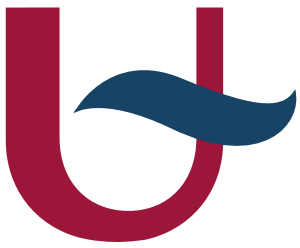


Universiteit Antwerpen

Faculteit Toegepaste Ingenieurswetenschappen
Elektronica-ICT

A Contract-Based Approach for Multi-Viewpoint Consistency in the Concurrent Design of Cyber-Physical Systems

Proefschrift voorgelegd tot het behalen van de graad van
doctor in de toegepaste ingenieurswetenschappen
aan de Universiteit Antwerpen te verdedigen door

Ken Vanherpen

Prof. Dr. Paul De Meulenaere
Prof. Dr. Hans Vangheluwe

Antwerpen, 2018

Jury

Chairman
Prof. Dr. Peter Hellinckx, University of Antwerp, Belgium

Supervisors
Prof. Dr. Paul De Meulenaere, University of Antwerp, Belgium
Prof. Dr. Hans Vangheluwe, University of Antwerp, Belgium

Members
Prof. Dr. ing. Joachim Denil, University of Antwerp, Belgium
Prof. Dr. ir. Pieter J. Mosterman, MathWorks, USA
Dr. ir. Klaas Gadeyne, Flanders Make, Belgium

Contact

Ken Vanherpen
CoSys-Lab, Faculty of Applied Engineering, University of Antwerp
Groenenborgerlaan 171, 2020 Antwerpen, Belgium
M: ken.vanherpen@uantwerpen.be
T: +32 3265 1866

Acknowledgments

I almost finished my master degree when one day Paul asked me if I would be interested in pursuing a PhD. Although I considered it as an honor to be given this opportunity, there was some doubt. After all, I did not had the ambition to pursue an academic career. A few weeks later, Paul introduced me to Hans who overwhelmed me with his enthusiasm. I guess a lot had to do with the fact I did some modeling in my master's project. Combined with Paul's vision on how a PhD can be useful in industry I finally agreed. As I experienced myself, the road towards obtaining a PhD title is typed by highs and lows. Fortunately, I was surrounded with two supervisors who, in a subtle way, always encouraged me to push my boundaries. Thank you Paul and Hans for giving me this opportunity, to believe in me, and to support my business interests (that sometimes interfered with my research).

During the past years, I have had the chance to discuss my research with numerous people. In particular, I would like to thank former and current members of the CoSys-Lab research group, of the AnSyMo/MSDL research group, and of the IDLab research group. A special thanks goes to Joachim who always found the time to organize some memorable brainstorm sessions. I would also like to thank the people of Atlas Copco, Dana, Van de Wiele, Picanol, Siemens PLM Software, Tenneco, and Verhaert to provide me some industrial insights.

As with every PhD dissertation, this work has also been accepted by the scientific community. Thank you Peter Hellinckx, Paul De Meulenaere, Hans Vangheluwe, Joachim Denil, Pieter J. Mosterman, and Klaas Gadeyne for being part of my jury. It was a pleasure to answer your questions and to read your suggestions. In a constructive way they have contributed to the final result of this dissertation.

This PhD would not have been possible without financial means. As such, I would like to thank the Flanders Innovation & Entrepreneurship agency (VLAIO) and Flanders Make vzw to support the *MBSE4Mechatronics* project (grant nr. 130013) and the *CSE_Codesign_ICON* project (grant nr. HBC.2017.0391). In particular, I would like to thank project managers Gregory Pinte, Maarten Witters, and Davy Maes to give me the opportunity to gain experience in both research and industry.

My family and friends have been of a tremendous value during the past four years. One way or another they have contributed to the successful journey I have made so far. I will always be grateful to my parents for their unconditional support.

Last but not least, thank you Sandrine for being a great girlfriend who I always can rely on. Thank you for your love, your support, your believe in me, and to take care of our lovely daughter Zita. Although Zita is only nine months old, her unstoppable smile and energy encouraged me in finalizing this dissertation. Sandrine and Zita, this work is dedicated to you!

Ken Vanherpen
August 2018
Antwerp, Belgium

Abstract

The exponential rate at which Cyber-Physical Systems evolve, while getting increasingly complex, poses new challenges in terms of their design. Already, engineers practice a Model-Based Systems Engineering approach, in which models are used to support their requirements engineering, design, verification, and validation activities. Because of the heterogeneity of these systems, the design of Cyber-Physical Systems is often allocated to a multidisciplinary team combining expertise in different engineering areas (e.g., control logic, embedded systems design, and mechanics). Hence, Multi-Paradigm Modeling as a method is practiced enabling engineers to model each aspect of the system explicitly at the most appropriate level(s) of abstraction using the most appropriate formalism(s), while modeling the development process(es) explicitly. Typically, this well-defined development process comprises three main phases: a common architectural phase, a domain-specific implementation phase and an integration phase. Often, the different domains enact the implementation phase concurrently to each other so that the time to market and design costs are reduced.

Unfortunately, we observe that the enactment of the process is characterized by costly, iterative design cycles partly due to the involvement of various engineering disciplines, each with a different viewpoint on the system under design. This heterogeneity often leads to inconsistent decisions on shared design parameters, causing unexpected behaviors while integrating the system. Contract-Based Design as a method has already been proposed to preserve consistency between different design artifacts. Therefore, a contract consisting of a set of assumptions and guarantees is defined between engineers prior to the design phase. The assumptions and guarantees describe the conditions under which a system promises to operate while satisfying desired properties. The method, however, appears to be only applicable when reasoning about consistency within one and the same viewpoint. Focusing on the interactions between control and embedded viewpoints, this dissertation extends the current method by combining the principles of Contract-Based Design with ontological reasoning to ensure consistency across viewpoints.

The proposed method, and in particular the notion of ontological reasoning, not only ensures multi-viewpoint consistency but enables the automatic translation of design parameters from one domain to another. This appears to be extremely important since engineers lack in the ability

to reason about each others domain. As such, model simulations, used to verify and validate different aspects of the system throughout the implementation, do not correctly represent the behavior of the integrated system. We propose a Round-Trip Engineering method allowing for a semi-automatic annotation of simulation models, incorporating properties from domains that may influence its behavior.

As contracts are defined prior to the design of the system, contracts constrain the design space to a finite set of possible implementations and integrations. As such, Design-Space Exploration techniques can be applied to assist engineers while designing and integrating the system. We elaborate on how contracts enable Design-Space Exploration and initiate a pattern catalog categorizing the embedding of different Design-Space Exploration techniques in a Model-Based Systems Engineering context.

The complexity of multi-viewpoint design process and the practicalities of our methods are tackled by means of an academic case study. We also demonstrate how our methods and techniques can be integrated in concurrent design processes for Cyber-Physical Systems. Although this dissertation focuses on the interactions between control and embedded engineering, the presented methods and techniques can be applied to any multi-domain engineering process.

Nederlandstalige Samenvatting

De exponentiële groei in de evolutie van cyber-fysische systemen en hun toenemende complexiteit zorgt voor nieuwe uitdaging tijdens hun ontwerp. Daar wordt reeds gekozen voor een *Model-Based Systems Engineering*-aanpak waarbij modellen worden gebruikt ter ondersteuning van het ontwerpproces gaande van het opstellen van de systeemvereisten tot het ontwerpen, verifiëren en valideren van het systeem. Door de heterogeniteit van cyber-fysische systemen worden meestal ingenieurs uit verschillende domeinen (zoals controlelogica, embedded systemen en mechanica) samengevoegd in een ontwerpteam. Tijdens het ontwerp maakt men vaak gebruik van de *Multi-Paradigm Modeling*-methode om alle aspecten van een systeem te modelleren op het (de) meest geschikte abstractieniveau(s), gebruikmakende van de meest geschikte modeleertaal of set van modeleertalen, en waarbij het (de) gevolgde ontwerpproces(sen) expliciet wordt (worden) gemodelleerd. Dit ontwerpproces bestaat uit drie fasen: een gezamenlijke architecturale fase, een domein-specifieke implementatie fase en een integratiefase. Om de ontwerptijd en ontwerpkosten te reduceren zal men bovendien opteren voor een parallel ontwerpproces waarbij de verschillende ingenieursdomeinen in parallel (een deel van) het systeem implementeren.

Helaas merken we dat het ontwerpproces wordt gekenmerkt door dure, iteratieve ontwerpcycli, deels vanwege de betrokkenheid van verschillende ingenieursdisciplines die elk een andere zienswijze hebben op het te ontwerpen systeem. Deze heterogeniteit leidt vaak tot inconsistente ontwerpbeslissingen op gedeelde ontwerpparameters wat leidt tot onverwacht gedrag van het geïntegreerde systeem. In de literatuur wordt de *Contract-Based Design*-methode voorgesteld om consistentie tussen verschillende ontwerpartefacten te garanderen. Daartoe wordt een contract tussen verschillende ingenieurs gedefinieerd voorafgaand aan de ontwerpfase. Het contract bestaat uit een verzameling van aannames en garanties die beschrijven onder welke voorwaarden een systeem zal voldoen aan een verzameling van gewenste eigenschappen. De *Contract-Based Design*-methode blijkt echter enkel consistentie te kunnen garanderen wanneer de betrokken ingenieurs eenzelfde zienswijze op het systeem hanteren. In de context van co-design tussen het controle domein en het embedded domein, breidt deze dissertatie de huidige methode uit door de principes van de *Contract-Based Design*-methode te combineren met het ontologisch redeneren zodat consistentie tussen verschillende zienswijzen gegarandeerd kan worden.

De voorgestelde methode zorgt niet alleen voor consistentie tussen ingenieurs met een verschillende zienswijze op het systeem, maar maakt het ook mogelijk om ontwerpparameters te vertalen tussen ingenieursdomeinen. Dit blijkt nuttig aangezien ingenieurs vaak moeite hebben om te redeneren over elkaars domein. Dit kan als gevolg hebben dat simulatiemodellen, gebruikt om de verschillende aspecten van het systeem te verifiëren en te valideren, het gedrag van het geïntegreerde systeem niet correct voorstellen. Wij stellen de *Round-Trip Engineering* methode voor die een halfautomatische annotatie van de simulatiemodellen mogelijk maakt, zodat eigenschappen die hun oorsprong hebben in een ander domein en een invloed kunnen hebben op het systeemgedrag mee in rekening kunnen worden genomen.

Aangezien contracten worden gedefinieerd in een onderhandelingsfase voorafgaand aan de (parallelle) implementatie van het systeem, beperken zij de ontwerpruimte tot een eindig aantal mogelijke implementaties en integraties. Aldus kunnen *Design-Space Exploration*-technieken worden toegepast om ingenieurs te ondersteunen bij het ontwerpen en integreren van het systeem. In deze dissertatie zullen wij dan ook aantonen hoe contracten *Design-Space Exploration* mogelijk maken. Verder wordt een aanzet gegeven naar een catalogus van patronen waarin verschillende *Design-Space Exploration*-technieken, gebruikt in een *Model-Based Systems Engineering* context, worden gecategoriseerd.

De complexiteit van ontwerpprocessen waarbij ingenieurs een verschillende zienswijze hebben op het te ontwerpen systeem en de praktische aspecten van onze methoden worden aangetoond aan de hand van een academisch voorbeeld. We tonen ook aan hoe onze methoden en technieken kunnen geïntegreerd worden in parallelle ontwerpprocessen van cyber-fysische systemen. We focussen ons in deze dissertatie enkel op de interacties tussen de controle en embedded ingenieursdomeinen. De gepresenteerde methoden en technieken kunnen echter worden toegepast in elk ontwerpproces waarbij meerdere domeinen, elk met een verschillende zienswijze op het systeem, moeten samenwerken.

List of Publications

The following list of peer-reviewed papers and technical reports, where I was a co-author, serve as a basis of this dissertation:

— Ken Vanherpen, Joachim Denil, Paul De Meulenaere, and Hans Vangheluwe. Ontological Reasoning as an Enabler of Contract-Based Co-design. In *6th International Workshop on Cyber Physical Systems. Design, Modeling, and Evaluation (CyPhy)*, pages 101-115, 2017. The idea of using contracts was proposed by Ken and Joachim after some great, yet intensive, brainstorm session. It was further refined by Ken who also wrote the paper. Hans, Paul and Joachim commented on the work and reviewed the draft of the paper.

— Ken Vanherpen, Joachim Denil, István Dávid, Paul De Meulenaere, Pieter Johannes Mosterman, Martin Törngren, Ahsan Qamar, and Hans Vangheluwe. Ontological Reasoning for Consistency in the Design of Cyber-Physical Systems. In *1st International Workshop on Cyber-Physical Production Systems (CPPS)*, pages 1-8, 2016. The idea of relating design parameters to ensure consistent design originates from Ken and was further refined during discussions with Hans, Paul, Joachim, Martin, Ahsan. Ken defined the foundational patterns and wrote the majority of the paper. Joachim and István assisted in writing. Hans commented on the work and reviewed the draft of the paper.

— Ken Vanherpen, Joachim Denil, Hans Vangheluwe, and Paul De Meulenaere. Model Transformations for Round-Trip Engineering in Control Deployment Co-Design. In *Proceedings of the 5th International Workshop on Model-driven Approaches for Simulation Engineering (Mod4Sim) as part of the Spring Simulation Multi-Conference (SpringSim)*, pages 820-827, 2015. As the presented technique builds upon earlier work of Joachim, he proposed the idea. Ken implemented the technique, while attention was being paid in terms of speed optimization, and wrote the paper. Paul, Hans and Joachim commented on the work and reviewed the draft of the paper.

— Ken Vanherpen, Joachim Denil, Paul De Meulenaere, and Hans Vangheluwe. Design-Space Exploration in Model-Driven Engineering: an Initial Pattern Catalogue. In *Proceedings of the 1st International Workshop on Combining Modelling with Search- and Example-Based*

Approaches (CMSEBA) co-located with 17th International Conference on Model Driven Engineering Languages and Systems (MODELS 2014), *CEUR Workshop Proceedings, vol. 1340*, pages 42-51, 2014. Based on Ken's exploratory work on Design-Space Exploration, and in particular constraint-based approaches using Alloy, Hans and Joachim proposed the idea. It was refined during discussions between Ken and Joachim. Ken implemented the constraint-based approach and wrote the majority of the paper. Joachim implemented the genetic algorithm and rule-based approach while assisting in writing. Hans and Paul commented on the work and reviewed the draft of the paper.

— Ken Vanherpen, Joachim Denil, Paul De Meulenaere and Hans Vangheluwe. Design-Space Exploration in Model-Driven Engineering: an Initial Pattern Catalogue. *School of Computer Science, McGill University, Technical Report, SOCS-TR2014.4*, 2014. Ken extended the workshop paper with the case study. He was assisted by Joachim for the implementation of the case study. Joachim commented and reviewed the report.

The following list of peer-reviewed papers and technical reports, where I was a co-author, are not included within this dissertation:

— István Dávid, Bart Meyers, Ken Vanherpen, Yentl Van Tendeloo, Kristof Berx, and Hans Vangheluwe. Modeling and Enactment Support for Managing Inconsistencies in Heterogeneous Systems Engineering Processes. In *Proceedings of the 2nd International Workshop on Collaborative Modelling in MDE (COMMitMDE) co-located with ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MoDELS)*, *CEUR Workshop Proceedings, vol. 2019*, pages 145-154, 2017.

— István Dávid, Eugene Syriani, Clark Verbrugge, Didier Buchs, Dominique Blouin, Antonio Cicchetti, and Ken Vanherpen. Towards Inconsistency Tolerance by Quantification of Semantic Inconsistencies. In *Proceedings of the 1st International Workshop on Collaborative Modelling in MDE (COMMitMDE) co-located with ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MoDELS)*, *CEUR Workshop Proceedings, vol. 1717*, pages 35-44, 2016.

Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

A/G	Assume/Guarantee
ADL	Architecture Description Language
API	Application Program Interface
AUTOSAR	AUTomotive Open System ARchitecture
B2B	Back-to-Back
CAN	Controller Area Network
CAS	Computer Algebra System
CBCD	Contract-Based Co-Design
CBD	Contract-Based Design
Co-Design	Concurrent Design
CPS	Cyber-Physical System
CPSs	Cyber-Physical Systems
CWA	Closed World Assumptions
DSE	Design-Space Exploration
DSL	Domain-Specific Language
ECU	Electronic Control Unit
ECUs	Electronic Control Units
EMF	Eclipse Modeling Framework
FTG+PM	Formalism Transformation Graph and Process Model
GUI	Graphical User Interface
HHD	Hybrid Hydraulic Drivetrain
HHV	Hybrid Hydraulic Vehicle
HiL	Hardware-in-the-Loop

HPF	High Pass Filter
HPP	High Performance Processor
HRC	Heterogeneous Rich Components
HVAC	Heating, Ventilation and Air Conditioning
ICE	Internal Combustion Engine
IoT	Internet of Things
IPL	Integration Property Language
IT	Information Technology
LHS	Left-Hand Side
LPF	Low Pass Filter
LPP	Low Performance Processor
LTM	Linguistic Type Model
MA	Multi-Abstraction
MARTE	Modeling and Analysis of Real-Time and Embedded systems
Massif	MATLAB Simulink Integration Framework
MBSE	Model-Based Systems Engineering
MDE	Model-Driven Engineering
MiL	Model-in-the-Loop
MOEA	Multi-Objective Evolutionary Algorithm
MPM	Multi-Paradigm Modeling
MS	Multi-Semantics
MV	Multi-Viewpoint
NAC	Negative Application Condition
NEDC	New European Driving Cycle
NSGA	Non-dominated Sorting Genetic Algorithm
OEM	Original Equipment Manufacturer
OS	Operating System
OSLC	Open Services for Lifecycle Collaboration
OTM	Ontological Type Model
OWA	Open World Assumptions
OWL	Web Ontology Language

PBD	Platform-Based Design
PiL	Processor-in-the-Loop
PM	Process Model
PN	Petri-Net
PSA	Peugeot Société Anonyme
pv	performance values
PWC	Power Window Control
PWM	Pulse-Width Modulation
RDF	Resource Description Framework
RHS	Right-Hand Side
RTE	Round-Trip Engineering
RTOS	Real-Time Operating System
SBO	Search-Based Optimization
SD	Semantic Domain
SDLC	Software Development Life Cycle
SiL	Software-in-the-Loop
StA	Sampling-to-Actuation
SWC	Software Component
SWCs	Software Components
SysML	Systems Modeling Language
TRL	Technology Readiness Level
UML	Unified Modeling Language
W3C	World Wide Web Consortium
WCET	Worst-Case Execution Time
WCRT	Worst-Case Response Time
XMI	XML Metadata Interchange

CHAPTER 1

Introduction

1.1 Context

For over 230 years, our society has been confronted with some (r)evolutions that have profoundly changed the manufacturing industry. These (r)evolutions are known as the three Industrial Revolutions. The first dates from the end of the 18^{th} century where mechanical production machines were powered by water and steam. At the end of 19^{th} century the second Industrial Revolution allowed for mass production by using electric powered assembly lines. In the late 60s of the previous century electronics and Information Technology (IT) enabled the third, and currently last, Industrial Revolution by automating the manufacturing process. Today, Cyber-Physical Systems (CPSs) are considered as one of the enablers of the upcoming fourth Industrial Revolution (i.e., Industry 4.0) by turning today's factories into smart factories [LFK+14]. As its name implies, CPSs are characterized by the integration of *Cyber* and *Physical* components, where the *Cyber* part relates control algorithms deployed on a (distributed) processing platform. These embedded systems are tightly coupled to the physical processes, often using feedback loops. Through this interaction, *Cyber* and *Physical* components influence each other which must be taken into account when designing the system. Besides being an enabler of the fourth industrial revolution, CPSs can already be found in a plethora of applications nowadays, such as automotive systems, aerospace, telecommunication, healthcare, energy distribution, climate control, robotics, etc. Given the expected revolutions in the field of manufacturing, Internet of Things (IoT), and autonomous systems, CPSs tend to take an even more prominent role in our daily lives.

The exponential rate at which CPSs evolve, while getting increasingly complex, poses new challenges in terms of their design. To tackle the increasing complexity, engineers already practice a Model-Based Systems Engineering (MBSE) methodology [Est08], in which models are used to support their requirements engineering, design, verification, and validation activities of a system beginning in the conceptual design phase and continuing throughout development and later life cycle phases [INC07]. These models typically operate at the same or different levels of abstraction within a single viewpoint. In literature, these abstraction levels are called horizontal and vertical abstraction, respectively [SPHP02, GNNS10]. Model transformations can be applied to manipulate models from one abstraction to another. They are typically used for code synthesis, integration, analysis, simulation, and optimization purposes. Multi-Paradigm Modeling (MPM) as a method consolidates these modeling methods and techniques, enabling engineers to model each aspect of the system explicitly at the most appropriate level(s) of abstraction using the most appropriate formalism(s), while modeling the development process(es) explicitly [MV04].

The heterogeneous nature of a Cyber-Physical System (CPS), however, requires a collaboration of teams with expertise in a particular engineering domain (e.g., control logic, embedded system design, and mechanics). In that respect, each team is considered as a stakeholder that has a particular interest, also called concern, in the system under design. For example, a team of control engineers is concerned about the response time of the designed control algorithm while a team of embedded engineers is concerned about the processor load. In this dissertation we refer to

these concerns as *properties*. They need to be satisfied by a (set of) *design parameters* that are defined when designing a domain-specific implementation of (a part of) the system. Continuing the example, the response time property can be satisfied by defining a control algorithm with a reasonable settling time while the processor load property is satisfied by choosing an appropriate sample rate for the software tasks running the control algorithm. Both settling time and sample rate are design parameters of the control and the embedded domain, respectively. In a MBSE context, a (set of) model(s) is used for designing, interpreting, and analyzing the particular view(s) that addresses a stakeholder's concern(s). In doing so, each stakeholder follows a set of guidelines, specific to its domain, that includes information on languages, notations, modeling methods, and analysis techniques used during the design process. This set of guidelines is formalized by the ISO/IEEE 42010 standard [ISO11] as a *viewpoint* and is adopted by the literature in [TQB+14, BLTT12]. Inferred from the description in [ISO11], Figure 1.1 illustrates how the concepts stakeholder, property, viewpoint, and view are related to each other.

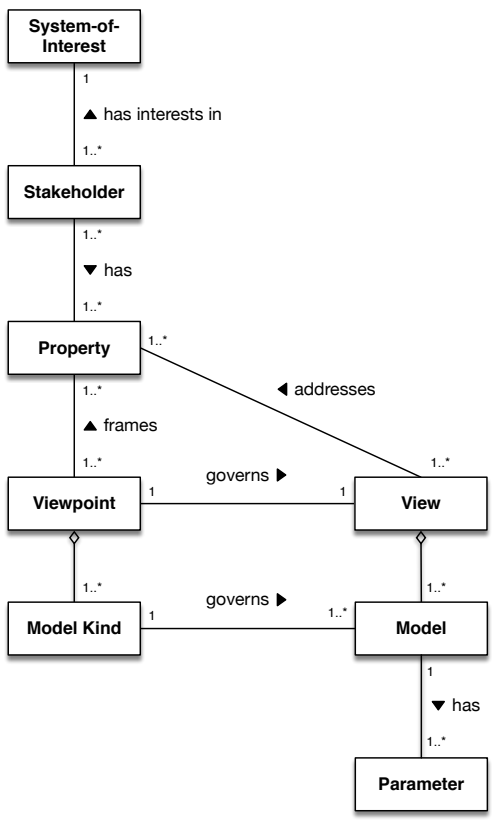


Figure 1.1: Conceptual overview of the relationship between *stakeholder*, *property*, *viewpoint*, and *view*, based on [ISO11]

The overview indicates that different viewpoints may exist within one single engineering domain. For example, within the embedded domain one viewpoint may be concerned with the architecture while another viewpoint is concerned with end-to-end timing. Within the scope of this dissertation, however, we focus on relationship between viewpoints of different engineering domains and, in particular, the inconsistencies that can arise between viewpoints. After all, requirements, representing the behavior of the real world system in a certain context, are –often implicitly– implemented by multiple stakeholders so that their respective properties (that needs to be satisfied by a set of domain-specific design parameters) are also related. Unawareness of these relationships may eventually lead to an inconsistent system.

We experience this in current CPS design processes in which engineers of both the control and the embedded domain derive properties that should hold for their view on a system with an implicit knowledge of each others domain. For a control engineer it is, for example, important to control a step motor with a precision of 0.5 degree. In this case, precision is a property for the control domain while degree is a design parameter. On the other hand, the embedded engineer might be concerned about the cost (a property) so that a processor with a limited Pulse-Width Modulation (PWM) resolution (a design parameter) is chosen. Because of the limited knowledge of the control engineers with respect to the embedded domain, they may overestimate available hardware resources such as the resolution of the PWM or, even worse, may not be aware of floating point approximations on the hardware platform. This might lead to a lower precision than what was assumed by the control engineer. As a result, integration of both viewpoints affects the performance of the system.

1.2 Motivation

The introductory example in the previous section demonstrates how both engineering domains (stakeholders) use incomplete assumptions of each other's view. Shared properties are not (fully) taken into account, such that consistency between design parameters 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.

Similar to the different levels of abstraction, (in)consistency is classified as *horizontal (in)-consistency* or *vertical (in)consistency*. Horizontal consistency pertains to models at the same (horizontal) level of abstraction. An example is the modeling 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 parameters (such as power). To be consistent, analysis of both models must always yield identical values for each of these parameters. On the contrary, the notion of vertical inconsistency refers to models, and related parameters, at a

different (vertical) level of abstraction. In this case, there might be no straightforward relation between the domain-specific parameters as they might be reasoning over the satisfaction of a different property or set of properties. For example, during the modeling of a control algorithm, a state machine is used at a higher level of abstraction to explore the possible states in which the control algorithm may operate. At a lower level of abstraction, behavior may be added to these states by defining some control logic. Vertical consistency should be maintained during the refinement process so that the satisfaction of (behavioral) properties of the abstract control model are maintained after refinement. As such, transactions between the different states might not be changed or added at the lower abstraction level.

Different techniques exist enabling engineers to manage (in)consistencies. They can be classified as techniques that (i) detect and resolve inconsistencies or (ii) avoid inconsistencies. For the former, the design process should be explicitly modeled so that it can be restructured by some consistency management tool if inconsistencies occur [DDGV16, DMV+17]. For the latter, Contract-Based Design (CBD) as a method [BCF+08, BCN+12, BCN+15a, BCN+15b] has been proposed to avoid inconsistencies between design artifacts (i.e., design parameters) prior to the detailed design phase of a design process, by allocating contracts to design components [SVDP12, DLTT13, TQB+14]. A contract classifies (domain-specific) design parameters as a set of assumptions and guarantees. They can be regarded as preconditions and postconditions of a component, respectively, defining under which conditions a system promises to operate satisfying desired parameters. When combining components, each typed by a contract, well-defined contract operators can be used to combine their respective contracts.

