Ontological Reasoning as an Enabler of Contract-Based Co-Design

Ken Vanherpen[1,3], Joachim Denil[1,2,3], Paul De Meulenaere[1,3], and Hans Vangheluwe[2,3,4]

[1] CoSys-Lab (FTI), University of Antwerp, Antwerp, Belgium
[2] AnSyMo (FWET), University of Antwerp, Antwerp, Belgium
[3] Flanders Make vzw, Belgium
[4] MSDL, McGill University, Montréal, Québec, Canada
{ken.vanherpen,joachim.denil,paul.demeulenaere}@uantwerp.be
hv@cs.mcgill.ca

Abstract. Because of the combination of computational, networking and physical artifacts, different engineering disciplines are involved in the design of a Cyber-Physical System (CPS). This multidisciplinary approach leads to different, often contradicting, views on the system under design which in the end might lead to inconsistencies between domain specific properties. Contract-Based Design (CBD) aims to prevent these contradictions by defining possible conflicting properties in a contract. These contracts consist of a set of pre- and postconditions.
Although the current state-of-the-art describes the abstraction/refinement, composition and multi-view analysis and verification principles of CBD, it lacks methods and techniques to identify the shared properties in concurrent design processes. By combining the theory of CBD with the principles of ontological reasoning, this paper intents to provide a framework which enables Contract-Based Co-Design (CBCD). The feasibility of this framework will be explained by means of a running CPS example.

Keywords: Co-Design · Contract-Based Design · Cyber-Physical Systems · Ontological Reasoning · Ontologies

1 Introduction

Increasingly more, Cyber-Physical Systems (CPS) [1, 2] take a prominent role in a wide range of application areas such as transportation, manufacturing, health care, etc. They extend traditional mechanical systems with computational and networking capabilities making (daily life) products smarter, faster, more accurate, remotely controllable, and so forth. Therefore, CPS are considered as one of the key enablers of the fourth industrial revolution.

Despite the extended capabilities of CPS, its development process is characterized by costly, iterative, design cycles partly due to the involvement of various engineering disciplines, each with a different view and set of concerns of the system under design [3]. The involvement of these different stakeholders can lead to inconsistencies between shared properties, causing unexpected behaviors during the integration of the different design artifacts. To preserve consistency between those different views, Contract-Based Design (CBD) [4–6] is increasingly being used by system engineers to formalize an agreement between two or more engineering domains. Originating from contracts used in software engineering, such an agreement consists out of a set of assumptions and guarantees. These assumptions and guarantees describe the conditions under which a system promises to operate while satisfying desired properties.

Given the increasing complexity of Cyber-Physical Systems, aggravated by the need for cost-efficient products and shorter development time, the need for concurrent design (co-design) processes arises. Concurrent design makes engineers reason about common design properties to allow the independent development of parts of the system. In that sense, Contract-Based Design seems to be a useful methodology. Different contributions have been made elaborating on abstraction/refinement, verification and validation of contracts (see Section 2).

However, the current state-of-the-art does not allow the engineers to reason about the content of such a contract. It thus lacks in its applicability to the co-design of Cyber-Physical Systems. This paper intents to provide a framework which enables Contract-Based Co-Design (CBCD) by combining the current state-of-the-art of CBD with the principles of ontological reasoning [7]. The latter enables one to make the implicit knowledge of each engineer explicit by using ontological properties and certain influence relationships between them.

The rest of this paper is structured as follows. Section 2 gives an overview of the related work. The running CPS example is introduced in Section 3, while an overview of the currently used contract operators is given in Section 4. Similar to the proposed methodology in the current state-of-the-art, Section 5 investigates the applicability of the current theory in a co-design engineering process. However, some shortcomings will emerge which are resolved by our proposed CBCD methodology in Section 6. Finally, Section 7 concludes our contribution and gives an overview of our future work.

2 Related Work

Contract-Based Design finds its origin in the late 80's when Bertrand Meyer introduced the Eiffel programming language to enable contract-based software development [8, 9]. Eiffel introduces *Require* and *Ensure* clauses that correspond to respectively a set of pre- and post-conditions under which a software routine ensures to operates.

More than a decade later, the use of contracts during the design of CPS came to the attention of some researchers, including Damm [10, 11]. He introduced the concept of 'rich components' to deal with uncertainty when designing Cyber-Physical Systems. Rich components extend model components such that: (a) they cover all the specifications of the involved viewpoints, (b) they contain a set of assumptions and guarantees with respect to the context of the component, and (c) they provide classifiers to the assumptions.

