Power and Energy Communication Services for Control-software Models

Reynaldo Cobos Mendez
Robotics and Mechatronics
University of Twente
Enschede, The Netherlands
r.cobosmendez@utwente.nl

Douwe Dresscher
Robotics and Mechatronics
University of Twente
Enschede, The Netherlands
d.dresscher@utwente.nl

Jan Broenink
Robotics and Mechatronics
University of Twente
Enschede, The Netherlands
j.f.broenink@utwente.nl

2021 IEEE/ACM 3rd International Workshop on Robotics Software Engineering (RoSE) | 978-1-6654-4474-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/ROSE52553.2021.00016

***Abstract*—Implementing energy-based controllers in software represents a challenge for software engineers, as additional expertise is required to abide by the physics-domain constraints of energy exchange in the design and structure of the control software. Our paper bridges the gap between software engineering and the physics domain by conveying energy exchange to control-software modelling. We use principles of physical systems and the bond-graph modelling language to identify the mechanisms and constraints of energy exchange and represent them as data-communication services for software models. This work resulted in metamodels and models for power and energy communication that can facilitate the *first-time-right* implementation of robot-control software.**

***Index Terms*—domain-specific ontologies, domain-specific constraints, energy exchange, software modelling, real-time systems, bond graph.**

I. Introduction

Controllers that enable the robot to react with compliant behaviour when engaging in physical interactions are critical for their safe operation in dynamic and unknown environments. Levering the development of such controllers has been one of the goals of EU Robotics in recent years, as stated in its Strategic Research Agenda (SRA2020)[1]. On robots dealing with physical interactions, passivity-based control (PBC) offers robust and stable compliant behaviour in contact with unknown environments [1]–[3]. PBC relies on *energy* as shared property of control and physical systems, allowing shaping compliant behaviour using energy-shaping techniques [4]. Applications such as robotic tele-operated manipulation benefit from the energy-based approach of PBC (see for instance the work of [5]–[7]). Given that physical interaction can be exclusively characterised by *energy exchange* [8], the communication between the controller and the physical system can also be described as energy exchange when generalised forces and generalised velocities are used as steering and feedback signals.

However, implementing energy-based controllers in software represents a challenge for software engineering. Software engineers are required to have additional expertise on physics and physical-systems modelling to abide by the domain-specific constraints of energy exchange in the design and structure of the control software. Conforming to physics in the communication of steering and feedback signals is essential for stability and performance of energy-based controllers. An example is the destabilising effect of time delayed communication in bilateral teleoperators [9], [10], as lateness in the communication of steering and feedback signals breaks the physics conformance of power exchange. When *time-to-market* requirements come into play, abiding by the physics-domain constraints of energy exchange at modelling level can enable *first-time-right* implementation of energy-based control software.

Domain-specific languages (e.g., bond graph, Modelica[2], SIDOPS+ [11]) and tooling solutions (e.g., Simulink[3], 20sim[4]) exist for representing physical interaction in models of physical systems. However, such representation methods are not entirely compatible with software models, as they rely on domain-specific knowledge not associated with software engineering. Therefore, definition, representation and implementation guidance in software models of the physics-domain ontologies of energy exchange is needed.

Our paper bridges the gap between software engineering and the physics domain in designing and structuring robot-control software. We convey the mechanisms and constraints of energy exchange to software engineering by representing them as data-communication services for software models. A communication service, as proposed by RobMoSys[5], defines the semantics by which different parts of the software exchange data. This is convenient as it facilitates defining domain-specific constraints. We define these communication services in metamodels to facilitate the clear and unambiguous structuring of energy-based control-software models. Additionally, we present example models that conform to these metamodels and a use-case example.

To achieve this, we use principles of physical systems and the bond-graph modelling language to identify the mechanisms and constraints of energy exchange. For modelling and control

This research has received funding from the RobMoSys project (EU project No. 732410) under the subproject EGCS https://robmosys.eu/egcs/

[1]Strategic Research Agenda for Robotics in Europe 2014-2020. https://www.eu-robotics.net/sparc/upload/topic_groups/SRA2020_SPARC.pdf

[2]https://www.modelica.org

[3]More details about Simulink at https://www.mathworks.com

[4]https://www.20sim.com

[5]https://www.robmosys.eu/

978-1-6654-4474-3/21/$31.00 ©2021 IEEE
DOI 10.1109/RoSE52553.2021.00016

of physical systems, both the energy and its rate of change – the *power* – are of interest. The bond-graph modelling language is useful for describing the exchange of power among physical elements in a graphical and object-oriented way [12]. Therefore, we use entities of the bond-graph notation as guidance to provide physics conformance to power and energy communication services.

This paper is structured as follows: Section 2 concerns the principles of energy exchange in software models. Section 3 presents the physics-domain constraints and our approach to incorporate them in software models. Section 4 provides the metamodels and example models of power and energy communication services. Section 5 presents a use-case example. Finally, the conclusion to this paper and recommendations are addressed in Section 6.