Contracts are defined during a negotiation phase prior to the implementation of the system in which domain-specific design parameters are balanced against each other, keeping in mind the stakeholder's concerns [DLTT13]. As such, CBD is considered as an enabler of Concurrent Design (Co-Design), guaranteeing both horizontal and vertical consistency during implementation.

1.3 Challenges and Contributions

When restricting to (i) viewpoints in a single domain (e.g., the behavioral and safety viewpoint when modeling a control algorithm) or (ii) multiple domains operating at the same level of abstraction while sharing a single viewpoint (e.g., the interaction between the model of the control algorithm and the model of the physics), CBD can indeed be applied as a method to avoid inconsistencies between design parameters. Designing CPSs involves, however, multiple domains each with their own viewpoint(s) on the system under design while operating at different levels of abstraction. In this case, the current CBD method, and associated theory, lacks in the ability to reason about consistency, limiting its use in heterogeneous (co-)design processes.

As discussed before, engineers operating in different domains, using different viewpoints, have

limited aids in estimating the impact of their design choices. Often they are not aware of the implicit relationships that exist between design parameters originating from different domains. As a consequence, it is infeasible for engineers to reason about consistency when negotiating contracts. The following research questions can therefore be formulated:

RQ1 *At what level of abstraction can we relate different domains, and their related design parameters, to enforce consistent design of CPSs?*

RQ2 *What is the most appropriate formalism to represent these relations?*

In the current state of the art it is already shown that contracts can be negotiated in a preliminary design phase [DLTT13] and that contract operators can be used to combine contracts [BCN+15a]. It is yet unclear, however, how contracts should be used in concurrent design processes to ensure inter-domain consistency. When applying the current state of the art in such design processes, it becomes clear that there is no strict relation between what is guaranteed by one domain and what should be assumed by another domain. This raises the following research questions:

RQ3 *How should contracts be used to ensure consistency between heterogeneous viewpoints, that is, viewpoints related to different domains)?*

RQ4 *How should contracts syntactically and semantically be interpreted by different engineering domains?*

By definition, contracts ensure consistent design of a system by negotiating the design parameters prior to its (concurrent) implementation. As such, the minimum and maximum boundaries of design parameters are known beforehand. Often, non-functional requirements (e.g., cost or energy) impose restrictions on the possibly infinite set of implementations. Knowing that Design-Space Exploration (DSE) techniques can be used to (semi-)automatically optimize the design and/or deployment of a system with respect to a (set of) non-functional requirement(s), the following research question remains:

RQ5 *Can contracts be used as an input for design and/or deployment optimization methods?*

A plethora of DSE techniques already exist to search for an optimal design, such as evolutionary algorithms, constraint satisfaction, and (Mixed Integer) Linear Programming. It is, however, not obvious for engineers to select the most appropriate technique for a given optimization problem. As such, the following research question is formulated:

RQ6 *How can we structurally organize the plethora of DSE methods to assist engineers in the evaluation of the most appropriate method for a given optimization problem?*

This dissertation addresses the listed research questions resulting in the following contributions:

— A method supporting a contract-based design approach for heterogeneous design processes. This includes the negotiation of a (set of) common contract(s), ensure consistency between

negotiated design parameters and automatically deriving viewpoint-specific contracts to enable concurrent design. Therefore, the method relies on the syntactic and/or semantic translation of design parameters from one viewpoint to another.

— Methods and techniques supporting engineers in the interpretation of the derived viewpoint-specific contracts and the (semi-)automatic exploration of an optimal design implementation.

— An initial pattern catalog categorizing the plethora of DSE techniques using a well-defined structure similar to software design patterns.

— An integrated framework enabling consistent concurrent design processes. The framework supports specifying domain architectures and contracts while model transformations can be used to create domain-specific views in dedicated tools. Its modular approach makes it possible to extend the framework for other domains and/or tools.

— Demonstration of the applicability of the methods and techniques, and the supported integrated framework, within a concurrent design process using an academic case study.

1.4 Delimitations and Assumptions

The listed contributions of this dissertation are achieved by limiting the research scope. We therefore list the assumptions that are made throughout the course of the research:

— Although the methods, techniques, and integrated framework presented in this dissertation can be applied to any multi-domain design problem, we only focus on the synergies between the control and embedded domain. As a design decision in one of these domains affects the intended behavior/implementation of the other domain, while using thoroughly different terminology, we consider that focusing on these domains is representative for a broad range of multi-domain design problems.

— With respect to control-embedded co-design, we focus on timing-related issues that may emerge when integrating the system, leading to unintended behavioral changes. We therefore make use of a task-driven, processor-based computation platform using a single core to execute the deployed system. The Real-Time Operating System (RTOS) is configured to schedule tasks using the fixed-priority preemptive scheduling algorithm.

— As part of the method, an ontology is used to explicitly model the domain knowledge. A significant amount of effort has been spent to correctly model the embedded domain so that it can be used in an industrial setting. A rather academic approach has been chosen to model the control domain; e.g., only one viewpoint is modeled. We therefore acknowledge that the modeled control knowledge requires some rework before being usable on an

industrial scale.

— Some of the techniques that have been developed during the course of this research use and/or extend previous work of members of the CoSys-Lab and MSDL research groups. The framework, integrating the presented method and techniques, is solely used to demonstrate the usability of our fundamentally new approach in an academic context. It is therefore inevitable that the framework should be refactored to achieve a sufficiently high Technology Readiness Level (TRL) [Man95, Man09] so that it can be used in an industrial context.

1.5 Case Studies

Throughout this dissertation, the *power window* is used as an academic case study to demonstrate the feasibility of the presented method and techniques. Their integration within the framework, and as such the scalability of our fundamentally new approach, is demonstrated using the *Hybrid Hydraulic Vehicle (HHV)* as a sufficiently complex industrial case study.

1.5.1 The Power Window

Power window systems are typically found in automobiles enabling an occupant to lower and raise a window by pressing a button or switch. Although the primary operation and implementation was rather simple, additional features have been added by automotive manufacturers to improve occupants' comfort and security. In addition, a number of (deadly) incidents [BBR+06] have led to stricter safety regulations imposed by various government agencies [Dep09]. For example, the automatic reversal of the upward movement when an object gets stuck between the window and the door frame. This resulted in increasingly complex systems, forcing manufactures to use software to enable the operation of such systems. Based on the set of functional requirements described in [PM04], combined with safety regulations of [Dep11], we define the requirements for the power window case study as follows:

1. A window shall start moving within 200 ms after a command is issued.
2. A window shall automatically move to a final position when the up or down command is issued for less than 500 ms.
3. A window shall be fully opened or closed within 4.5 s.
4. When closing a window, a force of no more than 100 N may be present.
5. The detection of an object when closing a window should result in lowering the window by approximately 12.5 cm.

6. The operations of the driver have priority in case a passenger window is simultaneously operated by the driver and a passenger.

In this dissertation we consider a power window system containing two windows, one at the driver side and one at the passenger side. Its functional control architecture is shown in Figure 1.2. As mechanical buttons tend to generate an unstable signal when pressed, input signals from the driver (drvCmd) and the passenger (psgCmd) are debounced by the *Debounce* component. Using a bus, the debounced signals are transferred to the Power Window Control (PWC) component of the driver or to a control exclusion component. The latter ensures that the driver's operations have priority over those of the passenger. Both PWC components, one for each window, contain the control logic to determine the operation of the window. To detect a pinched object, the motor's requested power is measured using a current sensor; i.e., due to reactive force of the pinched force on a closing window, the requested power will increase. An end of range detection (EODR) detects that the closing window slides into the rubber at the top of the door, such that the control logic may ignore the higher current request (i.e., the reactive force). Note how each component contains a trigger input port to indicate that the component is periodically executed.

A power window system is an elementary illustration of a CPS: a window (physical component) actuated by software running on an processing device (cyber component). Using a feedback loop between the physical and cyber component (not shown in Figure 1.2), safety related information about the operation of the window can be taken into account by the software. As there are often multiple power windows in today's vehicles, a communication channel is used to transmit information from the various controllers used to sense and actuate the windows. Given the limited, yet sufficient, complexity of the system, the power window system is considered as an excellent academic case study [MV04, PM04, Den13, LDMH17].

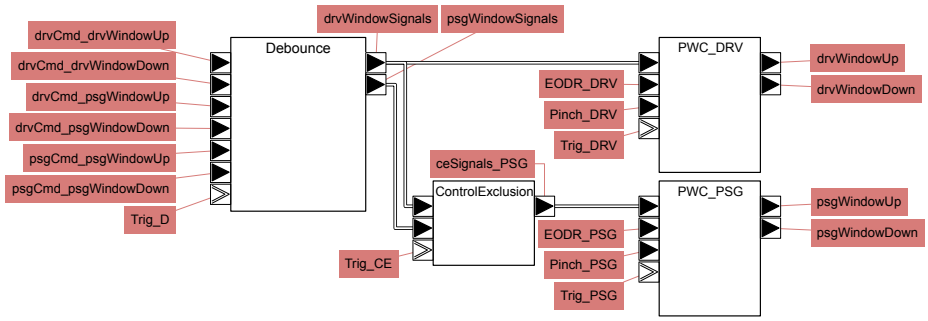


Figure 1.2: Functional control architecture of the power window system

1.5.2 The Hybrid Hydraulic Vehicle

The HHV industrial case study is based on a concept that has been devised by Peugeot Société Anonyme (PSA) for the development of a Hybrid Hydraulic Drivetrain (HHD) for a small passenger car (e.g., Citroën C3). The topology of the drivetrain, reverse engineered from available PSA patents [RR13], consists of two different power sources: (i) a conventional Internal Combustion Engine (ICE) and (ii) a hydraulic system. A schematic overview of the complete HHD and its components is given in Figure 1.3. The mechanical part of the drivetrain is illustrated in blue on the right-hand side, while the hydraulic components are shown in orange on the left-hand side.

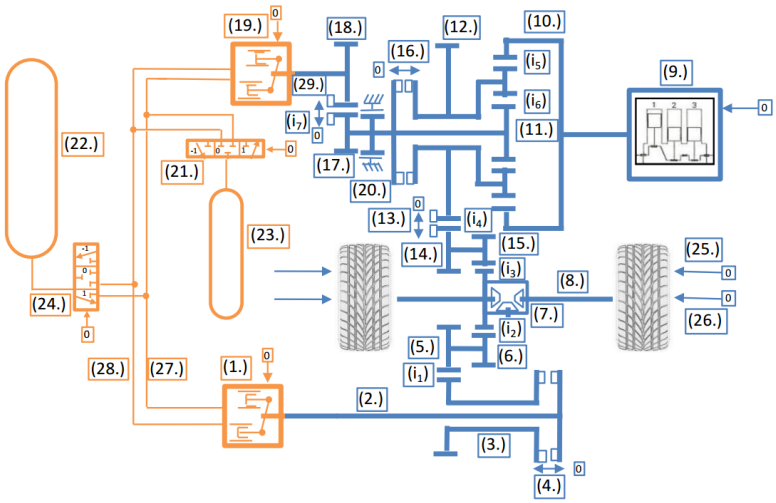


Figure 1.3: Conceptual overview of the Hybrid Hydraulic Drivetrain, based on [RR13]

According to PSA [SM12], this type of topology has a great potential to (i) optimize the operation of the ICE, (ii) reduce the fuel consumption and as such (iii) reduce a car's emissions. In order to achieve these improvements, a control algorithm must ensure that the ICE is operating within its optimum efficiency region [SM12, GY14]. Depending on the requested torque and the speed of

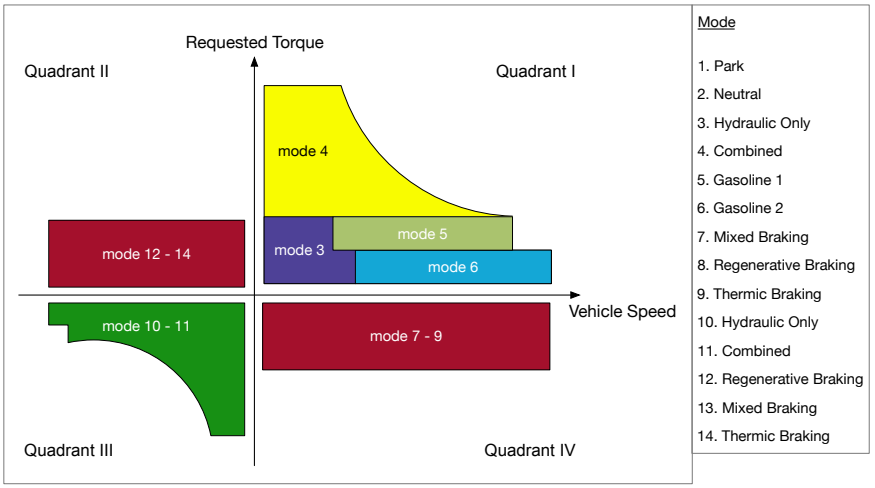


Figure 1.4: Driving modes of the Hybrid Hydraulic Vehicle, based on [SM12, GY14, Nas14]

the vehicle, 14 driving modes can be distinguished. They are categorized by four quadrants as shown in Figure 1.4. Quadrant I and quadrant IV categorize the forward driving modes, while quadrant II and III categorize the backward driving modes. In this dissertation, we limit ourselves to the forward operating modes. We distinguish four main modes in which the HHV can operate in forward driving, namely:

— Hydraulic mode—transmits the energy stored in the accumulator via the hydraulic motor and is typically used for urban driving (where low torque and low speed is required).

— Combined mode—uses power-split technology where a fraction of the power is transmitted through the hydraulic circuit and the remaining power through the mechanical parts. This mode is used when higher torque is requested by the driver, typically for acceleration and uphill driving. The ICE's power is distributed in two directions: (i) directly to the wheels (differential) and (ii) to the hydraulic motor passing through the hydraulic system.

— Conventional mode—power to the wheels is provided by the ICE, operating at lower engine speed in order to optimize fuel efficiency. Therefore, two conventional working modes (mode 5 and 6) are defined, where the only difference between the two is the gear ratio. Depending on the requested torque, the mode is selected for which the engine operates in its most efficient region so that fuel consumption is minimized.

— Braking mode—kinetic energy is stored in the accumulator while decelerating. If necessary, the thermal brakes can be activated to reduce the braking distance (e.g., in case of an emergency stop).

Based on the descriptions of PSA in [SM12, RR13, RG14, GY14], we define the requirements for the HHV case study as follows:

1. The HHV must achieve preset performance needs
   (a) A top speed of 180 km/h
   (b) A 0-100 km/h acceleration time of maximum 12.4 s (with a minimum acceleration of $2m/s^2$)
   (c) Throttle response must be less than 0.5 s
   (d) Braking distance for 100-0 km/h must be lower than 54 m
2. The HHV must achieve preset autonomy specifications
   (a) A minimum range of 700 km with a full gasoline tank (New European Driving Cycle (NEDC) [Uni])
   (b) Able to accelerate for 10 s on maximum hydraulic power only
3. The HHV must operate within the 4 quadrants of the Torque-Speed curve of Figure 1.4
   (a) Stand still in park mode
   (b) Ability to drive forward and backward
   (c) Capable of regenerative braking
   (d) Able to drive in hydraulic mode, combined mode, or ICE only mode

Figure 1.5 illustrates the functional control architecture that models the behavior of the HHV system. We distinguish between four main control components: *Driver Intention*, *High-Level Control*, *Low-Level Control* and *Dashboard*. The *Driver Intention* component determines the quadrant in which the driver intents to operate and how much torque is requested. Based on this input, the *High-Level Control* component, in which the graph of Figure 1.4 is modeled, determines the mode number and how much power is requested from the hydraulic motor and/or the ICE. The final actuation of the HHD components is orchestrated by the *Low-Level Control* component. The *Dashboard* component visualizes vehicle information which is relevant for the driver. Again, each component contains a trigger input port to indicate that the component is periodically executed.

As with the power window, the HHV contains all the elements of a CPS: a drivetrain (physical component) connected to a controller (cyber component) using a feedback loop such that the software is able to determine the most appropriate operating mode. Given the complexity of the system, a distributed hardware architecture is opted in which multiple controllers transmit information using a communication channel.

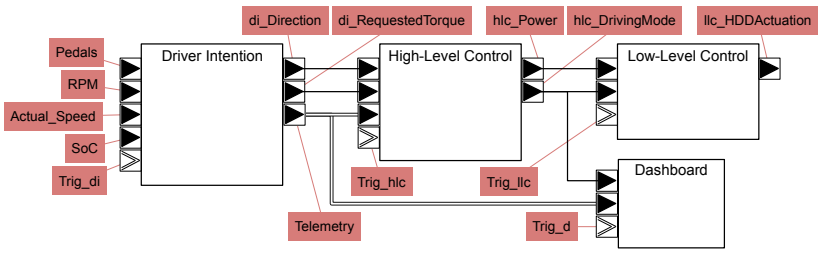


Figure 1.5: Functional control architecture of the hybrid hydraulic vehicle system

1.6 Outline of the Dissertation

The remainder of the dissertation systematically elaborates on our contributions. We illustrate the applicability of contributions by mean of the power window case study in Chapters 3 to 5, while the HHV case study is used in Chapter 6.

Chapter 2 gives a short overview of techniques, methods, and tools that are necessary to understand the novel method and techniques discussed in this dissertation.

Chapter 3 aims to answer **RQ1**, **RQ2**, and **RQ3**. The current state of the art regarding CBD, and its applicability in Co-Design processes, is discussed by means of the power window case study. The foundational concepts of our proposed method are introduced while we elaborate on explicitly modeling domain knowledge and how the method facilitates multi-viewpoint design processes.

Chapter 4 addresses **RQ4** by supporting the proposed method with a technique that (semi-)automatically augments the modeled control algorithm with timing-related deployment information stemming from the embedded domain.

Chapter 5 elaborates on **RQ5** by exploring how contracts may guide the search for an optimal design using DSE techniques. Furthermore, **RQ6** is addressed by initiating an initial pattern catalog categorizing different DSE techniques.

Chapter 6 illustrates how the integrated framework, consolidating the presented methods and techniques introduced in Chapters 3 to 5, can be used throughout the design process of a CPS; that is, a Hybrid Hydraulic Vehicle.

Finally, Chapter 7 concludes our contributions and elaborates on potential future research directions.

CHAPTER 2

Background

Contributions in the scientific domain often rely on a set of state-of-the-art techniques and methods. In what follows we elaborate on the techniques and methods, and their supported tools, that are relevant within the scope of this dissertation. Note that the descriptions are based on the cited sources.

2.1 Techniques

2.1.1 Modeling Languages

For the development of CPSs, engineers increasingly rely on models to represent different aspects of the system. In that respect, a model is an abstraction of (a part of) the real world system to be developed. These models enable engineers to verify and validate the properties of a system in a virtual environment using simulation techniques. In other words, engineers are able to verify whether requirements are met even before the system is built. When models are consistently used throughout the engineering process for verification and validation purposes, a Model-Driven Engineering (MDE) process is practiced in which each aspect of the system is explicitly modeled at the most appropriate level(s) of abstraction using the most appropriate formalism(s), while modeling the development process(es) explicitly. In the literature, this modeling approach is referred to as MPM [MV04] in which engineers rely on a Domain-Specific Language (DSL) to model (a part of) the system.

Each model in an engineering process is characterized by three main aspects [VSB04, Kle07]: its abstract syntax, its concrete syntax, and its semantics. The concrete syntax of a modeling language describes how a model is represented [CDLOP02, VTVMMV17]. This can either be visually (using icons), textually (using characters), by means of audio, or a combination of them. The abstract syntax preserves the essence of the concrete syntax. It defines how a modeling language is structured by means of a metamodel [Küh06] or a grammar for visual or textual languages, respectively. An instance of the modeling language (i.e., a model) is said to conform to the metamodel of the language. The semantics of a modeling language describes the meaning of a model [HR04]. In the literature, a distinction is made between operational (how a model is executed) and denotational semantics (how a model is represented in another language). A semantic mapping function explicitly describes how each model element is mapped onto an element of the semantic domain. In case of operational semantics, a mapping function might map a model to a (set of) behavioral trace(s). For denotational semantics, the mapping function maps every model element to another model element.

In this dissertation, the abstract syntaxes are described by means of a metamodel using class diagrams [RJB04]. For the concrete syntaxes both visual and textual representations are used. For the former, we opted for a connection-based syntax [CDLOP02] in which icons are connected to each other using lines. For the textual representation, a geometric-based syntax [CDLOP02]

is used in which textual characters are spatially arranged. With respect to the semantics of the models, both operational and denotational semantics are used.

2.1.2 Model Transformations

Model transformations play a key role in MDE to manipulate models and are even regarded as the "heart and soul of model-driven software and system development" [SK03]. They are typically used for code synthesis, integration, analysis, simulation, and optimization purposes.

Transformations are defined between the metamodels of a source model and the target model. If the source metamodel is different from the target metamodel, the transformation is classified as an exogenous transformation. If not, it is called an endogenous transformation. While transformations are defined at a higher abstraction level (i.e., the meta-level), they are executed at the model level where a distinction is made between in-place and out-place transformations. The former classification refers to a transformation that is executed within the model. If the result of a transformation is a new model, it is classified as an out-place transformation. A more detailed taxonomy of model transformations can be found in [MG06].

Different approaches can be taken to specify model transformations. Czarnecki and Helsen classify them by means of a feature model in [CH06]. At the top-level they distinguish between model-to-text and model-to-model transformation approaches. Although Czarnecki and Helsen detail the different approaches for each category, we elaborate on the approaches used throughout this dissertation. For both model-to-text and model-to-model transformations, a template-based approach is used. A template contains a set of rules, each of them typed by a Left-Hand Side (LHS) and Right-Hand Side (RHS). The LHS contains logic to read model elements of the source model, while the RHS consists of string patterns to construct the target model. The rule can optionally be extended with a Negative Application Condition (NAC) that constrains the LHS. The order in which the rules are executed is made explicit, resulting in operational transformations. With respect to model-to-model transformations, this dissertation complements the operational transformations with a declarative approach for which graph transformations are used [Sch95]. In that respect, both LHS and RHS of the transformation rule are expressed as a graph. When a match is found between the graph of the model and the LHS, the model's graph is transformed according to the graph structure defined in the RHS of the rule. The benefit of this approach is that the graph structures of the rule can be rendered in the modeling language. It allows users to define transformation rules without the need for expert transformation knowledge.

2.1.3 Design-Space Exploration

Model transformation are often used to explore a (possibly infinite) design space for an optimal solution for a particular design problem. DSE is an automatic process where possible alternatives

of a particular design problem are explored. The exploration is guided with imposed constraints and optimality criteria on the different candidate solutions.

Depending on the optimization problem, different search techniques can be used. For rather small problems, often an exhaustive search technique is used that explores the entire design space for feasible solutions. A less computationally intensive variant is the random search technique that only creates a predefined set of solutions of which the best one is selected. Using (meta-)heuristics, hill climbing searches for a more optimal solution, with respect to a given goal function, by incrementally modifying a single solution. Hill climbing techniques often require users to translate the problem to a generic search model. Using graph based rules, rendered in the user's modeling language, a set of model transformation patterns can be scheduled to optimize an original model. Again, meta-heuristics can be used to guide the model transformation schedule. As one may notice, exploring design spaces often requires one to balance between computational time and finding the optimal solution. In [Den13] a DSE technique is presented that prunes the design space more efficiently by combining multiple DSE techniques.

Within the scope of this dissertation the multitude of DSE techniques are further explored and categorized to assist engineers in selecting the most appropriate technique for their problem. Furthermore, the hill climbing technique is used the explore for the optimal deployment of a set of software components on a hardware platform.

2.1.4 Formalism Transformation Graph and Process Model

As already stated, MPM intends to model everything explicitly at the most appropriate level(s) of abstraction using the most appropriate formalism(s), while modeling the development process(es) explicitly [MV04]. In that respect, the Formalism Transformation Graph and Process Model (FTG+PM) is proposed in [LMD+12, LMD+13] as a language for modeling the workflow of a design process. The language enables one to explicitly model the used formalisms and how they are related to each other. As the name implies, the FTG+PM consists of two parts. The FTG part models all the involved formalisms (rectangles) and their relations using model transformations (circles). These can be either manual or (semi-)automatic transformations. The PM part models the workflow of the design artifacts (rectangles) and how they are processed throughout the design process (rectangles with rounded corners).

An example of an FTG+PM, describing the (partial) process of generating code from a control model, is shown in Figure 2.1. The FTG part is shown on the left-hand side of the FTG+PM, while the PM part is shown on the right-hand side. Given a set of requirements, often expressed in a textual formalism, the process starts by defining the specifications of the control algorithm. These specifications allow control engineers to model the control algorithm in their preferred (set of) formalism(s). The closed loop behavior of the control model is verified by means of Model-in-the-Loop (MiL) simulation, that requires a model of the physical system (i.e., a plant model). Often,

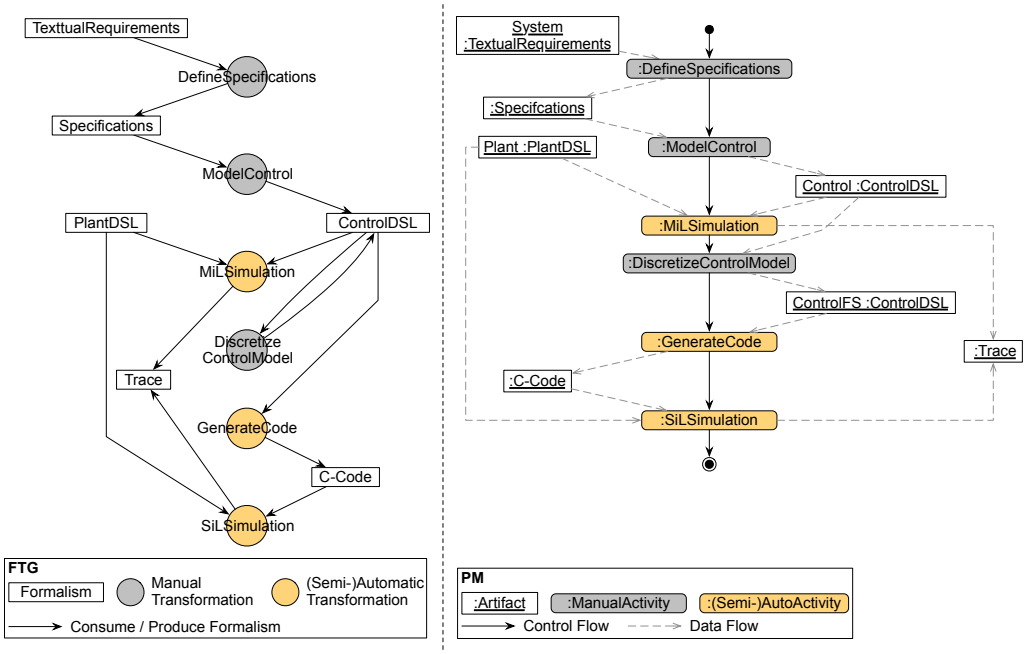


Figure 2.1: Example of a FTG+PM for generating code from a control model

control engineers prototype their control algorithms in the continuous-time domain. However, code is executed at a certain clock frequency, requiring control engineers to discretize their control algorithm. Once code is generated from the discretized model, a Software-in-the-Loop (SiL) simulation is executed in which the generated code is connected to the plant model. It is used to verify whether the logic of the code satisfies the specifications [MPE04].