In the framework of the European project SPEEDS[5], the work of Damm was extended by Josko et al. [12] and Benvenuti et al. [13] by means of 'Heterogeneous Rich Component' (HRC) which supports the integration of heterogeneous viewpoints on a system with different semantics originating from multiple design layers and tools. Therefore, a common meta-model was developed in [14]. Similar but less comprehensive approaches, however, were already introduced by the MARTE UML profile [15] and as a modelling framework called Metropolis [16]. The scope of the SPEEDS project resulted in the (first) use of contracts in a component based engineering context. In [17], Benveniste et al. present the mathematical foundations of CBD to enable the combination of contracts for different model components and the combination of contracts for different viewpoints on the same model component. According to the authors, a contract as such consists out of a pair of *Assumptions* and *Guarantees* formulated as $C = (A,G)$. Note that this relates to the *Require* and *Ensure* clauses introduced by Meyer [9].

In the scope of the European project CESAR[6], Benveniste et al. extended their theory and showed how contracts might be used through multiple application cases [4, 18]. They show that there exist three fundamental contract operators to combine contracts: refinement, composition and conjunction [4, 19].

Based on the work of of Benveniste et al., Graf et al. describe how circular and non-circular assume-guarantee reasoning can be used in order to check for contract dominance [20]. They make use of two frameworks, L0 and L1, which are focused on component refinement and component interactions respectively.

Sangiovanni-Vincentelli et al. address the emergent need of CBD in the context of system level design [6] . They present a design methodology that combines the concepts of CBD with Platform-Based Design (PBD) as a meet-in-the-middle approach. Related to the work of Graf et al. [20], Sangiovanni-Vincentelli et al. demonstrate how contracts may be dominated when combining subsystems (individually bounded by a contract). Furthermore, a clear distinction is made between horizontal and vertical contracts when combining the concepts of CBD with PBD. Similarly, Nuzzo et al. elaborate on the usefulness of CBD, and their formal analysis and verification methods, in a PBD methodology for Cyber-Physical Systems [21, 22]. Besides going into detail on the different methods and tools that are used to enable their methodology, an aircraft electric power distribution system is used as a demonstrator.

In [5], a more general framework of design contracts in the context of CPS design is given. Derler et al. focus on timing properties to facilitate the communication between control and embedded engineers. A non-exhaustive enumeration of contract types is given each with a specific set of parameters having a common interest to both engineering domains. Depending on the type of contract (and therefore the formalized set of parameters), an actual implementation of the contract is feasible for one or both of the engineering domains.

[5] www.speeds.eu.com

[6] http://www.cesarproject.eu

Törngren et al. describe the different viewpoints involved in the design of a mechatronic system [3]. Furthermore, they show how these viewpoints are interrelated by means of supporting models at different design levels, namely: (a) people level, (b) models level and (c) tools level. At each design level, some challenges and solutions (supporting models) are described. For the contributions of our work, the first two levels are particularly interesting. At people level, the authors point out that each stakeholder, involved in the design of a CPS, should be aware of the effect of his/her work on others. To enable this, the use of design contracts, as suggested by Derler et al. [5], is proposed. Moreover, they hint towards the use of assumptions and guarantees as discussed by [6]. Additionally, at models level, Törngren et al. describe the existence of dependencies between models implementing certain parts of the overall system requirements.

We conclude this section with the work of Persson et al. where the authors characterize model-based approaches used in the design of Cyber-Physical Systems [23]. To do so, a clear distinction is made between *views* and *viewpoints*. The former relates to the multitude of abstractions that can be made of a system while the latter refers to a set of all possible view instances. The authors show that there exist relations between views, and as such viewpoints, with respect to their content, process and operations which are not entirely exclusive to each other. This is illustrated by an academic case study of a wind-shield wiper system.

3 The Power Window as a Running Example

To clarify the current state-of-the-art in Section 5 and to detail our contribution in Section 6, we use the power window as a running example.

As every system, the power window is specified by a set of requirements. These requirements describe the expected behavior of the system given a certain context. Given that the power window system operates in a vehicle, we describe the most elementary behavior of the power window as follows [24]:

1. The power window should start moving within 200 ms after a command is issued.
2. The power window shall be fully opened or closed within 4.5 s.
3. When closing the power window, a force of no more than 100 N may be present.
4. Detection of a clamped object when closing the window should lower the window by 10 cm.

Given these requirements, the power window system can be seen as a black box controller with three inputs and two outputs as illustrated in Figure 1.

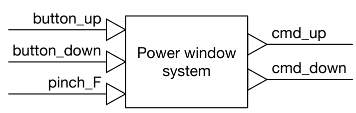

Fig. 1. Representation of the power window system

Using the definition of contracts for system design from [4] and [19], the set of requirements are formalized as a system contract, as shown in Table 1. The contract specifies certain assumptions on the context/environment the power window operates in, namely: (a) the input force is lower than 1000 N and (b) the minimum interval of button operations is 100 ms. Under these conditions, a safe operation of the system is guaranteed. It might be clear that the requirements of the system are the guarantees of the system. However, as one may notice in Table 1, certain functional requirements are refined given domain-specific knowledge. For example, requirement *3* and *4* are further refined in the spatial and temporal dimension to detail the safety requirement:

1. Spatial dimension: if a clamped object is detected, the power window may continue to close for a maximum of 0.2 mm before life threatening injuries occur.
2. Temporal dimension: given the spatial dimensions, safety can be guaranteed if the window lowers within 1 ms.