II. Energy exchange in control software

Previous work states that software components can be described as physical elements with physics-conformal interconnections by using entities of the bond-graph notation to represent energy exchange [13]. This provides a physical interpretation to the interconnection of software elements by representing it as a power flow. The same approach can be used to describe energy-consistent communication of control-software models.

See for instance the model described in bond-graph notation in Fig. 1a. An impedance controller and a physical system – a single mass, for instance – are connected through a half arrow denoting the power transfer between these two elements. In the bond-graph notation, such arrow is called *power bond*. The orientation of the power bond indicates the direction in which the power has a positive sign – i.e., if the power is positive, it flows in the direction of the half arrow; if the power is negative, it flows the other way around.

The communication between controller and robot can be interpreted as energy exchange when the product of the steering and feedback signals has a physical dimension of power (i.e., in Watt). This is the case when the control signals are composed by a generalised force, F (generalised in bond-graph terms as *effort*, e), and a generalised velocity, v (generalised in bond-graph terms as *flow*, f). Power, P, can be computed using (1), which means that a power bond can be computed using effort and flow as two signals calculated in opposite direction. Therefore, effort and flow are called *power-conjugated* variables.

$$P \;=\; e\,f \;=\; F\,v \tag{1}$$

In practice, input and output signals can be used to communicate power-conjugated variables, as shown in Fig. 1b. Moreover, *generalised positions* are preferred as feedback signals instead of velocities, as it is more practical to acquire and use position sensors. A *generalised position*, x (generalised in bond-graph terms as q-type variable), is the time integral of a generalised velocity (see (2)). Analogously, a *generalised momentum*, p (generalised in bond-graph terms as p-type variable) is the time integral of a generalised force (see (3)). In physical-systems modelling, the q-type and p-type variables are the *state variables* of a physical system. The elements of the model depicted in Fig. 1b, whether they use power-conjugated variables or state variables, can exchange energy as if they were physical elements.

$$q \;=\; \int f\,dt \;=\; \int v\,dt \tag{2}$$

$$p \;=\; \int e\,dt \;=\; \int F\,dt \tag{3}$$

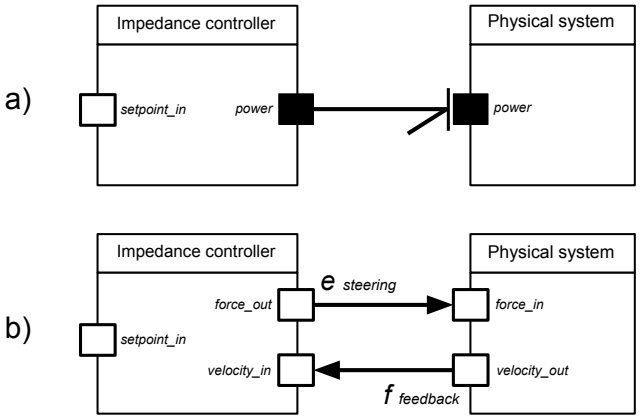

Fig. 1. Software models of an impedance controller exchanging power with a physical system: a) described in bond-graph notation and b) described using I/O signals.

Real physical elements are passive in nature – i.e., they only store or dissipate energy, never generate it. For the control software to conform to physics, the passivity property must hold. This is non-trivial, as energy generation in the controller compromises system stability and dependability. The too-late arrival of data can lead to a violation of passivity, as is the case of too large delays in the communication and computation of the control signals [14], [15]. To preserve passivity and stability, mechanisms that keep the energy bounded have to be implemented by carrying out a bookkeeping of the exchanged energy between the controller and the robot [6], [7], [16]. These methods rely on monitoring power exchange to enforce passive behaviour on the communication of energy data. See Appendix A for details on computation of energy in the control software.

The bond-graph language describes ideal energetic interconnections of physical elements. That is, a bond-graph model shows the exchange of signals between sub-models from the point of view of energy and energy exchange. As the bond-graph language is a domain-specific language of energy exchange, it does not dictate how the control software should handle effort and flow variables when used as steering and feedback signals, nor takes into account the limitations imposed by a real-world implementation of software – e.g., computation methods, discretisation, communication channels,

timing, hardware. Yet, the bond-graph language can serve to identify the physics-domain constraints that the control software has to abide by.

III. Domain-specific Constraints of Energy Exchange

The power-bond entity represents *instantaneous, point-to-point power transfer* among two interconnected elements. Energy leaves and enters an element through port-like entities called *power ports*. The power port comprises the *collocated* effort and flow variables of power exchange. For computation purposes, it is essential to determine whether the effort variable is computed as a function of the flow variable, or the other way around. This is called *causality*, and it defines inputs and outputs in the control software.

