A Bond-Graph Metamodel: Physics-Based Interconnection of Software Components

Reynaldo Cobos Méndez[1(✉)], Julio de Oliveira Filho[2(✉)], Douwe Dresscher[1(✉)], and Jan Broenink[1(✉)]

[1] Robotics and Mechatronics Group, University of Twente, Enschede, The Netherlands
{r.cobosmendez,d.dresscher,j.f.broenink}@utwente.nl
[2] TNO, The Hague, The Netherlands
julio.deoliveirafilho@tno.nl

Abstract. Composability and modularity in relation to physics are useful properties in the development of cyber-physical systems that interact with their environment. The bond-graph modeling language offers these properties. When systems structures conform to the bond-graph notation, all interfaces are defined as physical “power ports” which are guaranteed to exchange power. Having a single type of interface is a key feature when aiming for modular, composable systems. Furthermore, the facility to monitor energy flows in the system through power ports allows the definition of system-wide properties based on component properties. In this paper we present a metamodel of the bond-graph language aimed to facilitate the description and deployment of software components for cyber-physical systems. This effort provides a formalized description of standardized interfaces that enable physics-conformal interconnections. We present a use-case showing that the metamodel enables composability, reusability, extensibility, replaceability and independence of control software components.

Keywords: Bond-graph · Metamodeling · Power port · Component software · Cyber-physical systems

1 Introduction

On developing cyber-physical systems that exchange power with the environment (e.g., mechatronic and robotic applications), their relation to the physics domain brings additional concerns to software developers. These include exchange, transformation and conservation of power; change of state; and geometrical constraints. Additionally, the interaction with physical systems leads to tight

This research has received funding from the RobMoSys project (EU project No. 732410) under the subproject EG-IPC. https://robmosys.eu/eg-ipc/.

F. Arbab and S.-S. Jongmans (Eds.): FACS 2019, LNCS 12018, pp. 87–105, 2020.
https://doi.org/10.1007/978-3-030-40914-2_5

requirements on safety and reliability [26]. Examples are the teleoperation applications with force feedback, which control architectures require a reliable bilateral interconnection of force and velocity [8,27]. The interaction of controllers and other software components with the physical world determines stability, performance and safety properties. In other words, the interaction of the system follows physical laws [38].

Conformance to physics and preservation of composition-related properties (e.g., composability, reusability, replaceability and independance) of software components facilitate the development of cyber-physical systems. Such conformance not only allows software models to seamlessly interact with the physical world but also to behave as physical elements. This leads to the need of a description method to connect component-based software for cyber-physical systems with physical laws.

Energy-based modeling languages describe physical systems using *power* as the universal interaction currency or *lingua franca* between elements [21,40]. These methods rely on the principle of conservation of energy and changes of states to determine state variables, which are necessary and sufficient to unambiguously describe a system [5]. The automatic conservation of energy, the possibility to incorporate geometrical constraints, and its graphical representation make the bond-graph notation a straightforward form of object-oriented modeling language for physical systems [12,13]. A bond-graph model is a labelled and directed graph in which the edges represent an ideal energy connection between its vertices representing (real-world) physical elements [9]. When system structures conform to the bond-graph language, the format of all interfaces are guaranteed to exchange power [21], facilitating their design and later implementation.

In this paper we integrate physics description to cyber-physical system models using bond-graph notation. We present a metamodel of the bond-graph language that formalises the features of energy-based modeling. The metamodel describes standard interfaces that are guaranteed to exchange power and provide hierarchy, inheritance and encapsulation. This effort aims to facilitate the description and deployment of power-exchanging cyber-physical systems by closing relationship gaps between software components and physical models - e.g. impedance controllers in haptic devices. It is worth stressing that incorporating the metamodel in tooling is beyond the scope of this work. However, the metamodel serves as strict guideline to provide of physics interpretation to software components by interconnecting them in a power-consistent way.

This paper is structured as follows: Sect. 2 is a review of related work associating the bond-graph language to software engineering paradigms, metamodeling and the component-based approach. Section 3 is a discussion of the abstraction level of the bond-graph language and its higher-order relations. Section 4 presents the analysis of the separation and classification of the bond-graph constructs into metamodeling entities. Section 5 contains the formal definition of the bond-graph entities, their properties and constraints. Section 6 is a discussion of the developed metamodel. A use-case example is presented in Sect. 7, followed by the conclusions of this work in Sect. 8.

2 Related Work on Bond-Graph Language and Tooling

The concept of bond-graph was originated in the early 60's by Paynter [29] and further developed by Karnopp and Rosenberg [24]. Later, Breedveld [4,6] provided an insight to describe multi-dimensional physical systems using bond-graph notation. In this section, we mention work on mapping the bond-graph language to paradigms of software-engineering. Then, we make a brief review on effort relating the bond-graph language with the component-based approach for systems modeling.

2.1 Relation to Software-Modeling Languages

Broenink [12] defined the bond-graph notation as a form of object-oriented modeling language for physical systems. The elements and properties described by Object Oriented Modeling (OOM) (i.e., *objects*, *hierarchies*, *inheritances*, and *encapsulation*) are usually illustrated by graphical representations such as Unified Modeling Language (UML) [20]. Some works explore describing physical systems using UML [34] and System Modeling Language (SysML) [16,18]. Other works aim on integrating software models with physical models using architectural description languages (ADLs) [3,19]. The mentioned approaches describe physical models for cyber-physical systems from a software-modeling perspective using standard representations for software engineering. In contrast, we integrate physical laws to software architectures from a physical-modeling perspective using the bond-graph notation. The later is particularly relevant as a straightforward enforcement of physical constraints.

