Proceedings of the
44th IEEE Conference on Decision and Control, and
the European Control Conference 2005
Seville, Spain, December 12-15, 2005

Unified Modeling and Verification of Logic Controllers for Physical Systems

Marcello Bonfè, Cesare Fantuzzi and Cristian Secchi

Abstract— **The paper describes a modeling approach that aims to provide a unified framework for the specification and verification of logic controllers for multi-domain physical systems. The proposed modeling methodology is based on the cardinal principle of object orientation, which allows to describe both control software and physical components using the same basic concepts, like classes and interface ports, and the same modeling notation, based on the UML language. Thanks to this unified approach, it is possible to describe structural and behavioral aspects of any multi-domain system coupled with a logic control device. Moreover, the behavior of the closed-loop system can be analyzed with formal verification techniques for hybrid systems, in order to prove correctness properties otherwise difficult to verify considering only discrete-event models.**

I. INTRODUCTION

The object-oriented approach is a cardinal principle for many modeling, analysis and design techniques developed for different branches of engineering, not only related to software development. For example, emerging technologies for industrial control systems, like the IEC 61499-1 [1] standard for distributed controllers design, support engineers with many features oriented to the encapsulation and reuse of software modules. Moreover, modeling languages for complex physical systems adopting object-oriented principles, like Bond Graphs [2], are more and more popular, since they allow to capture in a very natural way the structure of systems that contain physical components from heterogeneous domains (mechanical, electrical, hydraulic, etc.).

Even though all the mentioned languages share the basic principles and are well-known in their application domain, it is hard, with the current tools, to integrate them, in order to describe all the aspects related to the design of complex industrial systems within a single modeling framework. A unified language embedding structural and behavioral aspects of control software and physical components would provide on one hand a *lingua franca* for the communication between engineers of different disciplines and, on the other hand, the possibility to verify the actual behavior of a computer-controlled system by considering its hybrid dynamics.

Starting from these remarks, we propose a practical approach for object-oriented modeling of multi-domain systems including physical components and control software, based on an extension of the specification language UML [3] that embeds the concepts of Bond Graphs and their mathematical formalization as port-Hamiltonian systems [4]. A model designed with the proposed language is formalized as an hybrid dynamic system, whose behavior can be analyzed with specific verification tools. In particular, the case of study presented in the paper has been verified using CheckMate [5]. The rest of the paper is organized as follows: in Sec. II we describe the background on UML and Bond Graphs, in Sec. III we specify the conceptual scenario of multi-domain object-oriented systems and formalize the mapping between Bond Graphs and UML, in Sec. IV this mapping is described using the UML notation and in Section Sec. V we provide an example of the application of the proposed modeling language to an industrial case of study. The paper ends with some concluding remarks and some directions for future work.

II. BACKGROUND ON MODELING LANGUAGES

UML - The UML language is defined in [3] by means of an extensible meta-model and its notation supports the specification of functional requirements, structural properties, objects interactions and objects' internal behavior. Structural views of a system are described with UML by means of *Class Diagrams*, while behavioral specifications can be described with UML by associating a *State Diagram* to each class of a model. In this way, the event-driven reactive behavior of all the instances of that class is represented exhaustively.

An interesting feature of UML is its extensibility, which allows to model domain-specific or methodology oriented concepts by means of *stereotyped* elements. A consistent set of UML stereotypes is called a *profile*. A well-known example of UML profile, called UML-RT, is the one described in [6]. The UML-RT profile allows to model real-time, event-driven and distributed software architectures, by means of highly encapsulated components called *capsules*, interacting with each other through well-defined *ports* and *protocols*. A port instance can be embedded in a composite class in order to provide controlled access to the internal behavior of the class itself or of one of its sub-components, according to a reference protocol. Adopting the definitions in [7], a protocol can be formally described as a 4-tuple $\mathcal{P} = (\mathcal{E}, \mathcal{R}, \mathcal{B}, \mathcal{Q})$, in which $\mathcal{E}$ is the set of *events* that can be exchanged between the participants in the protocol (i.e. operation calls, messages, etc.), $\mathcal{R}$ is the set of *protocol roles* that the participants can play, $\mathcal{B}$ is the reference behavior (i.e. the set of admissible strings from the event alphabet) and $\mathcal{Q}$ is the expected quality of service of the protocol. A port *implements* a given protocol role, which means that the knowledge of the reference protocol always allows to

M. Bonfè is with the ENDIF, Università di Ferrara, Via Saragat 1, 44100 Ferrara, Italy (mbonfe@ing.unife.it)

C. Fantuzzi and C. Secchi are with the DISMI, Università di Modena e Reggio Emilia, Via Allegri 15, 42100 Reggio Emilia, Italy

communicate correctly with a composite class including that port, independently of the actual internal details of the class.

