On the use of UML for modeling mechatronic systems

Cristian Secchi, Member, IEEE, Marcello Bonfè, Member, IEEE, and Cesare Fantuzzi, Member, IEEE

Abstract—This paper describes a modeling language, firstly proposed in [1]–[3], that aims to provide a unified framework for representing control systems, namely physical plants coupled with computer-based control devices. 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, particularly those of capsules, ports and protocols. Furthermore, it is illustrated how the well-known object-oriented specification language UML can be adopted, provided an adequate formalization of its semantics, to describe structural and behavioral aspects of control systems, related to both logical and physical parts.

Index Terms—Bond Graphs, System Modeling, Object-Oriented Design, Software Engineering, Unified Modeling Language.

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. In fact, object-orientation allows to define modular system architectures based on reusable and extensible components, whose features are appealing in any application domain, from modeling and simulation of dynamic systems to control design and software programming.

Since modularity and reusability are key properties for handling complexity, object-oriented approaches have been adopted for the implementation of modeling frameworks and simulation tools for large, complex and heterogeneous physical systems. Among the several object-oriented modeling languages for physical systems that have been developed within the last two decades it is worth to mention ASCEND [4], Omola [5], and ULM [6], to name only those that have some distinguishing features. Recently, the efforts to combine together the features of the various object-oriented modeling frameworks have led to a unified language for physical systems modeling called Modelica [7].

Modeling languages with graphical notations are commonly used also during analysis and design activities in complex software projects, in which software engineers are required to build first a conceptual model of the program and then to translate it in executable code. Object-oriented philosophy is by far the most successful strategy for modeling software applications and several development methodologies, supported by diverse specification languages, have been proposed within this framework; the interested reader is referred to [8] for a brief history of the object-oriented approach.

Currently, the leading visual notation in the software specification domain is the Unified Modeling Language (UML) [9], which has been defined as an integration of the most successful languages for software analysis and design. The aim of the consortium behind the definition of UML, the Object Management Group (OMG), is to provide “a standard language for specifying, visualizing, constructing and documenting all the artifacts of a software system” [10]. Thus, UML should be used to create platform independent models that can be mapped by a model compiler into any platform-specific application; for further details see [9]. Moreover, UML supports with a number of graphical views (i.e. Use Case diagrams, Sequence Diagrams, etc.) the specification of functional requirements, which is essential for any software project. Another reason that makes UML appealing is its extensibility, which allows to model domain-specific or methodology oriented concepts by means of stereotyped elements, derived from the basic UML meta-model elements. A consistent set of UML stereotypes is called a profile.

Also the emerging technologies for industrial control systems are more and more emphasizing concepts like modularity and reusability of components (both hardware and software), in order to increase efficiency of manufacturing systems design and simulation, reducing the time spent during the installation of machines and the operational qualification of production lines. Modern tools to design and program control devices for industrial settings (e.g. Programmable Logic Controllers, PLCs) support engineers with many features oriented to the encapsulation and reuse of software modules. For example, the well-known standard for PLC (Programmable Logic Controllers) programming IEC 61131-3 [11] and the newer standard IEC 61499-1 [12] for distributed control systems, define frameworks for the implementation of modular software architectures, based on program organization units called Function Blocks. Moreover, several authors have proposed UML as a specification language for software architectures based on Function Blocks [13], [14].

Even though the mentioned physical and software modeling languages share the basic principles and are well-known in their application area (i.e. systems simulation and software programming), it is hard, with the current methodologies and tools, to integrate them, in order to describe all the aspects related to the design of a complex computer-controlled system within a single modeling framework. On the other hand, since in this kind of systems the control software, the physical plant and their interconnections form a tightly coupled whole, it would be more useful to explicitly model them as an aggregate.
Moreover, a unified language embedding structural and behavioral aspects of control software and physical components would provide a lingua franca for the communication between mechanical, process and software engineers. In complex control systems the physical characteristics of the plant have a deep impact on the software itself [15] and, therefore, if it was possible to describe physical systems using a language widely accepted by the software community, this would simplify the programmers’ task.

Consider for example the motion control system reported in Fig. 1, which is a quite typical mechatronic system. The system consists of several physical components (a DC motor, a power converter, some gears and a crank-rod mechanism) and of a logical part (the control law, the trajectory generator and the control logic) that can be implemented as an object-oriented software application. This simple example highlights the heterogeneous nature of mechatronic systems, which are made up by the aggregation of physical and logical elements. Modeling languages like Modelica or the Bond-graphs [16], [17] allow to describe with an object-oriented approach the structure and dynamics of the physical part, namely the DC motor and the mechanical load, but they are not suitable for representing neither the control law nor the control logic. On the other hand, a network of Function Blocks [11], [12] would be a natural choice for the design of the control software, but this design framework, which is based on an event-driven model of computation, cannot be easily adapted for describing physical systems, which are essentially time based. Recently, an actor-based modeling strategy and a software tool for modeling systems with heterogeneous models of computation has been proposed within the Ptolemy project [18]. However, even if this framework allows to build and simulate a model of the overall control system reported in Fig. 1, it lacks of the expressiveness of UML: for example, it doesn’t allow any requirement specification, which is instead necessary during the first phase of the control system design. Thus, Ptolemy would not be able to support control engineers from the very beginning of the development process.

