Flexible Logic-based Co-simulation of Modelica Models

Luciano Baresi, Gianni Ferretti, Alberto Leva, and Matteo Rossi
Dipartimento di Elettronica e Informazione - Politecnico di Milano
20133 Milano, Italy - Email: baresi|ferretti|leva|rossi@elet.polimi.it

***Abstract*—The design of complex embedded software systems requires the careful analysis of the system and of the environment it interacts with. The different natures of these two elements are difficult to address by means of a single all-encompassing technique/notation. The paper proposes MCA, the MADES Co-simulation Approach, which allows designers to combine different, complementary formalisms in a seamless manner: the system is rendered through logic formulae, while the environment is demanded to Modelica. These two models are input to MCA to produce an execution trace that is "compatible" with them, that is, that does not violate either model. The paper introduces the theoretical basis of MCA and exemplifies it on a case study.**

I. Introduction

The design of complex embedded software systems requires a careful analysis of all involved elements. The software components, oftentimes referred to as the *system*, must be strictly aligned with the *environment* it must interact with. Strong timing requirements and complex interactions with controlled elements may further complicate the design as in the case of factory automation or manufacturing systems. The logic sequencing of operations must deal with motion control dynamics. In particular, robot cells are a clear example where the need for a verification of the integration between control software systems and motion control dynamic behavior is important for a safe and correct work-cycle programming.

The different natures of these two elements are difficult to address by means of a single all-encompassing technique/notation (e.g., hybrid automata [1]). The different parts require dedicated skills and also special-purpose modeling means. Hence, most approaches to the design of complex embedded software systems rely on different modeling and development tools for the system and the environment, and on co-simulation for assessing their coordinated behavior.

Co-simulation [2] allows individual components to be simulated by different simulation tools running simultaneously and exchanging information in a collaborative manner. A number of tools exist (e.g., [10], [12]) to realize co-simulation among different simulation environments in different application fields. Modeling techniques for the environment have been consolidated over the years: the environment is usually modeled as a continuous system by means of two well-known conceptual approaches. The *causal* approach —e.g., adopted by Simulink— decomposes the environment into a set of blocks, each modeled by an *ordinary differential equation* system, related through (sometimes artificially generated) causal relations between input and output variables. The *acausal approach*, supported by the object-oriented modeling language Modelica [3], requires that each (physical) component be specified through a *differential algebraic equation* system; their connections, which correspond to physical principles, are set by equating effort variables and by balancing flow variables. In contrast, the problem of modeling and designing the software control has been approached by means of different techniques. Informal approaches better match domain expertise, but they can only be partially analyzed, while formal methods are usually more difficult for the domain expert, but they typically provide richer support to analysis and verification.

Although the problem of integrating different approaches for modeling, designing, and analyzing embedded software systems together with the environment in which they operate has been widely studied, no conclusive solutions have been proposed, yet. The paper proposes **MCA**, the MADES Co-simulation Approach[1]. MCA aims to provide an innovative solution that blends the *acausal* modeling of the environment and a logic-based representation of the control to foster the formal verification of the resulting system. MCA is lightweight in that it allows designers to combine different, complementary formalisms (differential equations, logic formulae) in a seamless manner, instead of requiring them to fit their models to a single, all-encompassing notation, thus leveraging the respective strengths of involved formalisms. MCA leverages well-known modeling tools and standards, hides the formal representation used for verification to domain experts, and provides a comprehensive simulation environment to support both what-if analyses and the run of complete scenarios.

The MADES approach [20] proposes the use of a constrained version of UML, suitably enriched with some elements of MARTE (Modeling and Analysis of Real-Time and Embedded Systems [4]) for the design of the system. Designed models are automatically translated into predicates and axioms written in TRIO [5], which is a first-order linear temporal logic that supports a metric on time. ℤot [6], a bounded model/satisfiability checker, provides the verification capabilities. The definition of the environment is demanded to Modelica, which promotes non-proprietary and open source modeling of the environment. Moreover, it allows the user to describe components as differential algebraic equation systems, while more common and commercial tools (e.g., Matlab/Simulink) require that the system be described in

[1]The approach is named after the project in which it was developed.