Note that there exists a relation between the FTG and PM part. Artifacts and activities defined in the PM part are typed by the formalisms and transformations defined in the FTG, respectively. Also note that transformation and activities can be either manual or (semi-)automatic indicated by the grey or yellow colors, respectively. A more exhaustive use of the FTG+PM has been demonstrated by Mustafiz et al. [MDLV12] by means of a deployment case study in the automotive domain.

Within the scope of this dissertation, the FTG+PM is used to explicitly model the interactions between the control and embedded domain when designing cyber-physical systems. Furthermore, the FTG+PM is used to support the categorization of the different DSE techniques.

2.2 Methods

2.2.1 Model Based Systems Engineering

Due to the ever increasing complexity of systems, engineering activities are shifting from a document-centric to a model-centric process in which models are used as an integral part of the technical baseline that includes the requirements, analysis, design, implementation, and verification of a capability, system, and/or product throughout the acquisition life cycle [Mod11]. We refer to the term MBSE if these models are used to support system requirements, design, analysis, verification, and validation activities of a system beginning in the conceptual design phase and continuing throughout development and later life cycle phases [INC07].

As designing complex systems involves engineers from various disciplines, engineers follow a set of guidelines to ensure a product implements the given system requirements. The order in which the guidelines are followed is called the design process. Inspired by the software engineering community, that has defined Software Development Life Cycle (SDLC) models over the past decades [Rup10], we distinguish three models that are commonly used when designing CPSs: the waterfall model, the V model, and the agile model. They are conceptually shown in Figure 2.2.

The waterfall model is typed by a sequence of design steps in which a step can only be started if the previous is finished. Therefore, this model is considered when the project is not subject to changes. Otherwise, the design process must start over which may be costly in large engineering projects. Similar to the waterfall model, the V model is typed by a sequential design process. However, the V model distinguishes between *design* and *verification & validation* steps, respectively on the left-hand side and right-hand side of the V model, while keeping them tightly coupled as illustrated by the horizontal dashed lines. While going from high-level requirements to a low-level implementation, the specifications of the design are verified by executing acceptance tests, ranging from low-level unit tests to high-level system tests. Ultimately, at the top of the V model, it is validated whether the system satisfies the requirements. Although the verification steps are executed while designing the system so that late detection of implementation errors can be avoided, a changing product requirement still requires one to redesign (parts of) the system which, again, may be costly. Using an iterative design process, agile design processes avoid these costly redesigns. Their process model is typed by so called sprints (i.e., design iterations) in which a finalized product is delivered at the end of each cycle. Subsequent design cycles extend the previous prototype by implementing more requirements. As such, agile design processes are suitable when requirements are subject to change. On the downside, the lack of a thorough system analysis may result in a more complex, more expensive, design process. In general, we can conclude that each design process is typed by:

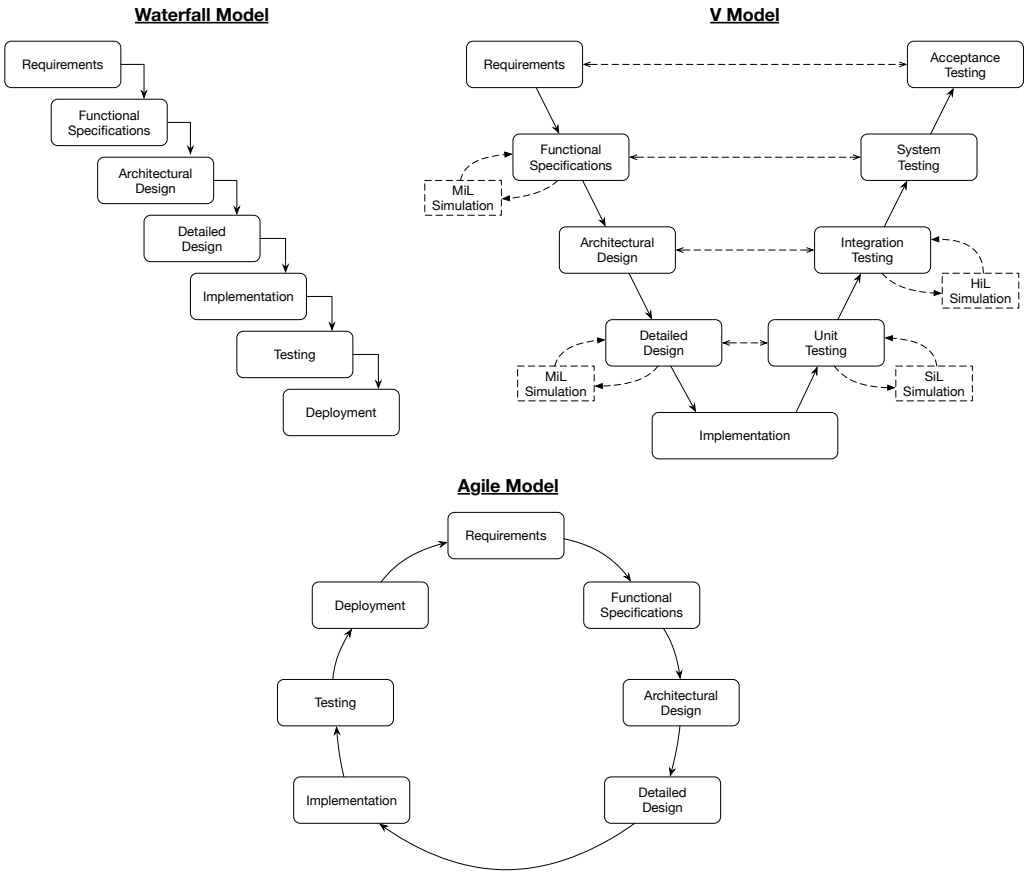


Figure 2.2: Conceptual overview of commonly used MBSE design processes

1. A specification phase: the set of system requirements is refined to a set of technical requirements called specifications.
2. A design phase: implementation of the specifications by the engineer(s).
3. An integration phase: integration of the implementation(s) and/or components in order to construct the system.
4. A verification/validation phase: executed during both the design phase and integration phase to verify whether specifications and requirements are met.

Nevertheless, in the context of designing (large) complex systems using a MBSE approach, the V model is widely accepted as the industry standard for designing and testing a CPS [Est08]. As such, we provide an example of a (possible) enactment of its conceptual representation (at the

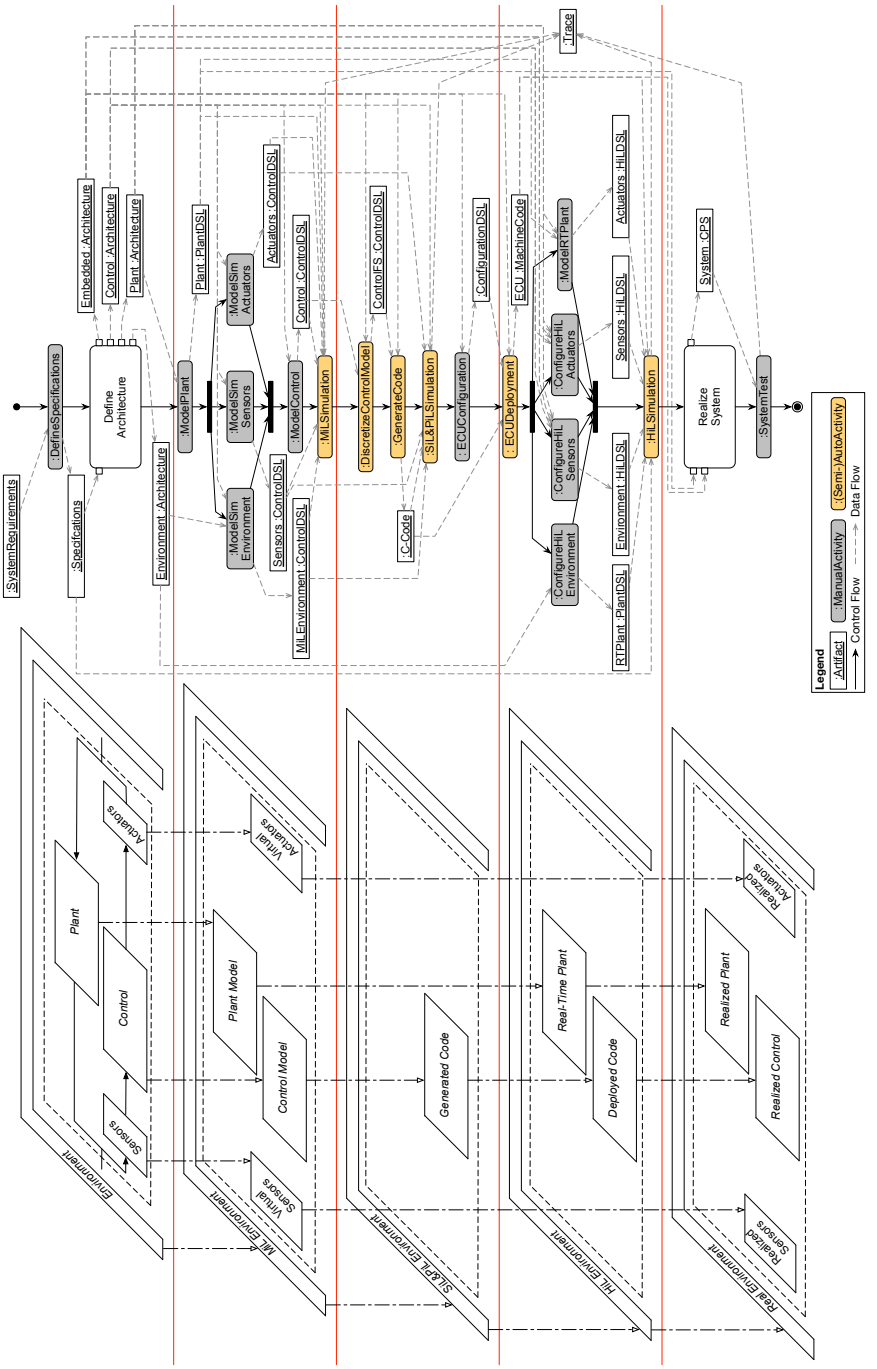


Figure 2.3: Possible enactment of a sequential design process using a MBSE approach

top right in Figure 2.2) by means of the Process Model (PM) shown at the right-hand side of Figure 2.3 while representing how the design process relates to the conceptual evolution of the system (left-hand side of Figure 2.3).

The design process starts off with a set of system requirements. Given these system requirements, the multi-disciplinary engineering team defines a set of technical requirements called functional specifications. They continue by discussing a global architecture whereby the models and peripherals (sensors, actuators, and environment) are considered as black-boxes so that signal interfaces can be defined. Subsequently, viewpoint-specific architectures, being subsets of the global architecture, are deduced. Viewpoints are typically associated to an engineering domain, so that there exist as many architectures as there are engineering domains. Focusing on control and embedded design in this dissertation, an embedded and control architecture are deduced from the global architecture. The viewpoint-specific architectures are used by each engineering domain while designing (a part of) the system. For example, the embedded architecture is used by the embedded engineer to configure the Electronic Control Unit (ECU) while the control architecture is used by the control engineer to design the control algorithm.

The implementation of the system typically starts with the design of the control algorithm. Therefore, control engineers rely on a plant model (i.e., a model of the physics) such that the behavior of the modeled control algorithm can be verified through a MiL simulation. In other words, a MiL simulation is executed to verify whether the specifications are met. If this is not the case, the control engineer will have to (partially) redesign the control algorithm. For reasons of simplicity, this decision is not shown in the PM. In case the simulation results fulfill the specifications, the control algorithm is handed over to the embedded engineer who prepares the control algorithm for deployment. This includes selecting a fixed-step solver, a sample rate and, optionally, converting floating-point representations to fixed-point. Given the final configuration details, code is generated from the modeled control algorithm and verified by executing a SiL simulation. Prior to the deployment, a Processor-in-the-Loop (PiL) simulation is often executed to verify the compiler and to test the linker and loader [MPE04]. Again, a (partial) redesign and/or reconfiguration may be necessary if specifications are not met (not shown in the PM). Once the specifications are met, the embedded engineer configures the ECU. This includes the configuration of the drivers, the AUTomotive Open System ARchitecture (AUTOSAR) stack, the RTOS, the communication matrix describing the messages on a communication medium, etc. Subsequently, a model-to-text transformation generates code from the modeled control algorithm so that it can be deployed on the ECU. Before interacting with the physical world, the ECU running the control algorithm is connected to a real-time platform executing the plant model so that a Hardware-in-the-Loop (HiL) simulation can be executed [MPE04]. Ultimately, the hardware operates in its intended environment controlling the physical system. By executing an acceptance test, one verifies whether the initial requirements are met.

Due to faster time to market and lower development costs demands, however, engineering teams tend towards the parallelization of design activities. To support these design processes, the

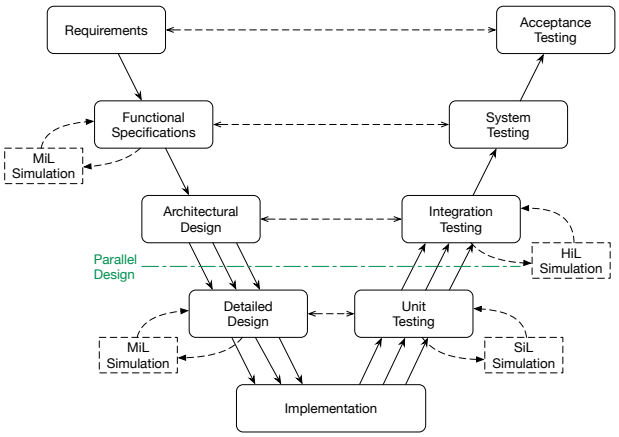


Figure 2.4: Conceptual overview of the V model for parallel design activities

VDI 2206 guideline [GM03, VDI04, Ise08] was issued that formalizes the necessary adaptions to the V model so that concurrent design can be enabled. As conceptually shown in Figure 2.4, viewpoint-specific workflows are parallelized once a high-level architecture is defined. This is denoted by the parallel arrows in Figure 2.4. As a viewpoint is typically associated with a certain engineering domain (e.g., control, embedded, mechanical), engineering activities are parallelized according to the involved engineering domains.

Again, we provide an example of a (possible) enactment of the parallel V model by means of the PM shown in Figure 2.5. As one may notice, the *detailed design* and *implementation* activities of the V model are situated between the subgraphs *Define Architecture* and *Realize System*. In contrast to a sequential design process, the development of the control model and the configuration of the ECU are conducted in parallel. A first synchronization between parallel design activities takes place once the control model is discretized and the ECU is configured. Given the final ECU configuration details, code is generated from the modeled control algorithm and verified using a SiL simulation. Afterwards, the software is cross compiled and verified on the target processor by executing a PiL simulation. If these simulation satisfy the specifications, the code is deployed by the embedded engineer who links the generated code with the ECU configuration. In parallel with the previous described design activities, a HiL simulation setup is configured. In that respect, the mechanical engineer prepares the plant model to run on a real-time target. Once the parallel design activities are finalized, a HiL simulation is executed to verify if the integrated system satisfies the initial specifications. Finally, the hardware operates in its intended environment controlling the physical system. By executing a system test, it is validated if the initial requirements are met.

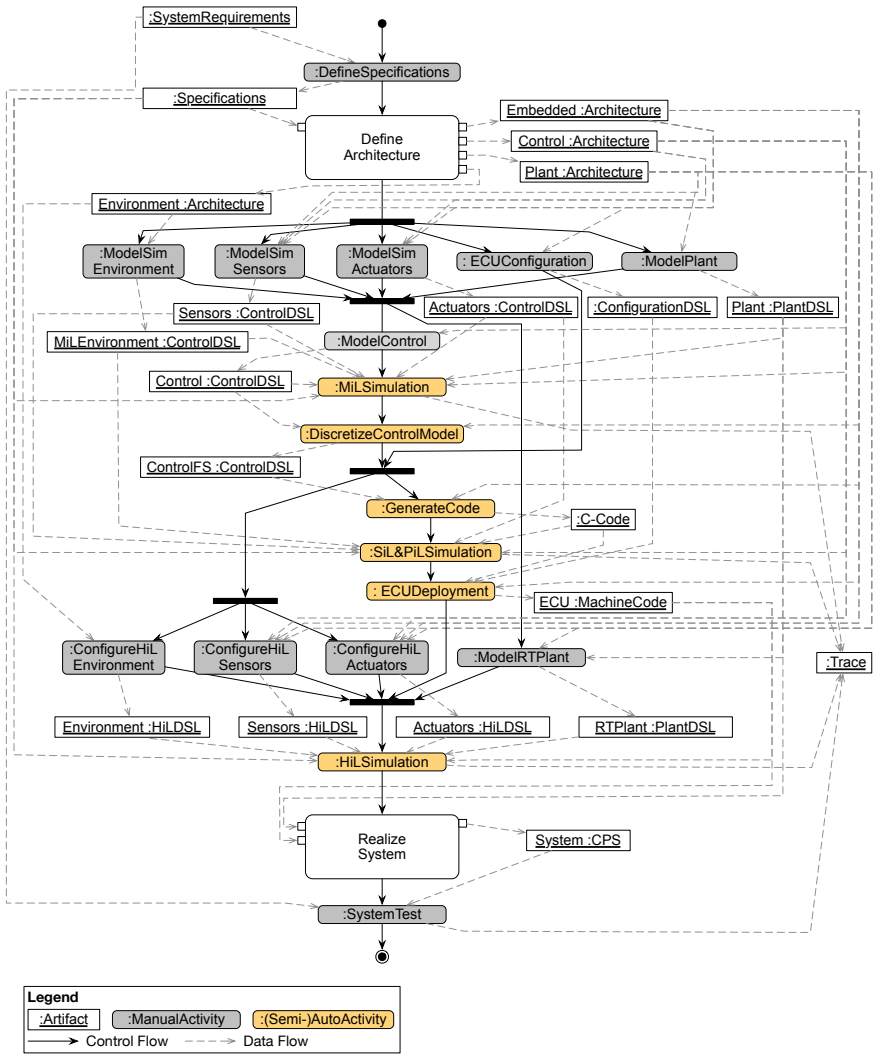


Figure 2.5: Possible enactment of a parallel design process

2.2.2 Contract-Based Design

Engineers intend to avoid inconsistencies within the same or between different domains, by defining an architecture prior to the (concurrent) design of the system. The architectural description outlines the design components implementing functional specifications, the interfaces between those components, value ranges and units of the interfaces, etc. When designing complex systems, however, components are often implemented by different engineers and combined/integrated at later stages, even within the same domain. During the design of a complex algorithm, for example, a component-based approach is mostly used in which the algorithm is divided into self-containing components implementing a well-defined functionality. We illustrate this by means of Figure 2.6 in which the design of the control algorithm for the power window, introduced in Section 1.5.1, is decomposed in terms of components. Note that we consider all the button signals as a single bus signal. As a benefit of this component-based approach, components can be designed by different engineers and stored in a library for later reuse. Although the architecture defines interfaces between components, a more formalized document is necessary that describes what a self-containing component may expect at its inputs and, given these conditions, what it should guarantee at its outputs. Hence, Contract-Based Design was introduced to formalize components in terms of Assume/Guarantee (A/G) reasoning.

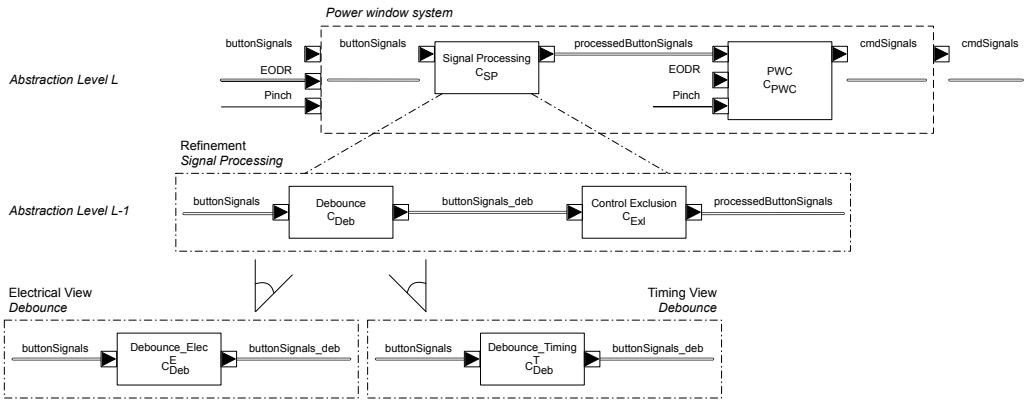


Figure 2.6: Decomposition of the power window control architecture

According to the CBD theory, formally described by Benveniste et al. in [BCN+15a], a contract C consists of a set of assumptions A and guarantees G describing the preconditions and postconditions of a system in terms of its set of design variables VAR, respectively. Or more formally:

$$C = (VAR, A, G) \tag{2.1}$$

In the example of the decomposed control algorithm, each component *M* can be typed by such a contract *C* in which their input variables are captured as a set of contract assumptions, while the output variables are expressed as a set of contract guarantees. Each component *M* is said to satisfy a contract *C*, formulated as $M \models C$, whenever it fulfills the set of guarantees under the given set of assumptions. Using set theory, this can be formally written as [BCN+15a]:

$$M \models C \text{ if and only if } M \cap A \subseteq G \tag{2.2}$$

A contract is said to be *consistent* if its set of implementations is nonempty ($M \neq \emptyset$) and *compatible* if its set of environments *E* is nonempty ($E \neq \emptyset$). Note that E is an environment of C if and only if $E \subseteq A$. A consistent contract, however, does not imply a consistent design. In case component M interfaces with component M', for example, consistency can only be guaranteed if variables belonging to the set of guarantees of component M equals the variables belonging to the set of assumptions of component M'.

When the final control algorithm is composed by integrating several interfacing components into one single component, as shown in Figure 2.6, one must ensure that their respective contracts are correctly consolidated to a single (top-level) contract. To do so, the CBD theory defines three operators: the conjunction operator $\wedge$, the composition operator $\otimes$, and the abstraction/refinement operator $\preceq$. In what follows, we briefly describe these operators by means of the power window example (Figure 2.6). We therefore rely on the theory and examples described in [BCN+12, BCN+15a].

The top-level *Signal Processing* component in Figure 2.6 is decomposed in two components implementing a single functionality: *Debounce* for debouncing the input signals and *ControlExclusion* to provide priority to driver's signals. At the bottom of Figure 2.6, it is shown that there exist two viewpoints on the *Debounce* component, namely, an electrical viewpoint and a timing viewpoint. Each viewpoint-specific implementation is typed by a (viewpoint) specific contract; that is C^{E}_{Deb} and C^{T}_{Deb}. They are formally described in Table 2.1.

According to the theory of [BCN+15a], contracts that relate to different aspects that may exist on a single component can be combined using the conjunction operator $\wedge$. These different aspects relate to the different concerns stakeholders have with respect to the component under design. Examples of such concerns (i.e., viewpoints) are: functionality, safety, quality of service, resource utilization, etc. Let $C' = (A', G')$ and $C'' = (A'', G'')$ be contracts related to the implementation of two different viewpoints on a single component *M*, then the resulting contract $C = C' \wedge C''$, can be obtained as follows [BCN+15a]:

$$\begin{aligned} A &= (A' \cup A'') \\ G &= (G' \cap G'') \end{aligned} \tag{2.3}$$

$$
C^{E}_{Deb}: \begin{cases} \text{Variables:} & \begin{cases} \text{Inputs:} & \text{x} \\ \text{Outputs:} & \text{y} \end{cases} \\ \text{Types:} & \text{x,y} \in \mathbb{R}_{+} \\ \text{Assumptions:} & 0\,\text{V} \leq \text{x} \leq 10\,\text{V} \\ \text{Guarantees:} & 0\,\text{V} \leq \text{y} \leq 5\,\text{V} \end{cases}
$$

$$
C^{T}_{Deb}: \begin{cases} \text{Variables:} & \begin{cases} \text{Inputs:} & \text{x} \\ \text{Outputs:} & \text{y} \end{cases} \\ \text{Types:} & \text{x,y} \in \mathbb{R}_{+} \\ \text{Assumptions:} & \text{True} \\ \text{Guarantees:} & t_y \leq 5\,\text{ms} \end{cases}
$$

Table 2.1: Electrical (E) and Timing (T) viewpoint contract for the *Debounce* component

$$
C_{Deb}: \begin{cases} \text{Variables:} & \begin{cases} \text{Inputs:} & \text{x} \\ \text{Outputs:} & \text{y} \end{cases} \\ \text{Types:} & \text{x,y} \in \mathbb{R}_{+} \\ \text{Assumptions:} & 0\,\text{V} \leq \text{x} \leq 10\,\text{V} \\ \text{Guarantees:} & \begin{cases} 0\,\text{V} \leq \text{y} \leq 5\,\text{V} \\ t_y \leq 5\,\text{ms} \end{cases} \end{cases}
$$

Table 2.2: Contract for the *Debounce* component

In the example of the power window, $C^{E}_{Deb} \wedge C^{T}_{Deb}$ results in contract C_{Deb} as described in Table 2.2.

Note that a conjunction of contracts relaxes the assumptions and enforces the guarantees. As such, if there exists an implementation M that satisfies the conjunction of two contracts, the implementation satisfies either contracts as well. This is formally written as [BCN+15a]:

$$
\text{If } M \models C' \wedge C'' \text{ then } M \models C' \text{ and } M \models C'' \tag{2.4}
$$

Complex systems often consist of different components that are interconnected through their interfaces. An example is shown at abstraction level L-1 of Figure 2.6 where the *Debounce* component is connected to the *Control Exclusion* component. Again, each component its implementation is typed by a contract shown in Table 2.2 and Table 2.3, respectively. In this case, the composition operator $\otimes$ is proposed by [BCN+15a] to combine contracts. Let now $C' = (A', G')$ and $C'' = (A'', G'')$ be the contracts of two connected components *M'* and *M"*, respectively.

$$
\mathrm{C_{CE}}\colon \begin{cases} \text{Variables:} & \begin{cases} \text{Inputs:} & \text{x} \\ \text{Outputs:} & \text{y} \end{cases} \\ \text{Types:} & \text{x,y} \in \mathbb{R}_+ \\ \text{Assumptions:} & 0\,\text{V} \leq \text{x} \leq 5\,\text{V} \\ \text{Guarantees:} & 0\,\text{V} \leq \text{y} \leq 3\,\text{V} \end{cases}
$$

Table 2.3: Contract for the *Control Exclusion* component

$$
\mathrm{C_{SP}^{Ref.}}\colon \begin{cases} \text{Variables:} & \begin{cases} \text{Inputs:} & \text{x} \\ \text{Outputs:} & \text{y} \end{cases} \\ \text{Types:} & \text{x,y} \in \mathbb{R}_+ \\ \text{Assumptions:} & 0\,\text{V} \leq \text{x} \leq 10\,\text{V} \\ \text{Guarantees:} & \begin{cases} 0\,\text{V} \leq \text{y} \leq 3\,\text{V} \\ \text{t}_\text{y} \leq 5\,\text{ms} \end{cases} \end{cases}
$$

Table 2.4: Contract for the refined *Signal Processing* component

Then the composition $C = C' \otimes C''$ can be obtained as follows [BCN+15a]:

$$
\begin{aligned} A &= (A' \cap A'') \cup \neg(G' \cap G'') \\ G &= (G' \cap G'') \end{aligned} \tag{2.5}
$$

In the example of the power window, $\mathrm{C_{Deb}} \otimes \mathrm{C_{CE}}$ results in contract $\mathrm{C_{SP}^{Ref}}$ as described in Table 2.4.