Of course, a critical reader would remark that UML, despite the fact that it offers tools for requirements specification, is still a language whose application domain is limited to software design. Nevertheless, the language itself can be thought as an abstract system modeling language, whose semantics can be adapted and extended to make it suitable for different domains. Actually, several researchers explored the possibility to use software engineering tools for modeling physical systems. In [19], [20] physical systems are modeled as the interconnection of blocks characterized by a continuous behavior and UML is properly extended to be able to represent this kind of systems. In [21] a physical system is modeled as a generic resource which provides some services and UML is used to describe a complex system made up both of a logical and of a physical part. Moreover, a consortium of industries and public organizations has recently proposed a profile for systems engineering, called SysML [22], which aims to support specification, design and verification of complex systems including hardware, software, information and processes. However, this profile is currently at an early stage of development and it is not clear enough, in our opinion, how to formalize with the SysML language the dynamic behavior of physical systems.

In order to provide a precise framework for integrated modeling of physical systems and control software, the main contribution of this paper is a proposal for an extension of UML based on an already existing and well-known profile, called UML-RT, introduced by Selic et al. in [23]. 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 ports following a certain communication protocol. Ports make the internal implementation of capsules independent of the communication with the external world, increasing significantly the reusability of the component. The concept of port as an interaction point between a component and its environment has been included also in the draft of the newer UML 2.0 specification [24], since it has been universally recognized as a necessary feature to emphasize modularity and reusability of design models. The concept of protocol, instead, allows to model the global behavior of the system independently of the particular implementation of the capsules, thus making the protocol specification both a functional requirement and a design specification. The key idea behind the paper’s contribution, is to unify the concepts of port and protocol of UML-RT with the concepts of power ports and interconnection structures of Bond-graphs, so that network-based modeling approaches for physical systems may be easily re-interpreted in order to adopt the graphical notation of UML.

In particular, the paper extends the results proposed in [1]–[3] with some remarks related the practical application of the proposed object-oriented approach to multi-domain systems modeling. This approach exploit the Bond-graphs based modeling framework [16], [17] and their recent mathematical formalization as port-Hamiltonian systems [25], [26], to show that any physical system can be described within the UML-RT profile.

Within this framework a control system can be modeled as a hierarchical set of physical and logical objects endowed with ports through which they exchange information with each other following proper communication protocols. In this way, it is possible to clearly distinguish the actors, both physical and logical, that compose the system and the global behavior that derives from their collaboration. Thanks to the
concepts of port and protocol, modularity and reusability are extended to all the elements that characterize any complex automated system, both from the software and the hardware point of view, so that engineers can analyze at any level of detail, following the hierarchical decomposition of the object-oriented model, both functional (i.e. sequence of operations) and non-functional (i.e. minimum torque or speed required by an electric motor) design requirements. We also provide a description of a general physical system using UML formalism and its extension mechanism in order to overcome some modeling aspects not directly addressed in the standard such as the continuous behavior of the physical capsules. In this way, UML can be used to describe both physical and logical elements becoming, consequently, a suitable unified language for modeling complex control systems.

The paper is organized as follows: in Sec. II we provide some background on UML-RT profile and on the Bond-graphs modeling language and in Sec. III we specify the conceptual scenario of multi-domain object oriented systems we refer to. In Sec. IV we show how it is possible to map the Bond-graphs description of a physical system into the UML-RT profile and in Sec. V we provide a formal description of a physical system using UML and its extension mechanism. In Sec. VI we provide an example to validate the results obtained in the paper and, finally, in Sec. VII some conclusions are drawn and future work is addressed.

II. BACKGROUND

A. The UML-RT Profile

The UML Real Time Profile (UML-RT) addresses modeling concepts that have proven suitable for modeling the software architecture of complex real-time event driven systems in application domains such as telecommunication, aerospace and industrial control. UML-RT extends standard UML (using the standard extension mechanism) by adding five new stereotypes through which real-time architectures are modeled: capsule, port, connector, protocol, protocol role. In the following we give a brief description of these elements focusing on modeling aspects rather than on their software aspects; for a more complete introduction see [23].