978-1-4673-0311-8/12/$31.00 ©2012 IEEE

the state space form, which requires additional mathematical manipulations —and consequent waste of time— to the user. Since the paper presents MCA, we do not describe the UML models of the systems, and how they can be transformed into logic. Interested readers can refer to [20] for an in-depth presentation of the translation.

MCA is not *yet another* co-simulation solution, but comes with some key features. MCA supports the simulation of nondeterministic models, and thus the simulator tries different alternatives. MCA allows the user to specify logical constraints in a "descriptive" way, transforms them in the way required by the two verification engines, and guides the simulator to check them. For example, one may want to check wether a given condition is true every 5 time units (t.u. for short). It can also handle numeric values (in the domain of real numbers) and it can assign them nondeterministically to variables. This means we can define statements such as "a variable v must assume a value between 2 and 3 within 5 t.u.", and the tool picks the value randomly within the interval.

In order to focus on the architecture of the approach rather than on the complexity of the particular application, the approach is exemplified on a simple example, namely the motion control of a two d.o.f. radar. This example, relevant to the mechatronic field, was chosen as representative of more general mechatronic systems, such as robots or machine tools (with respect to the motion control problem only).

The rest of this paper is organized as follows. Section II presents the formal basis of the MADES Co-simulation Approach, and sketches the prototype simulation tool. Section III illustrates a simple case study to exemplify the main characteristics of the proposed solution. Section IV surveys related approaches and Section V concludes the paper.

II. Integrated Simulation

In a nutshell, the goal of simulation is, given models of the behavior of the system being developed and of its environment, to produce a trace that is "compatible" with both models, i.e., that does not violate either model. A *trace* is a sequence of values over a time interval, which correspond to the values assumed over that time interval by a set of quantities (or *variables*) of interest (physical quantities, such as temperature, velocity, altitude, or logical ones, such as a switch being turned on or off, etc.). Suppose, for example, that the variables of interest are the temperature Tp in a room and the state St on/off of a fan, and that the time interval is the discrete one $[0, 5]$; then a trace could be $[\langle \mathsf{Tp} = 25, \mathsf{St} = \mathsf{off}\rangle, \langle \mathsf{Tp} = 26, \mathsf{St} = \mathsf{off}\rangle, \langle \mathsf{Tp} = 27, \mathsf{St} = \mathsf{on}\rangle, \langle \mathsf{Tp} = 28, \mathsf{St} = \mathsf{on}\rangle, \langle \mathsf{Tp} = 27, \mathsf{St} = \mathsf{on}\rangle, \langle \mathsf{Tp} = 27, \mathsf{St} = \mathsf{on}\rangle]$. Usually, the model S of the system and the model E of its environment are expressed through different formalisms and using different notions of time. Model S is typically described through formalisms such as automata or logics, using a discrete notion of time (in which case interval $[0, 5]$ is, as in the example above, the sequence $[0, 1, 2, 3, 4, 5]$); the behavior of its physical environment E, on the other hand, is normally described through differential equations, using a continuous notion of time. Then, a notion of "hybrid" simulation is necessary, one that seamlessly mixes the concepts above.

The basic premise of MCA is that the system and environment models communicate with each other through shared variables. The reference model is the one depicted in Figure 1, which was originally proposed in [7]: the system and environment models have private variables (spv_i's and epv_j's in Figure 1), whose values are not visible outside of the model itself, and also some shared variables (ssv_i's and esv_j's) that are accessible to both models and are used to communicate information between them. Each variable belongs to a model (e.g., shared system variables ssv_i belong to the system model), which is used to compute the variable's value. In the MCA, variables can be real-valued quantities such as a speed or an acceleration, or discrete, finite (e.g., Boolean) signals such as a switch being turned on or off.

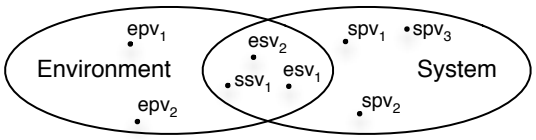


Fig. 1. A reference model.