In the previous, the components *Debounce* and *Control Exclusion*, and their associated contract, are related to each other through their interfaces. We therefore say that these components are operating at the same (horizontal) level of abstraction; hence the term horizontal (in)consistency. However, components may also be hierarchically structured. In that case, components operating at abstraction level L-1 are considered as a refinement of the parent component operating at abstraction level L. Let now $C = (A, G)$ and $C' = (A', G')$ be the contract of the component operating at abstraction level L and L-1, respectively. Then the refinement of C by C', formulated as $C' \preceq C$, is defined as follows [BCN+15a]:

$$
\begin{aligned} A &\subseteq A' \\ G &\supseteq G' \end{aligned} \tag{2.6}
$$

For the power window example, the combined implementation of *Debounce* and *Control Exclusion*, and the resulting contract $\mathrm{C_{SP}^{Ref}}$ (Table 2.4), is a refinement of the *Signal Processing* component. For the latter, its implementation is typed by the contract $\mathrm{C_{SP}}$ as formally described in Table 2.5.

$$
C_{SP}: \begin{cases} \text{Variables:} & \begin{cases} \text{Inputs:} & \text{x} \\ \text{Outputs:} & \text{y} \end{cases} \\ \text{Types:} & \text{x,y} \in \mathbb{R}_+ \\ \text{Assumptions:} & 0\ \text{V} \leq \text{x} \leq 5\ \text{V} \\ \text{Guarantees:} & \begin{cases} 0\,\text{V} \leq \text{y} \leq 5\,\text{V} \\ t_y \leq 5\,\text{ms} \end{cases} \end{cases}
$$

Table 2.5: Contract for the (abstract) *Signal Processing* component

Note that the assumptions of the refined component are more relaxed while guarantees are enforced. As a result, any implementation M_i of contract C' is an implementation of C as well, or more formally [BCN+15a]:

$$\text{If } M_i \models C' \text{ and } C' \preceq C \text{ then } M_i \models C \tag{2.7}$$

The same reasoning can be defined for the environment in which they operate [BCN+15a]:

$$\text{If } E \models C \text{ and } C' \preceq C \text{ then } E \models C' \tag{2.8}$$

For a more detailed description of the Contract-Based Design theory and more extensive examples we refer to [BCF+08, BCN+12, BCN+15a, BCN+15b].

2.2.3 Ontologies

An ontology enables one to represent the shared knowledge in a domain of discourse using a common language [Gru93]. It enables one to explicitly define the concepts of a domain of discourse, properties of each concept describing various features and attributes of the concept, and restrictions on these properties. An ontology together with a set of individual instances of concepts constitutes a knowledge base [NM01].

Figure 2.7 illustrates an example of a knowledge base in which real world persons are classified in an ontology as being a *Project Manager*, *Supervisor*, or *PhD Student* concept. Ontological concepts and real world instances are related to each other through a satisfaction relationship. As such, each classification divides the instances of the real world as being satisfied (*C*) or not satisfied ($\neg$ *C*) by the concept. In addition, there may be an undefined relationship (*DK_C*) between concepts in the ontological world and instances in the real world. Ontologies are therefore regarded as Open World Assumptions (OWA) models. Properties between ontological concepts are represented as arrows and are applicable to all the individuals belonging to the related concepts. Note that concepts are represented as sets such that binary operations (i.e., set theory[Can74]) can

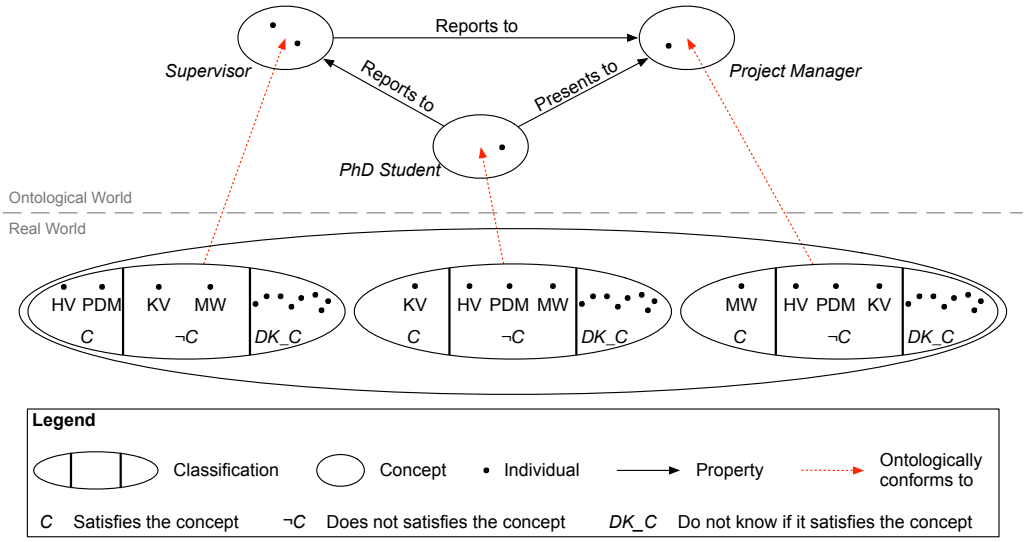


Figure 2.7: Introductory example of an ontology

be applied to the concepts.

Defining an ontology starts by choosing an appropriate ontology language. They can be divided into two categories [SI02]: traditional and web-based ontologies. The former category refers to languages based on first-order logic [Smu68], frame logic [KL89a, KL89b], and description logic [BCM$^+$03]. The latter, and more recent, category refers to languages based on the World Wide Web Consortium (W3C) standard for the Web of linked data (i.e., the Semantic Web) [Wor]. Besides these two categories, there also exist languages that combine the best of both worlds. An important criterion in the selection of the ontology language is its expressiveness (in function of the application). A language is more expressive if it is able to (i) express *Individuals*, (ii) relate *Concepts* (e.g., intersection), and/or (iii) characterize *Roles* (e.g., transitivity).

As ontologies are modeled using OWA, dedicated reasoners can be used that infer logical consequences from a set of explicitly asserted facts or axioms in the modeled ontology and typically provides automated support for reasoning tasks such as classification, debugging, and querying [DCtTdK11]. Note, however, that the expressiveness of a language is inversely proportional to its reasoning abilities. Among the large number of reasoning engines available [Abb12], well known reasoners are: Pellet [SPG$^+$07], FACT++ [TH06], and HermiT [SMH08, GHM$^+$14].

Within the scope of this dissertation, we have opted to use the W3C's standardized Web Ontology Language (OWL) (i.e., OWL 2) [HPSvH03, Bec09] to model our ontologies. In OWL 2, the terms *Classes*, *Properties*, *Instances*, and *Data Values* are used to construct the ontology and are synonyms for the general terms *Concepts*, *Roles*, *Individuals*, and *Data Values*, respectively.

Different flavors of the OWL exist: OWL Lite, OWL DL, and OWL Full. The difference lies in the expressiveness of the language, with OWL Lite being the least expressive. We have opted to use OWL DL as it enables one to express an ontology using description logic. In this dissertation the $\mathcal{SHIQ}(\mathcal{D})$ description logic is used. As syntactic and semantic interoperability is of importance within the scope of our contributions, we defined a so called upper ontology for CPSs that subsumes viewpoint-specific ontologies and relates them using a common set of *Classes* and *Properties*.

2.3 Tools

2.3.1 Eclipse EMF

The Eclipse Modeling Framework (EMF) [Theb] is a modeling framework and code generation facility for building tools and other applications based on a structured data model. From a model specification described in XML Metadata Interchange (XMI), EMF provides tools and runtime support to produce a set of Java classes for the model, along with a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor. The EMF is considered as the core of an extensible modeling environment enabling interoperability between technologies, defined as plugins, addressing a certain modeling concern. An example of such a plugin is Epsilon [KPP08], enabling one to define and execute model-to-model and model-to-text transformations.

2.3.2 T-Core

T-core is a framework providing a collection of primitive constructs, defined at the optimal level of granularity, for the design of model transformation languages [SV10, Syr11, SVL15]. It is not restricted to any form of specification of transformation units, be it rule-based, constraint- based, or function-based. It can also represent bidirectional and functional transformations as well as queries. T-Core modularly encapsulates the combination of these primitives through composition, reuse, and a common interface. It is an executable module that can be easily integrated into a programming or modeling language.

2.3.3 Protégé Desktop

Protégé Desktop is a feature rich ontology editing environment with full support for OWL 2, and direct in-memory connections to description logic reasoners such as HermiT and Pellet. It supports creation and editing of one or more ontologies in a single workspace via a completely

customizable user interface. Visualization tools allow for interactive navigation of ontology relationships. Advanced explanation support aids in tracking down inconsistencies. Refactor operations available include ontology merging, moving axioms between ontologies, renaming of multiple entities, and more [Sta16, Ala13]. Different formats for exchange are supported, including the standardized W3C's expressive Resource Description Framework (RDF), OWL, and Manchester OWL syntaxes.

2.3.4 SymPy

SymPy is an open source Python library for symbolic mathematics. It aims to become a full-featured Computer Algebra System (CAS) while keeping the code as simple as possible in order to be comprehensible and easily extensible. SymPy is written entirely in Python [Sym, JČMG12].

SymPy supports a wide array of mathematical facilities. These include functions for assuming and deducing common mathematical facts, simplifying expressions, executing common calculus operations, manipulating polynomials, pretty printing expressions, solving equations, and representing symbolic matrices. Other supported facilities include discrete math, concrete math, plotting, geometry, statistics, sets, series, vectors, combinatorics, group theory, code generation, tensors, Lie algebras, cryptography, and special functions. SymPy has strong support for arbitrary precision numerics, backed by the mpmath package. Additionally, SymPy contains submodules targeting certain specific physics domains, such as classical mechanics and quantum mechanics. This breadth of domains has been engendered by a strong and vibrant user community. SymPy is a dependency of many external projects across a wide spectrum of domains [MSP+17].

2.3.5 Simulink

Simulink® is a block diagram environment for multidomain simulation and Model-Based Design. It supports system-level design, simulation, automatic code generation, and continuous test and verification of embedded systems. Simulink® provides a graphical editor, customizable block libraries, and solvers for modeling and simulating dynamic systems. It is integrated with MATLAB®, enabling one to incorporate MATLAB® algorithms into models and export simulation results to MATLAB® for further analysis [Mat].

2.3.6 Massif

The MATLAB Simulink Integration Framework (Massif) for Eclipse is an Eclipse-based feature that supports the easy handling of Simulink® models by providing import and export capabilities to/from Eclipse EMF. It therefore relies on a generic EMF metamodel that was designed to

store all information for each Simulink® block and provide the type information as defined in Simulink® using library links [HSB+]. Importing and exporting is done using the command line interface of MATLAB®. For this, Massif relies on the Open Services for Lifecycle Collaboration (OSLC) [Joh13], a specification enabling integration of tools via linked data.

2.3.7 MAST

MAST is an open source suite of tools to execute schedulability analysis of real-time distributed systems that assesses a rich variety of timing requirements. For this, a textual input model of the system is used as an input of the analysis tools that calculates the response times, jitter, slack time, etc. MAST is also able to execute a design-space exploration to find an optimized assignment of priorities and deadlines. The result of the analysis is presented as a textual output model. Both input and output models are fully described by metamodels enabling the integration of tools in a model-driven engineering tool chain [Gon, GGPD01].

CHAPTER 3

Design Contracts Enabling Consistency in Multi-Viewpoint Design Processes

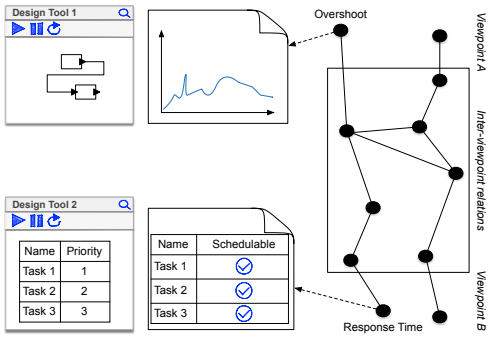
Abstract. Designing a Cyber-Physical System requires a multi-disciplinary team to collaborate such that a rather complex product can be built given certain constraints (e.g., time-related or resource-related). Key to efficient collaboration among engineering teams is the ability to understand each others' domain and, as a consequence, to understand how viewpoints are related to each other. Despite the many efforts that have been made in defining techniques and methods for collaborative design of Cyber-Physical Systems, engineers still face inconsistencies when integrating design artifacts. This leads to time consuming, iterative design processes where inconsistencies are resolved, in turn possibly creating new ones. Based on the principles of Contract-Based Design, using Assume/Guarantee contracts, and ontological reasoning, a method is proposed enabling engineers to reason about inter-viewpoint dependencies prior to the design of the system. The early detection and avoidance of inconsistencies enables the method to be used in a concurrent engineering setting while reducing the development time and effort. Hence the name Contract-Based Co-Design as reference to the approach.

3.1 Introduction

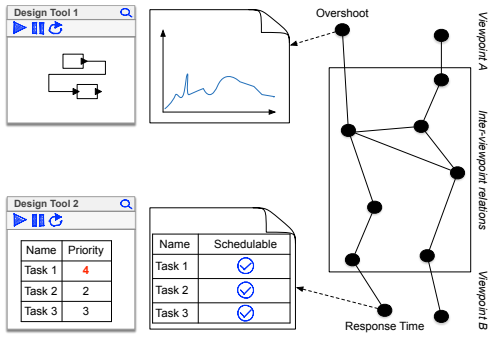
Despite the emerging need for intelligent systems to enforce the next industrial revolution (i.e., Industry 4.0) or the next automobile revolution (e.g., full autonomous driving), designing such CPSs remains a challenge. The integration of mechanical components with control algorithms running on computational units and the distributed nature of the system requires a multidisciplinary engineering team to collaborate, each with a different view and set of concerns on the system under design. As a result, there exist heterogeneous viewpoints on the system under design that may lead to contradicting design decisions [TQB+14]. This is illustrated in Figure 3.1, where we focus on the relations between a control and an embedded viewpoint and how a change in the embedded domain affects the control domain. Engineers in either domain use dedicated tools to design and evaluate, through simulations and analysis, a part of the system (Figure 3.1.(a)). In our example, control engineers are concerned about the maximum allowed overshoot of their control algorithm while embedded engineers are concerned with the maximum response time of the different tasks. Assume that the configuration of the embedded platform as shown in Figure 3.1.(a) results in an analysis for which the concerns of both control and embedded engineers are fulfilled. Because of additional tasks that must be scheduled, embedded engineers may decide to lower the priority of the task running the control algorithm. This is indicated by a higher priority number in Figure 3.1.(b). Although the system is schedulable, lowering the priority results in a higher response time of the task. Because of the inter-viewpoint relationships, illustrated by the graphs on the right-hand side of each figure in Figure 3.1, the behavior of the control algorithm is affected so that its overshoot is larger than expected. We illustrate this in Figure 3.1.(c).

As a systems engineer is often aware of these inter-viewpoint relationships, they typically merge the different viewpoints on a system and execute a tradeoff analysis between the various design parameters so that consistency between viewpoints is preserved. The increasing systems' complexity, however, demands for techniques to assist engineers in both the (consistency) tradeoff analysis and management. In that respect, CBD (Section 2.2.2) is a promising technique in which the various design parameters are formalized as an agreement (i.e., a contract) between two or more engineering domains. However, the CBD theory does not support engineers in reasoning about the inter-viewpoint relationships and, as such, about the content of a contract, while highly focusing on consistency within one single viewpoint. It thus lacks in its applicability to the (concurrent) design of CPSs; that is, processes that combine multiple viewpoints.

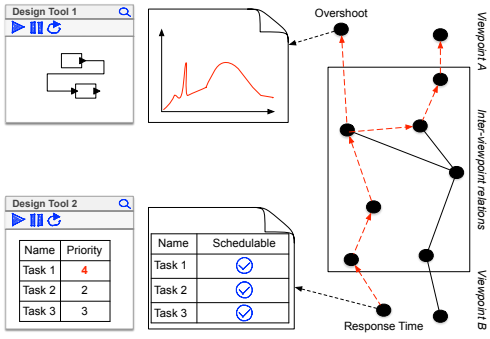
In particular, contract-based design supported (multi-viewpoint) processes are characterized by a common engineering phase in which viewpoint-specific architectures are defined and a common contract is negotiated [DLTT13]. As engineers consider the design from a different viewpoint they face difficulties in reasoning how design artifacts, and their corresponding design parameters, originating from different viewpoints are related to each other. As a result, the content of the common contract may be incomplete or inconsistent. Secondly, once a common contract is negotiated, it is highly recommended to derive viewpoint-specific contracts, containing information that is



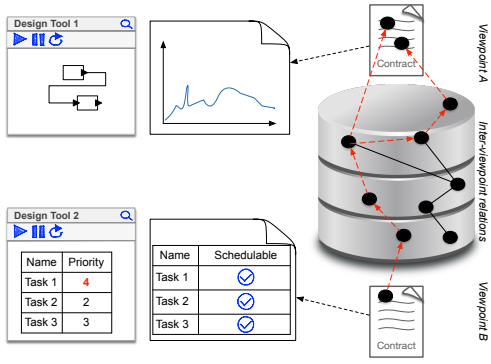
(a) Implicit relations between control and embedded viewpoint



(b) Parameter adjustment in the embedded viewpoint



(c) Changing parameter affects control viewpoint



(d) Contracts and knowledge base enabling consistent viewpoints

Figure 3.1: Overview of the Contract-Based Co-Design Method

only relevant for a particular viewpoint. This may require translation mechanisms from commonly negotiated design parameters to viewpoint-specific design parameters.

In this chapter we argue that reasoning over inter-viewpoint relationships can be established by adding the principles of ontological reasoning (Section 2.2.3) to A/G reasoning. In that respect, ontologies are used to make the tacit domain knowledge of engineers explicit while simultaneously linking design parameters to each other. This is illustrated by the database symbol shown in Figure 3.1.(d). It enables engineers to discuss a consistent mapping between viewpoint-specific architectures, formalized by a common (set of) *mapping contracts*, and enforces consistent multi-viewpoint design by automatically deriving viewpoint-specific contracts. As this is of particular interest for the concurrent design of CPSs, we introduce a Contract-Based Co-Design (CBCD) method in this chapter.

We consider that MBSE can only be successfully achieved when engineers are supported with tools. We therefore present a Contract-Based Co-Design (CBCD) framework acting as a single point of truth throughout the (concurrent) design process, supporting engineers in (i) negotiating consistent *mapping contracts*, (ii) deriving domain-specific contracts, (iii) detecting inconsistencies between design parameters (during both contract negotiation and design), (iv) updating the viewpoint-specific design with information from the other viewpoints (Chapter 4), and (v) exploring different design alternatives (Chapter 5). Key is the ontology, that is regarded as a knowledge base within the scope of this dissertation, that relates design parameters as illustrated in Figure 3.1.(d).

3.2 Related Work

Contract-Based Design finds its origin in software engineering where Hoare in 1969 introduced the Hoare triple PQR that states that the result R of a program Q is true if the input condition P is true [Hoa69]. In the late 80's, Meyer introduced the Eiffel programming language as an implementation of Hoare logic [Mey88, Mey92]. Using "Require" and "Ensure" clauses, Eiffel enabled a programmer to annotate a software routine with pre and post requirements respectively.

It took, however, almost a decade before Contract-Based Design (formerly called design by contract) was associated with model-based design. In 2001, Broy and Stolen introduced the FOCUS method for the development of software-intensive systems going from requirement specification to abstract implementation [BS01]. In their approach, component specifications are expressed in terms of assumed input streams and, under these conditions, guaranteed output streams, for which a stream can be regarded as a set of events. This so called A/G paradigm is considered as a reformulation and generalization of the pre/post style logic of Hoare. In parallel with Broy and Stolen, de Alfaro and Henzinger [dAH01] used an automata-based formalism to capture a component's temporal I/O behavior in terms of input assumption and output guarantees respectively while formalizing the notion of compatibility, composition, and refinement. A few years later, Damm et al. [Dam05, DV05] introduced *Rich Components* that extends traditional component models with (i) functional and non-functional information originating from different views on the system, (ii) explicit information on what a component guarantees under certain conditions while (iii) adding classifiers on the environmental conditions (i.e., the assumptions).

The work of Damm et al. and Alfaro and Henzinger served as an input for the results obtained in the framework of the SPEEDS[1] project. In [JMM08, BFMSV08, PDH+09] the definition of *Rich Components* was extended by supporting heterogeneous viewpoints (in a single domain) on a system. Hence the term *Heterogeneous Rich Components (HRC)*. The scope of the SPEEDS project resulted in the (first) use of contracts in a component based engineering context. In [BCF+08], Benveniste et al. present the mathematical foundations of CBD to enable the

[1] www.speeds.eu.com

combination of contracts for different model components and the combination of contracts for different viewpoints on the same model component. A few years later, within the scope of the CESAR[2] project, the CBD theory was further extended with formal contract definitions, among them the three basic contract operators (abstraction/refinement, composition, and conjunction), in [BCN+12, BCN+15a].

A more generic contract framework is shown by Bauer et al. [BDH+12], where the relation between specifications of component behaviors and contracts is shown. Moreover, they demonstrate that the trace-based contract theory of Benveniste et al. [BCF+08] is an instance of this generic, specification theory based, contract framework. In [DLTT13], a non-exhaustive classification of timing contracts is given. Focusing on control-embedded related timing parameters, Derler et al. demonstrate the feasibility of a contract type for one or both engineering domains.

Besides the theoretical aspects of CBD, its applicability in engineering processes has also been addressed in the literature. Sangiovanni-Vincentelli et al. address the emergent need of CBD in the context of system level design in [SVDP12]. 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. [GPQ14], 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 [NXO+14, NSVB+15]. Using the design of an aircraft electric power system, they illustrate how a hardware topology can be independently constructed from the control logic. In [BCN+15b], a method, combining CBD and AUTomotive Open System ARchitecture (AUTOSAR) [AUT, FNS+16] principles, is shown for the independent, concurrent, development of a distributed task allocation problem. The presented method consists of the specification of resource agnostic, functional, contracts and resource aware contracts that are both decomposed to enable concurrent engineering. In more recent work, Dal Lago et al. [DLFPF18] present how non-functional requirements (e.g., timing) of Service Oriented Cyber-Physical Systems can be practically validated using fault injection and CBD. The authors demonstrate how extensive fault tree analyses can be avoided by breaking the analysis down to the component level in a similar approach as system-level contracts are refined into component contracts. If an analysis is proven to be correct at the component level (i.e., if the analysis satisfies a contract of a component) compositional contract rules are used to verify a system's validity.

To avoid inconsistencies between multiple domains, and their respective viewpoints, Bhave introduces in [Bha11] the notion of a common base architecture that provides engineers a single reference point for multi-domain system models. As mappings are defined between the viewpoint-specific architectures and the common base architecture, structural consistency can be guaranteed. The framework has been further extended by Rajhans in [Raj13, RBR+14] so that semantic

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

consistency can be guaranteed throughout the design process. Therefore, mappings between heterogeneous semantic domains are formally described and verified during design. With respect to verification, Ruchkin [RSI+18] recently introduced the Integration Property Language (IPL) that enables system engineers to define properties over both behavioral and static semantics. Using model checking techniques, it is verified whether viewpoint-specific models are consistent.

As already mentioned in Section 3.1, we contribute to the current state-of-the-art of CBD by explicitly modeling the domain knowledge using an ontology enabling one to relate design parameters from different engineering disciplines using inter-viewpoint relationships. In that respect, Törngren et al. describe the different viewpoints involved in the design of mechatronic systems in [TQB+14]. They show how these viewpoints are interrelated by means of supporting models at different design levels, namely: (i) people level, (ii) models level, and (iii) 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 of particular interest. At people level, the authors point out that each stakeholder, involved in the design of a CPS, should be aware of the effect of their work on others. The use of design contracts is proposed to enable this awareness. Moreover, they hint towards the use of assumptions and guarantees as discussed by [SVDP12]. Additionally, at models level, Törngren et al. describe the existence of dependencies between models implementing certain parts of the overall system requirements. As a possible solution towards consistency management, a Dependency Modeling Language (DML) and a supporting Dependency Modeler [Qam13] is suggested to formally capture and manage dependencies.

The benefits of capturing and managing dependencies in the design of CPSs is shown by Gross and Rudolph in [GR16]. While not explicitly mentioning the term ontology, the authors construct a graph that relates domain-specific design parameters for the design of a satellite system. It enables them to execute sensitivity analysis across the entire design space enabling designers to find a tradeoff between different design decisions. Moreover, the presented ontology facilitates DSE to find the optimal satellite design satisfying the requirements.

We conclude this section with the work of Persson et al. where the authors characterize model-based approaches used in the design of CPSs [PTQ+13]. 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 that are not entirely exclusive to each other.

3.3 Design Contracts Supporting Multi-Viewpoint Design Processes

As described in Section 2.2.1, engineers are challenged to optimize their design processes so that the time to market is shortened while design costs are reduced. In order to achieve these demands, viewpoint-specific design activities are already parallelized once a common architecture is defined. It is often seen, however, that the integration of these different viewpoints fails as requirements, and as such design parameters, overlap. In other words, due to the lack of consistency management techniques iterative, time consuming, design loops may be necessary in which inconsistencies are resolved, in turn possibly creating new ones.

In [DLTT13], Derler et al. suggest the introduction of a negotiation step in (concurrent) design processes in which the different stakeholders formally define a set of design parameters, and their expected values, using a (set of) contracts so that consistency throughout the detailed design step can be assured. Figure 3.2 outlines the proposed design process as an adaptation of the widely used V model (shown in Figure 2.2). To enable the negotiation of contracts, it is assumed that the architectural design step is extended to include a preparatory design for each of the involved viewpoints. This implies that each viewpoint considers a possible (set of) design(s) based on the given specifications. In case of control-embedded design, for example, the control viewpoint reasons about a certain control strategy, sample period, etc. As being part of the embedded domain, the hardware viewpoint reasons about a certain execution platform, communication bus, etc.