On the search of a bond-graph metamodel, [35] presents a UML Class Diagram in bond-graph notation that specify the relations and data properties of the elements composing a certain physical model. [39] contributes on integrating the bond-graph language to SysML. This is done by mapping the bond-graph entities to SysML constructs. Another example is the work of [31], which propose a framework for component modeling using bond-graph-based metamodeling techniques. These efforts identify non-trivial relations of the bond-graph language to other metamodels and languages; however, details on bond-graph primitive constructs, constraints and relations to higher-order-knowledge are still missing. Our work tackles this gap by capturing the relation to physics and mathematics of the bond-graph notation into the metamodel.

2.2 Bond-Graph Language in Tooling

Various modeling tools supporting bond-graph notation are available. These tools use their own language to represent bond-graph models. General application platforms, like Simulink[1], use their native block diagram representation [2,22]. Other approaches explore implementing the bond-graph language in object-oriented environments using Modelica [12,15,17]. More dedicated tooling, like 20-SIM[2], use a bond-graph-based language for modeling, simulation and generation of implementable code [1,7,11].

Relating the component-based approach to bond-graph modeling language is non-trivial. An example is given in [10], which discusses how 20-SIM uses libraries of models and sub-models, exploiting the encapsulation provided by the bond-graph paradigm. [28] proposes (semi-)automating the generation of simulation models in bond-graph notation by using *off-the-shelf* component implementations. As a metamodel of the bond-graph language is missing (or at least inaccessible), the development and integration between tools is problematic. As a first step on the metamodel formulation, the following section positions the bond-graph language among the abstraction levels and addresses its relation to higher-order-knowledge.

3 Methods on Metamodeling

From the Object-oriented Modeling perspective, any bond-graph (sub-)model can be seen as an *object* containing the mathematical description of a physical system [9]. Mathematically speaking, the three models in Fig. 1 represent exactly the same system. The bond-graph model in Fig. 1 is a more abstract representation of the physical system compared to the iconic diagrams. In this section, the abstraction levels of physical system modeling is explored, as well as the relation of the bond-graph language to other known metamodels.

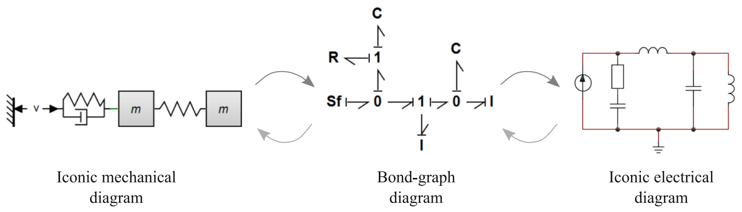

Fig. 1. Different representations of the same dynamic system.

3.1 Levels of Abstraction

In Fig. 1 there is a bond-graph model and two domain models represented by iconic diagrams - one mechanical and one electrical. The bond-graph model

[1] Available Simulink library for bond-graph: https://nl.mathworks.com/matlabcentral/fileexchange/11092-bond-graph-add-on-block-library-bg-v-2-1.
[2] More details about 20-Sim: https://www.20sim.com/.

represents both domain models as the same dynamic system. This identifies the bond-graph model as a Multi-Domain-Specific Language (MDSL) with respect to the iconic diagrams. The different levels of abstraction are represented in Fig. 2, where the bond-graph metamodel is located at a higher level (M2) with respect to the bond-graph model and iconic diagrams (M1). More details about levels of abstraction are presented in [14].

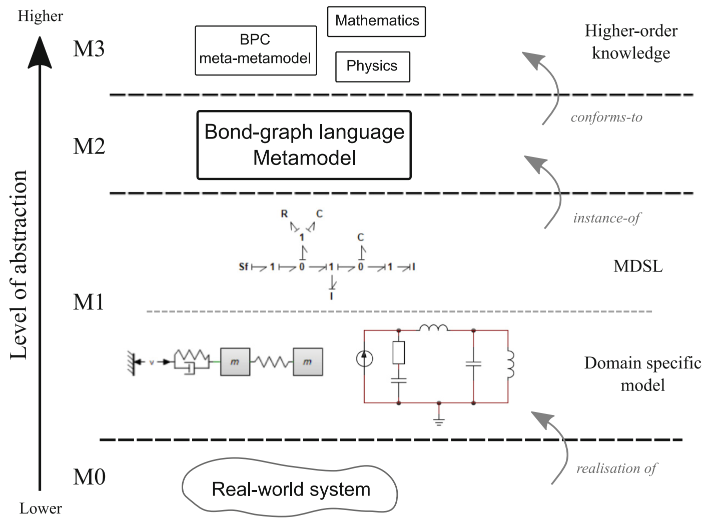

Fig. 2. Different levels of abstraction of a physical system.

As illustrated in Fig. 2, the bond-graph language conforms to higher-order knowledge (M3) - the Block-Port-Connector (BPC) metamodel, physics and mathematics. The bond-graph model in Fig. 1 would not be capable of describing physical interactions without its relation to other meta-metamodels, physics and mathematics. Therefore, it can be said that the bond-graph language adheres to other known metamodels.

3.2 Conforming to Higher-Order Knowledge

Mathematical abstractions of the bond-graph notation are port-Hamiltonian systems and Dirac structures [23]. The diagram in Fig. 3 is a port-Hamiltonian representation of an ideal physical system in bond-graph notation [37]. A port-Hamiltonian system can describe network models of physical systems that exchange power through ports [33]. The physical interaction among elements is done by the allocation of effort e and flow f variables on such ports and bonds. The mathematical relation between e and f characterizing the behavior of each (sub-)system is known as *constitutive relation*.