This refinement, that is made after discussions with experts and looking into regulations, results in the fifth guarantee of Table 1.

Table 1. Power window system contract C_{SYS}

Assumptions	*pinch_F* will be lower than 1000 N.
	button_up occurs sporadic with a minimum period of 100 ms.
	button_down occurs sporadic with a minimum period of 100 ms.
Guarantees	Delay between *button_up* and *cmd_up* within [0 ms, 200 ms].
	Delay between *button_down* and *cmd_down* within [0 ms, 200 ms].
	Maximum activation time *cmd_up* within [0 ms, 4.5 s].
	Maximum activation time *cmd_down* within [0 ms, 4.5 s].
	If *pinch_F* exceeds 100 N, delay between *pinch_F* and *cmd_down* within [0 ms, 1 ms].
	If *pinch_F* exceeds 100 N, activation time *cmd_down* within [0 s, 0.43 s].

4 Overview of the State-of-the-Art Contract Operators

In section 2 it is shown that a lot of contributions in the field of Contract-Based Design for Cyber-Physical Systems have been done in the context of the SPEEDS and CESAR projects. Therefore, this section gives a short overview of the currently used contract operators. Section 5 uses these operators to check their feasibility in a co-design engineering process.

Decomposition of a system

Concurrent engineering (co-design) can be realized by decomposing the system into components that are designed (semi-)independently of each other. From the perspective of a CPS, one can distinguish three independent components: (a) a hardware component, i.e. one or more embedded platforms which are connected to each other, (b) a control component and (c) a mechanical component. Each component is typed by a set of in- and outputs, a set of behaviors and a set of extra-functional properties like performance, timing, energy, safety, etc. Figure 2 shows the *decomposition* of the power window system (Figure 1) into its control and hardware component. Note that we neglected the mechanical component for the sake of clarity. As can be seen, components can be further *refined* and hierarchically structured to represent different levels of *abstraction*. They can be connected to each other by sharing certain ports and variables.

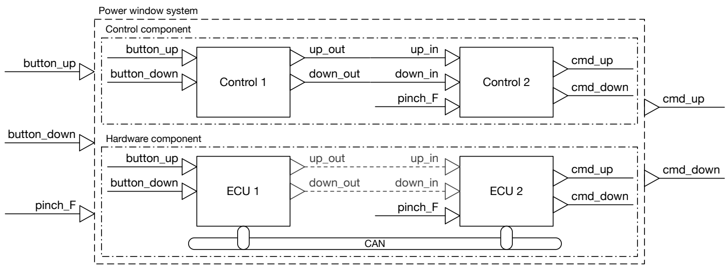

Fig. 2. Refinement of the power window system

The decomposition of the system results in a decomposition of the system contract as well. Indeed, each (sub-)component is typed by an individual contract that is derived from the system contract. By using different operators, the component contracts are merged and should satisfy or refine the system contract.

Contract operators

Because a contract is a set of assumptions and guarantees, set theory is used to merge component contracts. Three basic operators are defined in literature [19]: abstraction/refinement $\preceq$, conjunction $\wedge$ and composition $\otimes$. Before applying the current CBD theory to our example, we briefly discuss these basic operators.

Abstraction/Refinement As already stated, components might be hierarchical structured and as such, a component its contract might be further refined. Let $C^{'} = (A^{'}, G^{'})$ and $C = (A, G)$ be two contracts consisting out of a set of assumptions and guarantees. The refinement $C^{'} \preceq C$ holds if and only if:

$$\begin{aligned} A^{'} &\supseteq A \\ G^{'} &\subseteq G \end{aligned} \tag{1}$$

Given this constraint, it is clear that a refined contract should weaken the assumptions and strengthen the guarantees. Therefore, we say that any implementation M of contract $C^{'}$ is an implementation of C as well, or more formally:

$$\text{If } M \models C^{'} \text{ and } C^{'} \preceq C \text{ then } M \models C \tag{2}$$

A similar reasoning can be obtained for the environment E of both contracts:

$$\text{If } E \models C \text{ and } C^{'} \preceq C \text{ then } E \models C^{'} \tag{3}$$

Conjunction The conjunction operators enables one the merge different viewpoint contracts associated to one single component. In the example of Figure 2, the component 'Control 1' might be typed by a behavioral and a safety viewpoint contract. Let $C_1 = (A_1, G_1)$ and $C_2 = (A_2, G_2)$ be two viewpoint contracts consisting out of a set of assumptions and guarantees. The conjunction $C_1 \wedge C_2$ can then be obtained as follows:

$$\begin{aligned} A &= (A_1 \cup A_2) \\ G &= (G_1 \cap G_2) \end{aligned} \tag{4}$$

Similar to the abstraction/refinement operator, the conjunction operator weakens the assumptions and strengthens the guarantees.

