UNIFIED MODELING OF CONTROL SOFTWARE AND PHYSICAL PLANTS

C. Secchi * **C. Fantuzzi** * **M. Bonfé** **

* *DISMI, University of Modena and Reggio Emilia, Viale Allegri 13, 42100 Reggio Emilia, Italy e-mail: {secchi.cristian, fantuzzi.cesare}@unimore.it*
** *University of Ferrara, Via Saragat 1, 44100 Ferrara, Italy e-mail: mbonfe@ing.unife.it*

Abstract: The aim of this paper is to provide a unified language for modeling both control software and physical plants in real time control systems. This is done by embedding the bond graph modeling language for physical systems into the UML-RT framework, widely used to model distributed real-time software.

Keywords: Bond graphs, software engineering, formal languages

1. INTRODUCTION

Complex real time systems are made up of two fundamental parts whose collaboration allows to achieve a desired behavior for the overall system: the real-time control software and the physical plant. In order to master the complexity of software applications, it has been proven necessary first to build a model of the program and then to exploit it to write the software. Object oriented philosophy is by far the most successful strategy for modeling software applications and several development methodologies have been proposed within this framework. Recently an effort has been made to standardize the notation and this has led to the Unified Modeling Language (UML), (Rumbaugh *et al.*, 1999) and some powerful development methodologies have been proposed using UML as, for example, the ROPES strategy (Douglass, 2000). In order to specialize the UML for the development of real-time applications a specific UML Profile (namely a "standardized" language extension, typically created for a particular domain) has been defined: the real time profile (UML-RT), (Selic and Rumbaugh, 1998). Within this profile it is possible to easily model real-time complex, event-driven and possibly distributed software architectures as a set of capsules that exchange infomation following a certain protocol. In complex control systems the physical characteristics of the plant have a fundamental impact on the software itself (Selic and Motus, 2003) and, therefore, it is necessary to model the physical plant in order to successfully implement a control system. The language mainly used to describe physical systems, namely that of differential equations, is very different from that mainly used to model the software and, therefore, it is not possible to describe the overall control system within the same framework. On the other hand, since the control software and the physical plant form a whole, it would be beneficial to model them together in a complete model. A unified modeling language would provide a sort of *lingua franca* for the communication between control and software engineers. A very insightful modeling strategy for physical systems are bond graphs (Paynter, 1960), which describe a system as a set of elements exchanging energy. Recently

this graphical language has been formalized by introducing the concept of Dirac structure (van der Schaft, 2000). The aim of this paper is to exploit bond graphs and their mathematical formalization in order to provide a unified language for modeling both physical plant and control software. We illustrate how it is possible to describe a physical system as a UML-RT model, namely as a set of elements that exchange information along a connector over which a communication protocol is implemented. In this way, UML-RT can be used as a modeling language for both software and hardware of a complex, possibly distributed, control system. The paper is organized as follows: in Sec. 2 we provide some background on the UML-RT profile and bond graphs. In Sec. 3 we show how it is possible to map the bond graphs formalism into the UML-RT profile. In Sec. 4 we provide the UML notation for the description of a physical system and in Sec. 5 we provide an example to validate the results obtained in the paper. Finally, in Sec. 6 we draw some conclusions and address some future work.

2. BACKGROUND

The UML-RT profile. UML-RT extends standard UML (using the standard extension mechanism provided in UML) 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 software aspects, for a more complete introduction see (Selic and Rumbaugh, 1998). *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 it can interact with the rest of the world. A *connector* is the physical medium over which communication between capsules takes place. A *protocol* is a specification of a closed group of participants and of the rules that define the communication that can take place over the connector that joins the participants. A *protocol role* specify a specific part that a participant has to play during the communication. Formally a protocol $\mathcal{P}$ is defined by a 4-tuple $\mathcal{P} = (\mathcal{E}, \mathcal{R}, \mathcal{B}, \mathcal{Q})$ (Selic, 1999): $\mathcal{E}$ is the event alphabet, namely the set of all the event types that can be passed between the participants in a protocol, $\mathcal{R}$ are the protocol roles, namely the roles that the participants to the protocol can play, $\mathcal{B}$ represents the protocol reference behavior and identifies the set of legal behaviors that constitute a protocol and it is usually described by a state machine, $\mathcal{Q}$ 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 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 achieving a desired goal.