Bond Graphs - The modeling approach based on components and ports is the foundation also of the Bond Graphs formalism [2]. This language allows to model in a very elegant and, in a sense, object-oriented way any kind of physical systems, thanks to the fact that in every physical domain there is a pair of variables, defined on a pair of dual vector spaces[1] $\mathcal{F}_{p0}$ and $\mathcal{E}_{p0} = \mathcal{F}^*_{p0}$, whose dual product is *power*. The two factors of power are generically called, in the Bond Graphs terminology, *flow* and *effort*. A *power port* is defined by a pair $(f, e) \in \mathcal{F}_{p0} \times \mathcal{E}_{p0}$ and represents the means through which a physical system can interact with the rest of the world. With the bond graphs modeling strategy, any lumped parameters physical system is represented as a set of basic elements, distinguished as energy storing elements, energy dissipating elements and energy sources, each one endowed with one or more power ports through which it can exchange energy with the others. The dynamic behavior of any physical system is due to the exchange of energy that takes place among its constitutive elements. A bond graph shows explicitly the network structure along which the various elements interact, exchanging energy, by means of connectors called *bonds*, which relates basic elements and junctions, whose behavior is governed by Kirchhoff-like laws, or power preserving transformations (i.e. transformers and gyrators).

Each (and *only*) element that can store energy has states associated to it, in a way such that each state represents the storage of energy flowing through a power port. Even if bond graphs are in general acausal [2], it is possible to assign a causality to each element and thus to fix the role of input or output for each variable in a power port. Very often, energy storing elements have an integral causality associated and their behavior is represented by

$$\begin{cases} \dot{x}_i = u_i \\ y_i = \frac{\partial H}{\partial x_i} \end{cases} \qquad i = 1, \dots, m \tag{1}$$

where m is the number of power ports associated to the element, H is a function of the state that represents the energy stored in the element in a given configuration and u_i and y_i are dual power variables representing the i^{th} power port. Dissipative elements impose an algebraic relation between their input/output variables of the type $\langle u, y \rangle \geq 0$, namely such that power is absorbed by the element. Sources of energy can be either sources of flow (i.e. fixing an flow to a certain value), or sources of effort (i.e. fixing an effort). An elegant mathematical formulation of Bond Graphs is the one described in [4]. In the reference, Bond Graphs are mapped into port-Hamiltonian systems exploiting the concept of *Dirac structure*, as explained in [8], in order to provide a formal representation of the power-conserving interconnection structure of a physical system. Once that a coordinate system has been fixed, several matrix representation of a Dirac structure are available [8], of which a very effective one is the so called *kernel representation*. Given a system with m power ports and the following vectors:

$$f = \begin{pmatrix} f_1 \\ \vdots \\ f_m \end{pmatrix} \in \mathcal{F}_p \qquad e = \begin{pmatrix} e_1 \\ \vdots \\ e_m \end{pmatrix} \in \mathcal{E}_p \tag{2}$$

where f_i and e_i are the flow and the effort associated to the i^{th} power port, the behavior of any power preserving interconnection structure is defined in kernel representation by a relation of the kind:

$$F(x)f + E(x)e = 0 \tag{3}$$

where $x \in \mathcal{X}$ is the state of the physical system and $E(x)$ and $F(x)$ are matrices such that:

$$\begin{aligned} &E(x)F^T(x) + F(x)E^T(x) = 0 \\ &\dim[\ F(x) \quad E(x)\ ] = \dim \mathcal{F}_p \end{aligned} \tag{4}$$

III. Unified Modeling of Multi-domain systems

In order to model, analyze and design logic control systems for industrial processes within an object-oriented perspective, it is necessary to remark that software and physical aspects are tightly coupled and, therefore, they should be taken into account at the same time. In particular, a module in a control software will have a well-defined interface, related to input/output electrical signals, to interact with sensors and actuators on its physical counterpart, and an events/data interface to interact with other control modules. The first interface represents the *hardware port* of the control module, while the second is conceptually the *software port*. The aggregation of a control module, its hardware and software ports and the related physical components is what we consider as a *mechatronic class*, within an integrated object-oriented approach.