For a physics-conformal communication of power and energy using either power-conjugated variables or state variables, the control software has to abide by the physics-domain constraints applied to power-bond and power-port entities. Therefore, for the steering and feedback signals to represent power exchange, their variables must abide by:

1) The computational causality direction
2) Collocation – i.e., be expressed in the same coordinate frame
3) Instantaneous communication

A. Causality

A power-port entity can have one of the two causal arrangements: either effort as input and flow as output, or flow as input and effort as output. Power ports have *cardinality* 1 in their relation with power bonds – i.e., power ports can only be connected to a single power bond [17]. As power bonds cannot be split, two connected power ports have opposite causality. In control-software components this can be translated to output signals of *one* component being the input of *one* other component. For the exchange of power in control software to abide by the physical laws, such communication must be done exclusively *1-to-1* between an opposite-causal and collocated pair of ports communicating the steering and feedback control signals.

B. Collocation

In power exchange, collocation implies that both the effort and flow – alternatively, the generalised momentum and generalised position – are expressed in the same coordinate frame. For example, when measuring the angular position of a joint upon a given steering torque to that same joint, it is said that the torque and the angular position are collocated.

C. Instantaneous communication

The values used to compute power and energy should correspond to the same moment in time. This implies that the communication of steering and feedback signals should be done *instantaneously*. When this does not happen, the computation of power and energy is not accurate. However, such synchronicity is too strict for the control software, given the inherent delays in data computation and communication. Instead, *timeliness* can be used as a more relaxed – yet strict – property of the control system in the handling of these signals.

Timeliness[6] is "the fact or quality of [an event of] happening at the best possible time or at the right time". The difference between *at the best possible time* and *at the right time* lies in the tolerance given to an event to happen with respect to a *deadline*. In the control software, such *timely events* include the computation and communication of the variables involved in power exchange.

The control software requires mechanisms to guarantee the timely communication of its variables. Conveniently, *real-time* (RT) systems can fulfil such requirements. In a RT system the response time is essential for the correct operation of the software, as a task executed late might bring worse consequences than the computation of incorrect values [18].

Given the impact on system stability and performance, the computation and communication of physical variables from which power and energy is computed must be done in firm real-time (FRT) or hard real-time (HRT) – i.e., with strict deadlines. On the other hand, soft real-time (SRT) – i.e., a more tolerant deadline – is convenient for tasks involving physical variables for informative purposes. When timeliness cannot be guaranteed due to large communication delays, passivity-enforcing methods can be used to extend the stability range of networked robotic systems (as mentioned in Section II). See Appendix B for more details on the differences between FRT, HRT and SRT.

IV. Power & Energy Communication Services

This section addresses the incorporation of the physics-domain constraints of power and energy exchange into data-communication services, which are made explicit in metamodels.

A. Power Computation-Communication Service

The UML diagrams in Fig. 2 depict the *power-computation communication-service* metamodel. This service defines the *data stream* of two communication objects – an *Effort-type* variable and a *Flow-type* variable – arranged in two causal configurations. The causal configurations define the causality constraint when transferring power between two elements. A data stream is a *peer-to-peer* communication between components [19], which is necessary to guarantee *1-to-1* exchange of power.

The *Effort-type* communication object contains either effort, e, or a generalised momentum, p-type variable. The *Flow-type* communication object contains either flow, f, or a generalised position, q-type variable. Both communication objects are defined as *Power-conjugated quantities* (see the UML diagram in Fig. 3). Formally speaking, p-type and q-type variables are not power-conjugated variables; however, for simplicity and because (2) and (3), they are defined in the same communication object. Moreover, communicating both

[6] As defined in the Cambridge Diccionary, retrieved from URL https://dictionary.cambridge.org/dictionary/english/timeliness