In MCA, the dynamics of environment variables is governed by differential equations or algebraic laws expressed in the Modelica [3] language, and use a continuous notion of time. We assume that the differential equations defining the dynamics of environment variables admit a unique solution, hence the model E is deterministic [8]; that is, given an initial condition and a trace of the shared system variables, the trace of the environment variables is uniquely determined.

Model S, instead, is described as a set of constraints defined through temporal logic formulae in the TRIO syntax [5] that use a discrete notion of time. S can include constraints on shared variables. For example, one can define through TRIO formulae that "if the voltage remained greater than 3 for the last 2 t.u.'s, then a switch is turned on within 2 t.u.'s, and it stays on for 4 t.u.'s". The two different notions of time (continuous for the environment model, discrete for the system model) are reconciled through the notion of sampling [9]. If the "sampling period" is δ, every discrete t.u. k corresponds to a continuous time instant $k\delta$. Sampling, however, introduces approximations that must be dealt with carefully. Consider, for instance, the trace of variable v depicted in Figure 2. Its sampling satisfies, at instant 12, the condition "variable

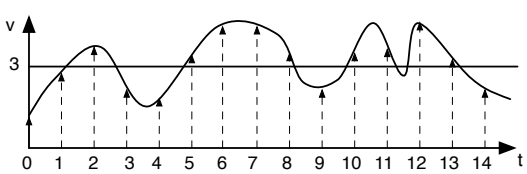


Fig. 2. Example of sampled variable.

v remained greater than 3 for 2 t.u.'s", but only because the

dip below 3 that occurred between samples 11 and 12 went undetected. To avoid such pitfalls we introduce a "regularity constraint" called non-berkeleyness [9], which requires that the conditions on environment shared variables appearing in model S do not change faster than δ continuous-time units (i.e., the distance between two changes in the conditions cannot be less than δ t.u.'s). Condition $v \geq 3$ in the trace of Figure 2 does not satisfy the non-berkeleyness constraint between sampling instants 11 and 12.

We consider a trace valid when the conditions on shared environment variables satisfy the non-berkeleyness regularity constraint, and we reject as unacceptable any trace that violates it, even if the sampled trace satisfies the system model. The non-berkeleyness constraint is introduced also as a "robustness" condition. In fact, it guarantees that, even if the sampling instants have an offset with respect to the ideal ones (e.g., values of the environment shared variables are taken at time instants $k\delta - \epsilon$, with $\epsilon < \delta$, instead of at times δk, which could be due to a small processing time of the collected data), conditions that are slightly weaker than the desired ones still hold, a less-than-ideal, but often still acceptable approximation. Consider, for example, condition $C =$ "the value of v was greater than 3 for the last 3 t.u.'s (current one included)", which holds at instant 8 in the trace of Figure 2. Suppose the actual sampling instant fell in between ideal instants 7 and 8. If the offset between actual and ideal sampling instants is big enough (though less than δ), condition C does not hold at the actual sampling instant. Nevertheless, a weaker condition $C' =$ "the value of v was greater than 3 between 1 and 2 t.u.'s ago" is still guaranteed to hold. This can correspond to the fact that the system is reacting to imprecise information, which can still be deemed acceptable as there is a bound on the imprecision (the bound being 2 t.u.'s, as shown by the approximation functions of [9]).

Temporal logic models are naturally nondeterministic, that is each model defines a set of traces even given the same trace of the environment shared variables. For example, consider constraint "if the voltage remained greater than 3 for the last 2 t.u.'s, then a switch is turned on within 2 t.u.'s, and it stays on for 4 t.u.'s": the system can react to condition "the voltage remained greater than 3 for the last 2 t.u.'s" by turning the switch on either 1 or 2 instants after the condition becomes true, which is a form of nondeterminism. Combined with the fact that, as mentioned above, the environment model is deterministic, this suggests the following algorithm for the advancement of the simulation from the state σ_k at sampling instant k to the state σ_{k+1} at sampling instant $k+1$ (where σ_i denotes the value of all variables at sampling instant i, which in turn corresponds to the continuous-time instant $i\delta$ if δ is the sampling period):