Bond-graphs. In every physical domain, there is a pair of variables, defined on a pair of dual vector spaces $\mathcal{F}_{p0}$ and $\mathcal{E}_{p0}$, 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 (van der Schaft, 2000) is defined by a pair $(e, f) \in \mathcal{E}_{p0} \times \mathcal{F}_{p0}$ and represents the means through which a physical system can exchange energy with the rest of the world. The bond graphs (Paynter, 1960) allow representing any lumped parameters physical system as a set of basic elements (that can be either energy storing or energy dissipating or source of energy) which are endowed with power ports through which they can exchange energy. The exchange of energy takes place through *bonds* that are interconnected by means of junctions whose behavior is governed by Kirchhoff like laws and that form the network structure along which the basic elements exchange energy. 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 graph are in general acausal (Paynter, 1960), it is possible to assign a causality to each element and thus to fix an input and an output for each power ports. Very often, energy storing elements have an integral causality associated and their behavior is represented by

$$\begin{cases} \dot{x}_i = u_i \\ y_i = \dfrac{\partial H}{\partial x_i} \end{cases} \qquad i = 1, \ldots, k \tag{1}$$

where k is the number of power ports associated to the element and H is a function of the state that represent the energy stored in the element in a given configuration and u and y 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 $u^T y \geq 0$, namely such that power is absorbed by the element. Encapsulation is possible within the bond-graphs framework: any physical system can be decomposed into a power preserving interconnection of *physical subsystems* that exchange

energy and that can be further decomposed in physical subsystems that exchange energy on their own. Recently, the power preserving interconnection structure has been mathematically formalized by introducing the concept of Dirac structure (van der Schaft, 2000). A very effective way to represent the physical interconnection behavior is the so called kernel representation. Suppose that there are m power ports exchanging energy through the interconnection structure and build the following vector:

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

where e_i and f_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:

$$E(x)e + F(x)f = 0 \quad (3)$$

where x is the state of the physical system and $E(x)$ and $F(x)$ are properly dimensioned matrices that satisfy the kernel representation conditions (van der Schaft, 2000).

3. A UML-RT MODEL OF PHYSICAL SYSTEMS

In this section we show how it is possible to interpret a physical system as a UML-RT model. To this aim we first give a system theoretic description of a UML-RT model. Consider a system made up of p capsules and m ports; a UML-RT system and its dynamic evolution can be described by the 5-tuple $(A, \mathcal{P}, \mathcal{C}, \alpha, \psi_{\mathcal{P}})$. The set

$$A = A_1 \times \cdots \times A_p \qquad a = (a_1, \ldots, a_p) \quad (4)$$

is the set of the attributes that characterize the overall system. Each set A_i is the set of the attributes of the capsule i. Since it is possible that some capsules have no attributes at all, some of the sets A_i can be empty sets. 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, etc. $\mathcal{P}$ is the protocol which is defined by the 4-tuple $(\mathcal{E}, \mathcal{R}, \mathcal{B}, \mathcal{Q})$ as detailed in Sec. 2. In particular the event set is given by:

$$\mathcal{E} = \bigcup_{i=0}^{m} \mathcal{E}_i \quad (5)$$

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:

$$E = \mathcal{E}_1 \times \cdots \times \mathcal{E}_m \qquad \varepsilon = (\varepsilon_1, \ldots, \varepsilon_m) \quad (6)$$

whose elements describe the ports configuration. Obviously it can happen that some or all of ε_i are empty, meaning that no message is crossing the corresponding port. $\mathcal{C}$ is the connector, the medium over which the capsules exchange information. The map $\alpha : A \times E \to A$ is the *attribute transition map* and it describes how the attributes of the capsules change in correspondence of a certain received message or of a spontaneous evolution due to some value of the attributes or to a transition of the state machine that can be associated either to a capsule or to the global system. The map $\psi_{\mathcal{P}} : A \times E \to E$ is the *port transition map* and it describes the dynamics of the messages crossing the ports. This dynamics depend both on the attributes of the system and on the signals crossing 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. Time does not play any role in the general description of the system since the UML-RT modeling can be used to describe both event based and time based systems. Now we can show how any physical system fits into this modeling framework.