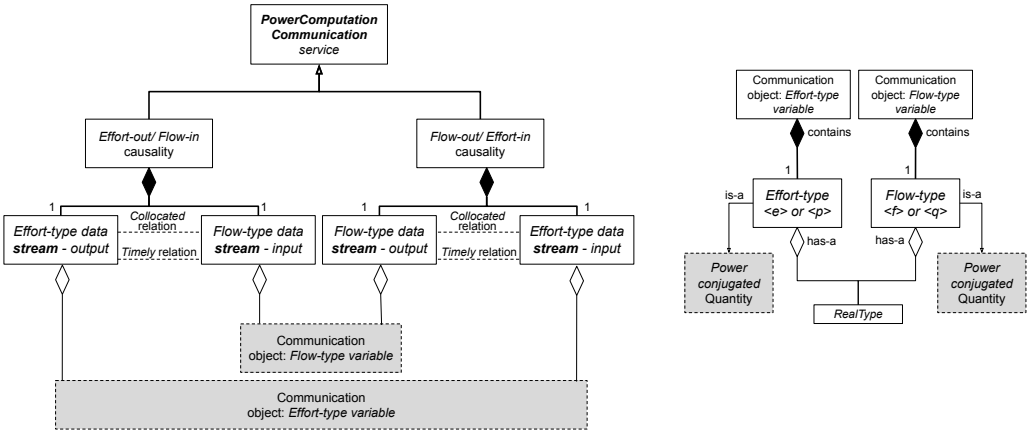

Fig. 2. Power Computation-communication service metamodel

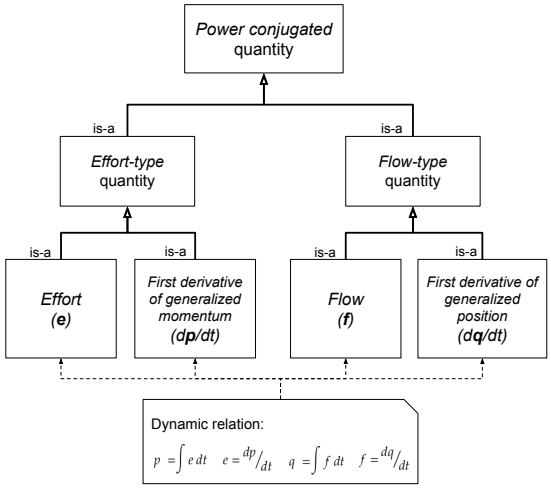

Fig. 3. Definition of Power-conjugated quantity

power-conjugated variables and state variables is convenient to avoid computing derivatives.

The physics-domain constraints are defined as follows:

Collocated-relation constraint: Any software element communicating power has a *Power Computation* communication interface composed of two ports, one containing an *effort-type* variable and the other containing a *flow-type* variable. Both variables are defined in the same reference frame.

Timely-relation constraint: Upon output data is ready for *streaming*, input data has to be received at the *collocated* input port within a firm/hard RT deadline.

Causality constraint: The communication of power among elements must be done exclusively *1-to-1*. The causal configuration of the *Power Computation* communication interfaces between two elements communicating power with each other is *always* opposite.

Fig. 4 shows the UML object-model diagram of a control-software model composed by components that associate to

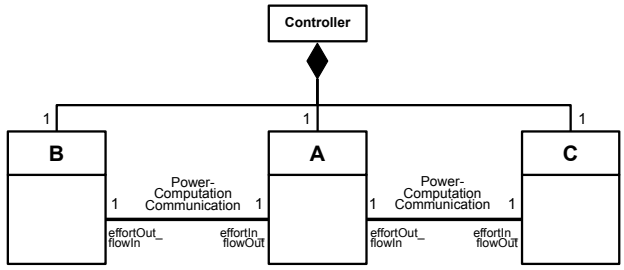

Fig. 4. UML object-model diagram of control software elements communicating power. The diagram conforms to Fig. 2.

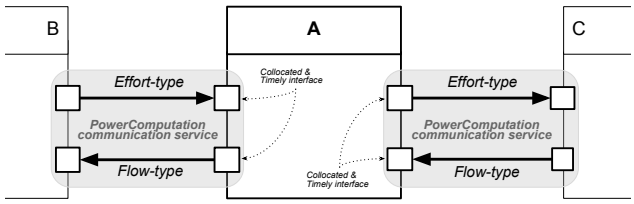

Fig. 5. Block-diagram model of control software elements communicating power via power-computation communication service. The model conforms to Fig. 4.

each other via *power computation communication* service. Fig. 5 is a signal block diagram derived from the model in Fig. 4. Component *A* shows both causal configurations, being the *effort-type* variable the input and the *flow-type* variable the output in the communication with component *B*, and being the *flow-type* variable the input and *effort-type* variable the output in the communication with component *C*. All pairs of ports – or interfaces – are *collocated*, *HRT/FRT timely-related* and *causal-constrained*. Power is computed using (1), (2) and (3) with the power-conjugated variables or state variables communicated from each *power computation communication* interface.

B. Energy Data Communication services

Having defined a communication service from which power can be transferred, energy can be computed and later communicated using the *energy state* and *energy packet* communication services. The UML diagram in Fig. 6 depicts these communication-service metamodels. Even though both services communicate energy, they have different purposes in the control software – hence, they have different behaviour and constraints.

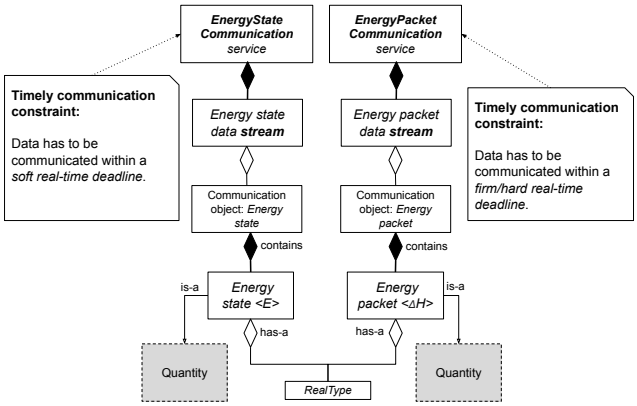

Fig. 6. Energy state communication service (left) and energy packet communication service (right) metamodels.