Capsules are the central modeling constructs in UML-RT. They represent the major architectural elements of complex real-time systems and their collaboration allows to model the whole software architecture. A capsule can have one or more ports through which it can communicate with the other capsules and it may contain one or more sub-capsules which collaborate together. The behavior of a (sub-)capsule can be described by a state machine. Ports are boundary objects that are “owned” by a capsule and that provide the only way through which a capsule can interact with the rest of the world. Ports can either be relay ports or end ports (also called behavior ports in UML 2.0 [24]). A relay port is connected with a port of a sub-capsule. It simply provides an opening in the encapsulation shell that allows to sub-capsules to communicate with the external world without being directly exposed. An end-port is a port that is directly connected to the state machine of the capsule, thus the messages flowing through the port directly affect the behavior of the capsule. A connector represents the means along which capsules communicate and it is the physical medium over which a communication protocol takes place. A protocol is a specification of a closed group of participants (protocol roles) and of the rules that define the communication that can take place over the connector that joins the participants. A protocol role specifies the messages that a specific participant can send and receive during the communication and how those messages must be exchanged. Formally a protocol \( P \) is defined by a 4-tuple [27]:

\[
P = (E, R, B, Q)
\]

\( E \) is the event alphabet, namely the set of all the event types that can be passed between the participants in a protocol. \( R \) are the protocol roles, namely the roles that the participants to the protocol can play. \( B \) represents the protocol reference behavior and identifies the set of legal behaviors that constitute a protocol. \( Q \) represents the expected quality of service of the protocol.

In the UML-RT framework, a complex real-time software application can be modeled as a set of capsules endowed with ports. The capsules are joined, through their ports, by means of some physical connector over which a communication protocol is implemented. Each port plays a protocol role and the collaboration between the capsules allows to achieve a desired goal.

B. Bond-graphs

In every physical domain, there is a pair of variables, defined on a pair of dual vector spaces\(^1\) \( F_{\phi_0} \) and \( E_{\phi_0} = F_{\phi_0}^* \) whose dual product is power. These variables are generally called flow and effort and, for example, in the mechanical domain they are velocity and force and in the electrical domain they are current and voltage. A power port [25] is defined by a pair \((e, f) \in E_{\phi_0} \times F_{\phi_0}\) and represents the means through which a physical system can exchange energy with the rest of the world.

The Bond-graphs modeling strategy [16] allows to represent any lumped parameters physical system through a set of basic elements: elements that store potential or kinetic energy, elements that dissipate energy and sources of energy. Each of these basic elements is endowed with one or more power ports through which it can exchange energy. The dynamic behavior of a physical system is due to the exchange of energy that takes place among these elements. A bond-graph shows explicitly the network structure along which the various elements interact; the paths along which energy is exchanged are represented by bonds: an effort and a flow are associated to each bond and their dual product represents the power exchanged through the bond. The network structure is represented by the interconnection of the various bonds by means of junctions, whose behavior is governed by Kirchhoff-like laws, and of energy preserving transformations (transformers, gyrators).

\(^1\)Given a vector space \( F_{\phi_0} \), its dual \( E_{\phi_0} = F_{\phi_0}^* \) is the vector space of the linear operators on \( F_{\phi_0} \). Given \( f \in F_{\phi_0} \) and \( e \in E_{\phi_0} \), their dual product is given by \( \langle e, f \rangle = e(f) \)
Each (and only) element that can store energy has states associated to it; each state models the storage of energy flowing through a power port. Even if Bond-graphs are in general acausal [16], it is possible to assign a causality to each element and thus to fix an input and an output for each power port. Very often, energy storing elements have an integral causality associated and their behavior is represented by

\[
\begin{aligned}
\dot{x}_i &= u_i \\
y_i &= \frac{\partial H}{\partial x_i}
\end{aligned}
\quad i = 1, \ldots, m
\]  

(2)

where \( m \) is the number of power ports associated to the element, \( H(\cdot) \) 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 the input and the output variable such that \( u(y) \geq 0 \), namely such that power is absorbed by the element. Sources of energy can be modeled in two ways: by means of sources of force, which fix the flow to a certain value, and by means of sources of effort, which fix the effort to a certain value. Encapsulation is possible within the Bond-graphs framework. A physical system is composed by an interconnection of \textit{physical subsystems} through their power ports. Each physical subsystem is composed by further physical subsystems which exchange energy on their own. This encapsulation process ends when basic elements are encountered.

Recently, the power preserving interconnection structure has been mathematically formalized introducing the concept of Dirac structure [25]. Once a coordinate set has been fixed, several representations of a Dirac structure are available; a very effective one is the so called kernel representation. Suppose that there are \( m \) power ports exchanging energy through the interconnection structure and build the following vector:

\[
f = \begin{pmatrix} f_1 \\ \vdots \\ f_m \end{pmatrix} \in \mathcal{F}_p \quad e = \begin{pmatrix} e_1 \\ \vdots \\ e_m \end{pmatrix} \in \mathcal{E}_p
\]  

(3)

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 can be represented by the following relation:

\[
E(x)e + F(x)f = 0
\]  

(4)

where \( x \) is the state of the physical system and \( E(x) \) and \( F(x) \) are matrices such that:

\[
F(x)E^T(x) + E(x)F^T(x) = 0
\]

\[
\text{dim} [ F(x) \quad E(x) ] = \text{dim} \mathcal{F}_p
\]  

(5)

III. MECHATRONIC OBJECTS

In this section we provide a conceptual framework for modeling complex mechatronic systems, like manufacturing machines together with their control software, and we show the importance of a unified modeling language for describing these systems.

