Ontological Reasoning for Consistency in the Design of Cyber-Physical Systems

Ken Vanherpen*, Joachim Denil*, István Dávid*, Paul De Meulenaere*, Pieter J. Mosterman†, Martin Törngren‡, Ahsan Qamar§ and Hans Vangheluwe*¶
*University of Antwerp
Email: {ken.vanherpen, joachim.denil, istvan.david, paul.demeulenaere}@uantwerp.be
†MathWorks
Email: pieter.mosterman@mathworks.com
‡KTH Royal Institute of Technology
Email: martint@kth.se
§Ford Motor Company
Email: aqamar2@ford.com
¶McGill University
Email: hv@cs.mcgill.ca

***Abstract*—The design of Cyber-Physical Systems (CPS) involves a multitude of stakeholders. Each of these stakeholders has a specific view on the system under design. Unfortunately, when designers create artefacts in their different views in a concurrent manner, the integration of the different views may reveal inconsistencies. This leads to time consuming, iterative design processes where inconsistencies are resolved, in turn possibly creating new ones. It is hence necessary to reason explicitly about the view-specific properties that depend on, and influence properties of other views. This enables consistency during integration and reduces the development time and effort. In this paper we formalise the interrelationships between the different views, in the context of different design processes, to allow designers to meaningfully and efficiently manage inconsistencies. Our formalisation introduces ontological domain properties and their relations as the link between the view-specific properties used by the stakeholders. Thus, our approach combines the state of the art of Model-Based Systems Engineering (MBSE) and Semantic Web. The relevance of this approach is demonstrated by means of a motivating example.**

I. Introduction

The development process of a Cyber-Physical System (CPS) is characterised by a collaboration of different teams in multiple engineering disciplines, which we call stakeholders [1]–[3]. Given a set of requirements, representing the behaviour of the real-world system in a certain context, stakeholders express their individual concerns through a set of properties specific to their own view on the system under design. As a consequence, requirements are –often implicitly– shared among stakeholders due to the overlapping sets of properties. This overlap puts constraints on the design processes of the domain-specific views.

We experience this in current CPS design processes. For example, both control and embedded software engineers derive the properties which should hold for their view of a system with an implicit knowledge of each other's domain. For a control engineer it is, for example, important to know how fast outputs can be computed and written, which depends on the hardware platform. Due to his limited knowledge of the embedded domain, however, a control engineer may overestimate available hardware resources such as processor speed. During integration, the embedded software engineer chooses a processor type based on, for example, its cost. This might lead to a lower processor speed than what was assumed by the control engineer. As a result, integration affects the performance of the control algorithm.

This introductory example clearly demonstrates how both engineering views (stakeholders) use incomplete assumptions of each others view. Shared properties are not (fully) taken into account, such that consistency cannot be guaranteed. Consistency means the absence of inconsistencies: situations where multiple views imply conflicting values for common properties that may be derived from them. This results in iterative, time consuming design processes where inconsistencies are resolved, in turn possibly creating new ones. Because the views in this example are at different levels of abstraction, we classify this type of (in)consistency as *vertical (in)consistency*. The notion of vertical consistency also occurs within the engineering view. For example, during the modelling of a control algorithm, a more abstract control model is used at a higher level of abstraction while its refinement adds more detail at a lower level of abstraction. Vertical consistency should be maintained during the refinement process as well, such that the (behavioural) properties of the abstract control model are maintained after refinement.

Horizontal (in-)consistency pertains to models at the same level of abstraction. An example is the modelling of an electrical motor subsystem. Electrical and mechanical views exist for this system. These are considered at a same level of abstraction if they allow reasoning about exactly the same set of properties (such as power). To be consistent, analysis of both models must always yield identical values for each of these properties.

During the last decade, many attempts were made to resolve

vertical inconsistency by supporting control engineers with tools enabling them to make hardware properties explicit [4]–[6]. To this end, blocks introducing a certain hardware property are added to the control model. The blocks encode an abstraction of property effects (such as time) due to the hardware platform on which the controller is deployed. Contract-Based Design (CBD) [7]–[9] as a methodology, is gaining popularity in the design of CPS. Its use originates from computer programming where a set of pre- and postconditions defines under which conditions a system promises to operate satisfying desired properties. Similarly, by defining contracts between different engineering views in CPS design, view-specific properties are balanced against each other in a preliminary negotiation phase. This design methodology enables concurrent engineering (co-design) in which synchronization occurs on a regular basis. Thus, both horizontal and vertical consistency can be guaranteed.

From our experience with design processes, however, translating view-specific properties from one domain to another seems to be hard for engineers and is often done in an ad hoc fashion. To address this issue and to facilitate CBD, we express these interrelations using an ontological framework. To build this framework, we make the implicit knowledge of each stakeholder explicit. During the translation of requirements to view-specific properties, each stakeholder keeps in mind certain domain properties, which we call *ontological properties*. For example, a control engineer implicitly thinks about *control performance*, *reaction time*, and *safety*. The embedded engineer reasons about *schedulability*, *processor load*, *cost*, etc. Due to overlap in requirements, some ontological properties will be shared and/or will influence each other such that the related view-specific properties will be shared or influenced as well. By making the influence relations between ontological properties explicit in our framework, we are able to explicitly translate related properties used in different views.