The *energy state* communication service communicates an amount of energy, E, as an informative value or state. This information is *streamed* to multiple elements in the control software as a system property. A timely communication constraint with a soft deadline is sufficient. This service is useful when communicating energy states across different functional levels in the software architecture – e.g., the user-interface displaying the energy used by the system.

On the other hand, the *energy-packet* communication service transfers an energy packet, ΔH, which transfers an amount of energy in the system. This energy is *streamed* to a single element in the control software to alter its energy content. Delivery guarantees are essential and a strict timeliness on its communication is recommended to avoid degradation of control performance. An example is the *energy-transfer protocol* used in *passivity layers* to balance their energy content [7], [16].

Fig. 7 shows the UML object-model diagram of a control-software model composed of components associated to each other via energy-data communication services. Fig. 8 is a signal block diagram derived from the model in Fig. 7. Component *A* communicates an energy packet, ΔH, to component *B* via *energy-packet communication* service. Component *B* communicates back an energy state value, E, to component *A* via *energy-state communication* service. The transfer of ΔH from *A* to *B* is intended to modify the energy content of both *A* and *B*. On the other hand, E being communicated from component *B* to *A* is only for informative purposes.

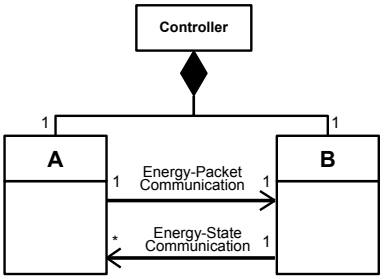

Fig. 7. UML object-model diagram of control-software elements communicating energy-data. The diagram conforms to Fig. 6.

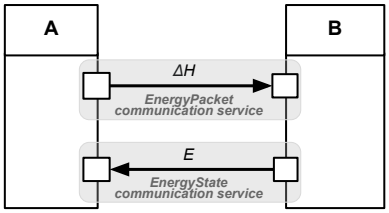

Fig. 8. Block-diagram model of control-software elements communicating energy data via energy-packet communication and energy-state communication services. The model conforms to Fig. 7.

V. Example Use Case

Fig. 9 depicts the schematic diagram of a robotic system – e.g., a robotic manipulator engaging in physical interactions. The control software must output generalised forces, F, while measuring the robot's generalised position, x, in return. Details on the composition and constraints of the controller are depicted in the UML object-model diagram in Fig. 10 and the block diagram in Fig. 11.

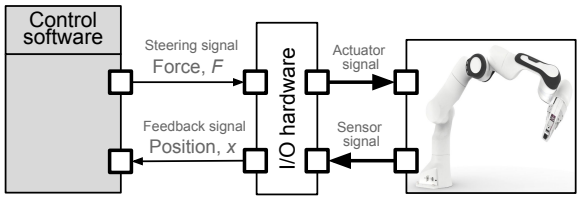

Fig. 9. Robotic system use-case example

The robot is controlled using an energy-based Cartesian loop controller that computes an output based on a Cartesian setpoint. The loop controller is organized in a stack of individual components (see for instance [16]) that exchange power through the *power-computation communication* service. The generalised forces, F, represent the effort-type variable (*F_out* and *F_in* in Fig. 11), while the generalised position, x, represent the q-type variables (*x_out* and *x_in* in Fig. 11) from which the flow-type variable (generalised velocity, v) can be computed using (2).

The loop-controller contains a Passivity-enforcing algorithm (see for instance [7], [16]) to add the right amount of damp-