Of course, within the physical domain, the bond graph modeling approach is one of the most effective tool and UML, especially including the concepts of ports and protocols, can be easily adopted to model industrial control systems. However, since UML can be extended to include domain-specific concepts, it is possible to define a proper mapping between bond graph elements and UML elements, in order to formalize the semantics of the interaction between control software and physical components. In particular, it is easy to see that the inter-domain access point is represented by the hardware port, which performs the sampling of sensor signals and updates commands for actuators. The first kind of operation is a measurement of either one of the state variables of the physical system (i.e. the position of a mechanical part) or one effort or flow variable (i.e. voltage, velocity, pressure, etc.), while the control action is executed by *modulating* either an effort/flow source or a power transformer/gyrator. Signal ports and modulating ports, as means to define the interaction of a physical system with a feedback controller, are standard tools in bond graphs [2].

[1]Given a vector space $\mathcal{F}_{p0}$, its dual $\mathcal{E}_{p0} = \mathcal{F}^*_{p0}$ is the vector space of the linear operators on $\mathcal{F}_{p0}$. Given $f \in \mathcal{F}_{p0}$ and $e \in \mathcal{E}_{p0}$, their dual product is given by $\langle e, f \rangle = e^T f$

In order to define a precise integration of bond graphs and UML, it is necessary to give first a system theoretic description of a UML model. To this aim, we consider a subset of UML which is consistent with the UML-RT profile [6], namely we assume that the interface of a class can be defined only in terms of ports. Therefore, we will equivalently use the word *class* or *capsule* to refer a composite structure having an internal behavior (i.e. the state machine), possibly a set of lower-level components and boundary interaction points, namely the *ports*. In this framework, a system made up of p capsules and m ports can be described as a 5-tuple $(A, \mathcal{P}, \mathcal{C}, \alpha, \psi_{\mathcal{P}})$, whose elements are defined in the following. The set

$$A = A_1 \times \cdots \times A_p \qquad a = (a_1, \ldots, a_p) \tag{5}$$

is the set of the *attributes* of the overall system, in which each set A_i represents the set of attributes of the capsule i. Each set A_i, and consequently A, in general has no structure and it can be composed of several and etherogeneous elements such as lists, integers, data structures, and so on. The interaction between capsules is defined by the protocol $\mathcal{P} = (\mathcal{E}, \mathcal{R}, \mathcal{B}, \mathcal{Q})$, as detailed in Sec. II. In particular the event set is given by:

$$\mathcal{E} = \bigcup_{i=0}^{m} \mathcal{E}_i \tag{6}$$

where $\mathcal{E}_i$ represents the event alphabet that the i^{th} port can exchange in the protocol. It is possible to define the following set:

$$E = \mathcal{E}_1 \times \cdots \times \mathcal{E}_m \qquad \varepsilon = (\varepsilon_1, \ldots, \varepsilon_m) \tag{7}$$

whose elements describe the events waiting to be processed in a given ports configuration. Obviously some ε_i may be empty, meaning that no message is crossing the corresponding port. $\mathcal{C}$ represents the interconnection structure, which is the medium over which the protocol $\mathcal{P}$ is implemented. The map $\alpha : A \times E \rightarrow A$ is the *attribute transition map* and it describes how the attributes of the capsules change, according to the behavior defined by the capsules' state machines. The map $\psi_{\mathcal{P}} : A \times E \rightarrow E$ is the *port transition map* and it describes the dynamics of events across the ports, according to the reference protocol and to the capsules' state machines.