Reasoning about ontologies (i.e., relating ontological properties) and tracing the properties at the modelling level to these ontologies allows us to examine current design processes. The processes can be restructured accordingly to reduce the number of costly design iterations. Note that we do not comply with the Semantic Web definition of ontologies since we have not yet committed to formalisms/languages/techniques/tools to use for ontological reasoning.

The rest of this paper is structured as follows. The usability of our approach is demonstrated through a motivating example in Section II. Section III presents three generalised design operations showing how ontological reasoning can be enabled. Given these generalised operations, we revisit the motivating example in Section IV. Section V discusses our fundamental approach and gives an outlook to future work. Section VI summarises the related work. Finally, Section VII concludes.

II. Motivating Example

We use the design of a power window at the passenger side of a vehicle to illustrate the usefulness of our approach. Although the power window seems more related to engineering an embedded system rather than a production system, it contains properties which are highly related to a production system: (a) it contains both computational and physical elements, (b) it contains sensors and actuators, and (c) when closing it might cause live threatening injuries such that safety is highly important. This section introduces the power window example and its design process. The case study will be revisited in Section IV using the foundations introduced in Section III.

For the embedded system to be designed, a set of requirements describe the behaviour of the system within a given context. In this case, the context of the power window is a vehicle, for which the behaviour at the passenger side can be described as follows [10]:

1) An electrical motor will operate the power window.
2) The window has a width and a height of respectively 1057 mm and 768 mm.
3) The power window can be operated by both driver and passenger. Priority is given to the driver.
4) The power window should start moving within 200 ms after a command is issued.
5) The power window shall be fully opened or closed within 4.5 s.
6) Detection of a clamped object when closing the window should lower the window by 10 cm.

For the design of the power window, three stakeholders are involved: a mechanical, control and embedded engineer. Depending on their view of the system, each stakeholder considers one or more requirements. The mechanical engineer chooses a motor whose characteristics (voltage, current, speed, torque, etc.) satisfy the requirements 1, 2 and 5. Furthermore, he creates a plant model representing the physical elements of the window and the motor.

The plant model is used by the control engineer, who is concerned with all requirements except 1 and 2, to synthesise the modelled control algorithm. It goes without saying that *control performance* will be of the utmost importance for the control engineer during the design of the control algorithm. Note that *control performance* is an ontological property which a control engineer implicitly keeps in mind when deciding on modelling properties such as ‘sample rate’.

An embedded engineer considers all of the requirements except 2 and the priority part of 3 in his view, such that the control algorithm deployed onto the Electronic Control Unit (ECU) is able to operate the window. He is concerned with hardware details (e.g. *processor speed* and *period of tasks*) instead of the behaviour of the control algorithm. Although there are shared requirements between the different views, the embedded engineer keeps in mind ontological properties such as *schedulability* when specifying properties like ‘processor speed’.

Let us consider the relationships between control and embedded engineer to get to the point of this illustrating example. As some of the requirements are shared among views, it may be clear that view-specific properties related to those shared requirements should be consistent. Due to the view-specific

interpretation of the requirements, however, engineers reason about different ontological properties: *control performance* for the control engineer and *schedulability* for the deployment engineer. Unintentionally, this leads to inconsistencies between views. In current design processes, for example, engineers have limited aids in estimating the impact of their design choices. Therefore, it is common for a control engineer to assume almost unlimited hardware resources such that view-specific properties such as 'computation and write time of outputs' are underestimated. As a consequence, *control performance* is verified using a wrong abstraction of the hardware platform. On the other hand, an embedded engineer strives for a schedulable system by deploying the control algorithm onto an ECU such that its *load* is regarded as *safe*. As a result, an ECU with enough resources (e.g., 'processor speed') is selected such that the system is *schedulable* without excessive *costs*. Moreover, the hardware platform's resources are typically shared among multiple software tasks. This results in extra time delays, expressed as 'Worst-Case Response Times' (WRTs), which where not taken into account by the control engineer.

From this example we conclude that relating view-specific properties such as 'computation and write time' and 'processor speed' may be difficult for engineers having a different view on the system. Moreover, they are often not aware of the relations between those view-specific properties and the ontological properties such as *control performance* and *schedulability*. By introducing an ontological framework, we make the relations between ontological properties explicit. This enables ontological reasoning such that view-specific properties can be interrelated and consistency can be guaranteed.

III. Foundations

Designing a CPS while keeping the views consistent is not obvious. Moreover, each design process is slightly different making consistency management of view-specific properties process dependent. We believe that using a combination of generic templates improves the usefulness of consistency management among views while reducing time and effort. By analysing current design processes, we experienced that they combine three fundamental relationships: Multi-Semantics (MS), Multi-Abstraction (MA) and Multi-View (MV).