1) From the values $\sigma_k(\Sigma_E)$ of the variables that are visible to the environment model (where, w.r.t. the reference model of Figure 1, Σ_E is the union of all variables accessible by the environment, i.e. $\{epv_i\}_i \cup \{esv_j\}_j \cup \{ssv_l\}_l$), using the Modelica interpreter, compute $\sigma_{k+1}(\Sigma_E - \Lambda_S)$ (where $\Lambda_S = \{ssv_l\}_l$ are the system shared variables), i.e., the part of the new state that is "owned" by the environment. Also, in this step the simulator checks that Boolean conditions on environment variables (e.g., $v > 3$) appearing in the model S satisfy the non-berkeleyness constraint. If the trace produced is not non-berkeley, then state $\sigma_{k+1}(\Sigma_E - \Lambda_S)$ is discarded, and the algorithm continues from step 4.
2) Check, using a satisfiability checker for metric temporal logic formulae, whether there exists a discrete-time trace of the system model S that has $\langle \sigma_0(\Sigma), \sigma_1(\Sigma), \ldots \sigma_k(\Sigma), \sigma_{k+1}(\Sigma_E - \Lambda_S) \rangle$ as prefix (where Σ is the set of all variables); if there is, complete state σ_{k+1} with $\sigma_{k+1}(\Sigma_S - \Lambda_E)$ (where Σ_S is the set of all variables visible to the system, and Λ_E are the environment shared variables), and repeat from step 1.
3) If the satisfiability checker cannot complete the prefix $\langle \sigma_0(\Sigma), \ldots, \sigma_{k+1}(\Sigma_E - \Lambda_S) \rangle$, then the simulation backtracks to $\sigma_{k-1}(\Sigma)$, and repeats from step 1, but feeding the satisfiability checker constraints to avoid producing again the same state $\sigma_k(\Sigma)$ that led to the dead-end.
4) If the Modelica interpreter determined that the trace of the environment variables is not non-berkeley, then the simulation backtracks to $\langle \sigma_0(\Sigma), \sigma_1(\Sigma), ...\sigma_k(\Sigma_E - \Lambda_S) \rangle$, feeding the satisfiability checker constraints that avoid producing $\sigma_k(\Sigma_S - \Lambda_E)$ that led to the non-berkeleyness.

A. Tool support

We implemented a tool[2] realizing the schema outlined above. The tool, whose architecture is outlined in Figure 3, is based on a coordinator that sends commands to the interpreters of the environment and of the system model. These

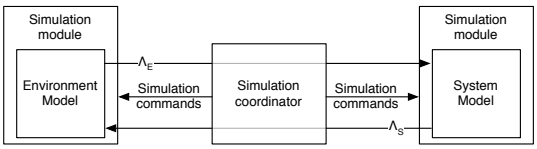


Fig. 3. Simulation architecture.

commands are essentially "advance one step" and "backtrack", plus suitable constraints (e.g., initial conditions, values to be avoided) to guide the two simulation modules.

The architecture depicted in Figure 3 comprises:

- The simulation module for the environment, which is based on the OpenModelica interpreter[3].
- The simulation module for the system based on the ℤot[4] tool, which can function as a satisfiability solver for temporal logic formulae.
- The coordinator, a stand-alone Java application which takes as input the environment and system models and some suitable parameters (e.g., sampling interval δ, length

[2]The tool is available from http://code.google.com/p/mades/downloads/list.
[3]http://openmodelica.org
[4]http://zot.googlecode.com

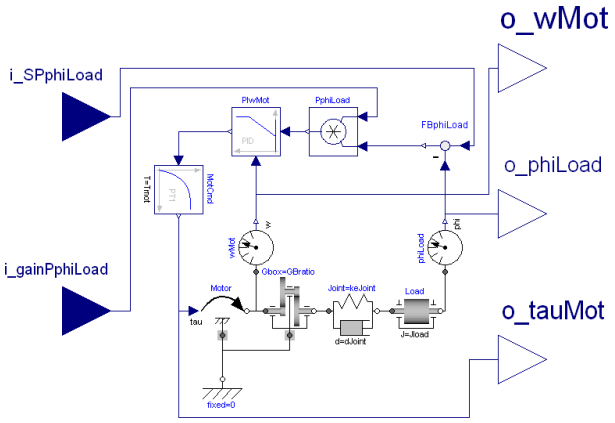


Fig. 4. Modelica scheme for the example.

of the trace to be produced measured in number of t.u.'s) then executes the algorithm described above to produce a trace of the overall system.