Now we can show how to map any physical system into this modeling framework. Since a physical system is made up by a set of basic physical elements that exchange energy, we consider these structural components as UML capsules. Let p be the number of capsules describing a physical system. The attributes of each capsule are represented by the physical states, therefore the set A becomes:

$$A = \mathcal{X}_1 \times \cdots \times \mathcal{X}_n = \mathcal{X} \qquad x = (x_1, \ldots, x_n) \tag{8}$$

in which $\mathcal{X}$ is usually a differentiable manifold. In general, $n \neq p$, since each capsule can be characterized by several states or by zero states (e.g. purely dissipative elements). In order to formalize the protocol modeling the dynamics of a physical system, we must first define which kind of information are exchanged between its components. The fundamental information is of course *energy*, which is exchanged through *power ports*. For a power port, we can define as the event alphabet the related space of power variables $\mathcal{F}_{p0} \times \mathcal{E}_{p0}$, in which $\mathcal{F}_{p0}$ is the space of flow vectors and $\mathcal{E}_{p0}$ is the space of effort vectors.

In addition, each physical capsule can have ports that are not directly related to the exchange of energy, but they only transmit signals related to the physical state of a component or related to the modulation of a source or of a transformer, which equals to modulating the exchange of energy along the interconnection structure. In a physical system, the *means* (e.g. electrical wires, pipes, etc.) through which capsules are interconnected represente the interconnection structure $\mathcal{C}$ introduced before, while the *way* in which the various subsystems are joined represent the energetic paths, namely the protocol $\mathcal{P}$, implemented over the interconnection structure $\mathcal{C}$. Since the behavior of physical systems is continuous, the event set $\mathcal{E}$ has infinite cardinality and, therefore, we call it *event space*. As efforts and flows are exchanged through the power ports of the interconnection, the event space contains both $\mathcal{F}_p$ and $\mathcal{E}_p$. Once causality has been fixed, each power port can play a specific role: it can provide a flow and, therefore, receive an effort, or it can provide an effort and receive a flow; we call these roles *energy roles*. In general, the way in which energy is exchanged depends on the states characterizing the interconnected capsules, as mentioned before, thus ports that carry signals that are used to modulate the interconnection structure play a further role in the protocol: a *modulating role*. Each capsule can participate to the protocol both by exchanging directly energy through ports that play energy roles and by modulating the energy transfer through ports that play a modulating role. Therefore, the state manifold $\mathcal{X}$ of the physical system is also part of the event space, which is, summarizing, given by $\mathcal{E} = \mathcal{F}_p \cup \mathcal{E}_p \cup \mathcal{X}$. Since the dynamics of a physical system is continuous, the protocol behavior $\mathcal{B}$ is continuous. The behavior of physical protocols, which must always have the property of being power preserving, is defined by a Dirac structure and, for example, its kernel representation as in Eq.(3), which allows to calculate the efforts or flows that have to be sent to the power ports in which they appear as received signals, using the efforts or flows incoming from power ports in which they appear as supplied signals [8]. When modeling physical protocols, we consider meaningless any quality of service assessment, therefore we do not formalize $\mathcal{Q}$.

Let $m \geq n$ be the number of power ports of the overall system. Once causality has been assigned, it is possible to distinguish an input signal u_i and an output signal y_i per each power port. Thus, it is possible to define the attribute transition map as a continuous function:

$$\alpha : \mathcal{F}_p \times \mathcal{E}_p \times \mathcal{X} \rightarrow \mathcal{X} \qquad (f(t), e(t), x(0)) \rightarrow x(t) \tag{9}$$

The function α defines the continuous internal behavior of each interconnected capsule. In particular, assuming integral causality, for each state we have that:

$$x_i(t) = \alpha(f, e, x(0)) = x_i(0) + \int_0^t u_i(\tau)d\tau \qquad (10)$$

where u_i can be either f_i or e_i depending on the port causality. The port transition map is given by:

$$\psi_{\mathcal{P}} : \mathcal{F}_p \times \mathcal{E}_p \times \mathcal{X} \to \mathcal{F}_p \times \mathcal{E}_p \times \mathcal{X} \qquad (11)$$