Composition The composition operator enables one to merge the contracts associated to different components. In the example of Figure 2, the contracts related to the sub-components 'Control 1' and 'Control 2' can be composed to compute the 'Control' contract. Let $C_1 = (A_1, G_1)$ and $C_2 = (A_2, G_2)$ be two components contracts consisting out of a set of assumptions and guarantees. The composition $C_1 \otimes C_2$ can then be obtained as follows:

$$\begin{aligned} A &= (A_1 \cap A_2) \cup \neg(G_1 \cap G_2) \\ G &= (G_1 \cap G_2) \end{aligned} \tag{5}$$

In the case of composition, both assumptions and guarantees are strengthened.

5 Applicability of the Current Methodologies on a Co-Design Engineering Problem

To the best of our knowledge, the current CBD theory has never been applied in a co-design engineering process. On the contrary, the examples shown in [4, 21, 22] are sequential engineering processes. Therefore, this section analyzes the feasibility of the current state-of-the-art/state-of-the-practice and identifies possible shortcomings using the power window example of Section 3.

Control component

As can be seen in Figure 2, the control component is decomposed into two control components. One component takes care of the user operations (button up and button down) and as such implements guarantee 1 and 2 of the system contract of Table 1. The other control component implements the main control loop which takes care of the remaining guarantees.

The composition of the two refined control components refines (the control view of) the system contract of Table 1 and will, using equation 1, strengthen (some) guarantees and weaken (some) assumptions. As an example, Table 2 shows a fragment of the contract of component 'Control 1' which is obtained by the conjunction of its functional and timing contract and the composition with the signal contract. The later one specifies a contract on the signals between 'Control 1' and 'Control 2'. The refined contract strengthens the guarantees of the system contract. In 'control 1' the delay between the component its input and its output (in fact, the input of the other component) is lowered from 200 ms to 52 ms. Together with the contract of 'Control 2', the total time is less than 200 ms. Furthermore, equation 3 holds because the environment of the composed refined components will be one for the system as well. Note that the actual refinement of the system contract is the conjunction of the composition of the hardware components and the composition of the control components.

Table 2. Fragment of contract C_{C1} for 'Control 1'

Assumptions	*button_up* occurs sporadic with a minimum period of 50 ms. *up_out* occurs sporadic with a minimum period of 2 ms.
Guarantees	Delay between *button_up* and *up_in* within [0 ms, 52 ms].

If we take a closer look at the content of the contract in Table 2, we may wonder to what extent a control engineer is able to guarantee these timing delays. Although a control engineer has several degrees of freedom (e.g. the order of the control algorithm) to influence the computational expensiveness of a algorithm, these timings highly depend on the hardware platform and thus on the hardware component. From our experience with industry, we know that control engineers have limited aids in estimating hardware properties and as such are not able to guarantee these delays once the control algorithm is deployed.

Hardware component

A similar conclusion can be made when looking to the contract for the hardware component, and in particular for the contract of 'ECU 1' (see fragment in Table 3) which implements the algorithm of 'Control 1'. Note that the composition of this contract with the 'ECU 2' and 'CAN' contract is again a refinement of the (hardware view of the) system contract, which one can verify using equations 1 - 3.

At a first glance, the contract contains everything an embedded engineer is able to reason about: timing period of a runnable, Worst Case Execution Time (WCET) and Worst Case Response Time (WRT). However, their position in the table is questionable. To be more specific, both parts of the table reason about these properties, while they should be clear guarantees of the platform.

Table 3. Fragment of contract C_{E1} for 'ECU 1'

Assumptions	*button_up* occurs sporadic with a minimum period of 40 ms.
	Runnable#actuation occurs each 40 ms.
	Delay between *Runnable#actuation* and *up_out* within [0 ms, 10 ms].
Guarantees	Timer occurs each 10 ms.
	Delay between *button_up* and *up_out* within [200 us, 10 ms + 1.3 ms].

Shortcomings

From this example, we conclude that the current state-of-the-art does not support a co-design process because the individual contracts: (a) contain properties on which the domain engineer lacks the ability to reason about and/or (b) make no clear separation between what is assumed from the other domain and what should be guaranteed under these conditions.

6 A Contract-Based Co-Design Methodology

To overcome the aforementioned shortcomings, a clear negotiation phase is included in the proposed engineering process. This results in a so called mapping contract. Based on this overall contract, the different domain-specific contracts are derived and further refined. This process for deriving domain-specific contracts from a negotiated contract was already suggested by Derler et al. in [5]. However, a clear methodology was not proposed. Therefore, we suggest to use domain ontologies to support this Contract-Based Design process.