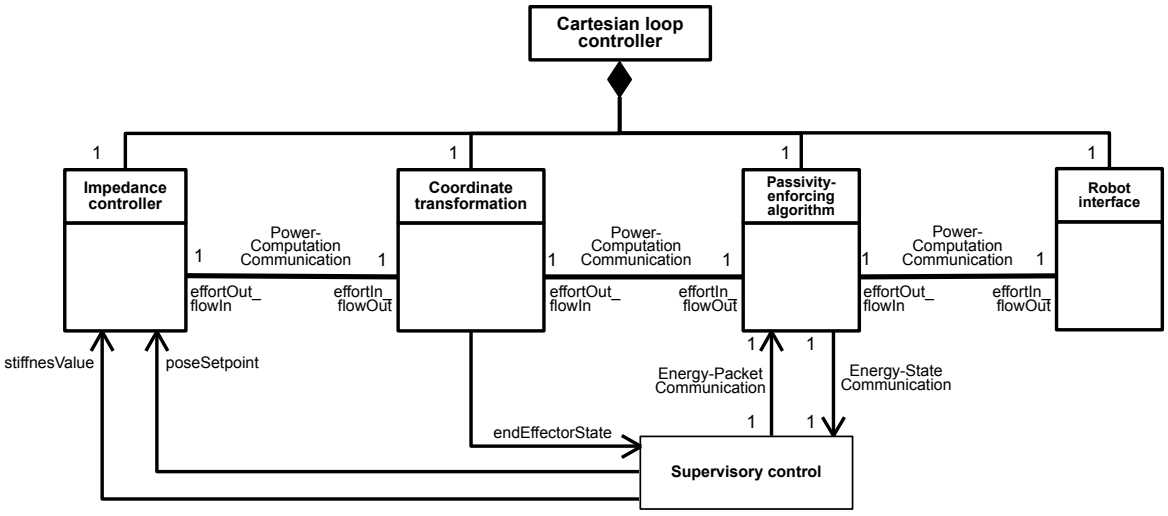

Fig. 10. UML object-model diagram of the Cartesian loop controller. The diagram conforms to Fig. 2 and Fig. 6.

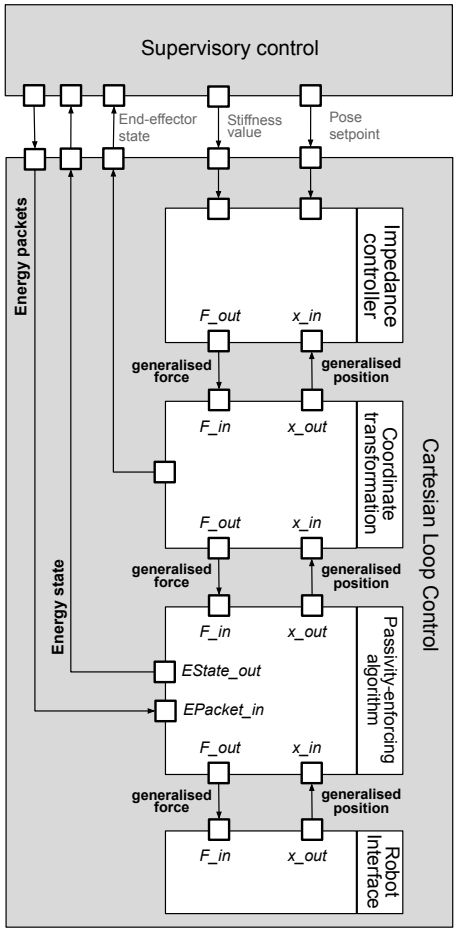

Fig. 11. Block-diagram model of the implemented Cartesian loop controller. The model conforms to Fig. 10.

ing to guarantee a passive system. This enables guaranteed stability properties in the presence of larger time delays. To achieve passivity, the loop controller receives energy packets (*EPacket_in* in Fig. 11) from a higher control level (i.e., the supervisory control in Fig. 10 and Fig. 11) using the *energy-packet communication* service. In return, the loop controller communicates its energy state (*EState_out* in Fig. 11) using the *energy-state communication* service.

The generalised force, F, and the generalised position, x, at each interface must be collocated and abide by the computational causality constraint. Moreover, F and x variables, and the energy packets require FRT/HRT guarantees, which needs to be facilitated by the deployment framework. These constraints are dictated by the power and energy communication services.

The UML object-model diagram in Fig. 10 and the block-model diagram in Fig. 11 describe two different views of the same controller. The UML object-model diagram provides software engineers a clear description of the structure and constraints of the controller. The same is the case of the block diagram for control/mechatronics/electrical engineers. Therefore, having both representations (i.e., Fig. 10 and Fig. 11) can facilitate the implementation of the controller in software.

Using this approach can bridge the gap between software engineering and the physics domain in the implementation of robotic systems. This can be achieved by having control software models conforming to the power and energy communication-service metamodels, so that models like Fig. 10 can be derived from models like Fig. 11 and the other way around.

VI. Conclusions

This paper has presented the properties of energy exchange of physical systems and translated them into domain-specific constraints for control-software models. This work resulted

in metamodels and models comprising the physics-domain constraints for the communication of power and energy data. The metamodels can be used to guide the implementation of constraints in software models and, in the future, lead to their automated construction, verification and code generation.

Constructing models according to the presented metamodels can bridge the gap between software engineering and the physics-domain in the design and structure of robot-control software. Physics-conformance of the control software can enable energy to be further used as property of control systems. Future work will focus on embedding energy in the control-software architecture for "energy-aware" robotic systems.

All-in-all, the work presented in this paper provides *freedom from choice* in the development of robot-control software by conveying the principles and approaches of the physics domain. This approach contributes to enhance functionality and dependability in robotics, and broaden the application range of robots in complex, high-demanding physical interactions.