The energy of the system in Fig. 3 is characterized by a Hamiltonian equation, $H(e, f)$, and a Dirac structure, $D(e, f)$, representing power-conserving interconnections. Energy (defined as integral of power over time) is a conserved quantity, meaning that, in a closed system, it is at most transferred, converted or dissipated to the environment as free energy[3]. The energy conservation property is described by the power-conserving composition of $D(e, f)$ [32]. Thus, the port-Hamiltonian theory and Dirac structures serve as higher-order mathematical formulations of bond-graph models.

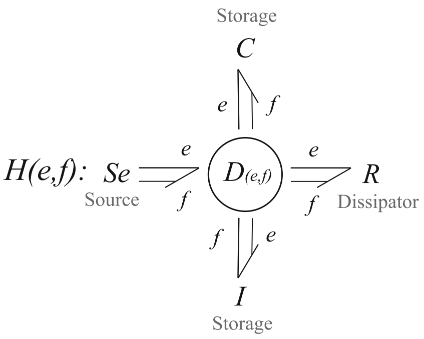

Fig. 3. A physical system represented as a port-Hamiltonian in bond-graph notation. A Dirac structure interconnects the source, storage and dissipative elements. The bonds denote power exchange as a product of effort and flow variables.

3.3 Conforming to Known Meta-Metamodels

When a model conforms-to a metamodel, the elements that are being used in the model satisfy the constraints on the relations that are made explicit in the metamodel [14]. This is the case of the Block-Port-Connector (BPC) meta-metamodel, whose modeling primitives are present in the bond-graph language known as *elements*, *junctions*, *power port* and *power bond*. In Table 1, a relation of the bond-graph constructs with the BPC primitives is shown, along with their higher-order knowledge link.

However, there are properties and constraints in the bond-graph language that cannot be sufficiently described by BPC to represent real physical interactions. For instance, physical interaction among systems require a bi-directional property in the connectors, as the energy is exchanged in both directions [6,29,33]. Given this, the bond-graph metamodel has to make such constraints explicit to avoid ambiguity when conforming to other meta-metamodels and paradigms.

The properties of OOM (i.e. *encapsulation*, *inheritance* and *hierarchy*) are essential when modeling physical systems [12]. The encapsulation into submodels

[3] Also known as the first law of thermodynamics.

and the inheritance property allow maintaining libraries of basic bond-graph elements and exchanging classes of components within a model. The hierarchy property allows complex bond-graph models to be embedded within other systems through power ports [21]. By conforming-to other metamodels (e.g, BPC), we can bring those entities, properties, attributes and relations required to describe the bond-graph language.

The realization of the metamodel is based on the separation of the elements of the modeling language into metamodeling concepts. In the following section, we describe the entities, relations and constraints of the bond-graph modeling language by identifying the properties and attributes of its elements.

Table 1. Relation between Block-Port-Connector primitives to bond-graph constructs.

BPC primitive	Bond-graph constructs	Physics/mathematics link
Block	- Dissipative elements - Storage elements - Source elements	*Hamiltonian theory*
	- Power junction - Transformer - Gyrator	*Dirac structures*
Port	- Power port	*port-Hamiltonian theory*
Connector	- Power bond	

4 Analysis of Bond-Graph Entities

As mentioned earlier, a bond-graph model is a graph which edges and vertices represent energetic interactions. Breedveld [6] provided a classification of the bond-graph vertices based on their energetic behavior. Such classification is represented in Table 2. The diagram in Fig. 4 is a representation of a bond-graph model whose elements are classified in four classes: *elements*, *junction structures*, *power port* and *power bond*. These classes have specific purposes in the bond-graph language and can be allocated and described in a higher level of abstraction.

Song [36] defines an *entity* as a primary thing that exists as itself and can be identified. Following this definition, the bond-graph *elements*, *junction structures*, *power ports* and *power bonds* can be classified as entities as they are independent things that can be clearly identified. Table 3 contains the properties and attributes of the bond-graph entities, which are essential to later formalize the modeling language.

Table 2. Classification of the bond-graph vertices based on their energetic behavior.

	Classification		Bond-graph constructs
Block	*Elements*	*Energic*	- Storage
		Entropic	- Dissipator
		Boundary	- Source
	Junction structures		- Power junction - Transformer - Gyrator

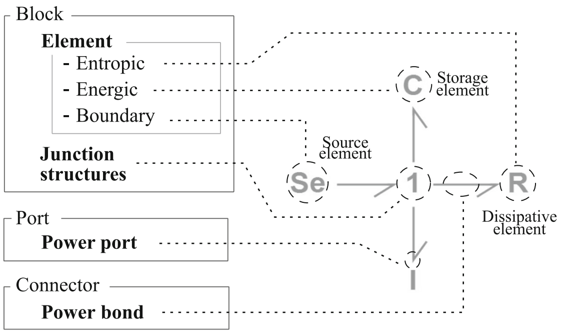

Fig. 4. Classification of bond-graph entities into 'classes'.

4.1 Classification of Bond-Graph Vertices

The UML diagram in Fig. 5 illustrates the classification of the bond-graph vertices based on their energetic properties and presence of parameters. The storages, dissipators and sources have at least one parameter - i.e., capacitance/compliance, inductance/mass, resistance/friction. We label these parametric elements as *BondElements*. The rest of the vertices are labeled as *JunctionStructures* as they interconnect *BondElements* (and other *JunctionStructures*) in a power continuous way.