In order to model, analyze and design a computer-controlled physical system within an object-oriented perspective, it is necessary to treat software and physical \textit{objects} as tightly coupled elements, which should never be perceived separately. For example, consider a manufacturing machine with its control device. In general, these machines are composed of physical sub-systems each one performing a specific functionality of the manufacturing process (e.g. material transportation, sealing, etc.). In order to design a modular control software, an effective approach would be to split the system according to the physical-functional decomposition and then design a control module for each one of the resulting machine components. A control module 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 \textit{hardware port} of the control module, while the second is conceptually the \textit{software port}. The aggregation of a control module, its hardware and software ports and the related physical components is what we consider as a \textit{mechatronic class}, within an integrated object-oriented approach. A schematic representation of a mechatronic class instance is shown in Fig. 2, in which the control module is depicted with the notation of IEC 61131-3 Function Blocks [11], which are by definition software components with an interface of input/output ports. Notice that the hardware port and the physical components are encapsulated into the mechatronic class concept, to emphasize their close relationship with the control software. Moreover, it is worth to note that the software module itself is a composite object, which contains, for example, trajectory generators and servo control algorithms for motion tasks required by the machine functionality, sequencing and exceptions handling logic and so on. In a similar way, the physical part is an aggregations of electrical motors, mechanical loads, heaters, etc. interacting with each other by means of the exchange of energy.

Therefore, an object-oriented approach to mechatronic systems modeling should allow to describe both physical and software parts as a network of generalized objects, exchanging energy, if they have a physical nature, and events/data, if they are purely logical. Of course, Bond-graphs have this feature for modeling physical systems, while UML, especially including the concepts of ports and protocols, can be adopted to model industrial control software, as described in [13], [14]. In addition, since UML can be extended to include domain-specific concepts, it is possible to define a unique object-oriented modeling language for mechatronic systems, by means of a proper mapping between Bond-graphs elements.
and UML elements and an adequate definition of the semantics of the interaction between software and physical components. In particular, it is easy to see that the inter-domain access point is represented by hardware ports, whose role in the model is to embed operations like time-triggered sampling of sensor signals and update of 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 second one consists in applying the control action 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 [16], in addition to power ports modeling the exchange of energy.

In next section, we will show how to formalize all of the Bond-graphs elements within the UML framework.

IV. USING UML-RT FOR MODELING PHYSICAL SYSTEMS

Bond-graphs can be interpreted in an object-oriented fashion, as shown in [28], but thanks to the concept of Dirac structure we show that any physical system can be described as a set of actors that cooperate for achieving the physical behavior characterizing the system as it happens in UML-RT modeling framework. In this section we show how it is possible to describe a physical system with a UML-RT model. This will be done by mapping the concepts of Bond-graphs [16] and port-Hamiltonian modeling [25] into the UML-RT framework. To this aim we first give a system theoretic description of a UML-RT model. A system made up of $p$ capsules and $m$ ports can be described as a 5-tuple $(A, \mathcal{P}, C, \alpha, \psi_P)$, whose elements are defined in the following. The set

$$\begin{align*}
A &= A_1 \times \cdots \times A_p \quad a = (a_1, \ldots, a_p) \quad (6)
\end{align*}$$

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 other homogeneous elements such as lists, integers, data structures, and so on. The interaction between capsules is defined by the protocol $\mathcal{P} = (\mathcal{E}, \mathcal{R}, B, Q)$, as detailed in Sec. II. In particular the event set is given by:

$$\begin{align*}
\mathcal{E} &= \bigcup_{i=0}^m \mathcal{E}_i \quad (7)
\end{align*}$$

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:

$$\begin{align*}
\mathcal{E} &= \mathcal{E}_1 \times \cdots \times \mathcal{E}_m \quad \varepsilon = (\varepsilon_1, \ldots, \varepsilon_m) \quad (8)
\end{align*}$$

whose elements describe the events waiting to be processed in a given ports configuration. Obviously some $\varepsilon_i$ may be empty, meaning that no message is crossing the corresponding port. $C$ represents the interconnection structure, which is the medium through which capsules exchange information and 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 $\psi_P : A \times E \rightarrow E$ 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. This dynamics depend both on the attributes of the system and on the signals across the ports and, implicitly, on the protocol through which all the capsules exchange messages. In general when a specific event takes place both the port and the attributes configurations can change.