The presented metamodels have the maturity level to assist on the development of software tools and standards. Nonetheless, work needs to be done on using the metamodels to perform automated conversion and consistency check of models. This can further facilitate *first-time-right* implementations to better fulfil the *time-to-market* requirements in the development of robot-control software.

Data Availability

Data sharing not applicable to this paper as no datasets were generated or analysed during this research.

Appendix A
Methods on Computation of Energy

The exchanged energy between two elements can be computed in terms of effort and flow (i.e., generalised force and generalised velocity) using (4). When energy is exchanged in the communication of steering and feedback control signals, the energy-flow between the control software and the physical system should be the same to preserve the energy consistency of the entire system. This is non-trivial as the sampling and hold actions during discretization can lead to an inaccurate computation of the exchanged energy (see for instance [14]).

To preserve energy consistency, [20] proposes an algorithm that computes energy exchange by sampling the generalised position, q-type variable, of the physical system instead of the flow. The exchanged energy, ΔH, between both systems is then computed using (5) for an effort-out/flow-in causal configuration of the control software. In (5), e is the output effort, q is the (sampled) generalised position, and k is a positive integer that denotes the kth sample time T. Nonetheless, a similar approach can be applied to a flow-out/effort-in causal configuration. Equation (6) denotes the computation of ΔH by sampling the generalised momentum, p-type variable, instead of the effort. In (6), f is the output flow, p is the (sampled) generalised momentum and k is a positive integer that denotes the kth sample time T.

$$\Delta H \;=\; \int P \;=\; \int e\, f\, dt \tag{4}$$

$$\Delta H \;=\; e((k-1)\,T)\,(q(k\,T) - q((k-1)\,T)) \tag{5}$$

$$\Delta H \;=\; (p(k\,T) - p((k-1)\,T))\, f((k-1)\,T) \tag{6}$$

Equations (5) and (6) are useful methods for an energy-consistent computation of power exchange. Still, the physical variables and their communication must abide by the physics-domain constraints to preserve the energy consistency of the control system.

Appendix B
Methods on Timeliness Monitoring

In computer systems, value functions can be used to describe hard and soft real-time environments [18] - i.e., the timeliness tolerance of computations and communication actions. More recent implementations of real-time systems use time-utility functions (TUF's) to measure the usefulness of computed data for the system [21]–[23]. Time-utility functions can be used as formal constraints and monitoring mechanisms for timely events in the control software.

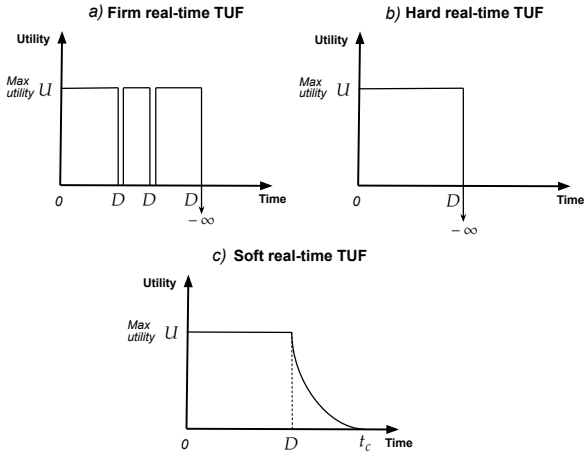

Fig. 12. Time-Utility Functions: a) Firm real-time, b) Hard real-time, and c) Soft real-time

Firm real-time (FRT) is used if missing a deadline does not cause any damage to either the system or its environment, but the output has no significant value. As described in [23], in a Firm real-time TUF (see Fig. 12a), the usefulness, U, of the data drops immediately from maximal to zero once the deadline, D, is due. Only a k number of missed deadlines in a certain period of time, t, is acceptable. If the number of missed deadlines is greater than k in the time period t, then the utility value drops to $-\infty$ as this might cause serious consequences. The parameters k and t are determined by the application.

Hard real-time (HRT) is used if missing its deadline may cause catastrophic consequences on the controlled system or its environment. In a Hard real-time TUF (see Fig. 12b), the usefulness, U, of the data drops immediately from maximal to $-\infty$ once the deadline, D, is due.

A more tolerant measures of usefulness can be applied to the computation and communication of data which variation

in timeliness does not (critically) impact the performance of the system. Soft real-time (SRT) is used if missing its deadline has still some utility for the system, although causing a performance degradation. In a soft real-time TUF (see Fig. 12c), the usefulness is degraded "softly" from maximal to zero in a time window determined by the deadline, D, and a critical time t_c.

ACKNOWLEDGMENTS

We want to thank Kees van Teeffelen, Julio de Oliveira, Bart Driessen (all from TNO, NL), Theo de Vries (VIRO, NL) and Herman Bruyninckx (Eindhoven University of Technology, NL, and KU Leuven, BE) for their contributions to the EGCS project.