The structure of each operation relies on the concepts of linguistic and ontological (meta-)modelling. In a model-driven-engineering context, a conformance relationship exist between a meta-model and a (possible infinite) set of models which are instances of the meta-model. According to Kühne [11] this conformance relationship can be either linguistic or ontological. Based on the work of Barroca et al. [12], Figure 1 represents these conformance relationships.

We clarify the different types of conformance relationships by means of the motivating example of Section II. A control algorithm is modelled by a control engineer using formalisms such as causal block diagrams, Statecharts, etc. Each model is typed by a meta-model: there exists a conformance relation between them. Since we are dealing with languages, this

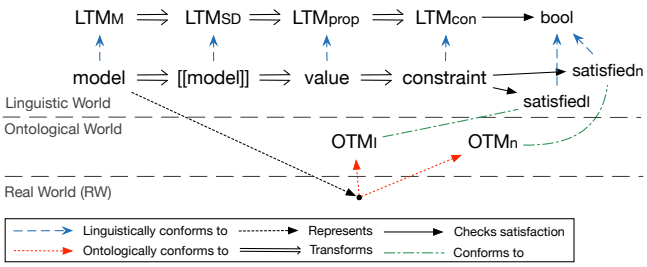

Fig. 1. Linguistic versus Ontological models

conformance relation is called linguistic and the meta-model is called a Linguistic Type Model (*LTM*). Semantics is given to a *model* by defining a Semantic Domain (*SD*) and a semantic mapping function (*[[.]]*) which maps a model onto its meaning, an element of the Semantic Domain. For example, the control model of the power window can be transformed to a Petri-Nets model, which linguistically conforms to the Petri-Net meta-model, to obtain a reachability graph. A second transformation is used to retrieve performance values such as liveness and boundedness. We specify them as linguistic properties since they are situated in the linguistic world. In that sense, the view-specific properties 'sample rate' and 'period of tasks' in Section II are linguistic as well. Subsequent transformations will check if a linguistic property satisfy a property using a function that returns a logical value (True or False). The linguistic models are modelled with Closed World Assumptions (CWA). This means that if a property is not modelled, it is assumed False.

As shown in the motivating example, each view interprets the requirements by means of some ontological properties (e.g., Control Performance High Enough? and Schedulable?). We use the question mark to make explicit that these are ontological properties that need to be checked based on the linguistic performance values. As a result, each *model* is typed by one or more Ontological Type Models (*OTM*) representing the implicit knowledge of the engineer. An *OTM* categorises or classifies real-world entities based on properties (concepts). These are logically related using some appropriate logic (e.g., description logic). Note that each ontology also conforms to a Linguistic Type Model (LTM) because the representation of the ontology must also be modelled using a language.

In our philosophy of ontological reasoning, ontologies and linguistic models are related to each other through a satisfaction relationship that must hold between their respective properties. In other words, each linguistic property stemming from a semantic domain can be linked to an ontological property. This implies that linguistic properties stemming from different semantic domains can be related to each other through a common ontology or a set of ontologies. Note that from an ontological viewpoint, no strict relation exists between a *model* and an *OTM*. If a relation does not exist, either within the ontology or with the linguistic type model, we do not assume that it is False. We could just be unaware of the

relation. Ontologies are therefore modelled with Open World Assumptions (OWA).

Based on these principles, the next subsections elaborate on three fundamental relationship patterns: Multi-Semantics (MS), Multi-Abstraction (MA), and Multi-View (MV). Figure 1 will serve as a basis to describe the structure of these patterns, each of them exemplified using isolated design operations on the power window example. By composing the different patterns, Section IV demonstrates the usefulness for the entire design process.

A. Multi-Semantics (MS)

Intent: The first pattern focusses on multiple semantic domains, for a single engineering domain, to give meaning to one specific view on the real-world system. It is useful when different performance characteristics can be analysed from a single model. For example, an electronic engineer analyses both the power consumption and heat dissipation of an electronic system-on-chip. Power consumption and heat dissipation are analysed in different semantic domains, using a different semantic mapping.

Structure: Figure 2 gives an overview of the relationships between linguistic and ontological properties. In the first phases of the design process, a written set of requirements formulates the desired properties of the real world system for a given context. Given these requirements, the engineer implic-

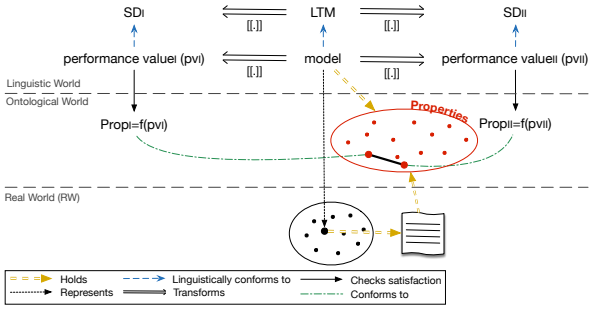

Fig. 2. Linguistic and Ontological relationships for *Multi-Semantics*

itly reasons about ontological properties. The solid oval in the *Ontological World* denotes the set of ontological properties covered by the requirements. Examples of such ontological properties include Safe?, Performance High Enough?, Schedulable?, Deadlock Free?, etc.

