. (eds) Cyber-Physical Systems (CPS) Foundations and Challenges. Available on-line: http://people.rennes.inria.fr/Albert.Benveniste/pub/NIST2012.pdf (2013) (**to appear**)
9. Berger, C., Mousavi, M.R., (eds): Cyber Physical Systems. Design, Modeling, and Evaluation—5th International Workshop, CyPhy 2015, Amsterdam, The Netherlands, 8 Oct 2015. Proceedings, Lecture Notes in Computer Science, vol. 9361, Springer (2015). https://doi.org/10.1007/978-3-319-25141-7
10. Bliudze, S., Furic, S.: An operational semantics for hybrid systems involving behavioral abstraction. In: Proceedings of the 10th International Modelica Conference, Linköping University Electronic Press, Linköpings Universitet, Linköping, Linköping Electronic Conference Proceedings, pp. 693–706 (2014). https://doi.org/10.3384/ECP14096693
11. Bliudze, S., Krob, D.: Modelling of complex systems: systems as dataflow machines. Fundam. Inf. **91**, 1–24 (2009). https://doi.org/10.3233/FI-2009-0001
12. Bliudze, S., Sifakis, J.: The algebra of connectors—structuring interaction in BIP. In: Proceedings of the EMSOFT'07, ACM SigBED, Salzburg, Austria, pp. 11–20 (2007)
13. Blochwitz, T., Otter, M., Arnold, M., Bausch, C., Elmqvist, H., Junghanns, A., Mauß, J., Monteiro, M., Neidhold, T., Neumerkel, D., Olsson, H., Peetz, J.V., Wolf, S., Clauß, C.: The functional mockup interface for tool independent exchange of simulation models. In: Proceedings of the 8th International Modelica Conference, Linköping University Electronic Press, vol. 63, pp. 105–114 (2011)
14. Bornot, S., Sifakis, J.: An algebraic framework for urgency. Inf. Comput. **163**(1), 172–202 (2000). https://doi.org/10.1006/inco.2000.2999
15. Bozga, M.D., Sfyrla, V., Sifakis, J.: Modeling synchronous systems in BIP. In: Proceedings of the Seventh ACM International Conference on Embedded Software, ACM, New York, NY, USA, EMSOFT '09, pp. 77–86 (2009). https://doi.org/10.1145/1629335.1629347
16. Broman, D., Brooks, C., Greenberg, L., Lee, E.A., Masin, M., Tripakis, S., Wetter, M.: Determinate composition of FMUs for co-simulation. In: Proceedings of the Eleventh ACM International Conference on Embedded Software, IEEE Press, Piscataway, NJ, USA, EMSOFT '13, pp. 2:1–2:12 (2013). URL http://dl.acm.org/citation.cfm?id=2555754.2555756
17. Caspi, P., Pilaud, D., Halbwachs, N., Plaice, J.: Lustre: a declarative language for programming synchronous systems. In: Conference Record of the Fourteenth Annual ACM Symposium on Principles of Programming Languages, Munich, Germany, 21–23 Jan 1987. ACM Press, pp. 178–188 (1987). https://doi.org/10.1145/41625.41641
18. Cellier, F.E., Kofman, E.: Continuous System Simulation. Springer, Berlin (2006)
19. Cellier, F.E., Kofman, E., Migoni, G., Bortolotto, M.: Quantized state system simulation. In: Proceedings of Grand Challenges in Modeling and Simulation (GCMS08), pp. 504–510 (2008)
20. Dabney, J.B., Harman, T.L.: Mastering Simulink. Prentice Hall, Upper Saddle River (2004)
21. Derler, P., Lee, E.A., Sangiovanni-Vincentelli, A.L.: Modeling cyber-physical systems. Proc. IEEE **100**(1), 13–28 (2012). https://doi.org/10.1109/JPROC.2011.2160929
22. Fitzgerald, J., Gamble, C., Larseny, P.G., Pierce, K., Woodcock, J.: Cyber-physical systems design: formal foundations, methods and integrated tool chains. In: 2015 IEEE/ACM 3rd FME Workshop on Formal Methods in Software Engineering (FormaliSE), pp. 40–46 (2015). https://doi.org/10.1109/FormaliSE.2015.14
23. Fritzson, P.: Introduction to Modeling and Simulation of Technical and Physical Systems with Modelica. Wiley, Hoboken (2011)
24. Furic, S.: Connection semantics: overview of some classical approaches and proposal for a novel one. (**unpublished, available on demand**) (2013)
25. Furic, S.: Physical connection proposal for the FMI. Technical Report, FMI Design Meeting, 9–10 Feb 2015, DLR, Germany (2015a)
26. Furic, S.: A physical connection proposal for the FMI. In: Workshop Sim@SL, ENS Cachan, Paris (2015b)
27. Gear, C.W.: Automatic multirate methods for ordinary differential equations. Technical Report, UIUCDS-R-80-1000, Illinois University, Urbana (USA) (1980)
28. Geisberger, E., Broy, M., (eds): Living in a Networked World: Integrated Research Agenda Cyber-Physical Systems (agendaCPS). acatech STUDIE, Utz Verlag GmbH (2015)
29. Henzinger, T.A.: The Theory of Hybrid Automata. Springer, Berlin (2000)
30. Hogan, N., Breedveld, P.: Chapter 15: the physical basis of analogies in network models of physical system dynamics. In: Bishop, R.H. (ed.) The Mechatronics Handbook, pp. 1–10. CRC Press, Boca Raton (2002)
31. Karnopp, D.C., Margolis, D.L., Rosenberg, R.C.: System Dynamics: Modeling, Simulation, and Control of Mechatronic Systems, 5th edn. Wiley, Hoboken (2012)
32. Konečný, M., Taha, W., Bartha, F.A., Duracz, J., Duracz, A., Ames, A.D.: Enclosing the behavior of a hybrid automaton up to and beyond a Zeno point. Nonlinear Anal. Hybrid Syst. **20**, 1–20 (2016). https://doi.org/10.1016/j.nahs.2015.10.004
33. Kübler, R., Schiehlen, W.: Two methods of simulator coupling. Math. Comput. Model. Dyn. Syst. **6**(2), 93–113 (2000)
34. Lamb, J.D., Asher, G.M., Woodall, D.R.: Network realisation of bond graphs. In: Granada, J.J., Cellier, F.E. (eds.) Proceedings of International Conference on Bond Graph Modeling (ICBGM '93), Society for Computer Simulation, Simulation Series, vol. 25(2), pp. 85–90 (1993)
35. Lee, E.A.: Cyber physical systems: design challenges. In: 2008 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC), pp. 363–369 (2008). https://doi.org/10.1109/ISORC.2008.25