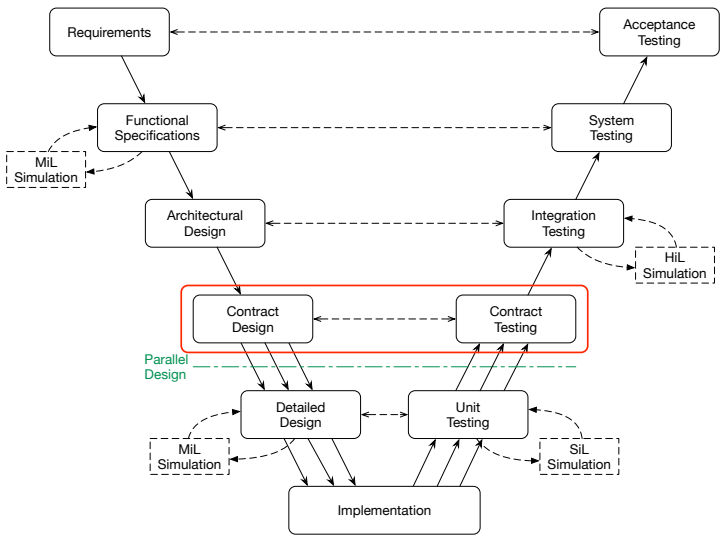


Figure 3.2: V model for consistent parallel design as proposed by Derler et al. in [DLTT13]

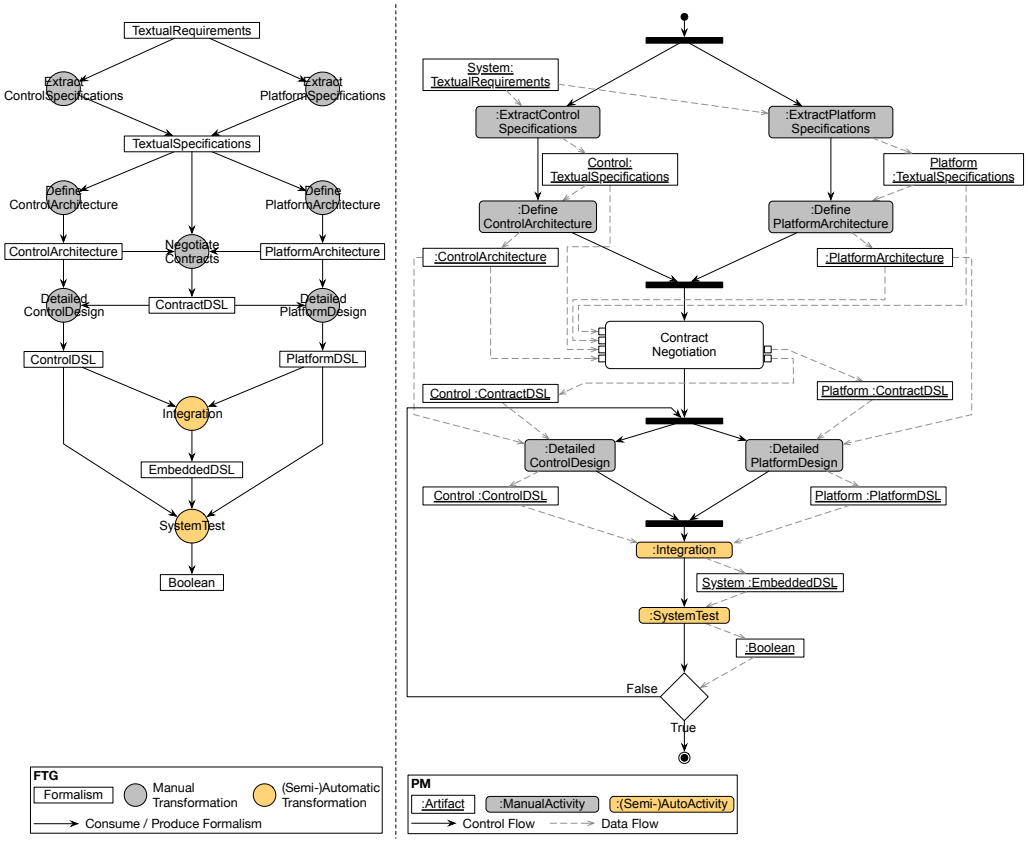


Figure 3.3: Possible enactment of a by contracts support design process as proposed by Derler et al. in [DLTT13]

Using a FTG+PM, introduced in Section 2.1.4, Figure 3.3 demonstrates a possible enactment of the conceptual representation of the by contracts driven design process of Figure 3.2. It clearly demonstrates how the different viewpoints, in this case the control and platform viewpoint, are required to negotiate a (set of) contract(s) before detailing their design. During contract negotiation, engineers are supported by heuristics/domain knowledge, simulation, analysis, and optimization techniques to achieve a tradeoff between the various design parameters [DLTT13]. Although the suggested methodology states that there may be potential iterations in the contract negotiation step, it is not clear to what extent consistency between the various design parameters, and thus between the different viewpoints, is ensured. The outcome of the negotiation step is a (set of) contract(s) with fixed design parameters. Preferably, viewpoint-specific contracts are derived from the negotiated set of design parameters. The methodology proposed in [DLTT13], however, does not discuss how they can be obtained. Additionally, the methodology does not

impose a verification step to verify that the implementation, as part of the detailed design step, satisfies the negotiated contract(s). In the next section, we elaborate on the question whether CBD as a technique is suitable to tackle these challenges.

3.4 Applicability of Contract-Based Design in Multi-Viewpoint Processes

To enable concurrent detailed design, the outcome of the negotiation phase must be (a set of) contract(s) that can be interpreted by the different viewpoints. In what follows we examine in more detail to which extent the current CBD theory supports this reasoning.

In the literature, some techniques, illustrated by means of a case study, have already been suggested that address the multi-viewpoint problem [BCN+15b, NXO+14, DLFPF18]. In particular, Benveniste et al. defines in [BCN+15b] a four-step approach to design a CPS (i.e., a multi-viewpoint design) using the principles of CBD:

1. Define a (hardware) resource agnostic top-level contract.
2. (Optional) For each component, decompose the top-level contract into resource-agnostic subcontracts.
3. Define a resource aware top-level contract.
4. Decompose the resource aware top-level contract into (resource aware) subcontracts.

Similar to the example shown in [BCN+15b], we illustrate the proposed approach for the design of the power window system (introduced in Section 1.5.1). Note that we only briefly discuss the approach to validate the applicability of the CBD theory in multi-viewpoint design processes. For more details, we refer to [BCN+15b] where the design of an AUTOSAR compliant *Exterior Light Management System* is used as an example.

Step 1 - Define a resource agnostic top-level contract

From a functional, resource agnostic, point of view, the control algorithm of the power window system consists of a *Debounce*, *ControlExclusion*, *PWC_DRV* and *PWC_PSG* component as shown in Figure 1.2. Note that the functional architecture addresses only one viewpoint; that is, the control viewpoint.

As described in Section 2.2.1, each design process starts by defining a set of specifications from the given set of system requirements. In that respect, the requirements specified in Section 1.5.1 are refined given domain-specific knowledge. For example, requirement 4 and 5 are further refined in the spatial and temporal dimension to detail the safety requirement:

4.1/5.1 Spatial dimension—if a clamped object is detected, the power window may continue to close for a maximum of 0.2 mm before safety-critical situations occur.

4.2/5.2 Temporal dimension—given the spatial dimensions and the inertia of the system, safety can be guaranteed if the window lowers within 1 ms.

They are formalized as the third and fourth guarantee of contract C_{Top}, shown in Table 3.1, using a pattern-based contract specification language as used in [BCN+15b] in which we solely focus on the assumptions and guarantees. Contract C_{Top} will be further extended by the engineer so that every requirement, and the derived specifications, listed in Section 1.5.1 is formulated as a contract entry. For instance, the first and second contract guarantee are derived from the first requirement. While defining the specifications, certain assumptions regarding the environment in which the system will operate are made. With respect to the context/environment in which the power window is operating, for example, it is determined that: (i) the minimum interval of button operations is 100 ms and (ii) the pinch force is not higher than 1000 N. These considerations are added to the contract as assumptions. For the sake of simplicity, we have limited ourselves to a subset of the available inputs and outputs in the definition of the resource agnostic top-level contract C_{Top}.

C_{Top}:		
	Assumptions	(1) *drvCmd_drvWindowUp* **occurs sporadic** with a **minimum interval of** 100 ms.
		(2) *psgCmd_psgWindowUp* **occurs sporadic** with a **minimum interval of** 100 ms.
		(3) *Pinch_DRV* **is lower than** 1000 N.
		(4) *Pinch_PSG* **is lower than** 1000 N.
	Guarantees	(1) **Delay between** *drvCmd_drvWindowUp* **and** *drvWindowUp* **within** [0 ms, 200 ms].
		(2) **Delay between** *psgCmd_psgWindowUp* **and** *psgWindowUp* **within** [0 ms, 200 ms].
		(3) **If** *Pinch_DRV* **exceeds** 100 N **and** *EODR_DRV* **equals** 0, **delay between** *Pinch_DRV* **and** *drvWindowDown* **within** [0 ms, 1 ms].
		(4) **If** *Pinch_PSG* **exceeds** 100 N **and** *EODR_PSG* **equals** 0, **delay between** *Pinch_PSG* **and** *psgWindowDown* **within** [0 ms, 1 ms].

Table 3.1: Power window system—top-level resource agnostic contract

Step 2 - For each component, decompose the top-level contract into resource-agnostic sub-contracts

In this second step, the resource agnostic top-level contract C_{Top} is decomposed so that each component of the functional architecture is typed by an individual contract. As the (similar) fourth step decomposes the top-level resource *aware* contract, this step is considered optional. Especially since the third step continues reasoning on the top-level contract. We agree with this reasoning and illustrate the decomposition in the final step of this approach.

Step 3 - Define a resource aware top-level contract

In order to define a resource aware top-level contract, the approach suggests a system architect (e.g., an Original Equipment Manufacturer) to reason about a possible hardware and software architecture for which a distributed computational platform and operating system tasks are defined, respectively. Figure 3.4 shows such an architecture for the power window system, that is, a resource aware architecture. As one can observe, the resource aware architecture consists of two Electronic Control Units (ECUs) connected using a Controller Area Network (CAN) communication bus. Based on the specifications of the chosen hardware platform and its current load, it is decided by the system architect that the tasks, denoted by the ovals in Figure 3.4, executing the implementation of components *Debounce* and *ControlExclusion* are allocated to the *Low Performance Processor (LPP)*. Tasks executing components *PWC_DRV* and *PWC_PSG* are allocated to the *High Performance Processor (HPP)*. For each task, the system architect budgets deployment related design parameter such as the Worst-Case Execution Time (WCET) indicating the upper bound on execution times of the software components running on the target hardware. For this, the system architect relies on heuristics/domain knowledge. In case of the deployed *Debounce* component, for example, the system architect specifies that an execution time between 1.5 ms and 2 ms is available for task T_d. Once defined, a resource aware top-level contract C_{Top}^{Res}, as shown in Table 3.2, is composed that refines the resource agnostic contract C_{Top}.

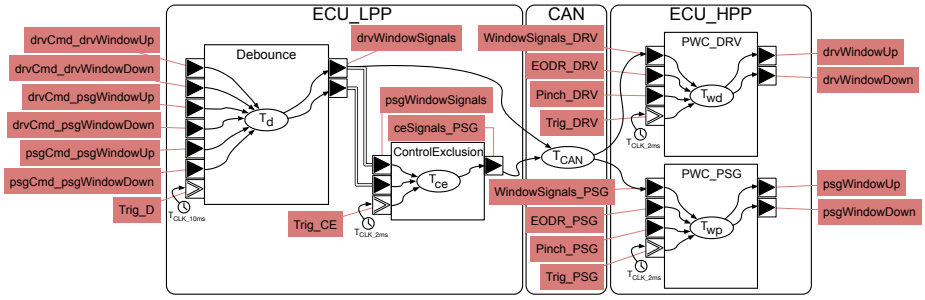


Figure 3.4: Embedded architecture of the power window system

C^{Res}_{Top}:		
	Assumptions	(1) *drvCmd_drvWindowUp* **occurs sporadic** with a **minimum interval of** 100 ms.
		(2) *psgCmd_psgWindowUp* **occurs sporadic** with a **minimum interval of** 100 ms.
		(3) *Pinch_DRV* **is lower than** 1000 N.
		(4) *Pinch_PSG* **is lower than** 1000 N.
	Guarantees	(1) **exeT**(T_d) **within** [1.5 ms, 2 ms], **exeT**(T_{ce}) **within** [750 μs, 1 ms], **exeT**(T_{CAN}) **=** 480 μs, **exeT**(T_{wd}) **within** [400 μs, 500 μs], **exeT**(T_{wp}) **within** [400 μs, 500 μs].
		(2) **Delay between** *drvCmd_drvWindowUp* **and** *drvWindowUp* **within** [0 ms, 200 ms].
		(3) **Delay between** *psgCmd_psgWindowUp* **and** *psgWindowUp* **within** [0 ms, 200 ms].
		(4) **If** *Pinch_DRV* **exceeds** 100 N **and** *EODR_DRV* **equals** 0, **delay between** *Pinch_DRV* **and** *drvWindowDown* **within** [0 ms, 1 ms].
		(5) **If** *Pinch_PSG* **exceeds** 100 N **and** *EODR_PSG* **equals** 0, **delay between** *Pinch_PSG* **and** *psgWindowDown* **within** [0 ms, 1 ms].

Table 3.2: Power window system—top-level resource aware contract

Step 4 - Decompose the resource aware top-level contract into (resource aware) subcontracts

Decomposing the resource aware top-level contract of C^{Res}_{Top} into subcontracts, requires the system architect to execute a global schedulability analysis to obtain the maximum response time, specified as Worst-Case Response Time (WCRT) for each task. It defines the maximum allowed time between the activation of the task and its completion. As such, it further refines the guarantees of C^{Res}_{Top}. For example, the second guarantee of C^{Res}_{Top} defining an end-to-end latency between 0 ms and 200 ms between *drvCmd_drvWindowUp* and *drvWindowUp* is refined into a range with a lower bound of 9.28 ms and an upper bound of 13.48 ms after analysis. Afterwards, the resource-aware top-level contract is decomposed into the subcontracts C^{LPP}_{Sub} and C^{HPP}_{Sub}, one for each ECU. They are shown in Table 3.3. Referring back to the example, one can notice how the second top-level guarantee is divided into the ranges [8 ms, 12 ms] and [800 μs, 1 ms] for the end-to-end latency on *ECU_LPP* and *ECU_HPP*, respectively. Note that a fixed delay of 480 μs (T_{CAN}) needs to be incorporated to obtain the schedulability analysis results (i.e., the range [9.28 ms, 13.48 ms]). Finally, the system architect assigns each subcontract to a supplier who is in charge of implementing the given contract.

C_{Sub}^{LPP}:	Assumptions	(1) *drvCmd_drvWindowUp* **occurs sporadic** with a **minimum interval of** 100 ms. (2) *psgCmd_psgWindowUp* **occurs sporadic** with a **minimum interval of** 100 ms.
	Guarantees	(1) **exeT**(T_d) **within** [1.5 ms, 2 ms], **exeT**(T_{ce}) **within** [750 μs, 1 ms]. (2) **Delay between** *drvCmd_drvWindowUp* **and** *drvWindowSignals* **within** [8 ms, 12 ms]. (3) **Delay between** *{drvCmd_drvWindowUp* and *psgCmd_psgWindowUp}* **and** *ceSignals_PSG* **within** [8.75 ms, 13 ms].
C_{Sub}^{HPP}:	Assumptions	(1) *WindowSignals_DRV* **occurs sporadic** with a **minimum interval of** 100 ms with **jitter within** [8.48 ms, 12.48 ms]. (2) *WindowSignals_PSG* **occurs sporadic** with a **minimum interval of** 100 ms with **jitter within** [9.23 ms, 13.48 ms]. (3) *Pinch_DRV* **is lower than** 1000 N. (4) *Pinch_PSG* **is lower than** 1000 N. (5) **Import Guarantees of C_{Sub}^{LPP}**.
	Guarantees	(1) **exeT**(T_{wd}) **within** [400 μs, 500 μs], **exeT**(T_{wp}) **within** [400 μs, 500 μs]. (2) **Delay between** *WindowSignals_DRV* **and** *drvWindowUp* **within** [800 μs, 1 ms]. (3) **Delay between** *WindowSignals_PSG* **and** *psgWindowUp* **within** [800 μs, 1 ms]. (4) **If** *Pinch_DRV* **exceeds** 100 N **and** *EODR_DRV* **equals** 0, **delay between** *Pinch_DRV* **and** *drvWindowDown* **within** [800 μs, 1 ms]. (5) **If** *Pinch_PSG* **exceeds** 100 N **and** *EODR_PSG* **equals** 0, **delay between** *Pinch_PSG* **and** *psgWindowDown* **within** [800 μs, 1 ms].

Table 3.3: Power window system—resource aware subcontracts

Discussion of the approach

When dealing with subcontractors, the approach ensures for a system integrator a schedulable system when integrating the implemented components received from its suppliers, if and only if the implementation(s) conforms to the contract(s). From the supplier's perspective, a team consisting of at least a control and an embedded engineer is charged with the implementation of the received (sub)contract. In that respect, they will follow a well-defined design process, as discussed in Section 2.2.1. To be able to design independently of one another, viewpoint-specific

contracts need to be derived from the received contract. However, both contracts C_{Sub}^{LPP} and C_{Sub}^{HPP} are highly focused on guaranteeing timing related design parameters, while control engineers have limited aids in estimating how their design decisions affect the execution time of a particular task and, related to that, the total end-to-end delay of a signal. For this reason, control engineers are unable to guarantee timing related design parameters. Furthermore, the contracts are in a sense imposed on the subcontractors implementing the system since there is no negotiation between engineers implementing the system. Consequently, design iterations might still be necessary due to incorrect resource budgeting of the systems engineer. Therefore, we say that the proposed approach results in a set of contracts that address only the system viewpoint(s) of the embedded domain.

Question remains how a contract addressing the control viewpoint can be obtained. Intuitively, one could argue that the guarantees of contract C_{Sub}^{LPP} and C_{Sub}^{HPP} should be considered as the control engineer's contract assumptions. Although, this is true, it is still unclear for a control engineer (i) how these assumptions should be interpreted or incorporated when designing the control algorithm and (ii) what (design parameters) should be guaranteed under these assumptions. Note that a similar problem exists when assuming control design parameters in the embedded contract.

We conclude that the approaches shown in the current literature [BCN+15b, NXO+14, DLFPF18], and by extension the state of the art in CBD theory, lack in the ability to infer contracts addressing a single viewpoint, such that multi-viewpoint design cannot be properly supported. As such, there is a need to derive viewpoint-specific contracts from a common (system) contract while ensuring that each contract contains parameters that can be interpreted by their respective viewpoint. In that sense, we consider that a translation between parameters stemming from different viewpoints is required. It should also be unambiguous what may be assumed by each viewpoint and what should be guaranteed under these conditions.

3.5 Multi-Viewpoint Consistency through Ontological Reasoning

Although relating design parameters between viewpoints is key to enable consistent concurrent design through the use of A/G contracts, translating design parameters from one viewpoint to another seems to be challenging for engineers and is often done in an ad hoc fashion. To address this issue and to facilitate the use of A/G contracts, 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 specifications in terms of viewpoint-specific design parameters, each stakeholder keeps in mind certain properties, which we call *ontological properties*. For example, a control engineer in charge of designing a control algorithm

for the system under development implicitly thinks about *control performance* and *response time* when deciding on modeling design parameters such as sample rate. An embedded engineer is concerned with embedded design parameters such as processor speed and period of tasks instead of the behavior of the control algorithm. As such, they reason about *schedulability*, *processor load*, *cost*, etc.

As some of the requirements are shared among views, it may be clear that viewpoint-specific design parameters related to those shared requirements should be consistent. Due to the viewpoint-specific interpretation of the requirements, however, engineers reason about different ontological properties; e.g., *control performance* for the control engineer and *schedulability* for the deployment engineer. Unintentionally, this leads to inconsistencies between viewpoints. 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 viewpoint-specific design parameters such as computation time 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 (i.e., WCRT) that where not taken into account by the control engineer.

By making the influence relations between ontological properties explicit in our framework, we are able to explicitly translate related design parameters used in different viewpoints. Moreover, reasoning about ontologies (i.e., relating ontological properties) and tracing the properties at the modeling 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.

3.5.1 Foundations of Ontological Reasoning

Our suggested consistency management approach relies on the concepts of linguistic and ontological (meta-)modeling in which a conformance relationship exists between a metamodel and a (possibly infinite) set of models that are instances of the metamodel. According to Kühne [Küh06] this conformance relationship can be either linguistic or ontological. Based on the work of Barroca et al. [BKV14], Figure 3.5 represents these conformance relationships. We clarify the different types of conformance relationships by means of the power window system. A control algorithm is modeled by a control engineer using formalisms such as causal block diagrams, statecharts, and so forth. Each *model* is typed by a metamodel: there exists a conformance relation between them. Since we are dealing with languages, this conformance relation is called linguistic and the metamodel is called a *Linguistic Type Model (LTM)*. The relation has to be strict in the sense that each structural element of a *model* must conform to some element in the *Linguistic Type*

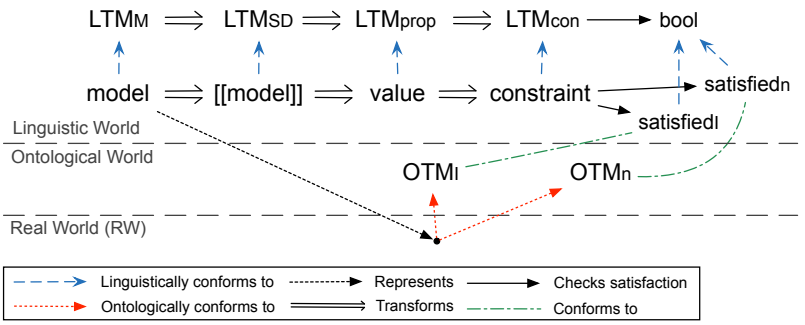


Figure 3.5: Linguistic versus ontological models, based on [BKV14]

Model. Semantics is given to a *model* by defining a *Semantic Domain (SD)* and a semantic mapping function (*[[.]]*) that maps a *model* onto its meaning, an element of the *Semantic Domain*. For example, the control model of the power window system can be transformed to a Petri-Net (PN) model, that linguistically conforms to the PN metamodel, 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 (i.e., design parameters) since they are situated in the linguistic world. Subsequent transformations will check whether a linguistic property satisfies a constraint using a function that returns a logical value (True or False). The linguistic models are modeled with Closed World Assumptions (CWA). This means that if a property is not modeled, it is assumed False.

As stated earlier, each viewpoint interprets (and implements) the requirements keeping in mind some ontological properties (e.g., Sufficient Control Performance? and Schedulable System?). We use the question mark to make explicit that these are ontological properties that must be checked based on the linguistic performance values. As a result, each *model* is typed by one or more *Ontological Type Model (OTM)* representing the implicit knowledge of the engineer. An *Ontological Type Model* categorizes 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* as the representation of the ontology must also be modeled 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 perspective, no strict relation exists between a *model* and an *Ontological Type Model*. 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

modeled with Open World Assumptions (OWA).

The described notion of ontological reasoning is regarded as a key enabler of Contract-Based Co-Design. Designing viewpoint-specific ontologies and combining them is, however, not obvious and highly depends on the design process. By analyzing different design processes used in industry, we detected three fundamental reasoning patterns that can be combined to enable ontological reasoning in any kind of design process: Multi-Semantics (MS), Multi-Abstraction (MA), and Multi-Viewpoint (MV). Each of these patterns rely on the foundational concepts of Figure 3.5 and is typed by a **Purpose**, **Structure**, and **Reasoning about Consistency** description.

Multi-Semantics (MS)

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

Structure: Figure 3.6 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 implicitly 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?, Sufficiently Performant?, Schedulable?, Deadlock Free?, and so forth.

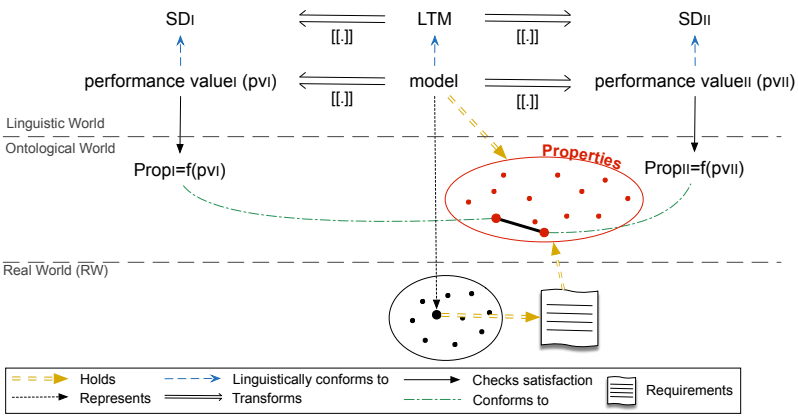


Figure 3.6: Linguistic and ontological relationships for the *Multi-Semantics* pattern

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*. This is denoted by the conformance relation in Figure 3.6. By mapping the *model*, using a semantic mapping function *[[.]]*, to a *Semantic Domain* 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 analyze different linguistic properties. The result of these analyses are called *performance values (pv)*. They can be expressed in any data type (numeric, boolean, string, etc.).

To check whether a linguistic property satisfies a certain ontological property, we test its related *pv* 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:

1. 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.
2. Both performance values must satisfy the same ontological property. In this case, the model is consistent with itself. Otherwise the model is (i) intra-model inconsistent, (ii) the semantic mappings are inconsistent, (iii) different linguistic properties are checked, or (iv) the model is infeasible.
3. There are (transitive) relations between the ontological properties that must be satisfied. According to [DDV15], these relationships can be categorized as being $\mathbb{L}1$, $\mathbb{L}2$, or $\mathbb{L}3$. $\mathbb{L}3$ relationships are the most precise. They define a precise mathematical relation between properties (and their related design parameters). $\mathbb{L}1$ relationships are regarded as the lowest level of precision, expressing the existence of a relationship between properties without specifying to which extent properties are related to each other. In between, $\mathbb{L}2$ relationships express the sensitivity between two related properties. Depending on the type and direction of the relations, the satisfaction relationship will lead to category 1 or 2.