As a first step in the design process, the engineer makes an abstraction of the real-world system by means of a *model*. This model strictly conforms to a *Linguistic Type Model* (LTM). This is denoted by the conformance relation in Figure 2. By mapping (using semantic mapping function *[[.]]*) the *model* to a *Semantic Domain* (SD) a meaning is associated with the model. The model thus obtained in the Semantic Domain may allow analysis of some pertinent (linguistic) properties. In this pattern, multiple semantic mappings to different semantic domains are available to analyse different linguistic properties. The result of these analyses are *performance values* (pv).

To check if a linguistic property satisfies a certain ontological property, we test its related *performance value* using a function that returns a logical value (True or False).

Reasoning about consistency: Different satisfaction relations can exist between the performance values and the ontology: (I) the performance values must satisfy two orthogonal properties. The two properties are orthogonal if they are not ontologically related (even after transitive closure of intermediate relationships). In this case, there are no consistency issues. (II) both performance values must satisfy the same ontological property. In this case, the model is consistent with itself. Otherwise the model is (a) intra-model inconsistent, (b) the semantic mappings are inconsistent, (c) different linguistic properties are checked or (d) the model is infeasible. (III) there are (transitive) relations between the ontological properties that must be satisfied: Depending on the type and direction of the relations, this will lead to category (I) or (II).

Motivating Example: We clarify the pattern by means of the example of Section II. The control model shown in Figure 3 represents the *model* which linguistically conforms to the meta-model of Simulink®. On the one hand, performing a simulation gives semantics to the control model (e.g., in the form of a simulation trace). From this, we obtain *performance*

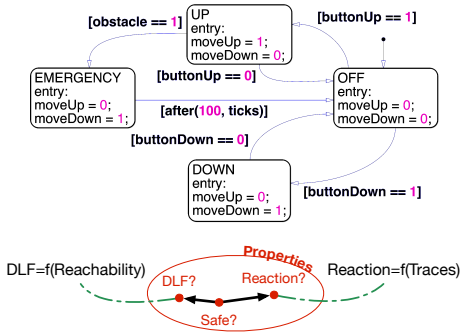

Fig. 3. Model of the power window controller and its ontology for *Multi-Semantics*

values such as the time to reverse the movement of a window when an object gets stuck between a closing window and the frame. This is then checked against the *Reaction Time High Enough?* property. On the other hand, a transformation to a Petri Net representation can be made to verify the *Deadlock Free?* property. Both ontological properties have a relation to the property *Safe?*. The influence relation between ontological properties is in this case: *Safe?* requires *Deadlock Free?* and *Reaction Time High Enough?*. *Deadlock Free?* and *Reaction Time High Enough?* are orthogonal.

B. Multi-Abstraction (MA)

Intent: In the multi-abstraction pattern, an abstraction-refinement relation exists between the different models. This implies that the abstract model's performance values must satisfy a subset of the ontological properties satisfied by the (performance values of the) more refined model.

Structure: The structure of the pattern is shown in Figure 4. As in every design process, a written set of requirements

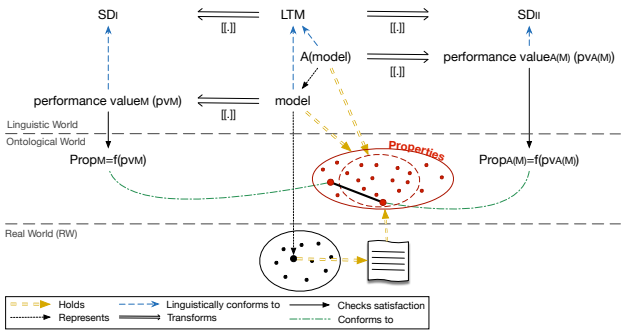

Fig. 4. Linguistic and Ontological relationships for *Multi-Abstraction*

formulates the real world system demands. Given this set of requirements, each engineering domain creates a set of ontological properties and relations between the ontologies that the system should satisfy. Because there is only a single view (engineering domain), there exists only one set of ontological properties the system should satisfy. This is denoted by the solid oval in the *Ontological World*. Similar to the previous pattern, linguistic properties are tested for both models by transforming them to a semantic domain. Again, the performance values are tested, using a function, for satisfaction with the ontological properties.

By definition of abstraction A, for an original model *model*, only a subset of the ontological properties satisfied by the performance values (in the *Linguistic World*) of the original model have to be satisfied by the performance values of the abstracted model $A(model)$. For each such ontological property op:

$$\{\text{A(model)} \models \text{op}\} \implies \{\text{model} \models \text{op}\}$$

If $A(model)$ satisfies an ontological property, this *must* imply that $model$ satisfies that same property.