Since a physical system is made up by a set of basic physical subsystems that exchange energy, the major architectural elements are physical subsystem which therefore can be modeled as capsules in UML-RT framework. Let p be the number of capsules describing a physical system. The attributes of each capsule are represented by a physical states. Thus:

$$A = \mathcal{X}_1 \times \cdots \times \mathcal{X}_n = \mathcal{X} \qquad x = (x_1, \ldots, x_n) \quad (7)$$

Usually, the set of attributes can be modeled as a differentiable manifold. The number of physical states is, in general, different from the number of capsules since each capsules can be characterized by several states or by zero states (e.g. purely dissipative capsules). The information that physical subsystems exchange in order to achieve a given behavior is *energy*. The exchange of energy happens through a power port and, therefore, it is associated to a pair of effort and flow variables. Thus one port is not sufficient to model the exchange of information that a physical subsystem has with the rest of the world, but a power port is needed. Within the UML-RT framework, each power port can be simply modeled by two ports, one relative to the flow and the other relative to the effort. Moreover, each capsule can have ports that are used to transmit signal not directly related to the exchange of energy, such as, for example, the state. Nevertheless these ports can play a role in the exchange of information among capsules. Each capsule can be further decomposed in subcapsules that exchange energy. In this case the energetic information exchanged by the capsule through its ports can be relayed to the ports of the subcapsules providing thus an

opening in the encapsulation shell. 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. The medium through which capsules are interconnected can be modeled as a connector $\mathcal{C}$ in the UML-RT framework and it 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 way in which the capsules exchange information. Thus, the physical structure of the interconnection is the connector while the topology of the interconnection is the protocol $\mathcal{P}$, implemented over the connector, through which the capsules exchange energy. Since the behavior of physical systems is continuous, the event set $\mathcal{E}$ has infinite cardinality and, therefore, in order to remark this difference, we call it *event space*. As efforts and flows are exchanged through the interconnection, the event space contains $\mathcal{F}_p \cup \mathcal{E}_p$. Energy is exchanged through power ports and, therefore, the pair of ports representing a power port participates to the protocol. Once causality has been fixed, each port can play a specific role: it can provide a flow, it can provide an effort, it can receive a flow or it can receive an effort; we call these roles *energy roles*. If a port is a flow (effort) receiver its companion port must be an effort (flow) supplier (this is a consequence of the first principle of thermodynamics, (Paynter, 1960)). 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*. Thus 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 part of the event alphabet; summarizing the event space is 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. 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.(3), which describe the relation among the efforts and the flows participating to the protocol, namely the way in which energy is exchanged. Since $E(x)$ and $F(x)$ satisfy the kernel representation conditions, it is always possible to calculate the signals (either efforts or flows) that have to be sent to the receiving ports (either effort or flow receiving) using the signals incoming from the supplying ports (either effort or flow supplying). When modeling physical protocol there is not any quality of service assessment since the protocol is just a model of how energy is exchanged and not an implementation of a certain messaging strategy. 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:

$$\alpha : \mathcal{F}_p \times \mathcal{E}_p \times \mathcal{X} \to \mathcal{X} \qquad (f(t), e(t), x(0)) \to x(t) \tag{8}$$

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

$$x_i(t) = \alpha(f, e, x(0)) = x_i(0) + \int_0^t u_i(\tau) d\tau \tag{9}$$

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

$$\psi_{\mathcal{P}} : \mathcal{F}_p \times \mathcal{E}_p \times \mathcal{X} \to \mathcal{F}_p \times \mathcal{E}_p \times \mathcal{X} \tag{10}$$

Thus, 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:

$$y_i(t) = \frac{\partial H}{\partial x_i}\Big|_{x(t)} \qquad i = 1, \ldots, n \tag{11}$$