As shown in Table 3, transformers and gyrators could also be classified as *BondElements* as they have parameters - namely transformation/gyration ratio. For the bond-graph metamodel, we propose classifying transformers and gyrators as parametric-*JunctionStructures* to distinguish them from the 0-/1-junctions. Having identified the entities and properties of the bond-graph language, we move forward on formally defining them into a metamodel.

5 Formalization of the Bond-Graph Metamodel

This section addresses the formal definition of the bond-graph language. Here, we capture the properties and constraints of the entities identified and classified in previous sections into a metamodel.

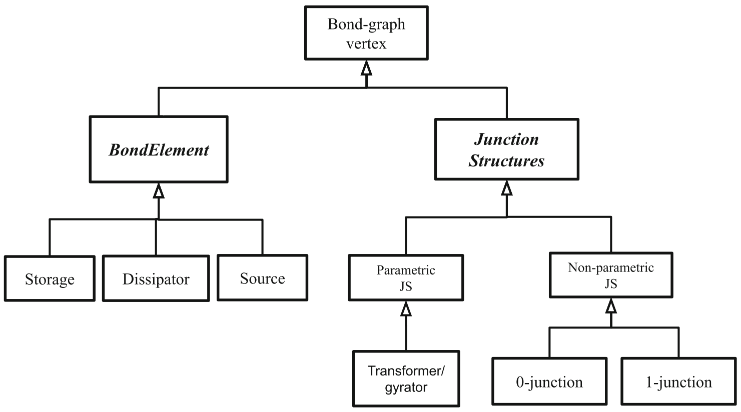

Fig. 5. Identification of bond-graph vertices based on their energetic behavior and presence of parameters

Table 3. Identification of properties and attributes of the bond-graph entities. Note: Transformers and gyrators are both Elements and Junction structures.

Bond-graph entity		*Property*	*Attribute*
Element	Storage	- Has constitutive relation	- Name - Symbol - Number of ports - Size of ports - Size of elements - Has parameters
	Dissipator		
	Source		
Element & Junction structure	Transformer/ gyrator	- Has power continuity - Has constitutive relation	- Name - Symbol - Number of ports - Has parameters
Junction structure	0-junction/ 1-junction	- Has power continuity - Has constitutive relation	- Name - Symbol - Number of ports
Power port		- Has 1 to 1 relation with bonds - Has 1 port has 2 variables in the constitutive relation	- Name
Power bond		- Has bidirectionality - Has power continuity - Connects 2 ports	- Name

5.1 Formal Definition of Entities, Properties and Constraints

The UML class diagram in Fig. 6a represents the association of entities of Table 3, based on the vertex classification of Fig. 5. The relation between *PowerPort* and *PowerBond* classes is represented in Fig. 6b. The definitions are enforced using *Description Logic* (DL) as UML is insufficient to describe the correct application of the bond-graph language. The formal semantics provided by DL let humans and computer systems exchange the same language, avoiding ambiguity [25].

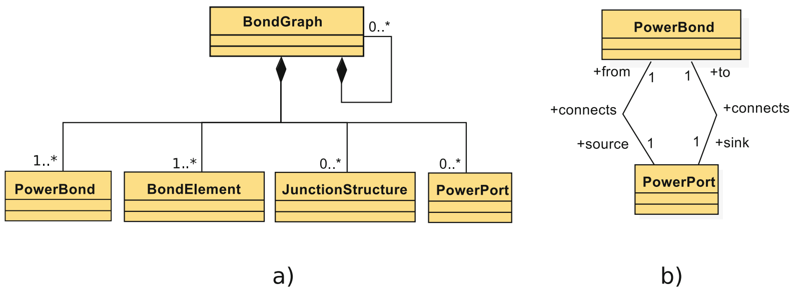

Fig. 6. UML diagram of bond-graph classes.

The bond-graph classes in Fig. 6 and their properties are expressed in detail in Table 4. The chosen language is the Web Ontology Language 2 (OWL2), which is based on DL. The axioms in Table 4 map the bond-graph classes to Block-Port-Connector (BPC) primitives (as in Table 1), facilitating the description of the bond-graph language.

Despite the bond-graph language conforms-to BPC, it is not allowed for a power bond to connect more than two power ports. Given this, it is required to constrain the relation between *PowerPort* and *PowerBond* with two '1-to-1 connects' shown in Fig. 6b. In the *PowerPort-PowerBond-PowerPort* relation, one *PowerPort* is declared as source and the other as sink. However, the energy exchange in both directions of the *PowerBond* is guaranteed by the *bidirectionality* property. Just like the previous example, the essential constraints of the bond-graph classes are formally expressed in Table 5 using DL[4].

5.2 Formal Definition of Power Variables

The bond-graph formalism indicates that the only variable to be 'transferred' among vertices through *PowerPorts* and *PowerBonds* is 'power' as a product of colocated effort and flow. To constrain this, it is proposed to define effort and flow as quantities containing a given value and a given unit. In the bond-graph metamodel ontology, we use the Ontology of Units of Measure described in [30]. The complete definition of the power variables is represented in Table 6.

5.3 A Note on Causality

The power-bond determines the effort and flow variable as a bilateral signal flow [5,29]. Meaning that, when one variable is given as an input for one port, its conjugated variable is automatically the output for that particular port. The inverse is true for the port at the other side. Causality is the policy that assigns the order of computations by determining whether the effort or the flow is either input or output. Causal analysis is addressed in [11] for computer-aided modeling and simulation purposes. In practice, causality is required to generate easy

[4] See the Appendix for more details about the symbols used in the formal definitions.

Table 4. Formal definitions of the bond-graph classes.