Reasoning about consistency: If the set of performance values of the refined model do not satisfy all the properties of the abstracted model, the two models are inconsistent. This case is called vertical inconsistency in the literature. The designer should mitigate the issue such that the refined model satisfies all the properties of the abstract model. This could be a redesign of the abstract or refined model.

Motivating Example: Figure 5 shows a refined model of the power window controller and the set of ontological properties it satisfies (via its performance values). Our refined model still satisfies the ontologies discussed in the multi-semantics example. However, it also satisfies a new ontological property: *Priority to Driver?*. This property denotes that the driver commands have priority over the commands of the passenger. The model is still deadlock free and the reversal of the window is still satisfied. Note that we also discovered a new relation in the ontology. *Safe?* now also requires the *Priority to Driver?* property to be satisfied. Our abstract model however cannot be regarded as *Safe?* anymore because it has no notion of priority. To keep the ontology consistent with the

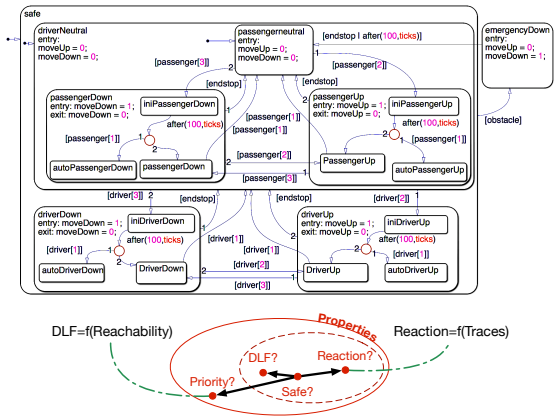

Fig. 5. Refined model of the power window controller and its ontology for *Multi-Abstraction*

different design artefacts, the *Safe?* property should be moved from the inner to the outer set of properties.

C. Multi-View (MV)

Intent: A final pattern operation focuses on multiple engineering domains involved when designing a CPS, each of them with a domain-specific view on the real-world system. It is useful when the view-specific models are somehow related to each other. For example, during the design of a control algorithm its model is synthesized using a plant model representing the physical elements of the real-world.

Structure: Similar to the previous patterns, Figure 6 depicts the structure of Multi-View design. Given the set of

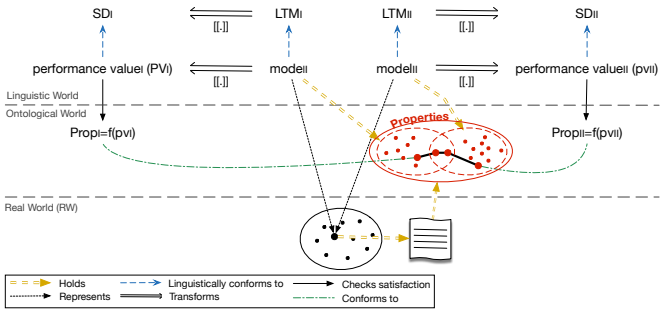

Fig. 6. Linguistic and Ontological relationships for *Multi-View*

requirements describing the behaviour of the real-world system for a given context, each view (engineering domain) reasons about certain linguistic properties and their related ontological properties. These sets are represented by the dashed ovals in Figure 6. However, since some of the requirements are shared among the views, properties will concern both views which implies that the ontological sets overlap. A semantic mapping function transforms both models to a semantic domain to test their linguistic properties. Using an appropriate evaluation function, the performance values are evaluated for satisfaction with the ontological properties.

Reasoning about consistency: Since the ontological properties at the intersection are related to the same requirement(s), an ontological relationship between them exists by default. Satisfaction between the performance values and the ontology can occur in two ways: (I) if the performance value(s) satisfy one or more of the properties in the intersection. (II) if the performance value(s) satisfy a view-specific ontological property which has a relation (after transitive closure) with a property in the intersection. If for (I) and (II) the view-specific performance values satisfy the property, the model is consistent with itself. Otherwise, the model is inter-model inconsistent. In the literature, this type of (in)consistency is specified as horizontal (in)consistency. Consistency can be guaranteed as well if performance value(s) satisfy orthogonal properties such that no relation with a property of the intersection exists.