Each signal crossing the port at time t can be calculated through the state information and the port configuration at time t. In particular, per each power port associated to an energy storing element we have that:

$$y_i(t) = \frac{\partial H}{\partial x_i}\Big|_{x(t)} \qquad i = 1, \ldots, n \qquad (12)$$

where y_i can be either e_i or f_i depending on the port causality. $H(x)$ is the function that expresses the energy stored into the system. In case of power ports associated to energy dissipation, we have that:

$$y_i(t) = g_i(u_i(t)) \quad i = n+1, \ldots, m \qquad (13)$$

where g_i is the algebraic function characterizing the port. In case of signal ports, those that play the modulating role in the communication protocol, we have that:

$$m_i(t) = z_i(t) \qquad i = 1, \ldots, n \qquad (14)$$

in which z_i can be any kind of physical variable e_i, f_i or x_i. Once the signals crossing the ports associated to the output of power ports and those that cross the modulating ports are available, it is possible to calculate, through the protocol behavior equation, the inputs of the power ports, thus completing the ports configuration at time t.

IV. UML STEREOTYPES FOR PHYSICAL SYSTEMS AND HYBRID MODELS

In this section, we define UML stereotypes for physical systems, extending those of the UML-RT profile [6], in order to provide a unified notation to model all the components of industrial systems with logic controllers.

In UML-RT capsules are modeled by the `<<capsule>>` stereotype of class. Ports are represented by the `<<port>>` stereotype of class and each capsule is in a composition relationship with its ports. A connector is modeled by an association between the ports that are interconnected. A protocol is modeled by the `<<protocol>>` stereotype of Collaboration and is in a composition relationship to each of its protocol roles that are represented by the `<<protocolRole>>` stereotype of ClassifierRole.

Physical capsules, modeled with the stereotype `<<Physical>>`, can interact with the other physical capsules by means of the physical protocol, which is a collaboration, stereotyped as `<<PhysicalProtocol>>`, of physical elements. The ports collaborating in a physical protocol can have one of the following roles: effort-supplier/flow-receiver (ES-FR), flow-supplier/effort-receiver (FS-ER) and modulating (M). The first two kind of roles are related to power ports. It is important to note that a power port always conveys two signals: effort and flow. Therefore, we can also assume a standard textual notation to refer these signals, with the style `PortName.e` and `PortName.f`. Moreover, we allow a physical capsule to have standard signal ports, like those of software capsules, which provide information (i.e. measurements) related to efforts, flows or states. These signal ports allow to model the sensors used by the control software to implement feedback strategies, representing the point of interaction between software and physical components and, therefore, the bridge between continuous and discrete domain. We also assume that these ports implicitly perform operations like time-triggered sampling or comparison of analog values with given thresholds to generate logic signals or events. An example of the usage of the stereotypes defined in this section is shown in the Class Diagram of Fig. 1.

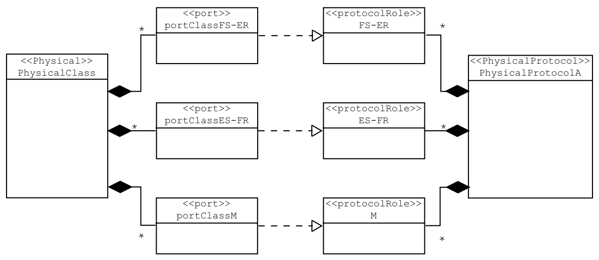

Fig. 1. UML representation of a physical subsystem

While in UML-RT capsules are event-driven entities, physical capsules represent time driven entities with continuous behavior. In order to include these features in UML, we refer to physical time as an abstract class that models a continuous and unbounded progression of time instants, belonging to a fully ordered dense set. Therefore, we assume that capsules representing physical subsystems are in an association relationship with a *PhysicalTime* class and that their attributes evolve continuously in time, depending on the energy exchanged with other physical capsules.