Motivating Example: We clarify the pattern by means of the power window system. The control model shown in Figure 3.7 illustrates an implementation of one of the PWC components in Figure 1.2. The *model* linguistically conforms to the metamodel of Simulink® Stateflow®. On the one hand, executing a simulation gives semantics to the control model (e.g., in the form of a simulation trace). From this, we obtain *performance 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 *Sufficiently Quick Response Time?* property. On the other hand, a transformation to a PN 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 *Sufficiently Quick*

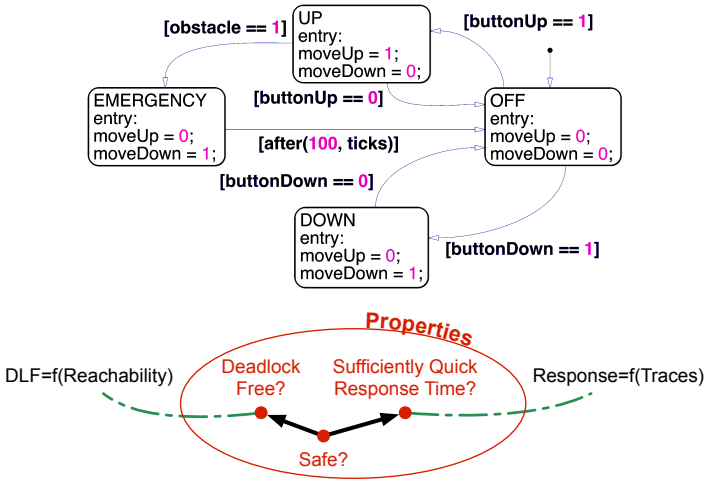


Figure 3.7: *Multi-Semantics* pattern example—model of the power window controller and its ontology

Response Time?. *Deadlock Free?* and *Sufficiently Quick Response Time?* are orthogonal.

Multi-Abstraction (MA)

Purpose: 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 3.8. As in every design process, a written set of requirements 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 viewpoint, 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*:

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

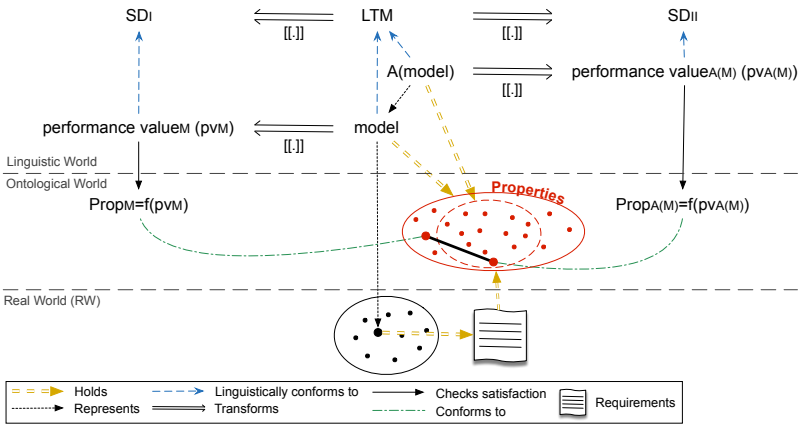


Figure 3.8: Linguistic and ontological relationships for the *Multi-Abstraction* pattern

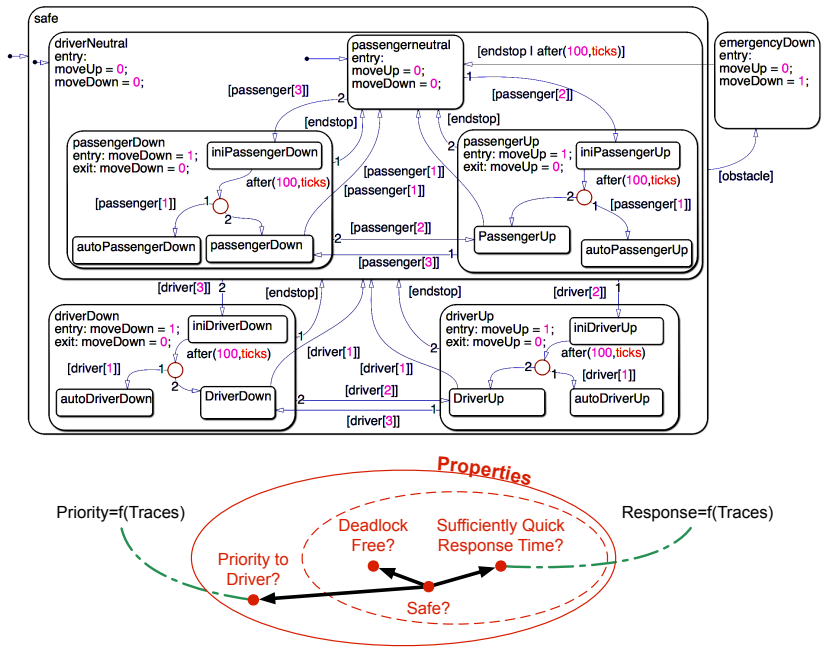


Figure 3.9: *Multi-Abstraction* pattern example—refined model of the power window controller and its ontology

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 3.9 shows a refined implementation of the power window controller and the set of ontological properties it satisfies (via its performance values). Our refined model still satisfies the ontological properties 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 different design artifacts, the *Safe?* property should be moved from the inner to the outer set of properties.

Multi-Viewpoint (MV)

Purpose: The MV pattern is related to the multiple viewpoints that exist on the real world system when designing a CPS. 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 3.10 depicts the structure of the Multi-Viewpoint pattern. Given the set of requirements describing the behavior of the real world system for a given context, each viewpoint reasons about certain linguistic properties and their related ontological properties. These sets are represented by the dashed ovals in Figure 3.10. However, since some of the requirements are shared among the views, properties will concern both viewpoints 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: As the ontological properties in 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 or (ii) if the performance value(s) satisfy a viewpoint-specific ontological property that has a relation (after transitive closure) with a

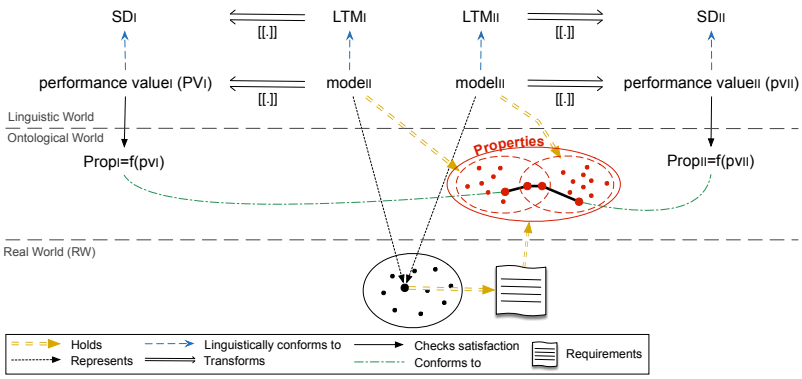


Figure 3.10: Linguistic and ontological relationships for the *Multi-Viewpoint* pattern

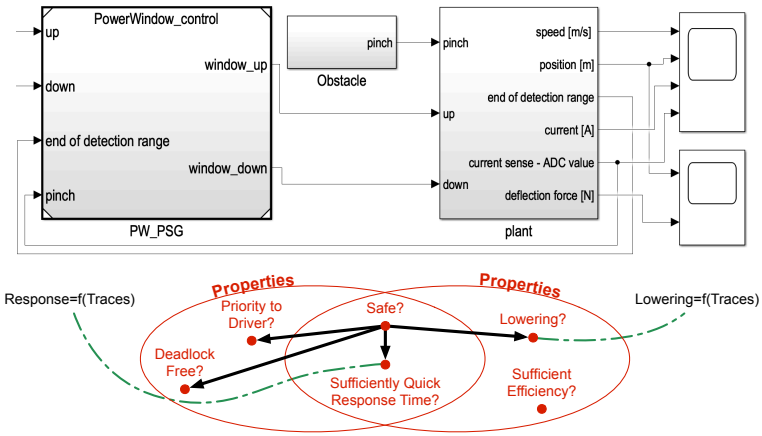


Figure 3.11: *Multi-Viewpoint* pattern example—control and mechanical viewpoint of the power window and their respective ontologies

property in the intersection. If for (i) and (ii) the viewpoint-specific performance values satisfy the property, the model is consistent with itself. Otherwise, the model is inter-model inconsistent. When operating at the same level of abstraction, this type of (in)consistency is specified as horizontal (in)consistency in the literature. 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 again illustrate this pattern by means of the power window system. The upper part of Figure 3.11 depicts how the power window controller is connected to a plant model, while the lower part of Figure 3.11 shows relationships between ontological properties. As already shown in the previous patterns, the power window controller is modeled 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 *Sufficient Efficiency?*, *Lowering?*, *Safe?*, and *Sufficiently Quick Response Time?*. 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 *RSufficiently Quick Response Time?* for the control view. Since the properties *Safe?* and *Sufficiently Quick Response Time?* are part of the intersection, inter-model consistency can be guaranteed.

3.5.2 Ontological Reasoning in Multi-Viewpoint Design Processes

While demonstrating the fundamental design operations in the previous subsection, it became clear no single pattern can be used on its own in a complete design process. This subsection reflects on the design of the power window system using the fundamental pattern operations, validating how consistency can be guaranteed during the design of a CPS.

Figure 3.12 shows the ontologies related to the design of the power window system. Centralized in the figure, one may recognize the MV pattern combined with the MA pattern that where demonstrated in Figure 3.11 and Figure 3.9 respectively. Both patterns are concurrently used by the control and mechanical engineer. Note that we have added the ontological property *Performant System?* to indicate control engineers reasons about control performance as well during their design. As already mentioned before, the third stakeholder (i.e., the embedded engineer) is concerned about the deployment of the control algorithm onto an ECU. For that, they use a hardware platform that is designed by the same or an additional engineer keeping in mind ontological properties such as *Balanced Processor Load?* and *Acceptable Cost?*. A relation

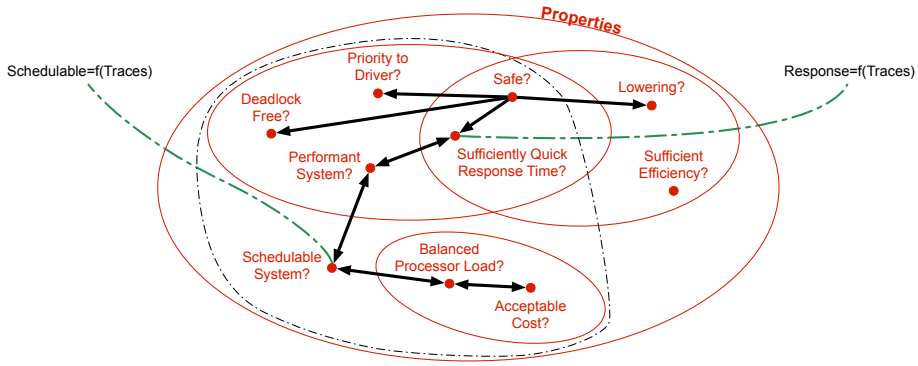


Figure 3.12: Ontologies related to the design process of the power window

between the properties exists since more hardware resources (resulting in a lower load) leads to a higher cost and vice versa. This is symbolized by an ontology that has a disjoint relation with the ontologies regarding the design of the control and plant model. To this end, we say that the viewpoints related to the control and embedded domain are orthogonal.

However, a consistency relationship between the software (control algorithm) and the hardware (ECU) exists from the ontological property *Balanced Processor Load?* to *Performant System?* through the property *Schedulable System?*. 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 *Sufficiently Quick Response Time?*, an indirect link between the schedulable system and the response time (of the control algorithm), to reverse the movement of the window, exists. Due to this ontological reasoning, a schedulable system implies a system to be safe.

3.6 Contract-Based Co-Design Driven Multi-Viewpoint Design Processes

Ontological reasoning ensures that semantically different parameters can be related to each other using an ontology. On the other hand, contracts are used to formally describe a component's behavior using a set of assumptions and guarantees over its variables (i.e., design parameters). To deal with heterogeneous viewpoints on a system, possibly at different levels of abstraction, an ontology for designing CPSs should be able to relate viewpoint-specific contracts such that a change in one system viewpoint is propagated to the other system viewpoint. In this section, we propose a method that combines A/G contracts (Section 2.2.2) with the notion of an upper ontology (Section 2.2.3). Using a common set of *Classes* and *Properties*, the upper ontology syntactically and semantically relates viewpoint-specific ontologies such that an ontology as the one in Figure 3.12 is constructed. The proposed method allows engineers to reason about consistency during contract negotiation so that concurrent design is facilitated. Hence, we name the method Contract-Based Co-Design.

3.6.1 Combining Assume/Guarantee Contracts with Ontologies

We conceptually illustrate the proposed approach using the example of Figure 3.13. On the left-hand side, contracts are specified for the control and platform viewpoint of a particular system. The latter viewpoint relates to the computational platform, combining hardware and software, of the embedded domain. As a hardware engineer selects a computational platform and a software engineer configures a RTOS, both are able to influence the WCRT for a particular (software) component. Consequently, WCRT belongs to the set of parameters that must be guaranteed

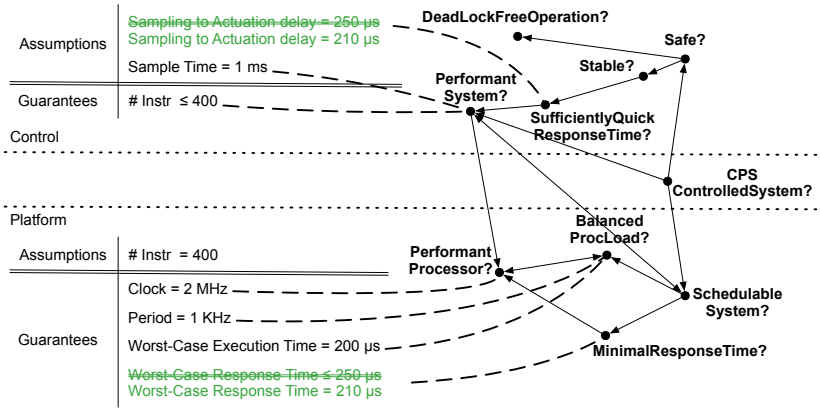


Figure 3.13: Conceptual representation of semantic interoperability between viewpoint-specific contracts (left-hand side) using an upper ontology (right-hand side)

by the platform (hardware and software engineer). On the other hand, the control algorithm is highly influenced by the timing related parameters of the hardware platform. In particular, the WCRT is responsible for a Sampling-to-Actuation (StA) delay, indicating the maximum duration between sampling an input and actuating an output, of the control algorithm's component. Since the control engineer is not able to influence this parameter, the StA delay is part of the set of parameters that are assumed by the control engineer. To keep these syntactically different, yet semantically equivalent, design parameters consistent, the upper ontology on the right-hand side of Figure 3.13 is used. The upper and lower part represent the, rather simplified, domain ontologies of the control and platform viewpoint, respectively. The middle part represents the, conceptually simplified, overall CPS ontology that enables one to relate domain ontologies and, as such, subsumes them. Each node represents an ontological property while a link between properties denotes a *Requires* relationship. For example, a controlled CPS requires a system to be safe, performant, and schedulable. To be able to relate syntactically, and possibly semantically, different design parameters, there need to exist a *Conformance* relationship between the evaluation of the parameter and one or more ontological properties. This implicit domain knowledge is conceptually made explicit using the dashed lines in Figure 3.13.

Suppose an embedded engineer concludes, after executing a schedulability analysis, that the WCRT can be refined from an estimated range $0\,\mu s$ - $250\,\mu s$ to a fixed value of $210\,\mu s$. Using the ontological reasoning capabilities it can be deduced which design parameters, from the same or other viewpoint, may be influenced by this change. Therefore, the directed *Requires* relationships are reversed such that they become *Influences* relationships. As such, it is deduced that changing the WCRT in the embedded domain influences all three design parameters of the control domain. *Requires*/*Influences* relationships are regarded as the *lowest level of precision* ($\mathbb{L}1$) relationship that can exist between two properties, that is, it is known that there exists a

semantic relationship between them. They can be elevated to $\mathbb{L}2$ or $\mathbb{L}3$ relationships. For example, in the case of Figure 3.13 it is derived that the StA delay has an *Equivalent* relationship to the WCRT design parameter as they are semantically equivalent. Other control design parameters have a *Mathematical* (i.e., $\mathbb{L}3$) relationship with respect to the WCRT as they are semantically different from each other.

As it is not the intent to unilaterally strengthen the contract of another viewpoint (i.e., strengthen the guarantees), but rather refine the other viewpoint's contract according to definition 2.6 of the CBD theory, it is concluded to weaken the StA delay assumption such that its value equals the WCRT guarantee. Suppose, however, that after analysis it would have been concluded that it is impossible to satisfy the maximum estimated WCRT of 250 μs, then both design parameters *#Instr* and *Sample Time* should be updated so that the platform guarantee can still be met. As already mentioned, unilaterally strengthening another one's viewpoint contract is not allowed such that special action is required. We will further elaborate on contract refinement in the next subsection where we discuss the CBCD method.

3.6.2 Contract-Based Co-Design Method

So far we elaborated on how ontologies can complement A/G contracts, we deliberately neglected how viewpoint-related contracts are achieved. Formulating viewpoint-related contracts, and in particular what should be assumed and guaranteed, is challenging for engineers considering the fact that they have limited knowledge of each others' domain and viewpoint on the system under design. By explicitly modeling both properties and design parameters in the ontology, however, the here presented Contract-Based Co-Design method enables to derive contracts specific to a particular viewpoint while ensuring consistency among them.

Inspired by the work of Derler et al. in [DLTT13], as discussed in Section 3.3, the CBCD method consists of four phases: (i) defining an architecture for each viewpoint, (ii) negotiating a (set of) contract(s), (iii) deriving contracts and detailing the behavior for each viewpoint, and optionally (iv) refining/renegotiating the contracts. Focusing on the control-embedded system viewpoints, Figure 3.14 illustrates by means of an FTG+PM how the CBCD method can be used within a concurrent design process. Note, however, that the CBCD method in itself does not impose a design process on the engineers. As such, Figure 3.14 only illustrates a possible enactment of the design process. To assist engineers in CBCD driven design processes, dedicated tool support is invaluable for each of the phases of our method. Section 3.7 elaborates on a framework that integrates those tools while in this subsection we detail the aforementioned four phases of the CBCD method.

Phase 1 - Defining an Architecture for each System Viewpoint

As shown in Figure 3.14, a design process typically starts by receiving a set of (textual) system requirements from which the viewpoint-related specifications are derived. Based on these, engineers are able to reason on possible architectures, also referred to as preliminary designs. For example, a control engineer is able to reason about: a concept for the control strategy and its related complexity, demarcating control functionality in terms of reusable components, how these components are related to each other in terms of input/output interfaces, and so forth. With respect to the platform, the hardware engineer is able to reason about: a set of possible ECUs and their related processing power, a communication channel to connect multiple ECUs, and so forth. Optionally, resources may be budgeted by the software engineer. This is particularly useful when the hardware platform will execute multiple control algorithms originating from different control engineering teams.

As an example, we refer back to the design of the power window system. Being a control engineer, it is decided that the control architecture consists of four functional components: *Debounce*, *ControlExclusion*, *PWC_DRV*, and *PWC_PSG* (see Figure 1.2). On the other hand, being a hardware engineer, it is opted for an architecture consisting of a *LPP* and a *HPP* connected using a CAN-bus (see Figure 3.4).

Phase 2 - Negotiating a (Set of) Contract(s)

In parallel with reasoning about the architectures, the textual requirements are translated to entries belonging to the set of guarantees of a system contract. Along with the architectures, the system contract serves as the input of the negotiation phase, shown by means of FTG+PM in Figure 3.15, in which it is decided how both architectures relate to each other, in effect, how the control architecture can be *mapped onto* the hardware architecture. As such, one or more *mapping contracts* are negotiated that are, in line with the CBD theory of Section 2.2.2, refinements of the system contract. Continuing the design of the power window system, it may be decided among the engineers to execute each control component by a dedicated software task. Components that implement less complex algorithms will run on the *LPP* while more complex components will be executed on the *HPP*. For each of these *mapping decisions*, a *mapping contract* is negotiated containing all design parameters that are of interest for the involved viewpoints. An example of such a mapping contract is shown in the middle section of Figure 3.16.

Keeping in mind the specifications and the architectural design, engineers estimate the value (ranges) for each design parameter to be negotiated. They therefore rely on tools to estimate design parameters from their conceptual designs (e.g., WCET analysis from a Simulink® model [KLFP02]), historical data, and/or domain experience. For example, when deciding that the software task holding the *Debounce* component will run on the *LPP*, the hardware engineers estimate that a clock speed of 100 KHz is needed to schedule the component. Therefore, negotiated

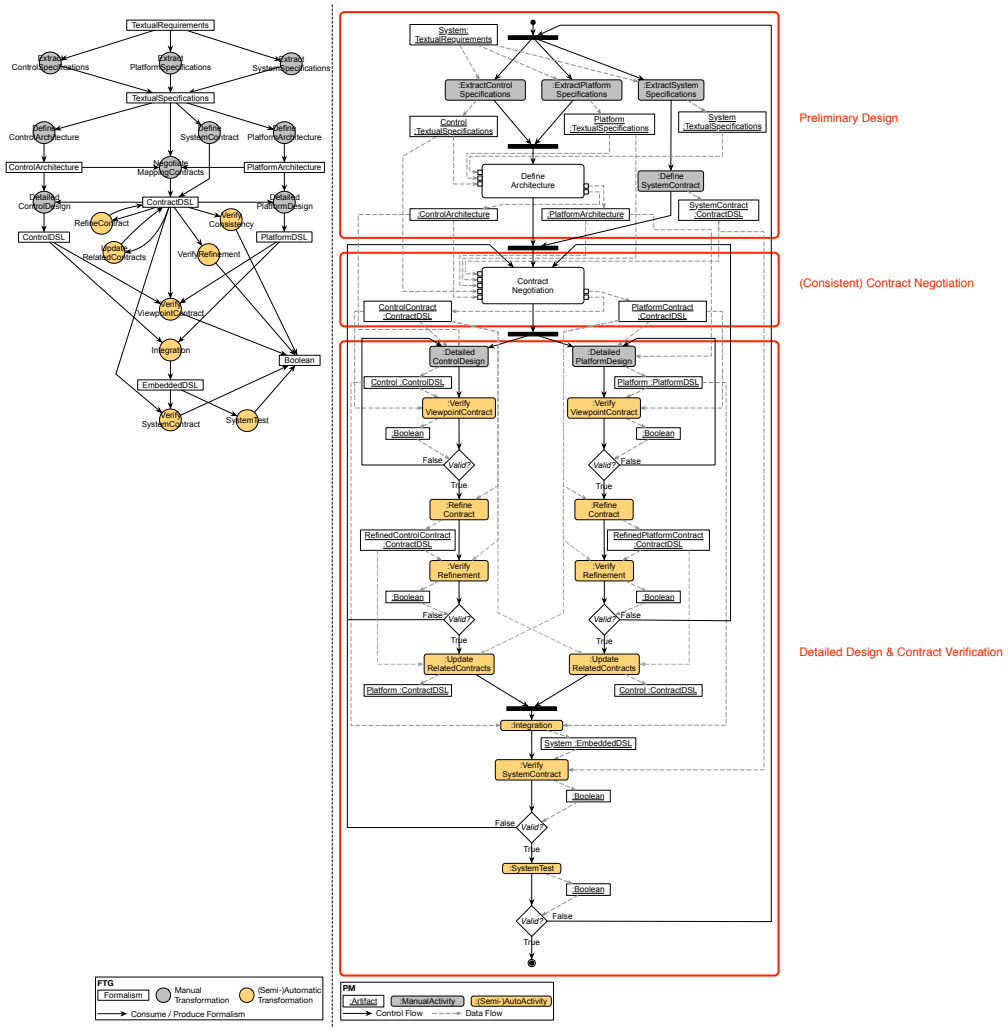


Figure 3.14: Possible enactment of a co-design process using the Contract-Based Co-Design method

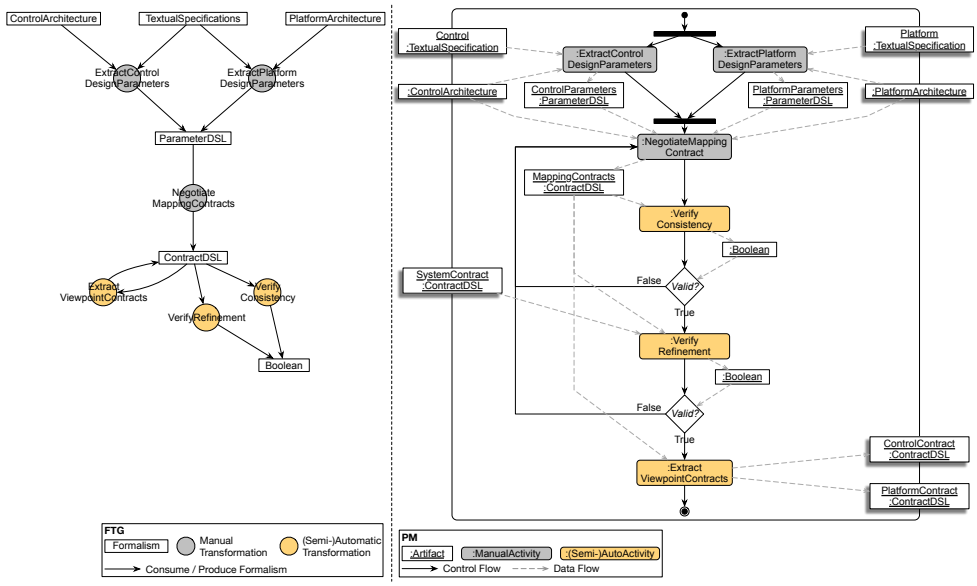


Figure 3.15: Detail of the negotiation process

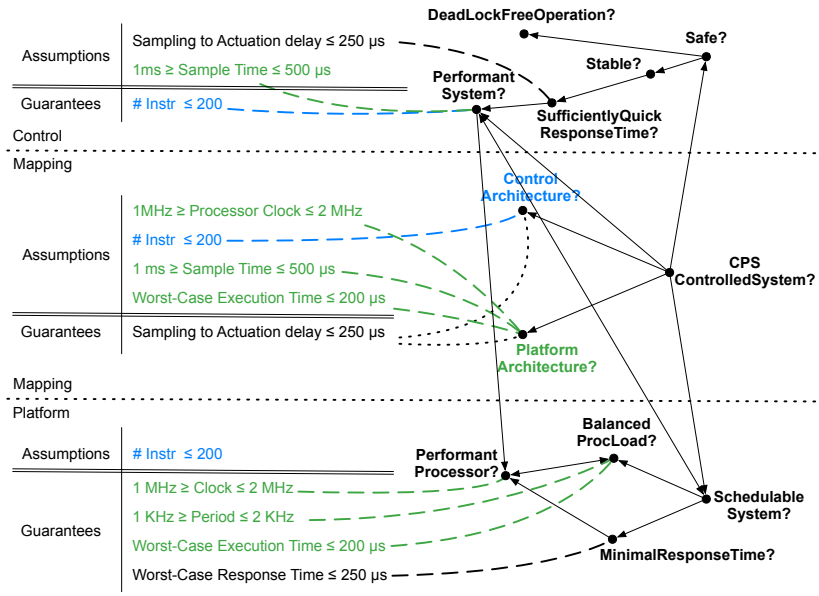


Figure 3.16: Example of a negotiated mapping contract and its derived viewpoint contracts

design parameters must be added to the set of assumed mapping contract values, while guaranteed design parameters are refinements of the (guaranteed) design parameters belonging to the system contract. In effect, the latter are requirements that should be guaranteed by the mapping. Note that a design parameter can be characterized by a range of values instead of one well-defined value. This can be useful if architectural constraints for certain parameters are known (e.g., the processor clock) while the final value has not yet been decided.