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:

$$y_i(t) = g_i(u_i(t)) \quad i = n+1, \ldots, m \tag{12}$$

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:

$$m_i(t) = x_i(t) \qquad i = 1, \ldots, n \tag{13}$$

Once the signals crossing the ports associated to the output of power ports and those that cross the modulating ports, it is possible to calculate, through the protocol behavior equation, the inputs of the power ports, thus completing the ports

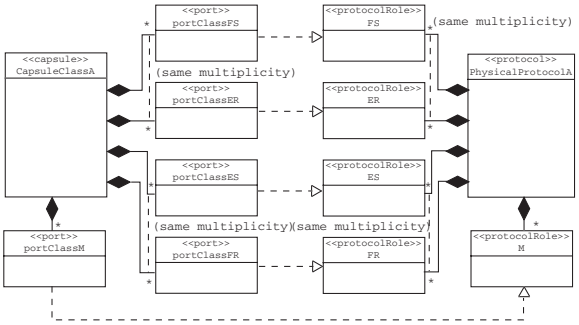

Fig. 1. UML-RT representation of a physical subsystem

configuration at time t. Thus we have proven the following:

Proposition. Any lumped parameters physical system can be represented as a UML-RT model

4. A UML DESCRIPTION

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 class stereotype `<<capsule>>`. 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 classes 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. In Fig. 1 is represented a capsule that models a generic physical subsystem. Each capsule can have several ports through which it can interact with the other capsules. Physical capsules can interact with the other physical capsules by means of the physical protocol, which is a collaboration indicated by the stereotype `<<protocol>>`. Each physical protocol has five kind of roles: effort supplier (ES), effort receiver (ER), flow supplier(FS), flow receiver (FR) and modulating (M). The multiplicity of the composition relationship of the protocol with each of its roles can be greater than one. The fact that for each port playing an effort (flow) supplier role there must be a port playing an effort (flow) receiver role is captured by the UML constraint `{same multiplicity}` applied both to the multiplicity of the composition relationship between the capsule and its ports and between the protocol and its roles. There are no constraints, instead, on the number of ports that play the modulating role. While in their standard use in UML-RT capsules represent event-driven entities, when modeling physical systems, capsules represent continuously time driven entities. As reported in (Selic and Motus, 2003), 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 then be modeled as a UML class and each continuously evolving system is in an association relationship with the physical time class. The attributes of each physical capsule evolve continuously in time and, therefore, a capsule is characterized by a continuous behavior. 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 equations modeling the behavior of a capsule can be represented through invariants relations between input, output and states.

5. EXAMPLE: DC MOTOR

Consider a DC motor that can be controlled either in position or in velocity, depending on the operator's choice. The physical plant interacts with the control software through velocity and position sensors and can be actuated by setting armature and field voltages. The operator set the kind of control to be performed through a simple graphic interface composed by one label, through which he/she can set either the position or the velocity setpoint and a radio button through which he/she can decide whether controlling the motor in position or in velocity. The structure of the overall system is reported in Fig. 2 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. This particular notation is allowed in UML-RT profile (Selic and Rumbaugh, 1998). The system is composed by three main capsules: the graphic user interface (GUI), the controller and the DC motor. The GUI communicates with the human operator and with the controller by the protocols **HG** and **GC** respectively. Both **HG** and **GC** are master slave protocols: The human operator (master of **HG**) sets the reference and the kind of control into the graphic interface (slave of **HG** and master of **GC**) which, on its turn, set the setpoint and the kind of control to the controller capsule (slave in **GC**). The controller communi-

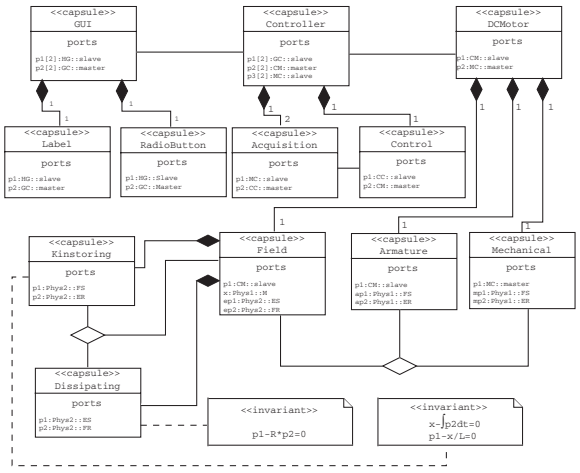

Fig. 2. The class diagram of the overall control system.