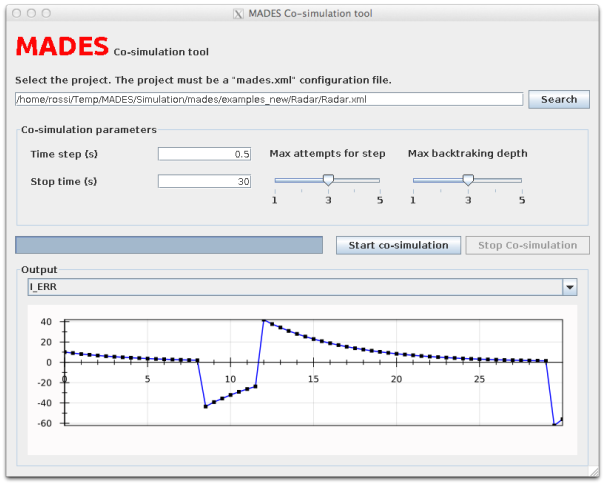
III. Example Simulation

In this section we discuss a case of simulation carried out on an example radar system. The environment model represents a motorized axis of a type compatible with typical radar pointing applications. An electric motor drives a flexible joint, and is endowed with cascade velocity-position control. The position set point and the gain of the outer controller are the manipulated inputs: the role of the former is obvious, while the latter can be modified if for example there is the necessity of smoothing the control actions to avoid over-currents. The motor angular velocity, the axis position and the motor torque (proportional to the current) are the output variables. Figure 4 shows the corresponding Modelica scheme.

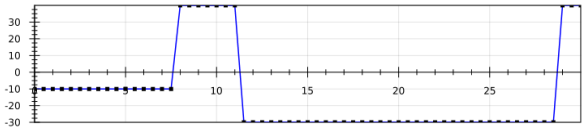
The software system interacting with the radar reads the current positioning error with respect to the desired set point and the motor current, and sets the controller gain depending on its internal logic; it also takes as input requests for set point change made by the user, and sets the set point depending on these. The model of the software system is expressed as a collection of temporal constraints which provide a high-level, partial description of the desired behavior of the software being designed. These constraints, which in a way represent requirements on the system under design, set rather loose boundaries on the design space, but leave many choices open, so the resulting system model is in many points nondeterministic. For example, one of the constraints defines that in each instant (i.e., "always"), if the user requests a set point change, then she does not ask again for a new set point for the next 5 t.u.'s (where a t.u. corresponds to a second). This is expressed by the following temporal logic formula [5], where reqSp is a predicate that holds in the instants in which the user asks for a set point change:

$$\mathrm{Alw}(\mathsf{reqSp} \rightarrow \mathrm{Lasts}(\neg \mathsf{reqSp}, 5)) \,. \tag{1}$$

We also require that, whenever the user requests a set point change, after the request the system will actually change the



(a)



(b)

Fig. 5. Results of the first simulation: error 5(a) and set point 5(b).

set point within the next 2 t.u.'s, as formalized by the next formula, where spChg is a predicate that holds in time instants in which the "set point change" command is issued; reqSpV and spV are variables representing, respectively, the value of the requested set point, and the current set point:

$$\mathrm{Alw}(\mathsf{reqSp} \rightarrow \mathrm{WithinF}(\mathsf{spChg} \wedge \mathsf{reqSpV} = \mathsf{spV}, 2)) \,. \tag{2}$$

A similar logic formula, shown next, expresses the requirement that a set point change occur only if the user has previously issued a change request that has not yet produced a change of set point since then.

$$\mathrm{Alw}(\mathsf{spChg} \rightarrow \mathrm{Since}(\neg \mathsf{spChg}, \mathsf{reqSp})) \,. \tag{3}$$

Other constraints have been imposed on the system, but they are not shown here for the sake of brevity.