In its essence, an ontology is typed by a set of ontological properties and certain influence relationships which exists between those properties. Each ontological property classifies a certain part of the real world.

In the context of Cyber-Physical Systems, ontologies are ideal to make the implicit knowledge of each domain engineer explicit. Based on our earlier work [7], Figure 3 shows a formal representation of ontological reasoning in a CPS design context. Given a set of requirements, which describes the real-world system for a certain context, the engineer reasons about certain domain properties (which might be related to each other). The solid oval in the *Ontological World* denotes the set of ontological properties covered by the requirements.

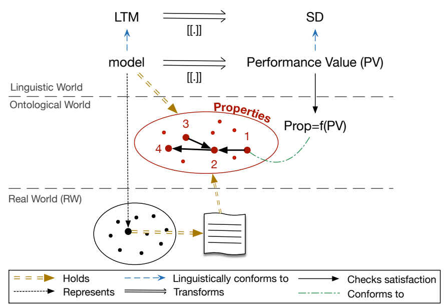

Fig. 3. Ontological Reasoning

As a first step in the design process, the engineer abstracts the real-world system by means of a model. This model is typed by a meta-model, called a

Linguistic Type Model (*LTM*). A linguistic conformance relationship exists between the model and the LTM. By transforming the model to a *Semantic Domain* (*SD*), using a semantic mapping function ($[[.]]$), a meaning is given to the model. This allows for analysis of linguistic properties which are called *Performance Values* (PV). The engineer evaluates these performance values to his own implicit knowledge of the system. This allows the engineer to conclude whether the system is conforming to the requirements or not.

By making this knowledge explicit using an ontology, a function returning a logical value can be used to evaluate these performance values against a certain ontological property. As discussed in [7], ontological reasoning enables us to reason about consistency between performance values related to different ontological properties. These properties in turn are connected by means of influence relationships.

We argue that ontological reasoning enables contract-based co-design of a CPS. Given the requirements of the system, an ontology of the overall system is created. The overall ontology is complemented with domain-specific ontologies for each engineering domain. The system ontology is linked to the domain-specific ontologies by means of influence relationships enabling us to reason about relationships between system and domain-specific properties and thus also about contracts. For example, Figure 4 shows the ontology (right side) and the different contracts (left side) of the power window system. We can clearly distinguish three areas: (a) a control area in the upper part, (b) a mapping area in the middle and (c) a hardware area in the lower part. The following subsections discuss these areas in more detail and how there are related to each other in a Contract-Based Design process consisting out of three phases: (a) negotiation, (b) deriving the domain contracts and (c) refinement of the domain contracts.

Phase 1 - Negotiation

A co-design engineering process, supported by Contract-Based Design, starts with a negotiation phase where the involved engineering domains discuss the system properties which need to hold. Therefore, each engineer represents the architecture of its domain given the system requirements. For example, the control engineer reasons about: (a) the amount of software components, (b) their in- and outputs, (c) connections between components, etc. On the other hand, the hardware engineer responsible for the hardware part of the system reasons about: (a) the number of Electronic Control Units (ECUs), (b) their processor, (c) communication between the ECUs, etc. These architectural parameters can also be ranged values. An example of such an architecture is shown in Figure 2 which, indeed, is a refinement of the system.

Given the architecture of the involved domains and the system requirements (e.g. Table 1), the engineers decide how these architectures are related to each other. For example, when focusing on control and hardware components, they decide how the control algorithm is mapped to the hardware. In the case of the power window, they decide on a one-to-one mapping between a control and ECU component. Based on this mapping, a mapping contract, as shown on the left