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-RT 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:

$$\begin{align*}
A &= \mathcal{X}_1 \times \cdots \times \mathcal{X}_n = \mathcal{X} \quad x = (x_1, \ldots, x_n) \quad (9)
\end{align*}$$

in which $\mathcal{X}$ is usually a differentiable manifold. The number of physical states is, in general, different from the number of capsules, 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 dynamic evolution of a physical system is determined by the exchange of energy that takes place among its constitutive parts. Thus, the fundamental information is energy which is exchanged through power ports. A power port can be modeled as a port whose event alphabet is given by the space of power variables, namely by $\mathcal{F}_{\rho 0} \times \mathcal{E}_{\rho 0}$, where $\mathcal{F}_{\rho 0}$ and $\mathcal{E}_{\rho 0}$ are the flow and the effort spaces relative to the port respectively.

Each capsule can be further decomposed in sub-capsules, namely in further interconnected subsystems that exchange energy, which means that it is possible to distinguish between power relay ports and power end ports.

A power relay port passes the energy, flowing through it, to a sub-capsule. It provides, as relay ports in UML-RT models for software architectures, an “opening” in the encapsulation that can be used by sub-capsules to exchange energy with the external world.

A power end port, instead, plays a specific role in the interconnection structure of the various sub-capsules. Physically speaking, it provides a means to inject (extract) energy into (from) the interconnection structure and, therefore, it affects the dynamic behavior of the capsule. In UML-RT end ports send events directly to the capsule state machine and, therefore, they can change the overall behavior of the capsule. Within the physical domain, the behavior of a capsule is determined by the interconnection structure along which sub-capsules exchange energy AND by the amount of energy that is exchanged. Energy end ports allow direct injection/extraction of energy on the interconnection structure and thus they allow to modify the overall behavior of the capsule. The key point is that while the overall behavior of a capsule in event based modeling is determined by the interconnection of the sub-capsules and by a state machine, the overall behavior of a capsule, in physical modeling, is
determined by the interconnection of the sub-capsules and by the amount of energy that is circulating, continuously, along the interconnection structure. Moreover, each physical capsule can have ports that are not directly related to the exchange of energy. For example, these ports transmit signals related to the physical state of a component, or related to the modulation of a source or of a transformer. These ports can only be considered relay ports since they cannot inject (extract) energy into (from) the interconnection structure. Nevertheless, these ports can play a role in the exchange of information among capsules.

When modeling physical systems, it is possible to distinguish between the means through which each capsule is interconnected to the others and the way in which the various subsystems are joined, namely the topology of the interconnection. More specifically, the means through which capsules communicate is represented by the interconnection structure $C$, which is made up, for example, by electrical wires, pipes, mechanical joints. The topology of the interconnection, on the other hand, represents the energetic paths, namely the energetic pathways $P$.

In the case of physical systems with an infinite event set $E$, the event set $E$ has in finite cardinality and, therefore, in order to remark this feature, we call it event space. As efforts and flows are exchanged through the power ports of the interconnection, the event space contains both $F_p$ and $E_p$. 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. There cannot be other roles apart from those of effort-supplier/flow-receiver and flow-supplier/effort-receiver, since a port cannot supply (or receive) both effort and flow as a consequence of the first principle of thermodynamics [16]. In general, the way energy is exchanged depends on the states characterizing the interconnected capsules. This dependence does not represent energy injection/dissipation, but rather a modulation in the transfer of energy along the interconnection structure. Thus ports that carry signals that are used to modulate the interconnection structure play a further role in the protocol, namely 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 $X$ of the physical system is also part of the event space, which is summarized, given by $E = F_p \times E_p \times X$. Since the dynamics of a physical system is continuous, the protocol behavior $B$ is continuous. While in software applications the behavior of the communication protocol can be arbitrarily imposed, all protocols used for modeling the interconnection of physical subsystems share the same characteristic: they are energy preserving, meaning that along the interconnection energy is neither stored nor dissipated nor produced but simply transferred. The behavior of physical protocols can be represented through the mathematical object of Dirac structure and through, for example, a pair of state dependent matrices $E(x)$ and $F(x)$ as reported in Eq.(4). Since $E(x)$ and $F(x)$ satisfy Eq.(5), it is always possible 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 [25]. Physical protocols, or, equivalently, Dirac structures, describe the way in which the components of a physical system exchange energy. In Sec. VI they will be used to describe the interconnection between the electrical and the mechanical part and between the electrical sub-components of the DC motor of the example reported in Sec. I. Dirac structures, and consequently physical protocols, can be used to model any lumped parameter system; some examples of the use of Dirac structures for modeling complex multi-domain physical systems can be found in [25], [26]. When modeling physical protocols, we consider meaningless any quality of service assessment, therefore we do not formalize $Q$.