Thanks to the logic-based nature of the MCA we can use temporal logic constraints also to "guide" the simulation by restricting the behavior of some variables. For example, for a first simulation we imposed the following constraints on the trace to be produced.

- One of the first 10 instants the set point must be 40, and one of the first 15 it must be -30 (notice that the constraint does not impose an ordering between the values: the set

point might be -30 *before* it becomes 40):

$$\text{WithinF}(\mathsf{spV} = 40, 10) \wedge \text{WithinF}(\mathsf{spV} = -30, 15)\,. \tag{4}$$

- The gain of the controller changes only once during the simulation (noChG is a predicate that holds in those instants in which the gain does *not* change):

$$\text{Until}(\mathsf{noChG}, \neg\mathsf{noChG} \wedge \text{AlwF}(\mathsf{noChG}))\,. \tag{5}$$

Figure 5 shows the results of the first simulation, which was carried out with a sampling period of $\delta = 0.5$ over a period of 30 t.u.'s. In particular, Figure 5(a) shows the window of the simulation tool with the set point error; Figure 5(b) shows the set point. We notice that the set point changes twice before 15 t.u.'s have elapsed, once when it is set to 40, and once when it is set to -30. In addition, the set point is changed one more time before the simulation ends, at time 29.

For a second simulation we also introduced a set of formulae that constrain the controller gain to remain within certain ranges depending on the value of the error in the last 2 t.u.'s. For example, we imposed that if the error (represented by variable err) was between 60 and 90 degrees for the last 2 t.u.'s, then the gain (represented by variable g) must be between 1.6 and 2 (other formulae stating similar constraints are not shown for brevity):

$$\text{Alw}(\text{Lasted}(60 < \mathsf{err} \leq 90, 2) \rightarrow 1.6 < \mathsf{g} \leq 2) \tag{6}$$

For this second simulation, we changed the set of constraints that are used to guide the simulation. In particular, we replaced formulae (4)-(5), with the following ones:

- Within the first 25 t.u.'s of the simulation, there must be a request of set point change from the user, which asks for a set point of -50.
- Within instants 25 ad 33 there must be also another set point change request, whose value is left unconstrained.

Figure 6 shows the results of the second simulation, which was carried out with $\delta = 1$ and a horizon of 40 t.u.'s. In particular, it shows that, due to constraint (6) (and similar ones not shown here), as the error changes (Figure 6(a)), the gain also changes (Figure 6(b)).

For a third simulation we added a further constraint on the rate of change for the gain, a variable that is controlled by the software system. More precisely, we introduced the following formula, which states that between two changes of the gain there must be at least 4 t.u.'s:

$$\text{Alw}(\neg\mathsf{noChG} \rightarrow \text{Lasts}(\mathsf{noChG}, 4))\,. \tag{7}$$

However, with this new constraint the simulator was not able to produce a trace, and it stopped after it reached the maximum numbers of retries and backtracks (both parameters of the simulation, as seen in Figure 5(a)). This is due to the fact that in the trace produced by the simulator the gain was changed right before a set point change request, which requires a further change in the gain, thus violating constraint (7). As the simulator does not explore the whole state space of the possible executions, this does not necessarily mean that the closed-loop system is unfeasible, but it highlights a source of potential problems in the set of constraints. In this case, if one cannot predict when the user will decide to change the set point of the radar, imposing minimum delays between changes in the controller's gain might produce a situation of conflict.

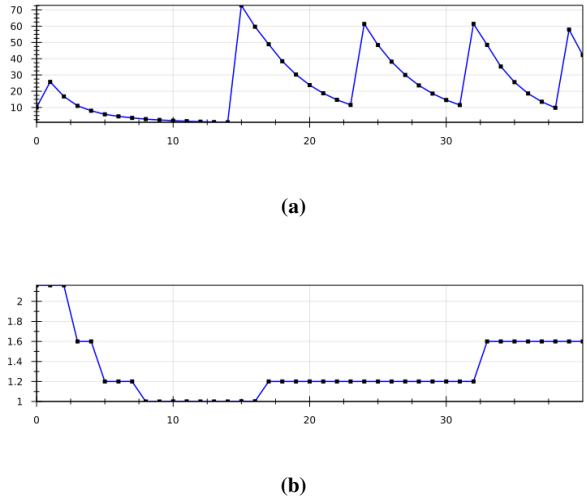


Fig. 6. Results of the first simulation: error 6(a) and gain 6(b).