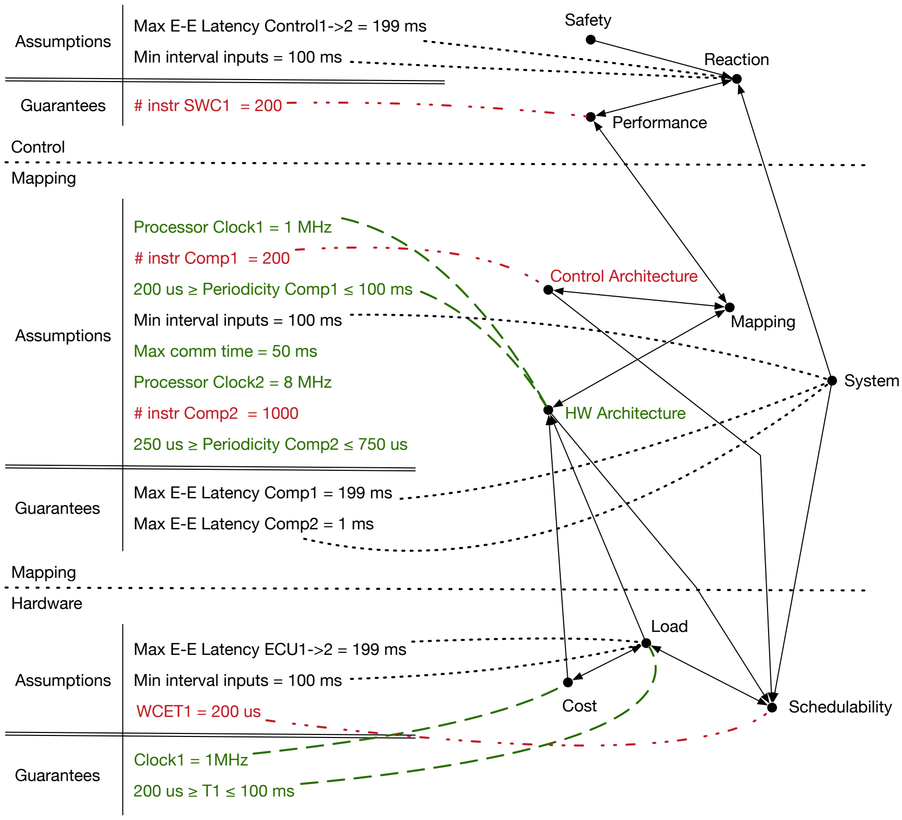

Fig. 4. Fragment of the mapping contract and the derived engineering contracts for the power window example

side of Figure 4, is defined that consists out of a set of assumptions and guarantees. Properties related to the architectures are best guesses. Therefore, they are assumptions of the mapping contract. Examples of such estimated properties are: clock speed, number of instructions, periodicity, minimum interval times of the inputs, maximum communication time between ECUs, etc.

Keeping in mind the defined architectures, the given system requirements are translated to system properties as well. For example, one requirement of the power window example states that '*the power window should start moving within 200 ms after a command is issued*'. This maximum latency is refined into two guarantees of the mapping contract: (a) a maximum latency of 199 ms for mapping component 1 and (b) a maximum latency of 1 ms for mapping component 2. A mapping component refers to the one-to-one mapping of a control to an ECU component. It might be clear that the system requirements, such as these latencies, are considered as guarantees of the mapping contract. As we notice, the mapping contract as shown in Figure 4 is a refinement of the system contract shown in Table 1. As a result, equations 1 - 3 are valid.

Phase 2 - Deriving the domain contracts

In the second phase of the process, the elements of the mapping contract are subdivided into three categories using the ontology shown on the right of Figure 4: (a) Control architecture, (b) Hardware architecture and (c) System. Based on this categorization it is decided if a contract element should be an assumption or a guarantee of the domain contract. Moreover, due to relations there exist between the ontological properties, it is decided whether a certain element is relevant for the domain contract and how it should be translated. The decision whether a contract element is translated to the domain contract is relevant when one wants to focus on one particular (extra-)functional requirement (e.g. timing, safety, etc.).

Contract elements which are related to a certain architecture become part of the guarantees of the domain contract related to that architecture. Given the mapping contract in Figure 4, for example, the element *Processor Clock* and *Periodicity of Component 1* are translated as guaranteed elements of the hardware contract as these are design decisions the hardware engineer should take care of. Likewise, the element *Number of instructions for Component 1* is translated as a guaranteed element of the control contract. Indeed, the control engineer is responsible for maintaining this limited amount of instructions which can be influenced by the order of the control algorithm.

Contract elements which are related to the system requirement, i.e. which are part of the system contract or which are a refinement of them, are translated as assumed elements of all the involved domain contracts. Based on these assumptions, domain engineers are able to make domain specific decisions in phase 3 of the design process. Those decisions are again the guarantees of their domain contracts.

Note that every element of the mapping contract is translated to at least one domain contract over the ontological relations such that completeness is guaranteed.

Phase 3 - Refinement of the domain contracts

As a final phase of the co-design engineering process, the domain engineers extend and refine their own contracts, keeping in mind equations 1 - 3, as shown in Figure 5. For example, the hardware engineer might decide to strengthen the periodicity of component 1, i.e. increase the periodicity from 100 ms to 50 ms. He is allowed to refine this contract element since it is a design parameter he has to guarantee. However, the refinement has to be taken under the given assumptions which might be relaxed (e.g. decreasing the *maximum end-to-end latency*).