Class	*OW2 axiom*	*Related properties*
BondGraph	Declaration (Class(:BondGraph))	- hasPowerBond - hasJunctionStructure - hasBondElement
BondElement	Declaration (Class(:BondElement))	- hasParameter - hasConstitutiveRelation - hasPowerPort - DisjointWith Junction-Structure - Contains max 1 BondGraph
	SubClassOf (:BondElement :Block)	
PowerPort	Declaration (Class(:PowerPort))	- isPortOf exactly 1 (BondElement or Junction-Structure)
	SubClassOf (:PowerPort :Port)	
PowerBond	Declaration (Class(:PowerBond))	- Connects exactly 2 PowerPort - hasBidirectionality
	SubClassOf (:PowerBond:Connector)	
JunctionStructure	Declaration (Class(:JunctionStructure))	- hasPowerPort - hasConstitutiveRelation - DisjointWith BondElement
	SubClassOf (:JunctionStructure :Block)	

simulatable cyber-physical models. However, a model with the same structure but different causality is in fact the same model; therefore, this policy is not considered part of the bond-graph metamodel itself.

This section has addressed the structural allocation of the entities of the bond-graph language along with their formal definition and constraints. The following section is a discussion of the properties of the developed metamodel.

6 Metamodel Discussion

The bond-graph metamodel has to contain the properties and elements of other known meta-metamodels, in addition to the intrinsic characteristics of the bond-graph notation, as described in Sects. 3 and 4. This section is a discussion of the properties of the bond-graph metamodel as well as the resulting ambiguities as consequence of the limitations of the formalization language.

6.1 Properties of the Bond-Graph Metamodel

As mentioned earlier, the bond-graph language has *encapsulation*, *hierarchy* and *inheritance* properties. In the metamodel, encapsulation is represented in the

Table 5. Essential constraints of bond-graph classes

DL syntax	Description
Class: BondElement	
$BondElement \sqcap JunctionStructure = \perp$	No bond-graph vertex can be at the same time a BondComponent and a JunctionStructure
$BondElement \sqsubseteq \forall\ hasPowerPort.Port$	BondElements have any number of power ports
$BondElement \sqsubseteq \leq 1\ contains.BondGraph$	BondElements may contain at most 1 BondGraph
Class: PowerPort	
$PowerPort \sqsubseteq 1\ isPortOf.(BondElement \sqcup JunctionStructure)$	Any PowerPort is a port of exactly one BondElement or one JunctionStructure
$PowerPort \sqsubseteq \leq 1\ connects.PowerBond$	PowerPorts can connect to at most 1 PowerBond
Class: PowerBond	
$PowerBond \sqsubseteq 2\ connects.PowerPort$	A PowerBond connects exactly two PowerPort. Equivalently, a PowerBond cannot connect to more than two power ports, nor have one of its connection points loose
Class: JunctionStructure	
$JunctionStructure \sqcap BondElement = \perp$	No bond graph vertex can be at the same time a BondElement and a JunctionStructure
$JunctionStructure \sqsubseteq \forall\ hasPowerPort.PowerPort$	JunctionStructure have any number of power ports
$JunctionStructure \sqsubseteq 0\ contains.\top$	JunctionStructure is a block that contains nothing
Class: BondGraph	
$BondGraph \sqsubseteq \forall has\ BondComponent.BondComponent$	BongGraphs have BondElements
$BondGraph \sqsubseteq \forall has\ PowerBond.PowerBond$	BondGraphs have PowerBonds
$BongGraph \sqsubseteq \forall has\ JunctionStructure.JunctionStructure$	BondGraphs have JunctionStructures

relation between the *BondGraph* class and the rest of the classes in Fig. 6. As expressed in Table 5, an instance of a *BondGraph* is a collection that can contain elements, junction structures and power bonds. In a similar way, a *BondGraph* instance can contain other *BondGraph* models itself.

The encapsulation of bond-graph models is illustrated in Fig. 7 as follows: Model 4 is a *BondGraph* model composed of *BondElements*, *JunctionStructures* and other *BondGraph* models interconnected by *PowerBonds* (represented by half arrows) and *PowerPorts* (represented by black squares). Model 1, Model 2 and Model 3 are *BondGraphs* models containing at least one *BondElement* each. These constructions are allowed by the metamodel, providing *hierarchy* to the system.

Given the guaranteed interconnection of vertices and the encapsulation property, the construction of complex models can be simplified by using generic submodels and only modifying parameters and attributes. For instance, Model 2

Table 6. Definition of power variables and their constraints

Data property	DL definition	Description
effort	$\exists$ *effort*. $\top \sqsubseteq$ *PowerBond*	Domain of effort is always a PowerBond
	Quantity $\sqsubseteq \forall$ *effort.PowerBond*	Range of 'effort' is always a 'Quantity'
	Quantity $\sqsubseteq \leq 1$ *effort.PowerBond*	'effort' is functional, and additionally for each individual moment in time
flow	$\exists$ *flow*. $\top \sqsubseteq$ *PowerBond*	Domain of flow is always a PowerBond
	Quantity $\sqsubseteq \forall$ *flow.PowerBond*	Range of 'flow' is always a 'Quantity'
	Quantity $\sqsubseteq \leq 1$ *flow.PowerBond*	'flow' is functional, and additionally for each individual moment in time

and Model 3 in Fig. 7 can be described as storage elements with different *ConstitutiveRelation*, parameter value and symbol. Therefore, *inheritance* is supported by the metamodel.