The MCA can be used not only to carry out simulation activities of the closed-loop system including both the software being designed and its environment, but also to prove properties on the former. In particular, one can, from the set of temporal logic constraints describing the desired behavior of the software system, prove (or disprove) properties on its internal mechanisms (hence, independent of the behavior of the environment). Hence, we used the ℤot tool to prove some properties for the radar example discussed in this section. The properties to be checked are, as usual in formal verification approaches, expressed as temporal logic formulae; in the MADES approach, the TRIO [5] logic is used to formalize both the system constraints and the properties to be proved. Hereafter, we present an example of verification activity carried out on the radar system.

More precisely, we proved that there cannot be two consecutive set point change requests without a set point change in between, which is formalized by the following formula:

$$\text{Alw}(\mathsf{reqSp} \rightarrow \text{Until}(\neg\mathsf{reqSp}, \mathsf{spChg}))\,. \tag{8}$$

We fed formula (8) to the ℤot tool, together with the set of constraints describing the system, and the tool determined that in fact the property holds for the system. Intuitively, the reason for this lies in the timing of change requests and set point changes: change requests cannot occur more often than every 5 t.u.'s, per constraint (1), while set point changes follow change requests after 2 t.u.'s, per constraint (2).

IV. Related work

Co-simulation has been widely studied over the years. For example, Stateflow, developed by MathWorks [10], is a control

logic tool used to model reactive systems via state charts and flow diagrams within a Simulink model. Stateflow uses a variant of the finite-state machine notation established by Harel [11], enabling the representation of hierarchy, parallelism and history within a state chart. Using add-on code generation products, a Stateflow user can automatically generate C, HDL, and PLC code from a Stateflow chart.

While Simulink actually implements a causal approach to modeling, where sub-models are connected by input and output ports, AMESim [12] pushes the idea that the variables, which are shared at the ports between sub-models, are physical entities and operate in both directions, thanks to a Bond graph theory approach. This solution facilitates the link among different physical domains.

The Modelica standard library StateGraph [13] also provides components to model discrete event and reactive systems in a convenient way. It is based on the JGraphChart [14] method and takes advantage of Modelica features for the *action* language. JGraphChart is an evolution of Grafcet [15] to include elements of StateCharts [11] that are not present in Grafcet/Sequential Function Charts. Therefore, the StateGraph library has a similar modeling power as StateCharts, but it avoids some of its deficiencies. Some improvements have been recently added to the new library StateGraph2 [16]. Note however that the model is always deterministic due to Modelica's single assignment rule.

ModelicaDEVS [17], which is a free library for modeling discrete event-oriented systems using the DEVS methodology, has also been developed in the Modelica/Dymola environment [18] as an implementation of a Modelica version of PowerDEVS [19]. However, it adopts a quite different approach for the numerical integration of the environment model: it adopts *state quantization* rather than applying *time discretization* to the solution of the DAE system describing the environment. Thus, state variables evolve individually, with no need to update them simultaneously.

As already said, the main difference between the MCA approach and the other mixed discrete events/continuous time simulation tools is the description of nondeterministic models, entailing the need for a mechanism that allows the exploration of different alternatives for the coupled discrete/continuous simulation, in case of constraint violation. MCA also allows, from the system side of the architecture, a formal verification of the software system, which is also not considered by more traditional approaches.

V. Conclusions and future work

The paper presents MCA, the MADES Co-simulation Approach, an innovative proposal for the co-simulation of the software system and its surrounding environment. The former is modeled through metric temporal logic formulae; the latter is rendered by using Modelica, along with its eco-system. The result can be seen as a lightweight solution where users can combine different complementary formalisms (differential equations, logic formalisms) and expertise in a seamless manner. These two models are the inputs for MCA to produce an execution trace that is "compatible" with them, that is, that does not violate either model. In the MADES approach, logic formulae can be derived from UML models, suitably augmented with a temporal logic semantics [20].