cates also with the DC motor through two other master slave protocol named **CM** and **MC**. The motor communicates the position and velocity information through the **MC** protocol while the controller set the control through the **CM** protocol. The GUI has two sub-capsules whose task is to implement the input label and the radio button. The controller has three sub-capsules: two sub-capsules are responsible of conditioning the signal received from the sensors and one sub-capsule has to use these data to calculate the control input to give to the DC motor. Thus the sub-capsules have to communicate and they do it through the master/slave protocol **CC**, where the ports of the Acquisition sub-capsules play the master role and those of the Control sub-capsule play the slave role. The DC motor capsule can be decomposed into three sub-capsules: the field sub-capsule, the armature sub-capsule and the mechanical sub-capsule that represent the armature circuit, the field circuit and the mechanical part of the motor respectively. Both the armature and the field sub-capsule are endowed with a port that receives the input voltage from the controller capsule. Furthermore the three sub-capsules communicate, namely exchange energy, through the protocol **Phys1**. The armature and the mechanical sub-capsules have a pair of ports that implement a power port and each of these port play a specific energy role while the field sub-capsule is endowed with a port that plays a modulating role within the physical protocol. The multiple association is represented by means of the standard UML diamond notation. The protocol behavior is represented by:

$$\underbrace{\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}}_{E(x)}\begin{pmatrix}mp2\\ ap2\end{pmatrix}+\underbrace{\begin{pmatrix}0 & -kx\\ kx & 0\end{pmatrix}}_{F(x)}\begin{pmatrix}mp1\\ ap1\end{pmatrix}=0 \quad (14)$$

where k is the electro-mechanical constant of the DC motor. Furthermore, each sub-capsule can be further decomposed in the interconnection of basic sub-capsules. For example, the field capsule can be decomposed in two sub-capsules: *Kinstoring*, an element storing kinetic energy, and *Dissipating*, an element dissipating energy. These elements communicate through the protocol **Phys2**. The behavior of the sub-capsules Kinstoring and Dissipating is reported exploiting the `<<invariant>>` constraint available in UML, where L and R are the inductance and the resistance of field circuit respectively. In order to keep the diagram simple, the association between the DC motor and the Physical time class has been omitted.

6. CONCLUSIONS AND FUTURE WORK

In this paper it has been shown how it is possible to model a complex control system as a set of, possibly compound, objects that achieve a certain behavior by exchanging information. These objects can be either software object that exchange information through a connector that implements an event based communication protocol or physical objects that exchange information, namely energy, through a physical connector which implements a certain continuous physical protocol, that can be modeled through a Dirac structure. UML-RT profile has been used to provide a unified language for modeling both physical systems and software architecture. Future work aims to investigate how to describe interaction between capsules with the new collaboration diagrams proposed in the new UML 2.0 specifications.

REFERENCES

Douglass, B.P. (2000). *Real-time UML - developing efficient objects for embedded systems*. second ed.. Addison-Wesley.

Paynter, H.M. (1960). *Analysis and Design of Engineering Systems*. M.I.T. Press.

Rumbaugh, J., I. Jacobson and G. Booch (1999). *The Unified Modeling Language Reference Manual*. Addison-Wesley.

Selic, B. (1999). Protocols and ports: reusable inter-object behavior patterns. In: *Proceedings of the Symposium on Object-Oriented Real-Time Distributed Computing*. Saint-Malo, France.

Selic, B. and J. Rumbaugh (1998). Using UML for modeling complex real-time systems. ObjecTime Limited/Rational Sofware Corp. white paper.

Selic, B. and L. Motus (2003). Using models in real-time software design. *IEEE Control Systems Magazine* **23**(3), 31–42.

van der Schaft, A.J. (2000). *L_2-Gain and Passivity Techniques in Nonlinear Control*. Communication and Control Engineering. Springer Verlag.