Motivating Example: We illustrate this pattern operation by using the motivating example of Section II. The upper part of Figure 7 depicts how the power window controller is connected to a plant model, while the lower part of Figure 7 shows relationships between ontological properties. As already

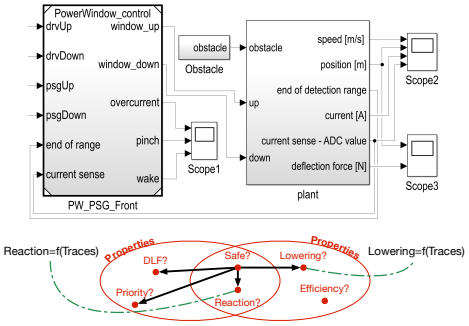

Fig. 7. Control and plant model of the power window and their ontologies for *Multi-View*

shown in the previous patterns, the power window controller is modelled using a statechart diagram and satisfies the ontological properties discussed in the multi-abstraction example. On the other hand, the plant model describes the physical elements of the real-world (i.e., the motor and the window mechanism) using causal-block diagrams. For this view, performance values should satisfy the properties *Efficiency High Enough?*, *Lowering?*, *Safe?* and *Reaction Time High Enough?*. Since lowering the window ensures that a clamped object can be released, a relation exists between *Lowering?* and *Safe?*. From the example in the multi-semantics pattern, we have shown how *Safe?* is related to *Reaction Time High Enough?* for the control view. Since the properties *Safe?* and *Reaction Time High Enough?* are part of the intersection, inter-model consistency can be guaranteed.

IV. Motivating Example Revisited

While demonstrating the fundamental design operations in the previous section, it became clear no single pattern can be used on its own in a complete design process. This section revisits the motivating example of Section II using the fundamental pattern operations, validating how consistency can be guaranteed during the design of a CPS.

Figure 8 shows the ontologies related to the design of the power window example. Centralized in the figure, one may recognize the Multi-View pattern combined with the Multi-Abstraction pattern which where demonstrated in Figure 7 and Figure 5 respectively. Both patterns are concurrently used by the control and mechanical engineer. Note that we have added the ontological property *Performance High Enough?* to indicate a control engineer reasons about control performance as well during his design. As already mentioned in Section II,

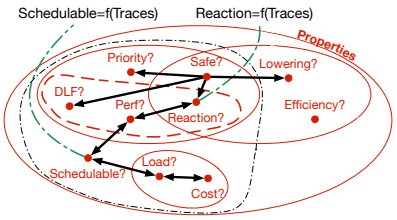

Fig. 8. Ontologies related to the design process of the power window

the third stakeholder (embedded engineer) is concerned about the deployment of the control algorithm onto an ECU. Therefore, he uses a hardware platform which is designed by the same or an additional engineer keeping in mind ontological properties such as *Balanced Load?* and *Low Cost?*. A relation between them exists since more hardware resources (resulting in a lower load) leads to a higher cost and vice versa. This is symbolized by an ontology which has no (direct) relation with the ontologies regarding the design of the control and plant model. To this end, we say that both ontologies are orthogonal.

However, a consistency relationship between the software (control algorithm) and the hardware (ECU) exists from the ontological property *Balanced Load?* to *Performance High Enough?* through the property *Schedulable?*. This latter property refers to the implicit knowledge of the embedded engineer who strives for a schedulable system in which the load for the ECU and the control performance is balanced. Since control performance has a consistency relationship with the property *Reaction Time High Enough?*, an indirect link between the schedulable system and the reaction time, to reverse the movement of the window, exists. Due to this ontological reasoning, a schedulable system implies a system to be safe.

One may notice that the pattern composition is not entirely valid for this design process of our power window. Since the *Schedulable?* property concerns both control and embedded engineer it would have been better to reason upfront about the performance values of timing.

V. Discussion

The foundations and motivating example show how inconsistencies arise because of the link between linguistic and ontological properties and their interrelations. We discuss some ramifications of reasoning in the ontological world for

the design of complex engineered systems and subsequent tools that are needed.

Engineers use approximation in a similar way as abstraction. From an engineering perspective, we rarely deal with true abstraction. Because of measurement errors, numerical techniques, order reduction of our physical quantities and phenomena, we usually approximate our performance value in the linguistic domain. Our foundations deal with these approximation by tuning the function that checks a linguistic performance value against the ontological property. The function has to take the tolerated error into account, and thus works with a range of values. Reasoning about consistency is in this case much harder. We need to take a distance metric between the non-approximated performance value and the approximated performance value into account to allow for substitutability. This is considered future work for our approach.

During the process of designing systems, engineers learn about the system they are designing. It is therefore also necessary to allow the ontology to be extended and updated. This also has to reflect within the tool support. However, the mechanisms of relating process, models and ontologies requires tools to be usable by engineers. We will use the model verse tool to enact the different modelling environments [13]. The model verse tool allows us to model languages, models, processes, ontologies and the different dependencies between the different artefacts. It also requires us to trace the different properties back to the linguistic world such that efficient consistency management can be done at the modelling level. Of course, the use of an additional tool(set) might lead to additional complexity. In certain development processes, however, ontologies can be reused, extended and/or merged which reduces the additional complexity and enhances scalability. Nonetheless, the feasibility of extending and merging two or more ontologies will depend on their heterogeneity. Note that the use of an incomplete ontology doesn't necessary lead to an inconsistent system. However, consistency can not be guaranteed for those properties which are not linked to an ontology.