The paper introduces the theoretical basis of MCA and exemplifies them on a simple case study. The flexibility provided by the approach and its key characteristics are interesting and encouraging, but they also demand for further analyses and experiments. This is the main gaol of our future work. A more thorough assessment will provide us the opportunity to conduct further experiments with the flexibility of the approach, and it will also give us more insights about it.

Acknowledgments: Work supported by the European Community's Seventh Framework Program (FP7/2007-2013) under grant agreement n. 248864 (MADES). We thank Michele Sama for his work on the co-simulator.

References

[1] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi, "HYTECH: A model checker for hybrid systems," *Int. J. on STTT*, vol. 1, no. 1–2, 1997.
[2] K. Hines and G. Borriello, "Dynamic communication models in embedded system co-simulation," in *Proc. of DAC*, 1997, pp. 395–400.
[3] P. A. Fritzson, *Principles of object-oriented modeling and simulation with Modelica 2.1*. John Wiley and Sons, 2004.
[4] Object Management Group, "UML Profile for Modeling and Analysis of Real-Time Embedded Systems," Tech. Rep., 2009, formal/2009-11-02.
[5] E. Ciapessoni, P. Mirandola, A. Coen-Porisini, D. Mandrioli, and A. Morzenti, "From formal models to formally based methods: An industrial experience," *ACM TOSEM*, vol. 8, no. 1, pp. 79–113, 1999.
[6] M. Pradella, A. Morzenti, and P. San Pietro, "The symmetry of the past and of the future: bi-infinite time in the verification of temporal properties," in *Proceedings of ESEC/SIGSOFT FSE*, 2007, pp. 312–320.
[7] C. A. Gunter, E. L. Gunter, M. Jackson, and P. Zave, "A reference model for requirements and specifications," *Software, IEEE*, vol. 17, no. 3, pp. 37 –43, 2000.
[8] C. A. Furia, D. Mandrioli, A. Morzenti, and M. Rossi, "Modeling time in computing: a taxonomy and a comparative survey," *ACM CSUR*, vol. 42, no. 2, pp. 6:1–59, 2010.
[9] C. A. Furia and M. Rossi, "A theory of sampling for continuous-time metric temporal logic," *ACM TOCL*, vol. 12, no. 1, pp. 8:1–8:40, 2010.
[10] MATLAB, *version 7.10.0 (R2010a)*. The MathWorks Inc., 2010.
[11] D. Harel, "Statecharts: A visual formalism for complex systems," *Sci. Comput. Program.*, vol. 8, pp. 231–274, June 1987.
[12] W. Jere, *Amesim*. International Book Marketing Service Ltd, 2011.
[13] M. Otter, K. E. Årzén, and I. Dressler, "Stategraph-a modelica library for hierarchical state machines," in *Proc. of the Int. Modelica Conf.*, 2005, pp. 569–578.
[14] K. E. Årzén, R. Olsson, and J. Akesson, "Grafchart for procedural operator support tasks," in *Proc. of the IFAC World Congress*, 2002.
[15] R. David and H. Alla, *Petri Nets and Grafcet: Tools for Modelling Discrete Event Systems*. Prentice Hall, 1992.
[16] M. Otter, M. Malmheden, H. Elmqvist, S. E. Mattsson, and C. Johnsson, "A new formalism for modeling of reactive and hybrid systems," in *Proc. of the Modelica Conf.*, 2008.
[17] B. P. Zeigler, H. Praehofer, and T. G. Kim, *Theory of Modeling and Simulation, Second Edition*, 2nd ed. Academic Press, 2000.
[18] T. Beltrame, *Design and development of a Dymola/Modelica library for discrete event-oriented systems using DEVS methodology*. ETH Zürich, 2006.
[19] F. Bergero and E. Kofman, "Powerdevs: a tool for hybrid system modeling and real-time simulation," *SIMULATION*, 2010.
[20] L. Baresi, A. Morzenti, A. Motta, and M. Rossi, "Towards the uml-based formal verification of timed systems," in *Proc. of FMCO*, ser. LNCS, vol. 6957, 2010, pp. 267–286.