When we consider a physical system together with its logic controller, the complete model described in UML will contain a set of elements stereotyped as physical capsules, interacting by means of a physical protocol, and a set of elements stereotyped as software capsules, interacting with the physical system by means of signal and modulating ports. Assuming that the behavior of software capsules is specified by UML State Diagrams, the dynamics of the aggregate system has a hybrid nature. In particular, if we focus on systems in which the interaction between hardware and software is based on boolean signals (i.e. logic sensors and controlled switches), the kind of hybrid systems which is more adequate to formalize a UML model including the proposed stereotypes is the one called *Threshold-Event-Driven Hybrid Systems* (TEDHS) [9]. A TEDHS consists of three subsystems: a *Switched Continuous System* (SCS), a *Threshold Event Generator* (TEG) and a *Finite State Machine* (FSM). The SCS interact with the FSM by means of a piecewise constant signal $u(t)$ taking discrete values in a finite set U, which are assigned according to the state of the FSM. The state transitions in the FSM are forced by events generated by the TEG according to the continuous state $x(t)$ of the SCS, while $x(t)$ evolves according to the differential equation $\dot{x}(t) = F_{u(t)}(x(t))$, selected by $u(t)$ at each instant. Within the UML-based modeling framework proposed in the paper, signal ports connecting physical capsules and software capsules play the role of the TEG, modulating ports connecting software capsules and physical capsules play the role of the switching signal $u(t)$, while the control logic and the complete physical system are, respectively, related to the FSM and to the SCS. Hybrid systems of the TEDHS class are the basis of the verification tool CheckMate [5], in which the system is modeled by means of a Matlab/Simulink® diagram that contains SCS blocks, TEG blocks and FSM blocks, the latter specified with Stateflow® charts. This modeling front-end allows an easy translation of UML multi-domain models, following the conceptual mapping described before. In order to perform state-space exploration and formal verification of properties specified in the temporal logic ACTL [9], CheckMate transforms the TEDHS first into an equivalent *Polyhedral-Invariant Hybrid Automaton* (PIHA), which is a hybrid automaton whose discrete locations are characterized by a given continuous dynamics and guards for transitions between locations are given by hyperplanes in the continuous state-space (i.e. transitions fire when the continous state cross the hyperplane guard). Then, the PIHA is mapped into a *Quotient Transition System* (QTS), a finite state system that is a conservative approximation of the hybrid system, in the sense that for every trajectory of the original hybrid system there is a trajectory in the QTS corresponding to the set of partitions of the continuous state space that the hybrid system passes through. The model checking problem on ACTL formulas is finally solved in CheckMate by performing the reachability analysis on the QTS. In next section, we will show how to model a simple case of study with the proposed UML extension and then how to use CheckMate for its verification.

V. Example

Consider the two tanks system illustrated in Fig. 2, taken from [10]. The plant is made up of two cylindric tanks

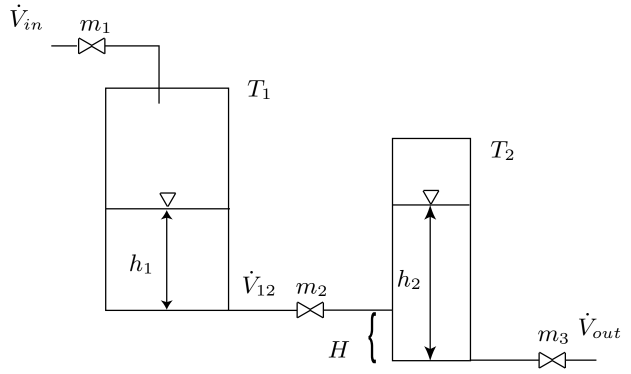

Fig. 2. The two tanks system

T_1 and T_2 which are situated at different levels and that exchange a fluid. The incoming flow, the amount of fluid flowing along the pipe interconnecting the two tanks and the outgoing flow are governed by the controlled valves V_1, V_2 and V_3 respectively. In the following, the control signal associated to the i^{th} valve will be called m_i; a valve can either be closed ($m_i = 0$) or open ($m_i = 1$). We assume that the input volume flow $\dot{V}_{in}$ is equal to a positive constant IN, if V_1 is opened, and equal to 0, otherwise. The volume flow $\dot{V}_{12}$ in the pipe interconnecting T_1 and T_2 depends on the position of V_2 and on the levels of fluid in the tanks. Thus, using Torricelli's law, we have that:

$$\dot{V}_{12} = \begin{cases} K_1\sqrt{h_1 - h_2 + H} & \text{if } m_2 = 1 \text{ and } h_2 > H \\ K_1\sqrt{h_1} & \text{if } m_2 = 1 \text{ and } h_2 \leq H \\ 0 & \text{if } m_2 = 0 \end{cases} \tag{15}$$