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:

$$\alpha : F_p \times E_p \times X \rightarrow X \quad (f(t), e(t), x(0)) \rightarrow x(t) \quad (10)$$

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

$$x_i(t) = \alpha(f, e, x(0)) = x_i(0) + \int_0^t u_i(\tau) d\tau \quad (11)$$

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

$$\psi_p : F_p \times E_p \times X \rightarrow F_p \times E_p \times X \quad (12)$$

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:

$$y_i(t) = \frac{\partial H}{\partial x_i}(z_i(t)) \quad i = 1, \ldots, n \quad (13)$$

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:

$$y_i(t) = g_i(u_i(t)) \quad i = n + 1, \ldots, m \quad (14)$$

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:

$$m_i(t) = z_i(t) \quad i = 1, \ldots, n \quad (15)$$

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 $t$. Thus we have proven the following:
Proposition. Any lumped parameters physical system can be represented as a UML-RT model

Summarizing, we have shown that also in the physical domain, as in UML-RT, it is possible to put in evidence a set of main actors collaborating together through a communication protocol in order to achieve a given task, i.e. a certain behavior. The key point for highlighting this structure is to recognize that the information that encodes the dynamic evolution of a physical system is energy and that the communication protocol can be described through a Dirac structure.

Thus, within the UML-RT modeling framework, a complex, possibly distributed, automation plant can be modeled as a set of physical and logical capsules exchanging information through communication protocols. What is important for the global behavior of the system is not the capsule itself but by the way in which it processes the information (either logical or physical) exchanged with the rest of the system. This make both logical and physical capsules, highly reusable and the automation plant easy to maintain both from an hardware and a software point of view.

V. UML STEREOTYPES FOR PHYSICAL SYSTEMS

Once it has been proven that physical systems can be modeled within the UML-RT framework, it is necessary to provide an unified formalism for describing both physical systems and software architectures. The most natural candidate is UML which is already widely used for modeling software. The aim of this section is to formalize, using UML, a physical system in order to provide an unified formalism to model both software and hardware (i.e. physical systems) of a control system.

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 with 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 stereotyped collaboration, whose stereotype is <<PhysicalProtocol>>, of physical elements. The formal semantics of the physical protocol is the one defined in previous section. 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, even if they cannot participate to the physical protocol because they are not able to affect its behavior, allow to model the sensors used by the control software to implement feedback strategies. An example of the usage of the stereotypes defined in this section is shown in Fig. 3.

Each physical capsule can have several physical sub-capsules. These sub-capsules are classes on their own and are in a composition relationship with the container class. Class diagrams are very useful to model the structure of a system, but in order to model explicitly the topology of the interconnection structure, stereotyped collaboration diagrams with the notation introduced in [23] are very useful. Multiple associations, quite common when modeling physical systems, are modeled through the standard UML diamond notation [9].

While in their standard use in UML-RT capsules represent event-driven entities, when modeling physical systems, capsules represent continuously time driven entities. There are two issues to be addressed in order to define the behavior of physical capsules in UML: the concept of time and the concept of continuous behavior.

Since physical subsystems change continuously in time we need to define a concept of time that is not discrete (e.g. associated to a clock), but continuous. As reported in [15], the general UML standard does not impose any restrictions on the modeling of time and it neither assumes that time is continuous or discrete nor that there is a single source of time in a system. This semantic flexibility allows several models of time that can be used to model both discretely an continuously time-driven systems. “Physical” time can be thought as a relationship that imposes a partial order on events and it is viewed as a continuous and unbounded progression of physical time instants. Each capsule representing a physical subsystem, thus, is in an association relationship with the PhysicalTime class, since the evolution of its attributes depends on time. Furthermore, the flow of information through the ports is continuous and, therefore, also the protocol roles are in association relationship with the physical time class.

The attributes of each physical capsule evolve continuously in time and depend continuously on the energy exchanged with the other physical capsules along the physical protocol. It is necessary, therefore, to model the physical capsule through a continuous behavior instead of through a state machine as it often happens when dealing with software applications. Since the UML is expressly a discrete modeling language, it does not provide any direct means to represent continuous behaviors. Nevertheless it is possible to use the extension mechanism of UML and to model the continuous behavior attaching the <<invariant>> constraint to a class. In fact the equation expressing the relation between the input and the variation of the state and the state and the value of the output is nothing else than an invariant between state and input and state and output respectively.