Tool builders can take the ontological properties and relations into account to support the engineers. An example of this is the use of round-trip engineering in the design process. With round-trip engineering, information about certain linguistic properties, that are only analysable at a less abstract level, are reintroduced at the higher abstraction level, essentially synchronising the two artefacts. However, we still need to reason about which linguistic properties need to be synchronised. Because of the link with ontologies, we can now reason on the dependant ontological properties and trace them to the linguistic properties. For example, the control models can be extended with timing information as shown in [6], [14]. Both papers introduce extra blocks into the control model that represent the delays of the computation time, scheduling time and communication time. This allows the control engineer to re-evaluate the control model for timing anomalies. By making the information explicit, the number of design iterations could be reduced.

By reasoning about the relations between the different ontological properties, we can also reason about the design process. An example of such a process restructuring can be the parallelisation of two sequential design activities where no ontological overlap is present. Tearing, partitioning and sequencing algorithms, introduced in design structure matrices, can be used to guide the restructuring process [15].

The linguistic models linked to the ontologies also help us in identifying activities that do not use a proper abstraction. For example, in the power window, the schedulability property influences the control performance property. However, the evaluation of the schedulability property requires more information that is only available after mapping the control components to the hardware components. Restructuring these activities into a real co-design process requires us to reason about the timing information before starting the design of the control and hardware models. Vertical design contracts allow us to mend this. The ontologies and relation with the linguistic world allows us to infer the view-specific properties that are needed for a specific design contract. By making the relations explicit, engineers can negotiate about the contracts with a common understanding of how properties are linked.

VI. Related Work

Introducing a common understanding between different disciplines through languages has been a common research theme in the mechatronic domain, for example in [16]. Close to our contribution is the work of Hehenberger et al [17] who introduce a domain ontology with structural elements of the design and relations to reason about consistent structures of the mechatronic device. In [18] OWL ontologies are used to formally represent the design of a production system. Combined with a SysML-based modelling approach, it enables engineers to evaluate the compatibility of their domain-specific models. Kovalenko et al [19] use an ontology-based approach to support engineers in the automated detection of defects between domain-specific models. Similar to our approach, it enables engineers to use their own tools and view on the system under design.

Persson et al [20] characterise model-based approaches used in the design of Cyber-Physical Systems. The authors identify *consistency* between the various *views* of the system as one of the main challenges in design of such complex systems. This is due to relations between views, with respect to their content (i.e., *semantic* relations), process and operations which are not entirely exclusive to each other.

Van der Straeten et al [21] emphasise the importance of proper characterisation of inconsistent model states, as a foundational activity in model inconsistency management. The authors propose a logic-based approach to address this activity, although, as it often the case in the state-of-the-art, they focus entirely on linguistic inconsistencies of pure software systems.

As an alternate approach, correspondence models have been used extensively as a formal underpinning to inconsistency characterisation. Qamar et al [22] propose explicit modelling

of dependencies among elements of various models in order to support change propagation and consistency management. Dávid et al [23] extend this framework by adapting it to explicitly modelled design processes and augmenting the reasoning by levels of precision of dependency links. As a special case of correspondence modelling, pivot models are often employed in mechatronics and CPS. Bhave et al [24] use an "architectural base model" and a set of model transformations for bi-directional mapping between various views and the base model. SysML and EAST-ADL are frequent choices for implementing pivot models, as presented in [25] and [26]. Adourian et al [27] show a technique for maintaining consistency between geometric and dynamic properties of mechanical systems using triple graph grammars (TGG). Due to the nature of TGGs, the approach does not only support unidirectional synchronisation, but also bidirectional change propagation and consequently, parallel evolution.

Characterisation of inconsistencies is a labor intensive task when carried out manually. This is especially true when ontological properties have to be taken into account. Herzig et al [28] propose a probabilistic approach to identify semantically similar model elements over multiple models, in order to relate them. Bayesian reasoning is used to infer likely semantic overlaps, based on similarity metrics of various models and model elements.

VII. Conclusions

This paper provides a foundation to facilitate communication among stakeholders having different concerns when designing a Cyber-Physical-System. We isolated three design operations and extended them with the notion of ontological reasoning. By re-composing fundamental design operations, we demonstrated how (in-)consistency can be managed. Furthermore, reasoning in the ontological domain gives us insights into the required content of contracts in Contract Based Design.

Acknowledgments

This work has been carried out within the MBSE4Mechatronics project (grant nr. 130013) of the agency for Innovation by Science and Technology in Flanders (IWT-Vlaanderen). This research was partially supported by Flanders Make vzw.

References

[1] P. Derler, E. Lee, and A. Vincentelli, "Modeling Cyber-Physical Systems," *Proceedings of the IEEE*, vol. 100, no. 1, pp. 13–28, January 2012.

[2] E. Lee, "Cyber Physical Systems: Design Challenges," in *ISORC '08*, May 2008, pp. 363–369.