6.2 Note on Completeness of the Bond-Graph Metamodel

The implementation of the bond-graph metamodel into tooling is out of the scope of this paper. This represent a limitation on the assessment of the completeness of the metamodel to describe physical systems. However, it is possible to determine whether or not the formal definitions adhere to the the bond-graph notation to construct models.

The interaction between vertices has to be done only through the association between *PowerPort* and *PowerBond* classes. The UML diagram in Fig. 6 and the DL definitions in Table 5 enforce this constraint. Still, there is no formal definition that prevents a *PowerBond* to be connected at both sides to the same *BondElement* or *BondGraph* model (through different ports) as shown in the system in Fig. 8a. Nevertheless, such an unusual connection does not compromise the correct application of the bond-graph language to describe the physical behavior of the system. For instance, Fig. 8b is a representation of the same system shown in Fig. 8a. On the other hand, the structure in Fig. 9a is not allowed due to lack of information required to describe the power exchange between *BondElements*. Changing the splitter to a *JunctionStructure* block - either 0-junction or 1-junction[5] - in Fig. 9b is essential to allow a physics-conformal interconnection.

[5] Depending on the model, the *JunctionStructure* could be either a 1-junction or a 0-junction as they denote different Diract structures.

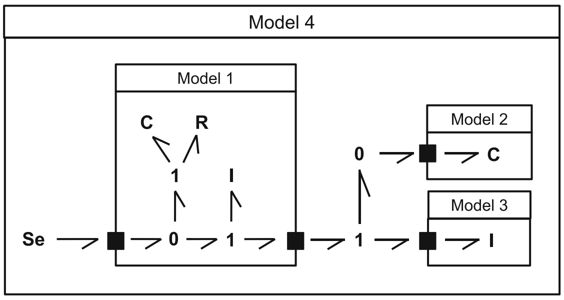

Fig. 7. A bond-graph model encapsulating other bond-graph (sub-)models.

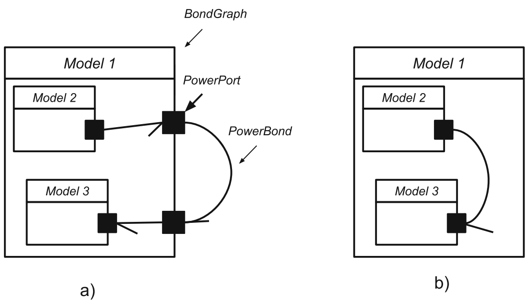

Fig. 8. Two equal systems (a) Unusual connection to same element. (b) Encapsulation of the PowerBond.

7 Interfacing Software Components Using Bond-Graph Entities

By incorporating the bond-graph metamodel to the component-based approach, software models like in Fig. 10 can be realized. The software components have instances of *PowerPort* class connected by instances of *PowerBond* class. The components themselves are *BondElement* instances which *ConstitutiveRelation* (see Table 4) is the implementation. Since power exchange (product of effort and flow variables) is enforced on the interfaces, the components can be exchanged or replaced depending on the application. Thus, the component implementation can be extended without compromising its independence from the interfaces. In this section we present a use case example where the bond-graph metamodel is applied.

7.1 Use-Case Example: Haptic Telemanipulation

We applied the entities described by the bond-graph metamodel on the teleoperation with force feedback use case depicted in Fig. 11. The system model is in bond-graph notation, which means that power is exchanged between the elements. A human operator telemanipulates a robot ('Slave') using another robot

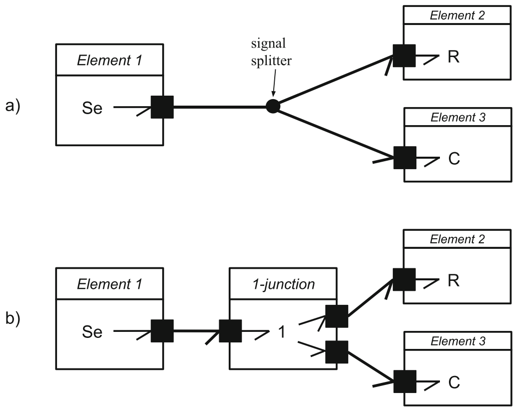

Fig. 9. Interaction between BondElements: (a) Ill-connection through a signal splitter not allowed by the metamodel. (b) Correct application of the bond-graph language. Note: the 1-junction in (b) could be a 0-junction.

('Master') as haptic device. The impedance controller component is provided of power ports and its implementation is bond-graph conformal - that is, dealing with effort and flow signals and other constraints as formalized in the metamodel. This is the same for the geometric Jacobian components.

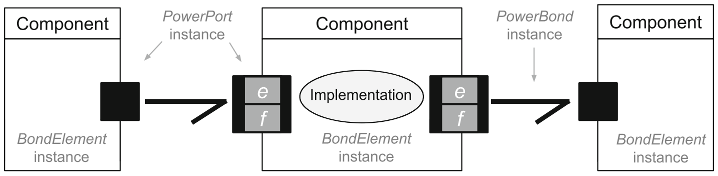

Fig. 10. Software components interacting via power ports and power bonds. Given the interfaces, it can be assumed that *power* is exchanged among them.

The *PowerPort* and *PowerBond* instances provide a power-consistent interconnection between the physical and virtual environments in Fig. 11. In other words, the impedance controller has a 'physical' link with the target environment and the operator. Thus, a (force) feedback loop is enforced. The interfaces allow the use-cased system to be composable and its components replaceable. The impedance controller and geometric Jacobians can be replaced by other components as long as they have the required power ports. The implementation of each component can also be extended. If the application changes - for instance, adding another 'slave' robot - the controller and Jacobian can be reused. Additionally, system developers can get the flexibility and independence offered by the component software approach by having *application-specific* and *tool-specific* components clearly separated.