36. Lee, E.A.: Constructive models of discrete and continuous physical phenomena. IEEE Access **2**, 797–821 (2014). https://doi.org/10.1109/ACCESS.2014.2345759
37. Lee, E.A., Zheng, H.: Operational semantics of hybrid systems. In: Morari, M., Thiele, L. (eds.) Hybrid Systems: Computation and Control, Lecture Notes in Computer Science, vol. 3414, Springer, Heidelberg, pp. 25–53 (2005). https://doi.org/10.1007/978-3-540-31954-2_2
38. Lelarasmee, E., Ruehli, A., Sangiovanni-Vincentelli, A.L.: The waveform relaxation method for time-domain analysis of large scale integrated circuits. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **1**(3), 131–145 (1982). https://doi.org/10.1109/TCAD.1982.1270004
39. Lindstrøm, T.: An invitation to nonstandard analysis. In: Cutland, N. (ed.) Nonstandard Analysis and its Applications, No. 10 in London Mathematical Society Student Texts, Cambridge University Press (1988)
40. Matsikoudis, E., Lee, E.A.: On fixed points of strictly causal functions. In: Formal Modeling and Analysis of Timed Systems, Springer, pp. 183–197 (2013)
41. Mattsson, S.E., Olsson, H., Elmqvist, H.: Dynamic Selection of states in Dymola. In: Proceedings of Modelica Workshop 2000, Lund, pp. 61–67 (2000)
42. Perelson, A.S., Oster, G.F.: Bond graphs and linear graphs. J Frankl. Inst. **302**(2), 159–185 (1976)
43. Rajkumar, R.R., Lee, I., Sha, L., Stankovic, J. Cyber-physical systems: the next computing revolution. In: Proceedings of the 47th Design Automation Conference, ACM, New York, NY, USA, DAC '10, pp. 731–736 (2010). https://doi.org/10.1145/1837274.1837461
44. Robinson, A.: Non Standard Analysis. North Holland, Amsterdam (1966)
45. Rust, H.: Operational semantics for timed systems: a non-standard approach to uniform modeling of timed and hybrid systems. Lecture Notes in Computer Science, vol. 3456. Springer (2005). https://doi.org/10.1007/978-3-540-32008-1
46. Sfyrla, V., Tsiligiannis, G., Safaka, I., Bozga, M., Sifakis, J.: Compositional translation of simulink models into synchronous BIP. In: 2010 International Symposium on Industrial Embedded Systems (SIES), pp. 217–220 (2010). https://doi.org/10.1109/SIES.2010.5551374 (2010)
47. Sifakis, J.: System design automation: challenges and limitations. Proc. IEEE **103**(11), 2093–2103 (2015). https://doi.org/10.1109/JPROC.2015.2484060
48. Sztipanovits, J., Bapty, T., Neema, S., Koutsoukos, X., Jackson, E.: Design tool chain for cyber-physical systems: Lessons learned. In: Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE, pp. 1–6 (2015), https://doi.org/10.1145/2744769.2747922
49. Tellegen, B.D.: A general network theorem, with applications. Philips Res. Rep. **7**(4), 259–269 (1952)
50. Trent, H.M.: Isomorphisms between oriented linear graphs and lumped physical systems. J. Acoust.l Soc. Am. **27**(3), 500–527 (1955)
51. Tripakis, S.: Bridging the semantic gap between heterogeneous modeling formalisms and FMI. In: 2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), IEEE, pp. 60–69 (2015)
52. Vladimirescu, A.: The SPICE Book. Wiley, Hoboken (1993)
53. Walther, M., Waurich, V., Schubert, C., Dr-Ing GubschBliudze, I.: Equation based parallelization of modelica models. In: Proceedings of the 10th International Modelica Conference, Linköping University Electronic Press, Linköpings Universitet, Linköping, Linköping Electronic Conference Proceedings, pp. 1213–1220, (2014). https://doi.org/10.3384/ECP140961213
54. Wolf, W.: Cyber-physical systems. Computer **42**(3), 88–89 (2009). https://doi.org/10.1109/MC.2009.81
55. Zeigler, B.P., Lee, J.S.: Theory of quantized systems: formal basis for DEVS/HLA distributed simulation environment. SPIE Proc. **3369**, 49–58 (1998)
56. Zheng, H., Lee, E.A., Ames, A.D.: Beyond zeno: get on with It! In: Hespanha, J.P., Tiwari, A.: (eds). Hybrid Systems: Computation and Control, 9th International Workshop, HSCC 2006, Santa Barbara, CA, USA, 29–31 March 2006. Proceedings, Springer, Berlin, pp. 568–582 (2006). https://doi.org/10.1007/11730637_42