Once a contract element is refined in one domain, the changes must be pushed to related contract elements which are part of the other domain contracts. This is made possible because every contract element is linked to an ontological property

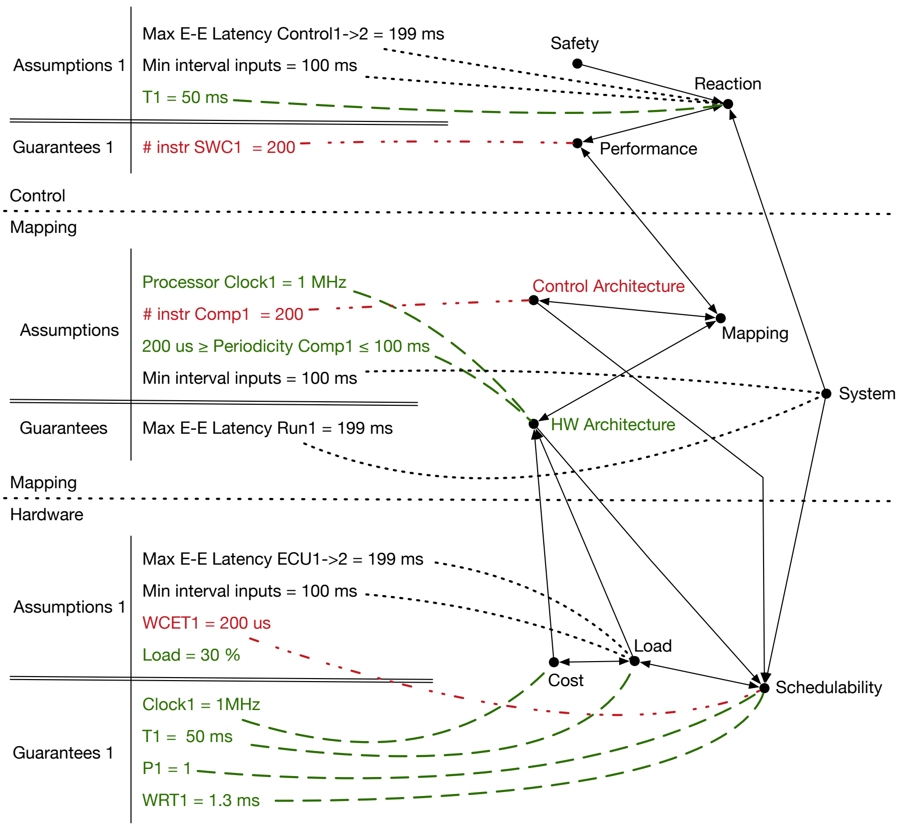

Fig. 5. Fragment of the refined engineering contracts for the power window example

which in turn are related to each other by means of influence relationships. For example, the refinement of the periodicity in the hardware contract results in an update of the assumed periodicity in the control contract via the ontological properties: Load → HW Architecture → Mapping → Performance → Reaction.

7 Conclusions & Future Work

The application of contract-based design in a concurrent engineering setting with multi-disciplinary teams is not well supported. Contracts contain elements that might be irrelevant for the engineer. Furthermore, there is no clear distinction between what is assumed from other domains and what is guaranteed under these conditions.

By combining the theory of CBD with the principles of ontological reasoning, we propose a three phased process that starts with a negotiation phase. A negotiation allows engineers to discuss a common mapping contract. Using an ontology, elements of the mapping contract are translated to domain-specific contract elements and, depending on the engineering, are defined in the assumption or guarantee part of the domain contract. By definition, our methodology ensures that what is assumed in one domain will be guaranteed by another domain. Furthermore, using ontological reasoning our methodology ensures consistency between contracts and as such keeps them synchronized at all times.

It might be clear that the applicability in an industrial context is only feasible when our methodology is supported by a user-friendly tool. Given an ontology, build by a system engineer, and the negotiated mapping contract we believe the supported tool should hide phase 2 and 3 of our proposed methodology allowing engineers to focus on their core business (i.e. designing the system). Providing this tool support is considered as future work. Once available, it will allow us to increase the complexity of the use case and investigate the feasibility of our methodology on models used in industry. Besides providing tool support, we are planning to verify the compatibility of our proposed design methodology with the current state-of-the-art contract operators. We believe an extension of the current contract operators is needed to support our vision of a mapping operator which assures that all the information is put forward to the (derived) domain contracts.

Acknowledgments This work has been carried out within the MBSE4Mechatronics project (grant nr. 130013) of the Flanders Innovation & Entrepreneurship agency (VLAIO). This research was partially supported by Flanders Make vzw.