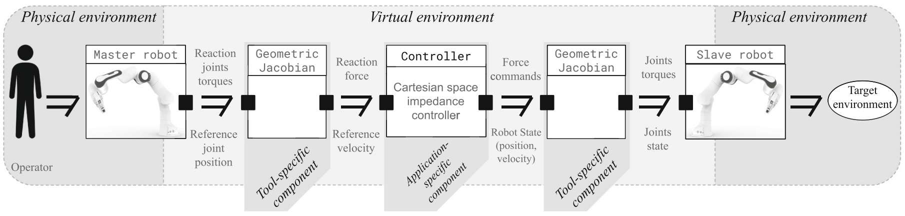

Fig. 11. Control software model of the teleoperation application in bond-graph notation.

8 Conclusions

We presented a metamodel that formalizes the bond-graph modeling language. This effort aims to ease the development of software components for cyber-physical systems that interact with their environment. This is done by enforcing the correct application of the bond-graph notation to describe physical interconnections among components. The elements, properties and constraints of the modeling language were identified and characterized using Description Logics. The result is a set of definitions of bond-graph classes, their object properties and relations that conform to physics, mathematics and Block-Port-Connector meta-metamodel.

As discussed in Sect. 6, the bond-graph metamodel supports encapsulation, hierarchy and inheritance, while providing of a physics interpretation to the component-based approach. The use case in Sect. 7 suggests that the metamodel can serve as a strict guideline to develop software components interfacing with physical systems. It was shown that composability, reusability, extensibility, replaceability and independence of components are present.

Further work can be done on integrating the bond-graph metamodel to software-modeling tooling. This means merging physical laws to *off-the-shelf* software components. Such incorporation can help reducing development times and improve cost-effectiveness by facilitating the task of control-software developers.

Moreover, further work can be on the formal description of causality. Thus the presented metamodel becomes more rich. Tools using this extended bond-graph metamodel can be enhanced with automated causal analysis functions, exploiting transformation from energy relations to signal input-output relations - as done by dedicated tooling for bond-graph modeling and simulation, e.g., 20-SIM.

Despite the tooling limitation, the presented metamodel is useful to provide the required descriptions to make already existent control components abide physics through guaranteed power exchange. The facility to monitor energy flows in the system through power ports can allow the definition of system-wide properties based on component properties, most notably passivity. Therefore, the bond-graph metamodel can contribute on reliability and safety of component-based cyber-physical systems.

Appendix

See Table 7.

Table 7. Symbols used in DL definitions

Symbol	Legend
$\sqcap$	Meet semilattice
$\perp$	Minimum element or bottom
$\sqsubseteq$	Partial order relation
$\forall$	For all
$\exists$	There exist
$\top$	Maximum element or top
$\sqcup$	Disjoint union

References

1. Automated modelling. In: Borutzky, W. (ed.) Bond Graph Methodology: Development and Analysis of Multidisciplinary Dynamic System Models, pp. 469–560. Springer, London (2010). https://doi.org/10.1007/978-1-84882-882-7_11
2. Antic, D., Vidojkovic, B.: Obtaining system block diagrams based on bond graph models and application of bondsim tools. Int. J. Model. Simul. **21**(4), 257–262 (2001). https://doi.org/10.1080/02286203.2001.11442210
3. Bhave, A.Y., Garlan, D., Krogh, B., Rajhans, A., Schmerl, B.: Augmenting software architectures with physical components. In: Embedded Real Time Software and Systems Conference (2010)
4. Breedveld, P.C.: Multibond graph elements in physical systems theory. J. Franklin Inst. **319**(1), 1–36 (1985). https://doi.org/10.1016/0016-0032(85)90062-6
5. Breedveld, P.: Integrated modeling of physical systems - dynamic systems, vol. 1. University of Twente, Enschede, The Netherlands (2014)
6. Breedveld, P.C.: Physical Systems Theory in Terms of Bond Graphs. Twente University of Technology, Department of Electrical Engineering, Enschede (1984). oCLC: 852801415
7. Breunese, A.P.J., Broenink, J.F.: Modeling mechatronic systems using the SIDOPS+ language. Simul. Ser. **29**(1), 301 (1997). oCLC: 106228295
8. Brodskiy, Y.: Robust autonomy for interactive robots (2014). https://doi.org/10.3990/1.9789036536202
9. Broenink, J.: Introduction to physical systems modelling with bond graphs (1999)