[3] "ISO/IEC/IEEE 42010 International Standard: Systems and Software Engineering: Architecture Description," 2011.

[4] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K.-E. Arzen, "How does control timing affect performance? Analysis and simulation of timing using Jitterbug and TrueTime," *IEEE Control Systems*, vol. 23, no. 3, pp. 16–30, Jun. 2003.

[5] D. Henriksson, A. Cervin, and K.-E. Arzén, "TrueTime : Real-time Control System Simulation with MATLAB / Simulink," in *Proceedings of the Nordic MATLAB Conference*, 2003.

[6] K. Vanherpen, J. Denil, H. Vangheluwe, and P. De Meulenaere, "Model Transformations for Round-trip Engineering in Control Deployment Co-Design," in *TMS/DEVS '15*. SCS, April 2015, pp. 820 – 827.

[7] P. Derler, E. A. Lee, S. Tripakis, and M. Törngren, "Cyber-Physical System Design Contracts," in *ICCPS '13*. ACM, 2013, pp. 109–118.

[8] 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.

[9] 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.

[10] 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.

[11] T. Kühne, "Matters of (Meta-) Modeling," *Software & Systems Modeling*, vol. 5, no. 4, pp. 369–385, Jul. 2006.

[12] B. Barroca, T. Kühne, and H. Vangheluwe, "Integrating Language and Ontology Engineering," in *MPM '14*, ser. CEUR, vol. 1237, September 2014, pp. 77–86.

[13] S. Van Mierlo, B. Barroca, H. Vangheluwe, E. Syriani, and T. Kühne, "Multi-Level Modelling in the Modelverse," in *MULTI '14*, 2014, pp. 83–92.

[14] O. Redell, J. El-khoury, and M. Törngren, "The AIDA toolset for design and implementation analysis of distributed real-time control systems," *Microprocessors and Microsystems*, vol. 28, no. 4, pp. 163 – 182, 2004.

[15] A. Yassine, "An introduction to modeling and analyzing complex product development processes using the design structure matrix (DSM) method," *Urbana*, vol. 51, no. 9, pp. 1–17, 2004.

[16] J. Gausemeier, M. Flath, and S. Mohringer, "Conceptual design of mechatronic systems supported by semi-formal specification," in *AIM '01*, vol. 2, 2001, pp. 888 – 892.

[17] P. Hehenberger, A. Egyed, and K. Zeman, "Consistency Checking of Mechatronic Design Models," in *CIE '10*, vol. 3, no. Parts A and B. ASME, 2010, pp. 1141–1148.

[18] S. Feldmann, K. Kernschmidt, and B. Vogel-heuser, "Combining a SysML-based modeling approach and semantic technologies for analyzing change influences in manufacturing plant models," *Procedia CIRP*, vol. 17, pp. 451–456, 2014.

[19] O. Kovalenko, E. Serral, M. Sabou, F. J. Ekaputra, D. Winkler, and S. Biffl, "Automating Cross-Disciplinary Defect Detection in Multidisciplinary Engineering Environments," in *Knowledge Engineering and Knowledge Management*. Springer International Publishing, 2014, pp. 238–249.

[20] 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*. Montréal, Québec, Canada: IEEE Press, September 2013, pp. 1–10.

[21] R. Van Der Straeten, "Inconsistency Management in Model-Driven Engineering: An Approach Using Description Logics," Ph.D. dissertation, VUB Brussels, Belgium, 2005.

[22] A. Qamar, J. Wikander, and C. During, "Managing dependencies in mechatronic design: a case study on dependency management between mechanical design and system design," *Engineering with Computers*, pp. 1–16, July 2014.

[23] I. Dávid, J. Denil, and H. Vangheluwe, "Towards Inconsistency Management by Process-Oriented Dependency Modeling," in *MPM'15*. CEUR, September 2015.

[24] A. Bhave, B. Krogh, D. Garlan, and B. Schmerl, "View Consistency in Architectures for Cyber-Physical Systems," in *ICCPS '11*, April 2011, pp. 151–160.

[25] A. A. Shah, A. A. Kerzhner, D. Schaefer, and C. J. J. Paredis, "Multi-view Modeling to Support Embedded Systems Engineering in SysML," in *Graph Transformations and Model-driven Engineering*. Springer, 2010, pp. 580–601.

[26] A. Y. Bhave, B. Krogh, D. Garlan, and B. Schmerl, "Multi-Domain Modeling of Cyber-Physical Systems using Architectural Views," in *AVICPS '10*, November 2010.

[27] C. Adourian and H. Vangheluwe, "Consistency Between Geometric and Dynamic Views of a Mechanical System," in *SCSC '07*. SCS, 2007, pp. 31:1–31:6.

[28] S. J. Herzig, A. Qamar, A. Reichwein, and C. J. Paredis, "A Conceptual Framework for Consistency Management in Model-Based Systems Engineering," in *ASME '11*. ASME, 2011, pp. 1329–1339.