VI. EXAMPLE

An important part of the software system controlling a manufacturing machine is related to motion control, including the generation of reference trajectories for the position or velocity
of a group of electric motors, the tracking of these trajectories and the management of electric power converters like, for example, PWM-controlled transistor bridges. In this section we represent the motion control system reported in Sec. I using the UML-RT based modeling framework developed in the paper. As an example of a system that realize the typical functionalities of an industrial motion controller, we consider a single permanent magnets DC motor toghether with its control software. The physical plant interacts with the control software through a position sensor coupled with the motor's shaft and through a controlled voltage source which affects directly the motor's armature voltage. The control algorithm is implemented as a real-time software program running over a microcomputer which, in its turn, interacts with an external entity, that may be a human operator or a higher level control unit. The UML-RT model of the overall system is reported in
which represents the fact that control system modeled here takes place through the master/slave protocol communication between the external entity and the controller sor unit, with the DC motor and with its sub-capsules. The interacts, through proper protocols, with the external supervi-

Fig. 4 where, in order to represent more concisely a capsule together with its ports, the ports of each capsule are listed in a specific port compartment. The name of the port is reported first, followed by the name of the protocol it participates to and finally by the protocol role it plays; furthermore, the attributes of each capsule are not listed in the diagram. The system is composed of two main capsules: the Controller, that represents the control software and that is characterized by a discrete behavior, and the DCMotor, that represents the physical plant and that is characterized by a continuous behavior.

The controller capsule contains three subcapsules:

- the PowerMonitor capsule, that is responsible of switching on and off the power converter required to control the DC motor’s armature voltage and detecting its abrupt failures;
- the TrajectoryGenerator, that computes at each sampling time the reference position on the basis of a predefined set of motion profiles stored in the memory of the microcomputer;
- the Control capsule, that computes the control action and sets the motor’s armature voltage on the basis of the current values of the reference position, received from the TrajectoryGenerator capsule, and of the measured position of the motor.

The Controller capsule is endowed with 4 ports by which it interacts, through proper protocols, with the external supervi-

or START signals, from the external world. The interaction between the controller and the DC motor takes place through two master/slave protocols: MC and CM. The port \( p_1 \) of the controller plays the slave role in the protocol HC which represents the fact that control system modeled here receives operating commands like, for example, ON, OFF

port by which it communicates with the external entity and the controller takes place through the master/slave protocol HC. The port \( p_1 \) of the controller plays the slave role in the protocol HC which represents the fact that control system modeled here receives operating commands like, for example, ON, OFF or START signals, from the external world. The interaction between the controller and the DC motor takes place through two master/slave protocols: MC and CM. The port \( p_2 \) plays the slave role in the MC protocol and this models the fact that the motor sets the value of the position that is used in the computation of the feedback control law, while the port \( p_3 \) plays the master role in the CM protocol and this models the fact that the armature voltage is set on the motor’s power source by the control software. Ports \( p_2 \) and \( p_3 \) are relay ports since the sensor value is acquired by the Control and the TrajectoryGenerator subcapsules and the voltage is set directly by the Control subcapsule.

The interaction protocol between the components of the control software is described by Fig. 5, which shows the behavioral state machines of the three subcapsules, playing the protocol roles denoted with CT (ConTrol) TG (Trajectory Generator) and PM (Power Monitor), and the statechart of the protocol itself, that describes the behavior of the software aggregate in response to both external and internal (i.e. from subcapsules) stimuli.

When the control system is switched off, the protocol’s statechart is in the “Idle” state. When it receives the On event (denoted as EX.On) from the actor playing the “external”, the system enters into the “Energize” state, in which the PowerMonitor capsule switches on the power converter and, if there are no faults, enables the system to wait for subsequent “running” commands. The first command that forces the system to actually move the motor is the Homing request: when this signal event is detected, the TrajectoryGenerator generates a motion profile that approaches at a very low speed the motor’s “zero” position. When the motor is “homed” (condition denoted as \( MP = 0 \) in Fig. 5), the system is able to accept the Start command, to which the system responds entering a state in which the trajectory generator computes a repeating motion profile, as required by the “production” cycle of the manufacturing machine connected to the motor.

When the control system is in the Run state, the repeating motion can be stopped in the following ways:

1) if a Stop command is received by the Controller capsule, the trajectory generator computes for the last time a complete cycle of the motion profile, so that the motor stops exactly in its “zero” position, which allows the control system to restart the cyclic motion as soon as another Start command is received;
2) if a SafeStop request is received, the trajectory generator computes a motion profile that reaches as soon as possible the condition in which the motor’s speed is equal to zero (condition denoted as \( MS = 0 \) in Fig. 5);
3) if the Control capsule detects that the following error is higher than a safety limit, the system enters an Error state, in which the armature voltage of the motor is set to zero and the power converter is switched in a configuration that brakes the motor as much as possible;
4) if the PowerMonitor detects a fault in the power converter circuitry, the system reacts in the same way described in the previous case.