10. Broenink, J.F.: 20-sim software for hierarchical bond-graph/block-diagram models. Simul. Pract. Theory **7**(5), 481–492 (1999). https://doi.org/10.1016/S0928-4869(99)00018-X
11. Broenink, J.F.: Computer-aided physical-systems modeling and simulation: a bond graph approach, March 1990
12. Broenink, J.F.: Object-oriented modeling with bond graphs and Modelica. In: Proceedings of the 1999 International Conference on Bond Graph Modeling and Simulation, pp. 163–168, February 1999
13. Brown, F.T.: Engineering System Dynamics: A Unified Graph-Centered Approach, 2nd edn. CRC Press (2006). https://doi.org/10.1201/b18080
14. Bruyninckx, H., Scioni, E., Hubel, N., Reniers, F.: Composable control stacks in component-based cyber-physical system platforms, April 2018
15. de la Calle, A., Cellier, F.E., Yebra, L.J., Dormido, S.: Improvements in BondLib, the Modelica bond graph library. In: 2013 8th EUROSIM Congress on Modelling and Simulation, pp. 282–287, September 2013. https://doi.org/10.1109/EUROSIM.2013.58
16. Cao, Y., Liu, Y., Fan, H., Fan, B.: SysML-based uniform behavior modeling and automated mapping of design and simulation model for complex mechatronics. Comput. Aided Des. **45**(3), 764–776 (2013). https://doi.org/10.1016/j.cad.2012.05.001
17. Cellier, F.E., Nebot, À.: The Modelica Bond Graph Library, p. 10 (2005)
18. Chen, R., Liu, Y., Cao, Y., Zhao, J., Yuan, L., Fan, H.: ArchME: a systems modeling language extension for mechatronic system architecture modeling. AI EDAM **32**(1), 75–91 (2018). https://doi.org/10.1017/S0890060417000245
19. Garlan, D., Monroe, R.T., Wile, D.: ACME: architectural description of component-based systems. In: Foundations of Component-Based Systems, pp. 47–68. Cambridge University Press (2000)
20. Garrido, J.M.: Object orientation. In: Garrido, J.M. (ed.) Object Oriented Simulation, pp. 51–58. Springer, Boston (2009). https://doi.org/10.1007/978-1-4419-0516-1_5
21. Gawthrop, P.J., Bevan, G.P.: Bond-graph modeling. IEEE Control Syst. **27**(2), 24–45 (2007). https://doi.org/10.1109/MCS.2007.338279
22. Geitner, G.: Power flow diagrams using a bond graph library under simulink. In: IECON 2006–32nd Annual Conference on IEEE Industrial Electronics, pp. 5282–5288, November 2006. https://doi.org/10.1109/IECON.2006.347232
23. Golo, G., van der Schaft, A., Breedveld, P.C., Maschke, B.M.: Hamiltonian formulation of bond graphs. In: Johansson, R., Rantzer, A. (eds.) Nonlinear and Hybrid Systems in Automotive Control, pp. 351–372. Springer, London (2003)
24. Karnopp, D., Rosenberg, R.C.: Analysis and Simulation of Multiport Systems: The Bond Graph Approach to Physical System Dynamics. MIT Press, Cambridge (1968)
25. Krötzsch, M., Simancik, F., Horrocks, I.: Description logics. IEEE Intell. Syst. **29**(1), 12–19 (2014). https://doi.org/10.1109/MIS.2013.123
26. Lee, E.A.: Cyber physical systems: design challenges. In: 2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC), pp. 363–369, May 2008. https://doi.org/10.1109/ISORC.2008.25
27. Mersha, A.Y.: On autonomous and teleoperated aerial service robots (2014). https://doi.org/10.3990/1.9789036536585

28. Novák, P., Šindelář, R.: Component-based design of simulation models utilizing bond-graph theory. IFAC Proc. Vol. **47**(3), 9229–9234 (2014). https://doi.org/10.3182/20140824-6-ZA-1003.01167
29. Paynter, H.M., Briggs, P.: Analysis and Design of Engineering Systems: Class Notes for M.I.T. Course 2.751. MIT Press, Cambridge (1961). Massachusetts Institute of Technology
30. Rijgersberg, H., van Assem, M., Top, J.: Ontology of units of measure and related concepts. Semant. Web **4**(1), 3–13 (2013). https://doi.org/10.3233/SW-2012-0069
31. Sampath Kumar, V.R., Shanmugavel, M., Ganapathy, V., Shirinzadeh, B.: Unified meta-modeling framework using bond graph grammars for conceptual modeling. Robot. Auton. Syst. **72**, 114–130 (2015). https://doi.org/10.1016/j.robot.2015.05.003
32. van der Schaft, A., Cervera, J.: Composition of Dirac structures and control of Port-Hamiltonian systems. In: Proceedings of the 15th International Symposium on the Mathematical Theory of Networks and Systems. University of Notre Dame (2002)
33. Scioni, E., et al.: Hierarchical hypergraphs for knowledge-centric robot systems. In: A Composable Structural Meta Model and its Domain Specific Language NPC4 (2016). https://doi.org/10.6092/JOSER_2016_07_01_p55
34. Secchi, C., Bonfe, M., Fantuzzi, C.: On the use of UML for modeling mechatronic systems. IEEE Trans. Autom. Sci. Eng. **4**(1), 105–113 (2007). https://doi.org/10.1109/TASE.2006.879686
35. Sen, S., Vangheluwe, H.: Multi-domain physical system modeling and control based on meta-modeling and graph rewriting. In: 2006 IEEE Conference on Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control. pp. 69–75, October 2006. https://doi.org/10.1109/CACSD-CCA-ISIC.2006.4776626
36. Song, I.Y., Froehlich, K.: Entity-relationship modeling. IEEE Potentials **13**(5), 29–34 (1995). https://doi.org/10.1109/45.464652
37. Stramigioli, S.: Intrinsically passive control using sampled data system passivity. In: Multi-point Interaction with Real and Virtual Objects, pp. 215–229, July 2005. https://doi.org/10.1007/11429555_14
38. Stramigioli, Stefano: Energy-aware robotics. In: Camlibel, M.Kanat, Julius, A.Agung, Pasumarthy, Ramkrishna, Scherpen, Jacquelien M.A. (eds.) Mathematical Control Theory I. LNCIS, vol. 461, pp. 37–50. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-20988-3_3
39. Turki, S., Soriano, T.: A SysML extension for Bond Graphs support (2005)
40. van der Schaft, A., Jeltsema, D.: Port-Hamiltonian systems theory: an introductory overview. Found. Trends® Syst. Control **1**(2), 173–378 (2014). https://doi.org/10.1561/2600000002