Although the value (ranges) of the negotiated parameters are estimations they must be consistent with each other. Only then, consistent parallel design can be guaranteed and, consequently, successful integration ensured. Consistency in the negotiation phase can be verified using the predefined upper ontology, conceptually shown at the right-hand side of Figure 3.16. Using the relationships that exist between ontological properties, parameters can syntactically and semantically be related to each other. For example, assume in Figure 3.16 the parameters *Number of instructions (# Instr)* and *ProcessorSpeed* are already defined while the parameter *WCET* is under negotiation. Although they are semantically different, their mathematical relationship is known within the ontology, that is, $WCET = \#Instr * 1/ProcessorSpeed$. We here simplify the calculation of the WCET by estimating the longest path of the conceptual control strategy, without taking into account the effect of memory, pipelining, and so forth. In case a *WCET* of 150 us would be decided, a possible inconsistent system may be designed as in some cases (i.e., when the chosen processorSpeed is lower than 1.33 MHz) the system contract will not be satisfied. Besides mathematical relationships (i.e., $\mathbb{L}3$ relationships), $\mathbb{L}1$ and $\mathbb{L}2$ relationships defined in the ontology may be used for evaluating design parameters using behavioral (simulation) traces.

The reader may notice that our definition of a mapping contract has similarities with the resource-aware contract shown in Table 3.2. Both contracts contain design parameter of multiple domains, although, their place in the contract differs.

Phase 3 - Detailed Design using the viewpoint contracts

Once the mapping contract is negotiated, domain engineers need to detail their preliminary design: behavior is added to the control components, parameters of the software/hardware platform are further detailed, etc. Preferably, these engineering activities are executed in parallel to reduce design time (and related costs). To facilitate this, contracts addressing only one viewpoint must be derived from negotiated mapping contracts such that it is clear for engineers implementing a particular viewpoint what can be assumed and what should be guaranteed.

Deducing on the viewpoint contracts is enabled using the defined upper ontology. Parameters defined in a mapping contract that are related to a certain architecture become part of the guarantees of the viewpoint contract related to that architecture. Given the mapping contract in Figure 3.16, for example, the elements *Processor Clock*, *Sample Time*, and *WCET* are translated as guaranteed

parameters of the platform contract. As such, it are the hardware engineers who should take care of their correct implementation. Likewise, the parameter *# Instr* is translated as a guaranteed parameter of the control contract. Indeed, the control engineer is responsible for maintaining this limited amount of instructions that can be influenced by modifying the complexity of the control algorithm. Note how design parameters can be (syntactically) translated to the terminology used within a particular viewpoint. The *Sample Time* defined in the mapping contract, for example, is translated to a *Period* entry for the platform viewpoint.

Deciding upon the assumptions of a viewpoint-specific contract depends on how guarantees from one viewpoint influence the guarantees of another viewpoint. Therefore, the *Requires* relations between ontological properties are used: if there exists an ontological path between design parameters guaranteed by different viewpoints, then the design parameter at the root of the path will be considered as an assumption of the other viewpoint. A similar reasoning can be used if their exists a more precise relationship between design parameters: if a guaranteed design parameter contains an equation (i.e., a *Mathematical* relationship) in which one of its terms refers to a guaranteed design parameter of another viewpoint, then that parameter will be considered as an assumption by the other viewpoint. For example, the *# Instr* can be calculated using the equation $WCET * ProcessorSpeed$. Since both *WCET* and *ProcessorSpeed* are guarantees of the platform contract, the (control) design parameter *# Instr* is regarded as an assumption of that contract. If a design parameter of the mapping contract is related to more than one viewpoint (e.g., because of an *Equivalent* relationship), then it is up to the engineers to decide who will guarantee the parameter. In case of the *StA delay*, that is equivalent to *WCRT*, it is decided that the platform should guarantee the requirement. After all, timing related parameters will highly depend on the computational power of the platform.

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

Phase 4 - Refining/Renegotiating a (set of) contract(s)

While it is obvious that the detailed design should satisfy the contract related to the viewpoint, engineers may decide to refine their contract(s) as shown in Section 3.6.1. In doing so, they keep in mind Equations 2.6 and 2.7 of the CBD theory. If and only if these equations hold, changes can be pushed to the other viewpoints without further notice. If a change in a contract cannot be regarded as a strict refinement, the mapping contract should be renegotiated such that the design process returns to phase 2.

3.7 An Integrated Framework Supporting the Contract-Based Co-Design Method

The fact that one has to derive relationships between parameters by means of domain knowledge stored in an ontology, makes it difficult and time consuming for engineers to manually apply the proposed method, especially when they are dealing with a large set of parameters. For this reason, we assert that the applicability of the CBCD method in an industrial context is only feasible when parts of the CBCD method are automated and/or hidden from the engineers. As such, an integrated framework as shown in Figure 3.17 is developed to assist engineers in applying our method in their current design processes. Note that the framework preserves design processes while acting as a single point of truth in the background of the preferred development chain of tools and methods. In that respect, the integrated framework is able to interact with third-party tools that provide an Application Program Interface (API) or that store a model in some kind of accessible text file. Within the scope of this dissertation, we limit ourselves to interactions with Simulink® (control domain), MAST (embedded domain), and Protégé (ontology editor). We refer to Section 2.3 for a detailed description of these tools.

We have opted to build the framework using the EMF, introduced in Section 2.3.1, in combination with Eclipse Epsilon [Thea], enabling us to define: (i) the (EMF) metamodel for both the control and embedded domain, (ii) the model-to-model transformations such that the architectures can be exported to, and imported from, third-party tools, and (iii) the services to verify refinement and consistency. Using Eclipse Sirius [Thed], a Graphical User Interface (GUI) is created for

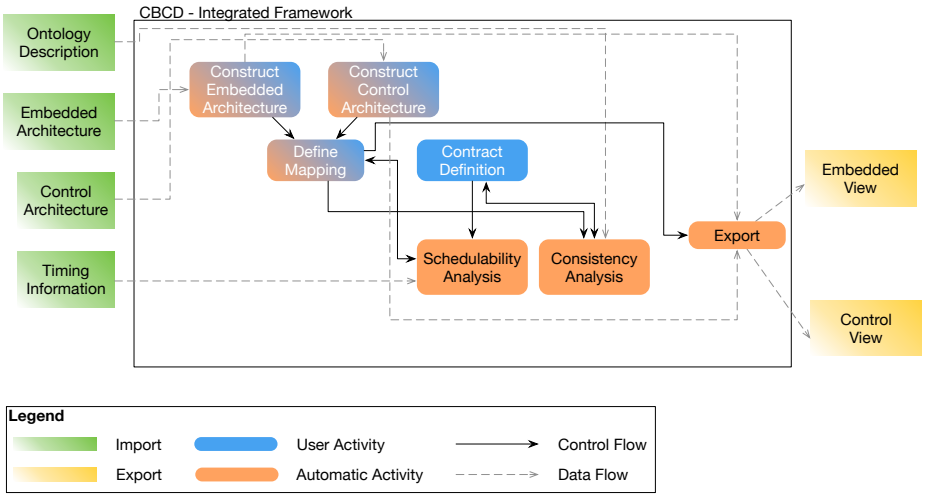


Figure 3.17: Architectural overview of the CBCD framework

constructing the architectures, defining the contracts, and initiating consistency analysis. In what follows, we elaborate on the services the integrated framework provides to its users.

3.7.1 Defining a Scalable and Reusable (Upper) Ontology

Although we have shown how A/G contracts can be complemented with ontologies, it is yet unclear how to define an (upper) ontology that is scalable and reusable among projects and/or organizations. In what follows we detail how we defined such an upper ontology that relates the ontologies of the control and platform viewpoint. For this, we use the W3C's standardized Web Ontology Language2 (OWL 2) in combination with Protégé [Sta16] as an ontology editor. In selecting a semantic reasoner we considered expressiveness and incremental classification as key features. In light of the design of the CBCD framework, integration with the Java programming language is a benefit. As such, we have opted to use the Pellet reasoner [SPG$^+$07].

In our definition of an ontology, a domain ontology explicitly describes the implicit engineering knowledge of a particular viewpoint on the system under design in which real world entities are represented by so called *Instances*. They are classified by one or more *Classes* that in turn can be structured using a superclass-subclass hierarchy in which a superclass subsumes its subclasses. They are regarded as sets such that binary operations (union, intersection, and so forth) can be applied to two or more *Classes*. *Properties* are used to express binary relationships between *Classes*, in effect, on their related *Instances*, using some appropriate logic (i.e., description logic). The upper ontology relates the semantically different domain ontologies using generic *Classes* that are common across all viewpoints.

Focusing on control-embedded co-design, we identify four domain ontologies, one for each viewpoint: control, hardware, software, and platform in which the latter subsumes both the hardware and software viewpoints and their related domain ontologies. As an example of a domain ontology, Figure 3.18 details the platform domain ontology. Let us now elaborate on how such a scalable domain ontology is constructed. As discussed in Section 3.5.1 and 3.6.1, each domain ontology should at least contain a set of (ontological) properties representing the engineer's implicit knowledge. Therefore, we have defined a superclass *Property* (not to be confused with the *Properties* term of OWL 2 to express binary relationships) in each domain ontology to capture these properties. In each domain ontology, the classification of its properties is preceded by a prefix (e.g., *Hw_*) to indicate the viewpoint to which the property belongs. This will be of use when constructing the upper ontology. Binary relations between properties may exist as well. They are denoted by the *Requires* relationship in an ontology. For example, in order to be able to obtain a *Hw_PerformantProcessor* a *HW_AcceptableProcessorLoad* is required (i.e., the load should not exceed a certain threshold). Note that we omitted the question mark (see Section 3.5.1) to facilitate querying of the ontology.

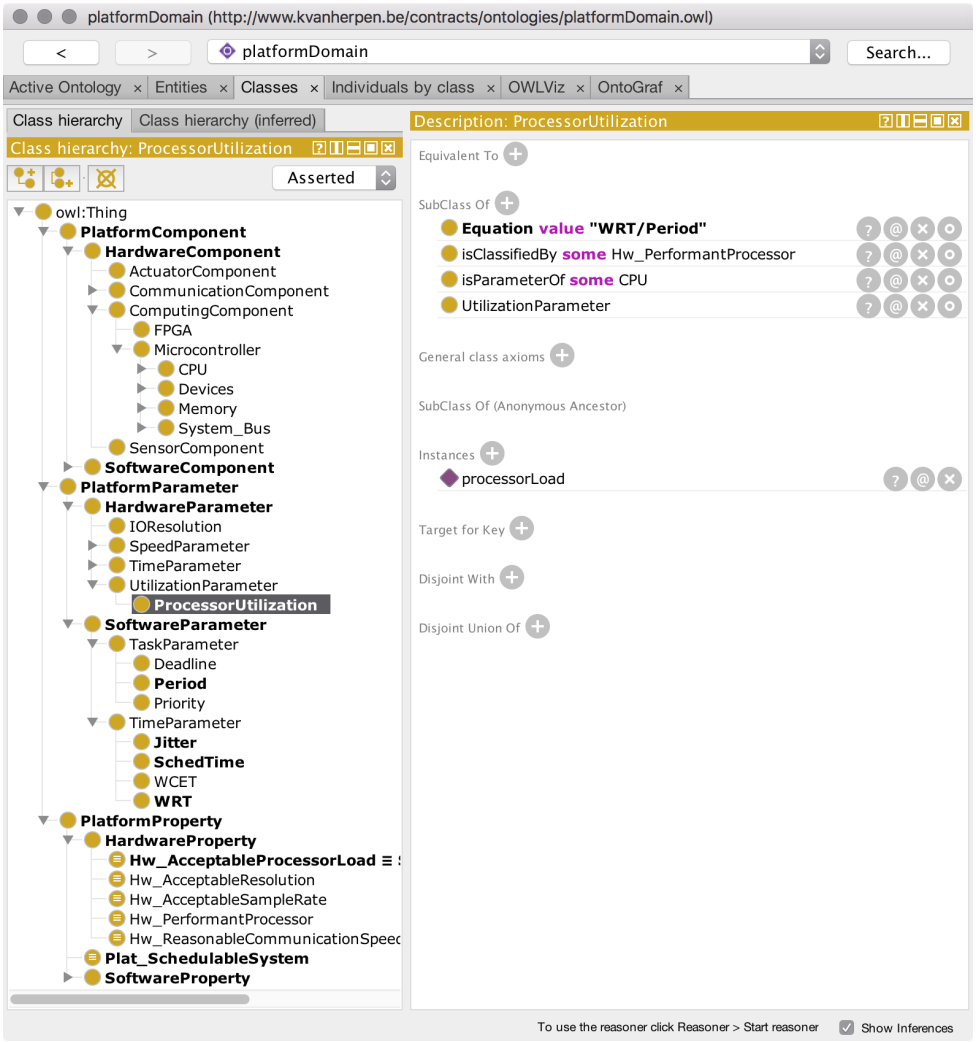


Figure 3.18: Ontology—example of the *Platform* domain ontology

In Section 3.5.1 we defined the existence of a *Conformance* relation between the function that evaluates the value of a design parameter and a(n) (ontological) property. While the evaluation function depends on the application, the implicit relation between a design parameter and one or more properties will remain the same. As such, we have extended each domain ontology with a superclass *Parameter* that contains a set of design parameters. Each parameter has an *isClassifiedBy* binary relationship with one or more *Property* classes. For the platform ontology, for example, the (design) parameter *ProcessorUtilization* is defined and linked to the property *Hw_PerformantProcessor* using an *isClassifiedBy* binary relationship. Explicitly modeling the set of design parameters at an ontology level has some additional advantages. First, it enables one to define more precise relationships between design parameters, in effect, *Equivalent* and *Mathematical* relationships. Secondly, it enables reuse of an overall CPS ontology in different projects or company settings. However, depending on the tools used or the company setting, domain engineers may use slightly different parameter terminology in their design than the classification chosen in the ontology. Using the notion of *Instances* in Protégé, that are real world entities belonging to a specific class, a domain ontology can be tailored to a particular setting. In case of the design parameter *ProcessorUtilization*, the instance *processorLoad* is defined to denote the real world terminology used by the engineer when designing the hardware.

Design parameters enable engineers to specify the behavior of components. To make the ontology complete, we have added the superclass *Component* containing a set of components used by the engineer to construct (part of) the system. Similar to the *isClassifiedBy* binary relationship, an *isParameterOf* relation may exist between a parameter and component (i.e., between their classes). In Figure 3.18, for example, it is defined that *ProcessorUtilization* is a parameter of a *CPU* component. Again, the taxonomy of the superclass *Component* classifies design components at a higher level, while *Instances* can be created to tailor the classification to a particular engineering setting (and related terminology).

One may notice that the platform ontology acts as an upper ontology for the hardware and software domain ontologies. To that end, three superclasses are defined that subsume the equally named classes of the domain ontologies: *Component*, *Parameter*, and *Property* (each of them preceded by the prefix *Platform*). Note how the upper ontology makes use of the MS and MV patterns described in Section 3.5.1 to relate syntactically and semantically different properties originating from different viewpoints. As we focus on control-embedded co-design, the platform domain ontology will be subsumed by the CPS upper ontology in conjunction with the control domain ontology, as shown in Figure 3.19. Typically, an upper ontology extends the *Property* superclass with additional ontological properties such that properties of the subsumed domains can be related to each other. For the CPS upper ontology, the created *CPSProperty* is extended with the properties *CPS_ControlArchitecture*, *CPS_ControlledSystem*, *CPS_LowCost*, and *CPS_PlatformArchitecture* to relate properties from the subsumed domain ontologies. In that respect, *Requires* relationships are added between (i) the added properties and (ii) the added properties and the subsumed ones. Given the set of asserted axioms (i.e., the constructed ontology), a dedicated semantic reasoner

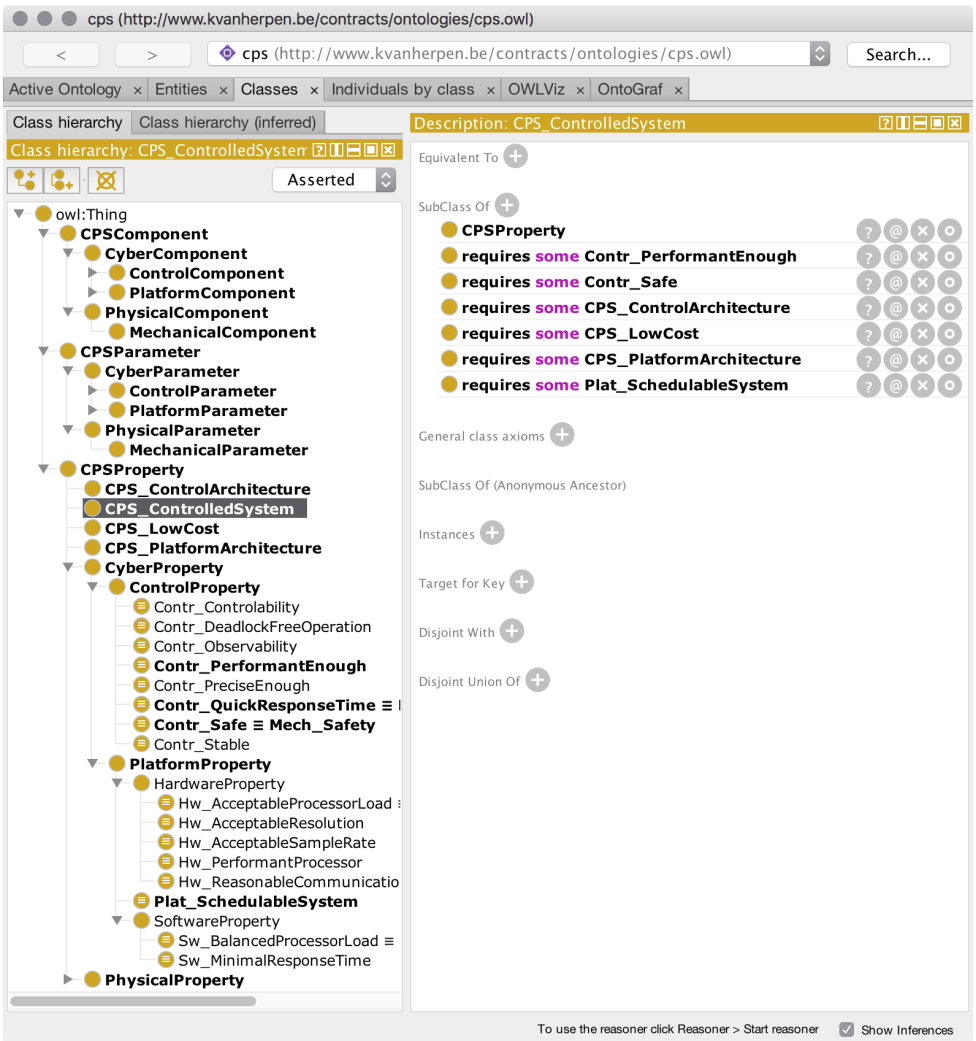


Figure 3.19: Ontology—upper ontology for Control-Platform Co-Design of a Cyber-Physical System

(i.e., Pellet) is used to verify consistency and satisfiability. In addition, a reasoner is able to infer additional classification and relationships from the asserted ontology.

Typically, the domain ontologies are build by the different domain experts while an upper ontology (e.g., the CPS ontology) is constructed by a systems engineer who is familiar with the different subsumed system viewpoints. Note, however, that by classifying parameters and components in the ontologies while using instances to represent the real world parameters, the ontologies can easily be tailored to different companies and/or engineers using different terminology while maintaining the relations. As such, the defined (upper) ontologies can easily be reused. Moreover, the domain ontologies can be easily extended with additional properties, parameters, and/or components using the proposed taxonomy. In case an additional viewpoint needs to be added to the upper ontology, for example the mechanical perspective, a domain ontology is created for that particular system viewpoint that is then imported into the upper ontology. Additional relations (e.g., equivalent relations) may need to be added after import.

3.7.2 Defining Viewpoint-Specific Architectures

Our CBCD method requires engineers to prepare a preliminary design, called architecture, before negotiating contracts. The framework supports this design phase by providing viewpoint-specific environments for each domain, in which a library of components can be used by the engineers to create their architectural models. These models conform to the predefined metamodels of the supported engineering domains (i.e., control and embedded), shown in Figure 3.20 and Figure 3.21. For the control domain, a Simulink® like environment is provided in which control engineers are able to model a top-level architecture consisting of subsystem and model reference blocks connected by single connections or buses. A hardware engineer is able to model a hardware architecture consisting of single and multi-core ECUs connected using a communication channel (e.g., CAN). A software engineer can further detail the hardware architecture by assigning an Operating System (OS), tasks, and runnables to the modeled processors.

As preliminary design architectures may already be modeled using viewpoint-specific tools Model-to-model transformations, possibly in combination with a tool-specific API, can be used to import from these dedicated tools. Currently, the framework supports importing models from Simulink®. To that end, we integrated Massif, as introduced in Section 2.3.6, in the CBCD framework. Using the command line interface, Massif imports Simulink® models to a model that is an instance of a Simulink® EMF metamodel. Using model-to-model transformations, the imported model is transformed to an instance of our control architecture metamodel. Likewise, preliminary design architectures defined in the framework can be exported to viewpoint-specific tools as well. Also for exporting architectures, the framework currently supports exporting models to Simulink®.

With respect to the deployment of the control architecture on the embedded architecture, the framework allows engineers to map the previously defined viewpoint-specific architectures. To

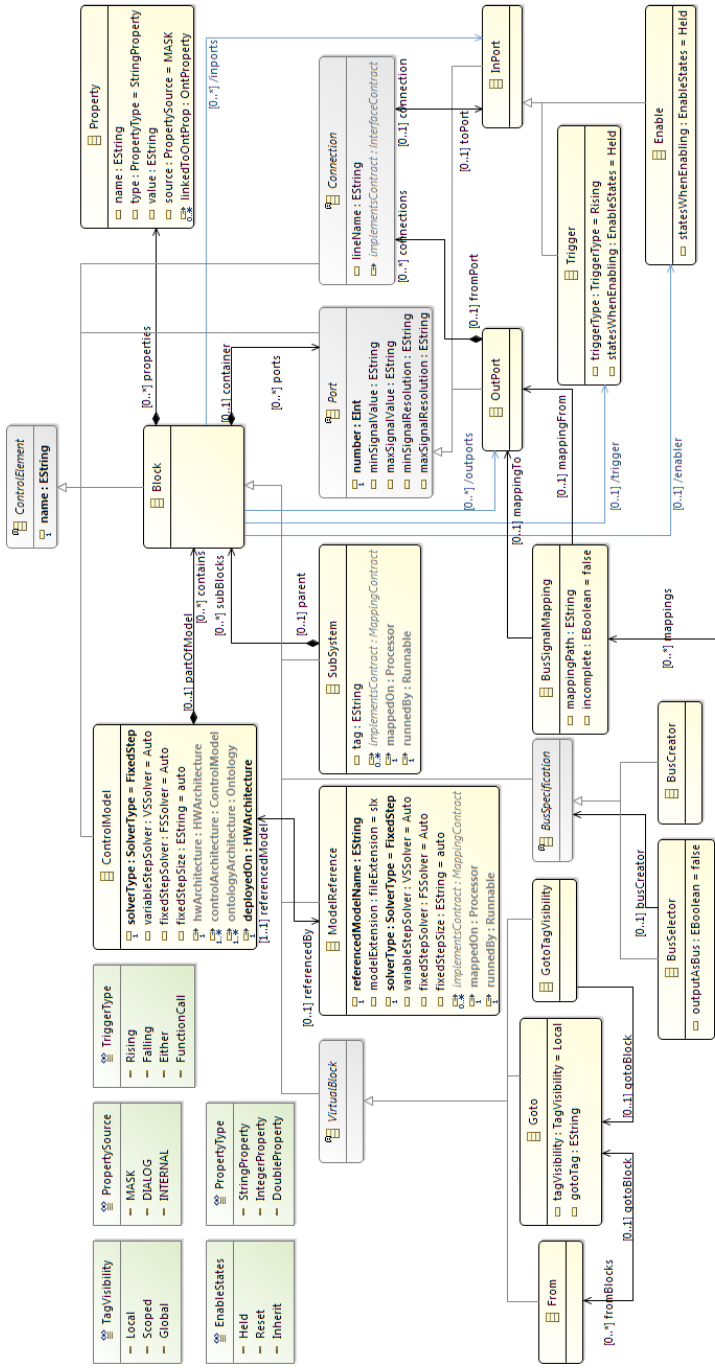


Figure 3.20: CBCD framework—metamodel for the architectural description of the control domain, based on [HSB^{+}]

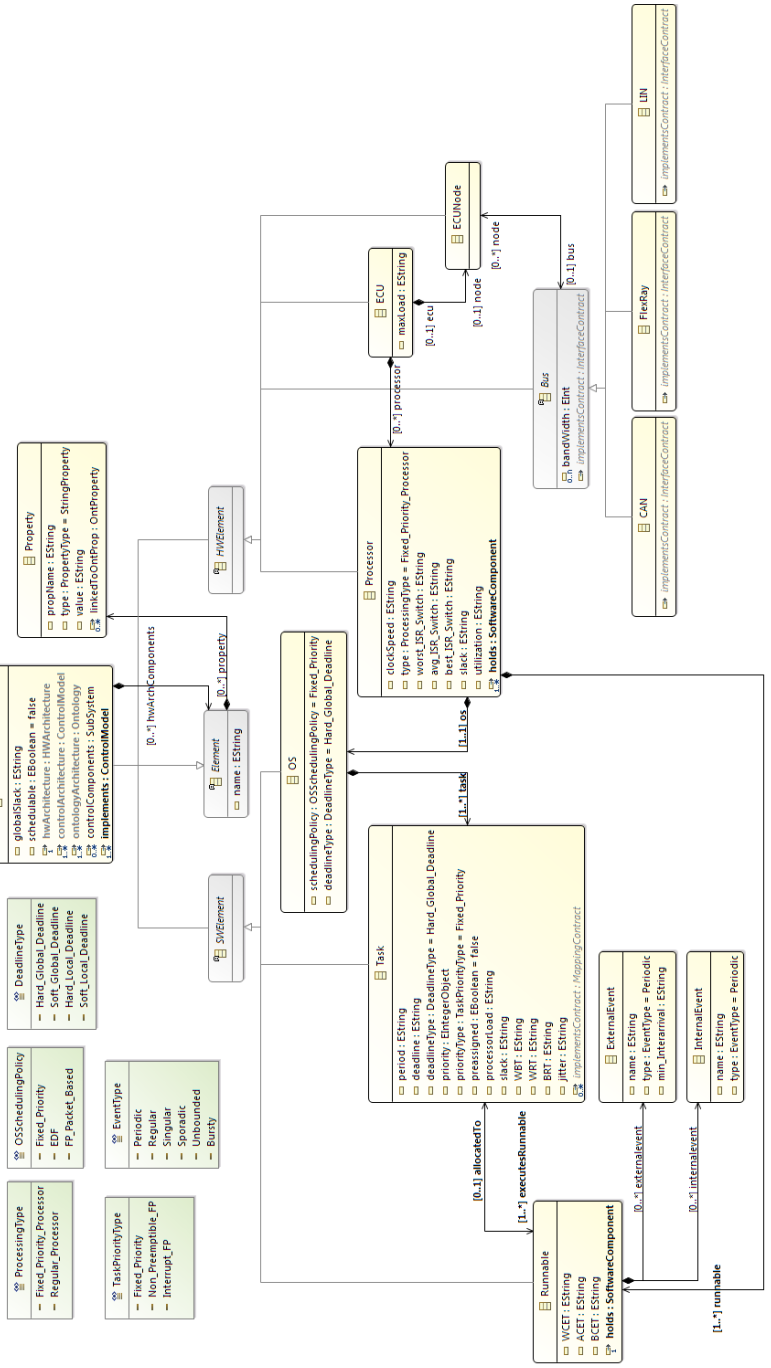


Figure 3.21: CBCD framework—metamodel for the architectural description of the embedded domain

do so, engineers explicitly need to relate architectural components using dedicated mapping connections provided in the available library. For example, a subsystem belonging to the control architecture is regarded as a Software Component (SWC) by the software viewpoint and can be allocated to a runnable, belonging to the software architecture, by relating both components using a *Holds SWC* mapping connection in the environment of the software engineer. It enables the framework to automatically verify whether mapped components implement the same negotiated mapping contract.