The DC motor capsule can be decomposed into two sub-capsules: the armature sub-capsule and the mechanical sub-

capsule, that represent the armature circuit and the mechanical part of the motor respectively. The armature sub-capsule is endowed with a port (\( p_1 \)) from which it receives the value of the input voltage from the controller capsule. Furthermore the two sub-capsules communicate, namely exchange energy, through the protocol Phys1. Both the armature and the mechanical sub-capsules have a power port that implements the role of effort-supplier/flow-receiver: this is due to the fact that their interconnection is by means of an ideal power gyrator [26], which relates proportionally the effort of one physical domain (i.e. electrical) to the flow of another domain (i.e. mechanical) and viceversa. In the proposed model, this is represented by the association between the two capsules and by the formalization of the interaction protocol by means of the Dirac structure described by the following:

\[
\begin{bmatrix}
1 & 0 \\
0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
mp.e \\
ap1.e \\
\end{bmatrix}
+
\begin{bmatrix}
0 & -k_t \\
k_t & 0 \\
\end{bmatrix}
\begin{bmatrix}
mp.f \\
ap1.f \\
\end{bmatrix}
= 0
\]

where \( k_t \) is the electro-mechanical constant of the DC motor.

The pair of matrices reported in Eq.(16) represent the behavior of the physical protocol which describes the way in which the mechanical and the armature sub-capsules exchange energy. Furthermore, each sub-capsule can be further decomposed in the interconnection of basic sub-capsules. For example, the
The armature capsule can be decomposed into two sub-capsules: *Kinstoring*, an element storing generalized kinetic energy, in this case an inductor, and *Dissipating*, an element dissipating energy, in this case a resistor. These elements communicate through the protocol *Phys2*, in which the Armature partecipates by providing a modulated voltage source through the port \( ap_2 \). The behavior of the sub-capsules Kinstoring and Dissipating is reported exploiting the **invariant** stereotype available in UML, where \( L \) and \( R \) are the inductance and the resistance of armature circuit respectively and their interaction is governed by the Dirac structure represented as follows:

\[
\begin{pmatrix}
1 & 1 \\
0 & 0
\end{pmatrix}
\begin{pmatrix}
ap_2 \cdot e \\
kp_1 \cdot e \\
dp_1 \cdot e
\end{pmatrix}
+
\begin{pmatrix}
0 & 0 \\
1 & -1 \\
0 & 1 & -1
\end{pmatrix}
\begin{pmatrix}
ap_2 \cdot f \\
kp_1 \cdot f \\
dp_1 \cdot f
\end{pmatrix}
= 0
\]  

(17)

The matrices reported in Eq.(17) represent another Dirac structure that describes the behavior of the physical protocol that models the way in which the inductor, the resistor and the voltage source exchange information (i.e. energy) through their power ports. Finally, in order to keep the Class Diagram simple, the association between the DC motor and the PhysicalTime class has been omitted.

Notice that the overall system can be represented as a set of capsules that exchange information following proper communication protocols. The capsules can be either discrete (as the capsules composing the Controller), and in this case their behavior can be represented through a statechart, or continuous (as the subcapsules of the DCMotor), and in this case their continuous behavior can be represented through the **invariant** stereotype. The protocols governing the exchange of information between discrete capsules can be of several kinds (master/slave, client/server, etc.) while those governing the exchange of information between continuous capsules have to be power preserving and therefore representable through a Dirac structure (e.g. *Phys1* and *Phys2*).
In this paper it has been shown how it is possible to model complex mechatronic systems with a unified approach based on Object-Orientation. The proposed approach allows to consider all the components of a mechatronic system, either belonging to the control software domain or to any physical domain, as elements of a set of, possibly compound, objects with given interface ports. The interactive behavior of the system is modeled as an exchange of information between its components. A generalized object of a mechatronic system can be either a software element, that exchange information through a connector that implements an event-based communication protocol, or a physical element, that exchange information, namely energy, through a physical connector which implements a certain continuous communication protocol, that can be mathematically formalized through a Dirac structure. The graphical language of UML, with the extensions included in the UML-RT profile, has been used to provide a unifying framework for modeling both physical systems and software architecture. Finally, it is important to remark that the main concepts of UML-RT have been included in the official UML 2.0 [24] and are fully supported by software development tools for real-time systems like, for example, Rational Rose® RealTime [29]. Since this tool can generate fully executable C++ or Java code based on UML-RT models, it is possible to develop a library of physical elements, either basic (i.e. energy storing blocks, sources, etc.) or complex (e.g. DC motors), whose internal implementation allows to simulate the dynamic behavior of the physical part of a control system. In this way, the project of a complex mechatronic system can be fully supported by requirements specification, design models and closed-loop simulations using a unified language and a single software tool.

When working on complex, possibly distributed, plants there is often the need to replace a physical component (e.g. with another component that performs a given physical process, like welding, in a different way). Future work aims to exploit the proposed unified language to formalize the concept of behavioral inheritance for multi-domain systems, which include software and physical components, extending the results described in [30]. In fact, the inheritance mechanisms supported by current object-oriented languages, both for software programming and physical systems modeling, are focused on the structure, rather than on the behavior of system components. However, preserving the behavior of a base class in all the classes derived by extension from it is very important to guarantee that the substitution of some components in an automated system do not affect the interactive behavior of the mechatronic aggregate, which is a necessity for manufacturing systems designers.

References