where h_1 and h_2 are the levels of fluid in T_1 and T_2 respectively and H is the height difference between the tanks. Finally the output volume flow $\dot{V}_{out}$ is equal to $K_2\sqrt{h_2}$ if V_3 is opened and to 0 if it is closed. K_1 and K_2 are positive constants that depend on the kind of fluid that is used. Recalling that in the hydraulic domain the power variables flow and effort are, respectively, volume flow $\dot{V}$ and pressure P and that the tanks can be modeled as elements that store hydraulic potential energy, we can define the UML model of the two tanks system with its logic controller as composed of three capsules, interacting with each other: the Plant, the Controller and PhysicalTime.
The Controller capsule represents the control logic and it interacts with the PhysicalTime capsule, in order to measure the progression of time, and with the Plant capsule in order to receive the information relative to the level of the fluid in T_2, through a port which plays the slave role in the master/slave protocol **PC**. The control action is transmitted to the plant through a port which plays a modulating role into the physical protocol **Phys** that governs the exchange of energy between the physical sub-capsules that compose the plant. The behavior of the Controller capsule is represented by the state machine reported in Fig. 3, which has been designed following the specification described in [10]: after a start-up phase, lasting a given period of time, in which the tanks are pre-filled in two steps, the level in T_2 is kept within a lower and an upper bound by opening/closing its output valve.

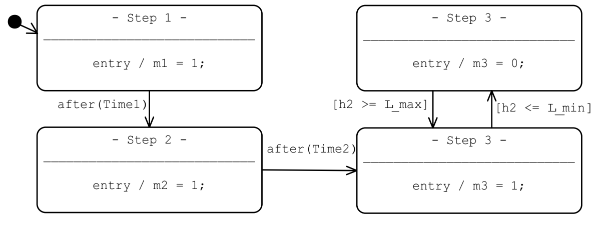

Fig. 3. The behavior of the Controller capsule

A detailed UML-RT class diagram of the plant is reported in Fig. 4. The Plant capsule has five sub-capsules that

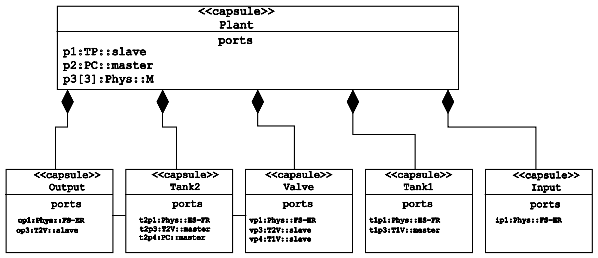

Fig. 4. UML-RT diagram of the physical system

represent the two tanks (Tank1 and Tank2), which are energy storing elements, two elements that inject/extract the fluid in the system (Input and Output), which are sources of flow, and an element that governs the flow along the pipe interconnecting T_1 and T_2 (Valve), which is a source of flow. Each sub-capsule has a port that implements a power port through which it exchanges energy with the other sub-capsules, following the protocol `Phys` whose behavior is represented by the Dirac structure described by:

$$\begin{pmatrix} m_2 & -m_1 & m_1 m_2 & 0 & 0 \\ m_3 & 0 & m_1 m_3 & -m_1 m_3 & -m_1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \texttt{ip1.f} \\ \texttt{vp1.f} \\ \texttt{t1p1.f} \\ \texttt{t2p1.f} \\ \texttt{op1.f} \end{pmatrix} + \\ + \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ m_1 & 0 & -1 & 0 & 0 \\ 0 & m_2 & 1 & 0 & m_3 \\ 0 & m_2 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} \texttt{ip1.e} \\ \texttt{vp1.e} \\ \texttt{t1p1.e} \\ \texttt{t2p1.e} \\ \texttt{op1.e} \end{pmatrix} = 0 \qquad (16)$$

Notice that also the Plant capsule participates to `Phys` through the port `p3` which plays a modulating role and which represents the control action on the plant. In fact we can see that m_1, m_2 and m_3 appear explicitly in the protocol behavior and that different values of the control action lead to a different way in which energy is exchanged among the sub-capsules.