REFERENCES

[1] R. Ortega and M. Spong, "Adaptive motion control of rigid robots: A tutorial," *Automatica*, vol. 25, no. 6, pp. 877–888, Nov. 1989.

[2] J. E. Colgate and N. Hogan, "Robust control of dynamically interacting systems," *International Journal of Control*, Oct. 2007.

[3] A. Albu-Schäffer, C. Ott, and G. Hirzinger, "A Unified Passivity-based Control Framework for Position, Torque and Impedance Control of Flexible Joint Robots," *The International Journal of Robotics Research*, vol. 26, no. 1, pp. 23–39, Jan. 2007.

[4] R. Ortega, A. van der Schaft, I. M. Y. Mareels, and B. M. Maschke, "Energy shaping revisited," *Proc. IEEE Conf. Control Appl.*, Feb. 2000.

[5] C. Melchiorri, S. Stramigoli, and S. Andreotti, "Using damping injection and passivity in robotic manipulation," in *1999 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (Cat. No.99TH8399)*, Sep. 1999, pp. 979–984.

[6] B. Hannaford and J.-H. Ryu, "Time-domain passivity control of haptic interfaces," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 1, pp. 1–10, Feb. 2002.

[7] M. Franken, S. Stramigioli, S. Misra, C. Secchi, and A. Macchelli, "Bilateral Telemanipulation With Time Delays: A Two-Layer Approach Combining Passivity and Transparency," *IEEE Transactions on Robotics*, vol. 27, no. 4, pp. 741–756, Aug. 2011.

[8] G. A. Folkertsma and S. Stramigioli, "Energy in Robotics," *Foundations and Trends® in Robotics*, vol. 6, no. 3, pp. 140–210, Oct. 2017.

[9] W. R. Ferrell, "Delayed Force Feedback," *Human Factors*, vol. 8, no. 5, pp. 449–455, Oct. 1966.

[10] R. J. Anderson and M. W. Spong, "Bilateral control of teleoperators with time delay," *IEEE Transactions on Automatic Control*, vol. 34, no. 5, pp. 494–501, May 1989.

[11] Breunese and J. F. Broenink, "Modeling Mechatronic Systems Using The Sidops+ Language," in *The Society for Computer Simulation International*, 1997, pp. 301–306.

[12] J. F. Broenink, "Bond Graphs: A Unifying Framework for Modelling of Physical Systems," in *Foundations of Multi-Paradigm Modelling for Cyber-Physical Systems*, P. Carreira, V. Amaral, and H. Vangheluwe, Eds. Cham: Springer International Publishing, 2020, pp. 15–44.

[13] R. Cobos Méndez, J. de Oliveira Filho, D. Dresscher, and J. Broenink, "A Bond-Graph Metamodel:," in *Formal Aspects of Component Software*, ser. Lecture Notes in Computer Science, F. Arbab and S.-S. Jongmans, Eds. Cham: Springer International Publishing, 2020, pp. 87–105.

[14] J. E. Colgate, P. E. Grafing, M. C. Stanley, and G. Schenkel, "Implementation of stiff virtual walls in force-reflecting interfaces," in *Proceedings of IEEE Virtual Reality Annual International Symposium*, Sep. 1993, pp. 202–208.

[15] J. E. Colgate and G. G. Schenkel, "Passivity of a class of sampled-data systems: Application to haptic interfaces," *Journal of Robotic Systems*, vol. 14, no. 1, pp. 37–47, 1997.

[16] Y. Brodskiy, "Robust autonomy for interactive robots," Feb. 2014.

[17] J. F. Broenink, "Bond-graph modeling in Modelica," *Simulation in Industry, 9th European Simulation Symposium 1997, ESS'97*, pp. 137–141, Oct. 1997.

[18] E. D. Jensen, C. D. Locke, and H. Tokuda, "A Time-Driven Scheduling Model for Real-Time Operating Systems," *Unknown Host Publication Title*, pp. 112–122, Dec. 1985.

[19] H. Bruyninckx, E. Scioni, N. Hubel, and F. Reniers, *Composable Control Stacks in Component-Based Cyber-Physical System Platforms*, Apr. 2018.

[20] S. Stramigioli, C. Secchi, A. van der Schaft, and C. Fantuzzi, "Sampled Data Systems Passivity and Discrete Port-Hamiltonian Systems," *IEEE transactions on robotics and automation*, vol. 21, no. 4, pp. 574–587, 2005.

[21] E. D. Jensen, "Asynchronous Decentralized Realtime Computer Systems," in *Real Time Computing*, ser. NATO ASI Series, W. A. Halang and A. D. Stoyenko, Eds. Berlin, Heidelberg: Springer, 1994, pp. 347–371.

[22] G. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, 3rd ed., ser. Real-Time Systems Series. Springer US, 2011.

[23] A. H. Boode, "On the automation of periodic hard real-time processes: A graph-theoretical approach," Jun. 2018.