Simon Bliudze is a Research Scientist at INRIA Lille—Nord Europe. He holds an MSc in Mathematics from St. Petersburg State University (Russia, 1998), an M.Sc. in Computer Science from Université Paris 6 (France, 2001) and a Ph.D. in Computer Science from École Polytechnique (France, 2006). He has spent two years at Verimag (Grenoble, France) as a post-doc with Joseph Sifakis working on formal semantics for the BIP component framework. Before joining INRIA in 2017, he has spent three years as a research engineer at CEA Saclay (France) and six years as a scientific collaborator at EPFL (Lausanne, Switzerland).



Sébastien Furic is a Research Engineer at INRIA Paris-Rocquencourt. He has twenty years of experience in programming language design and implementation, focusing primarily on the fields of formal verification, synchronous languages and physical modeling. He graduated from the University of Saint-Étienne and Mines Saint-Étienne, France. Before joining INRIA in 2017, he spent twelve years at Siemens PLM Software as a modeling language expert. His research interests include theoretical and practical aspects of modeling languages, with special emphasis on modular and compositional aspects of physical system modeling and simulation.

Joseph Sifakis is Emeritus Senior CNRS Researcher at Verimag. His current research interests cover fundamental and applied aspects of embedded systems design. The main focus of his work is on the formalization of system design as a process leading from given requirements to trustworthy, optimized and correct-by-construction implementations. Joseph Sifakis has been a full professor at Ecole Polytechnique Fédérale de Lausanne (EPFL) for the period 2011–2016. He is the founder of the Verimag laboratory in Grenoble, which he directed for 13 years. In 2007, Joseph Sifakis has received the Turing Award for his contribution to the theory and application of model checking. Joseph Sifakis has had numerous administrative and managerial responsibilities both at French and European level. He has actively worked to reinvigorate European research in embedded systems as the scientific coordinator of the ≪ ARTIST ≫ European Networks of Excellence. He has participated in many major industrial projects led by companies such as Airbus, EADS, France Telecom, Astrium, and STMicroelectronics. Joseph Sifakis is a member of the French Academy of Sciences, a member of the French National Academy of Engineering, Academia Europea, the American Academy of Arts and Sciences, and the National Academy of Engineering. He has been the President of the Greek Council for Research and Technology for the period February 2014–April 2016.

Antoine Viel is an R&D Engineer at Siemens PLM Software. He holds an M.Sc. degree in Control Systems from the Compiègne Technological University (France, 1994) and a Ph.D. in numerical analysis from INRIA (France, 1999). He has fifteen years of experience in system level simulation at OEMs, system engineering companies and software editors. His research interests include modeling languages, numerical solvers, and co-simulation technologies. He is a member of the Functional Mock-up Interface working group since its inception.