The translation of the UML model into a CheckMate block diagram is almost straighforward. The Controller capsule is mapped into an FSM block, modeling the same event-driven behavior. The events triggering the state transitions, which are related to Controller ports `p1` and `p2` since they depend from thresholds on time and level of T_2, will be generated by TEG blocks evaluating polyhedral constraints on the continuous state space. Finally, two SCS are necessary to simulate the dynamics of the physical system and the progression of time. The differential equation governing the continuous dynamics can be easily generated from the UML model of the Plant capsule, considering that the state variables are given by the volume of fluid in the two tanks, which are energy storing elements and that volume flows can be calculated from Eq.(16). Since Eq.(16) depends from the three valve control signals m_1, m_2 and m_3, it also specifies the dependence of the continuous dynamics from the switching signal generated by the Controller FSM, which can be defined as an integer variable u that can take any of the 2^3 values representing admissible configurations of the boolean values m_1, m_2 and m_3.

Once that the UML model has been translated in CheckMate, it is possible to simulate its behavior and to verify its properties. With regards to the formal verification task, an example of the properties that can be verified on the modeled system with the help of CheckMate is the fact that the tank T2 never overflows, which is expressed in ACTL as the formula `AG ~overfull`, being `overfull` an atomic proposition which is true if a given polyhedral constraint on the continuous state space is satisfied (i.e. the level of $T2$ is higher than 1 meter). Since the controller performs the start-up phase only considering timing events, it may happen that the system violates the specification because the time spent in the pre-filling phases *Step1* and *Step2* of Fig. 3 is too long, so that when the controller enters the actual regulating phase the overflow has already occurred. With a proper setting of the timing specifications *Time1* and *Time2*, it is possible to prove that the sytem verifies the desired property.

VI. Conclusion and future work

The paper has described an extension of the modeling language UML which can be adopted to describe complex computer-controlled physical systems. The fundamental modeling approach, which allows to unify the concepts used in software design with those used in systems modeling, is object-orientation. With the support of the standard notation of UML, the proposed language aims to increase multi-disciplinary approaches to the design of logic controllers for industrial processes and mechatronic systems. In the future, the authors aim to integrate the proposed concepts into a CASE tool that can support industrial control engineers in their design practice.

References

[1] I.E.C., "IEC 61499-1. Function Blocks for Industrial Process Measurement and Control - Part 1: Architecture," Public Available Specification (PAS), 2000.

[2] H. M. Paynter, *Analysis and Design of Engineering Systems*. Cambridge, MA: M.I.T. Press, 1961.

[3] O.M.G., "UML, v.1.4, OMG specification," Document N. formal/2001-09-67, 2001, www.omg.org/uml.

[4] G. Golo, A. van der Schaft, P. Breedveld, and B. Maschke, "Hamiltonian formulation of bond graphs," in *Nonlinear and Hybrid Systems in Automotive Control*, R. Johansson and A. Rantzer, Eds. Springer–Verlag, 2003, pp. 351–372.

[5] B. Silva, K. Richeson, B. Krogh, and A. Chutinan, "Modeling and verifying hybrid dynamic systems using CheckMate," in *Proc. 4th Int. Conf. on Automation of Mixed Processes ADPM*, 2000, pp. 237–242.

[6] B. Selic and J. Rumbaugh, "Using UML for complex real-time systems," IBM Rational Software Ltd, white paper, 1998, www-106.ibm.com/developerworks/rational/library/139.html.

[7] B. Selic, "Protocols and ports: reusable inter-object behavior patterns," in *Proc. of the 2nd IEEE Symposium on Object-Oriented Real-Time Distributed Computing*, Saint-Malo, France, 1999.

[8] A. van der Schaft, *L_2-Gain and Passivity Techniques in Nonlinear Control*, ser. Communication and Control Engineering. Springer Verlag, 2000.

[9] A. Chutinan, "Hybrid system verification using discrete model approximation," Ph.D. dissertation, Dept. of Electrical and Computer Engineering, Carnegie Mellon University, May 1999.

[10] S. Kowalewski, O. Stursburg, M. Fritz, H. Graf, I. Hoffmann, J. Preussig, M. Remelhe, S. Simon, and H. Treseler, "A case study in tool-aided analysis of discretely controlled continuous systems: The two tanks problem," in *Hybrid Systems V*, ser. LNCS. Springer–Verlag, 1999, vol. 1567, pp. 163–185.