References

1. P. Derler, E. Lee, and A. Sangiovanni-Vincentelli, "Modeling Cyber-Physical Systems," *Proceedings of the IEEE*, vol. 100, no. 1, pp. 13–28, 2012.
2. E. Lee, "Cyber Physical Systems: Design Challenges," in *ISORC '08*, pp. 363–369, 2008.
3. M. Törngren, A. Qamar, M. Biehl, F. Loiret, and J. El-khoury, "Integrating viewpoints in the development of mechatronic products," *Mechatronics*, vol. 24, no. 7, pp. 745–762, 2014.
4. A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. Henzinger, and K. G. Larsen, "Contracts for Systems Design," Tech. Rep. RR-8147, INRIA, 2012.
5. P. Derler, E. A. Lee, S. Tripakis, and M. Törngren, "Cyber-Physical System Design Contracts," in *ICCPS '13*, pp. 109–118, ACM, 2013.
6. A. Sangiovanni-Vincentelli, W. Damm, and R. Passerone, "Taming Dr. Frankenstein: Contract-Based Design for Cyber-Physical Systems," *European Journal of Control*, vol. 18, no. 3, pp. 217–238, 2012.
7. K. Vanherpen, J. Denil, I. Dávid, P. De Meulenaere, P. J. Mosterman, M. Törngren, A. Qamar, and H. Vangheluwe, "Ontological reasoning for consistency in the design of cyber-physical systems," in *CPPS'16*, pp. 1–8, 2016.
8. B. Meyer, "Eiffel: A language and Environment for Software Engineering," *Journal of Systems and Software*, vol. 8, no. 3, pp. 199 – 246, 1988.

9. B. Meyer, "Applying 'Design by Contract'," *Computer*, vol. 25, no. 10, pp. 40–51, 1992.
10. W. Damm, "Controlling Speculative Design Processes using Rich Component Models," in *ACSD'05*, pp. 118–119, 2005.
11. W. Damm and A. Votintseva, "Boosting Re-use of Embedded Automotive Applications Through Rich Components," *Proceedings of FIT*, pp. 1–18, 2005.
12. B. Josko, Q. Ma, and A. Metzner, "Designing Embedded Systems using Heterogeneous Rich Components," *INCOSE'08*, vol. 18, no. 1, pp. 558–576, 2008.
13. L. Benvenuti, A. Ferrari, E. Mazzi, and A. L. Sangiovanni-Vincentelli, "Contract-Based Design for Computation and Verification of a Closed-Loop Hybrid System," in *Hybrid Systems: Computation and Control*, vol. 4981 of *LNCS*, pp. 58–71, Springer, 2008.
14. R. Passerone, W. Damm, I. B. Hafaiedh, S. Graf, A. Ferrari, L. Mangeruca, A. Benveniste, B. Josko, T. Peikenkamp, D. Cancila, A. Cuccuru, S. Gérard, F. Terrier, and A. Sangiovanni-Vincentelli, "Metamodels in Europe: Languages, Tools, and Applications," *IEEE Design and Test of Computers*, vol. 26, no. 3, pp. 38–53, 2009.
15. OMG, *A UML Profile for MARTE (version beta 1)*. Object Management Group, omg document number: ptc/07-08-04 ed., 2007.
16. F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli, "Metropolis: An Integrated Electronic System Design Environment," *Computer*, vol. 36, no. 4, pp. 45–52, 2003.
17. A. Benveniste, B. Caillaud, A. Ferrari, L. Mangeruca, R. Passerone, and C. Sofronis, *Multiple Viewpoint Contract-Based Specification and Design*, pp. 200–225. Springer, 2008.
18. A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. Henzinger, and K. G. Larsen, "Contracts for Systems Design: Methodology and Application cases," Tech. Rep. RR-8760, INRIA, 2015.
19. A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. Henzinger, and K. G. Larsen, "Contracts for Systems Design : Theory," Tech. Rep. RR-8759, INRIA, 2015.
20. S. Graf, R. Passerone, and S. Quinton, "Contract-Based Reasoning for Component Systems with Rich Interactions," in *Embedded Systems Development*, vol. 20 of *Embedded Systems*, ch. 8, pp. 139–154, Springer, 2014.
21. P. Nuzzo, H. Xu, N. Ozay, J. B. Finn, A. L. Sangiovanni-Vincentelli, R. M. Murray, A. Donzé, and S. A. Seshia, "A Contract-Based Methodology for Aircraft Electric Power System Design," *IEEE Access*, vol. 2, pp. 1–25, 2014.
22. P. Nuzzo, A. L. Sangiovanni-Vincentelli, D. Bresolin, L. Geretti, and T. Villa, "A Platform-Based Design Methodology With Contracts and Related Tools for the Design of Cyber-Physical Systems," *Proceedings of the IEEE*, vol. 103, no. 11, pp. 2104–2132, 2015.
23. M. Persson, M. Törngren, A. Qamar, J. Westman, M. Biehl, S. Tripakis, H. Vangheluwe, and J. Denil, "A Characterization of Integrated Multi-View Modeling in the Context of Embedded and Cyber-Physical Systems," in *EMSOFT '13*, pp. 1–10, IEEE Press, 2013.
24. S. M. Prabhu and P. J. Mosterman, "Model-Based Design of a Power Window System: Modeling, Simulation, and Validation," in *Society for Experimental Machines IMAC Conference*, 2004.