3.7.3 Defining Contracts

Using a textual or graphical interface, system and mapping contracts can be specified by the engineers. The contracts can be automatically populated with the design parameters modeled in the ontology. To do so, the Protégé API is used to parse the ontology.

An example of a set of mapping contracts is shown in Figure 3.22. Orange colored design parameters denote the contract's assumptions while the red colored ones denote the guarantees. For each design parameter one can specify a (range of) value(s). Once the contracts are defined, they can be allocated to one or more components belonging to the architectures of the different viewpoints.

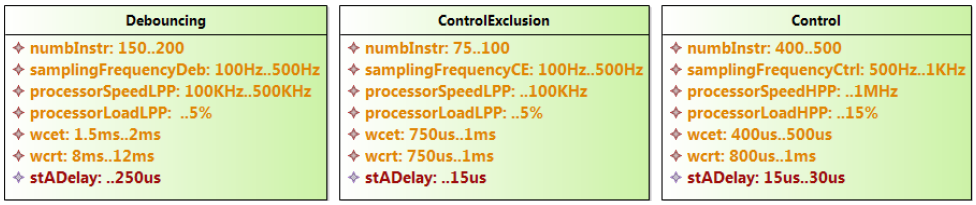


Figure 3.22: CBCD framework—mapping contracts for the power window system

Given the mapping contracts, viewpoint-specific contracts are derived, although, not made explicit to the engineers. Using a decision tree, it is determined which domain should guarantee a negotiated design parameter. Therefore, the ontology is parsed such that the tool can reason about the relations between design parameters using the ontological properties (i.e., ontological reasoning). The decision tree is graphically shown in Figure 3.23.

3.7.4 Executing Analysis

Using a dedicated analysis algorithm, parameters belonging to a negotiated mapping contract are automatically validated for consistency. For this, the relationships that exists in the ontology are explored. Remember that there exist three types of parameter relationships: (i) *Requires*

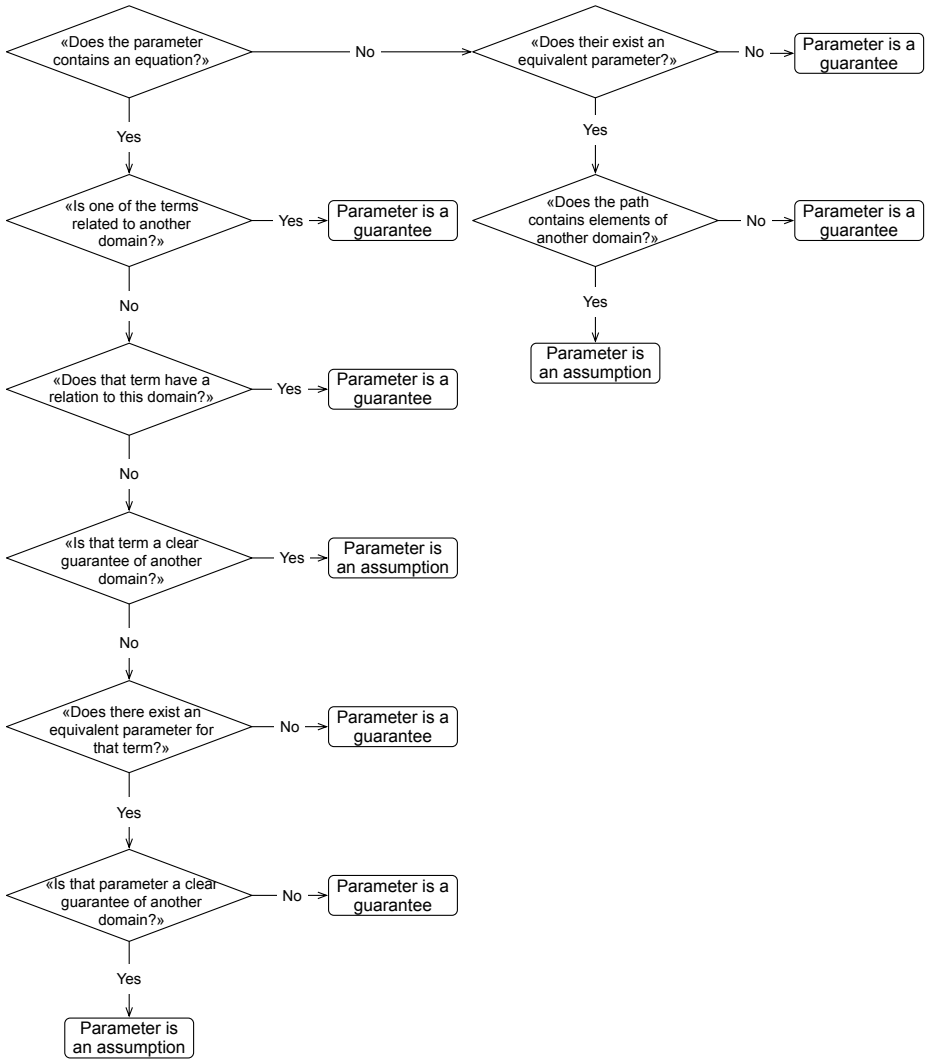


Figure 3.23: CBCD framework—decision tree to determine the viewpoint-specific contracts

relationships specified at the property level, (ii) *Equivalent* relationships specified at the parameter level, and (iii) *Mathematical* relationships specified at the parameter level. For both *Equivalent* and *Requires* relationships it is sufficient to infer the ontology, using the Pellet reasoner, and query for relations such that a graph, containing both parameters and properties, can be created. Given the graph, Dijkstra's algorithm [Dij59] is used to determine the existence of a (shortest) path between two parameters, so that a same reasoning method is used as discussed in Section 3.6.1. *Requires* relationships are elevated to *Mathematical* relationships if there exists a path between two or more parameters that are related to each other by means of a mathematical equation. As we currently limit ourselves to linear equations for defining a mathematical relationship between parameters, a linear solver can be used to solve the system. However, this only holds if all the independent variables of an equation are known. Since a contract does not need to be complete, in such a way that not all parameters belonging to the upper ontology need to be specified, it is likely that not all independent variables of an equation are known. As a result, linear solvers will not be able to solve the problem or even return an error message. On the other hand, symbolic solvers are able to compute mathematical objects symbolically, so that unknown independent variables remain in symbolic form. As such, we opted to use SymPy, introduced in Section 2.3.4, as a (symbolic) solver. Furthermore, the framework also validates whether the mapping contract(s) refines the system contract. To do so, the consistency algorithm implements Equation 2.6 of the CBD theory. If these consistency analyses turn out to be positive, viewpoint-specific contracts are automatically deduced as discussed before.

The schedulability of the deployed architectures is verified using dedicated third-party tools (e.g., MAST). For this, sufficient information about the deployed system should be provided by the embedded engineer; that is, the priority of the tasks, the WCET, period, and deadline. To enable the schedulability analysis, a model-to-text transformation is used to transform the deployment architecture to a textual model that can be used by the third-party tool. Afterwards, the viewpoint-specific architectures are annotated with the results of the analysis. To do so, ontological reasoning is again used to verify whether there exist semantically equivalent parameters for certain platform design parameters. For example, the schedulability analysis returns a value for WCRT, for which an equivalent design parameter in the control viewpoint exists, that is, the StA delay.

Finally, the framework is also capable of validating whether the implementation of a component satisfies its contract (Equation 2.2). In case of the deployed architectures, it is validated whether the schedulability analysis results (e.g., WCRT, processor load, and so forth) satisfy the parameters defined in the derived viewpoint-specific contracts.

3.8 Conclusion

The application of Contract-Based Design in a concurrent engineering setting with heterogeneous viewpoints on a system under design is not well supported. Because of syntactic and semantic

differences between design parameters of different viewpoints, it is virtually impossible to negotiate consistent A/G contracts. Moreover, it unclear for engineers what may be assumed from other viewpoints and what should be guaranteed under these conditions.

In this chapter we suggested to explicitly model the domain knowledge of each viewpoint using properties in so called domain ontologies. They are combined using general concepts in an upper ontology. Design parameters are related to one or more ontological properties, that in turn can be related to each other using *Require* relationships. This enables ontological reasoning whereby syntactically and semantically different design parameters are related to each other. As a result, consistency of negotiated A/G contracts can be validated so that consistent concurrent design is enabled.

To support engineers in this so called Contract-Based Co-Design process, we introduced a proof of concept integrated framework in which preliminary design architectures and A/G contracts can be defined. Given the upper ontology, the framework is able to validate for consistency among negotiated design parameters specified in the contract(s). In addition, the framework is able to verify whether parts of the implementation satisfy the viewpoint-specific contracts. As the framework acts as a single point of truth in the design process, interacting with third-party tools, these viewpoint-specific contracts are intentionally not made explicit to the engineers.

In summary, we return to the research questions formulated in Section 1.3:

RQ1 *At what level of abstraction can we relate different domains, and their related design parameters, to enforce consistent design of CPSs?*

RQ2 *What is the most appropriate formalism to represent these relations?*

Using domain ontologies, we explicitly modeled the domain knowledge by means of ontological properties. An upper ontology is used to relate these different domains such that ontological reasoning is enabled.

RQ3 *How should contracts be used to ensure consistency between heterogeneous viewpoints?*

We have shown a four phased design process in which top-level architectures are defined for each viewpoint. Defining these architectures enables engineers to reason about a certain mapping and, as such, discuss mapping contracts. Viewpoint-specific contracts are not made explicit, although, one can assign negotiated mapping contracts to the components belonging to the architectures. Using dedicated tool support, consistency, and satisfaction of the contracts is validated.

CHAPTER 4

A Round-Trip Engineering Method Supporting Contract-Based Co-Design Driven Processes

Abstract. When implementing a particular viewpoint on a system under design, engineers are provided an A/G contract that defines what an engineer may assume from other viewpoints and what they should guarantee under these conditions. Although ontological reasoning already provides syntactic and semantic transformations, it is often unclear to what extent these assumed design parameters influence the viewpoint-specific implementations and, as such, the contract parameters to be guaranteed. To tackle this issue, we propose a Round-Trip Engineering method allowing a control viewpoint to semi-automatically integrate hardware related assumptions, corresponding to the deployment, into the control model. The resulting method supports control and embedded engineers in the tradeoff analysis of design parameters when negotiating contracts and explicitly models assumed parameters at the most appropriate level of abstraction.

4.1 Introduction

CBCD as a method enables engineers to reason about multi-viewpoint consistency and provides viewpoint-specific contracts to the engineers in which it is clear what a viewpoint should guarantee under a given set of assumptions. These assumed design parameters are often semantically and/or syntactically guaranteed design parameters translated from another viewpoint. As such, there may exist a circular dependency between the different viewpoints. For example, the software viewpoint of the embedded domain should guarantee a maximum defined WCRT, that is translated to an assumed StA delay for the control viewpoint. Vice versa, the control viewpoint should design a control algorithm with a prenegotiated complexity. Given this assumed complexity, the software viewpoint is able to reason about a schedulable system so that the WCRT can be guaranteed for the software task(s) executing the control algorithm.

Although the CBCD method explicitly models the engineering knowledge using an ontology, engineers are often not aware of these possible circular dependencies. Moreover, given the viewpoint-specific contract they have limited aids in estimating the impact of the assumed design parameters on their modeled implementations. It is therefore opportune that the different viewpoint-specific implementations are (semi-)automatically updated with all the relevant assumptions such that they can be simulated as if they were running on the integrated system. To demonstrate this, this chapter presents a Round-Trip Engineering (RTE) method [SK04], in which the control viewpoint is augmented with software related timing information. Therefore, the top-level control architecture in Simulink® is updated with extra blocks so that control engineers can evaluate the behavior of the deployed algorithm at their level of abstraction.

4.2 Related Work

The literature describes multiple scientific contributions to introduce real-time execution behavior when modeling a CPS. In [EkT01] a tool is presented that enables engineers to co-simulate the functional logic of a (modeled) control algorithm with the behavior of the computing system, combining the view of both control and embedded domain at an appropriate level of abstraction. Eidson et al. [ELM+12] present the PTIDES design environment as an extension to the Ptolemy II framework. It allows a control designer to add a notion of physical time without actually deploying the system. Therefore, PTIDES extends discrete-event systems with a relationship between model time and physical time at sensors, actuators, and network interfaces. Another approach is presented by Guerra et al. [GSDA07] where triple graph transformations are used to back-annotate original models with analysis results. In [Nad13] Naderlinger demonstrates how to manipulate the Zero Execution Time (ZET) simulation behavior of Simulink® models to support real-time execution behavior by introducing building blocks consuming a finite amount of simulation time. In addition, a more general overview of integrating real-time execution behavior

at a functional model level is given by Derler et al. [DLTT13] where a framework of design contracts is proposed to facilitate interaction between control and embedded integration engineers designing CPS.

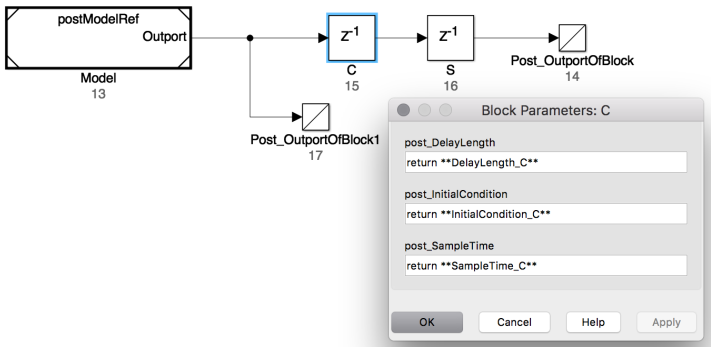
Ciccozzi et al. describe in [CSCS13, CCS13] an approach that is similar to ours. Their round-trip solution consists of three steps: (i) the generation of code from a source model, (ii) monitoring of extra-functional properties at system level, and (iii) back annotation of the source model. However, their back annotation consists of a textual description with implementation related properties meaning the system developer needs to be aware of these specific technical terms in order to optimize the deployment. In [MD14, CMDN15] the authors present the T-Res framework allowing for a co-simulation of the software model and the hardware execution platform. Inspired by TrueTime [HCÅ03, CHL$^+$03], they introduce kernel and task blocks into the Simulink® software model (i.e., the control model that is adapted for implementation). More recently, Li et al. demonstrate in [LMMHY16] a control-embedded co-simulation framework in which a multi-core processing architecture is modeled using the SimEvents® software. In their implementation, the execution of the control algorithm is triggered using the underlying discrete-event execution engine of SimEvents®.

4.3 The Round-Trip Engineering Method

As stated in Chapter 3, our CBCD method derives viewpoint-specific contracts from a (set of) negotiated contract(s) without making them explicit to the engineers. This decision has been made deliberately as engineers often are not aware of circular dependencies between viewpoints and have limited aids in estimating the impact of the assumed design parameters on their implementations. The RTE method complements the CBCD method to address these limitations, in particular for the control viewpoint. To do so, the RTE method replaces the generic export functionality of Figure 3.17 for the control viewpoint such that a control-specific view in Simulink® is created in which timing related information of the platform viewpoint (subsuming the hardware and software viewpoint) is represented using delay blocks.

To enable the RTE method, we rely on the "model transformations for, and in Simulink" technique introduced by Denil et al. in [DMV14]. The technique is further refined to enable parametrized rule-based model transformation. These rules act as a template of a model-to-model transformation, defining how the control view should be updated with timing related deployment information, and are created in the design environment of the control engineer (i.e., Simulink®). Figure 4.1 illustrates such a parametrized rule-based model transformation. Each rule consists of a LHS and a RHS. The LHS of the rule defines the precondition of the control view. As the RTE method complements the CBCD method, the precondition is an architectural control model consisting of model reference blocks (or subsystems) followed by some output (i.e., another Simulink® block). The RHS defines the post-condition of the model. As shown in Figure 4.1, it is defined

(a) LHS



(b) RHS

Figure 4.1: Parametrized model transformation rule

that the output of a model reference block is now followed by two delay blocks, representing the WCET (denoted by *C* as an abbreviation for *Computation*) and the scheduling time (denoted by the abbreviation *S*) of the deployed block. Note that both delays combine the WCRT of a component.

As illustrated by means of an FTG+PM in Figure 4.2, the RTE method transforms both control model and parametrized rule to a Himesis graph [Pro05] in which blocks are represented by nodes and connections by edges. These graph representations are used to create a set of model transformations for which the asterisks in Figure 4.1.(b) are replaced by timing related deployment information. Note that the rule creates a model transformation for each output signal of a control component. Next, the set of transformations is applied on the graph representation of the control model, resulting in a transformed Himesis graph that is then transformed back to a Simulink® model.

With respect to its integration in the CBCD method, the timing related deployment information may have its origin (i) in the negotiated contracts and/or (ii) in the results obtained from a schedulability analysis. In the former case, the RTE provides the control engineer with a top-level architecture in Simulink® in which the contract assumptions of the embedded domain (i.e., the software viewpoint) are explicitly modeled at a control engineer's level of abstraction. When behavior is added to the exported architecture, during the detailed design phase (Section 3.6.2), the control engineer is able to evaluate the control algorithm's behavior as if it were deployed on the hardware architecture. However, the explicitly modeled assumptions (i.e., the delay blocks

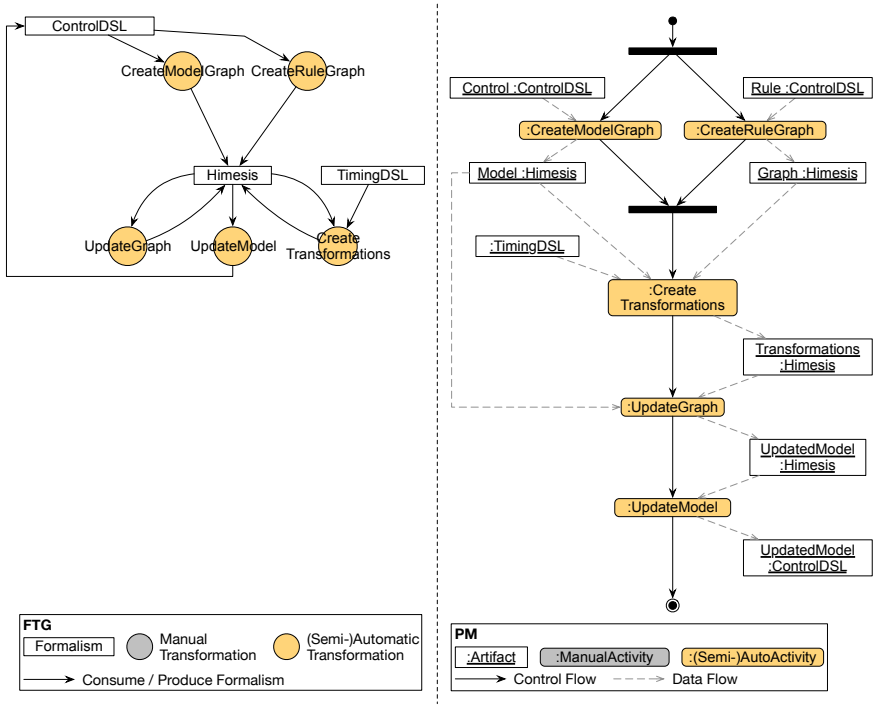


Figure 4.2: Overview of the Round-Trip Engineering (RTE) method

representing the WCRT) may be updated during the detailed design phase, which relates to the second origin of the timing related deployment information. Concurrently to the detailed control design, the embedded engineers will further detail the embedded architecture by defining a number of tasks, their priority, period, and deadline. Moreover, as behavior is added to the control algorithm, timing analysis may reveal a more precise WCET. Once this embedded related information is defined or obtained, a schedulability analysis is executed to evaluate whether the system is schedulable. The timing related results of this analysis are provided to the RTE method such that the initial delays in the control model can be updated.

While Figure 4.3 formalizes how the RTE method complements the CBCD method, Figure 4.4 illustrates how our RTE method extends the main framework of Figure 3.17. The export functionality is replaced by a RTE module for the control viewpoint. We rely on a python implementation of T-Core, introduced in Section 2.3.2, to enable the transformations to and from the Himesis graph representations. As such, our RTE method is implemented using the Python programming language. For interfacing with Simulink® we rely on its API. However, this results in considerable time-consuming transformations. Speed optimization can be achieved by accessing Simulink® diagrams using their xml representation (only applicable if the model is saved as an slx-file).

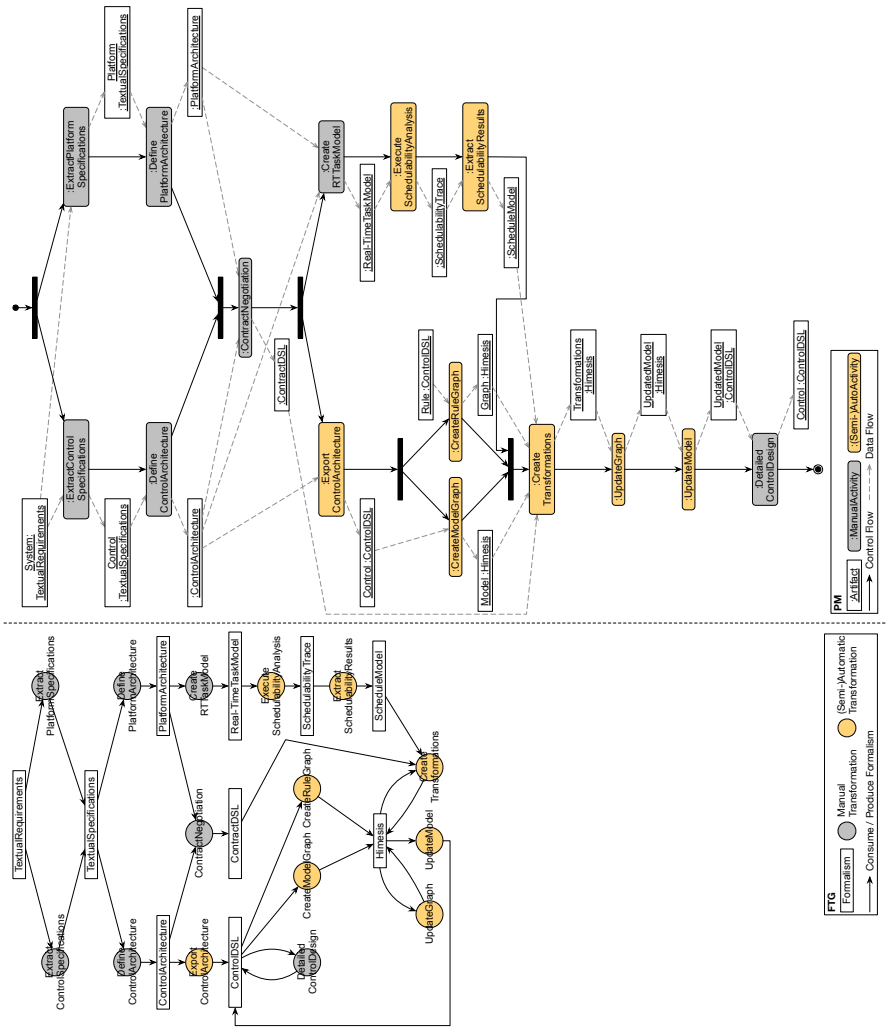


Figure 4.3: The RTE method complementing the CBCD method

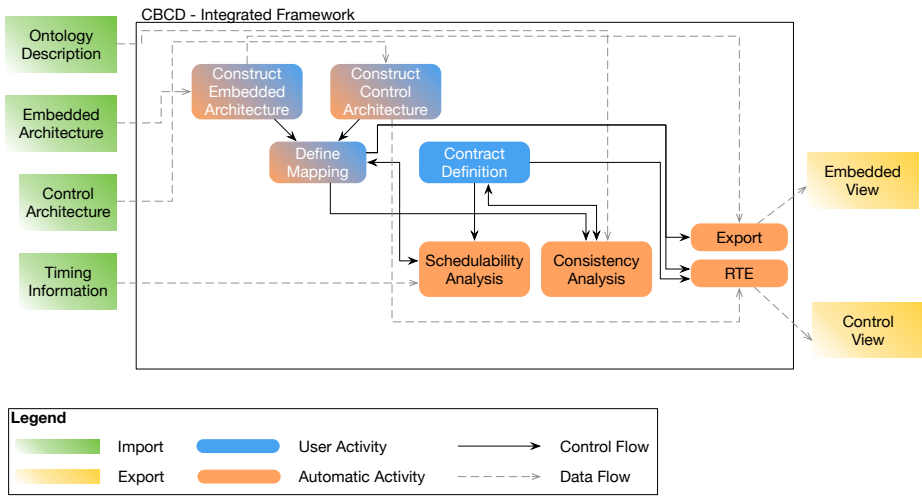


Figure 4.4: Architectural overview of the CBCD framework, including RTE support

4.4 The Round-Trip Engineering Method Applied on the Power Window Example

We demonstrate the applicability of our RTE method by means of the power window system (Section 1.5.1). We go through the relevant phases of a CBCD driven development process and discuss the evaluation of the control strategy after applying the RTE method.

Architectural Design and Contract Negotiation

Given the set of specifications, it is decided by the control engineer that the control algorithm consists of five (reusable) components: (i) an environment component emulating the actions of driver and passenger, (ii) a component to debounce the button signals, (iii) a control exclusion component prioritizing the driver signals, (iv) a main control component implementing the power window logic, and (v) a plant modeling the mechanical behavior of the system. Note that the debounce and main control component only model the operations for one set of buttons and window, respectively. As such, for each window a debounce and a main control component is modeled.

Control components (ii) through (iv) are deployed on a hardware architecture defined by the embedded engineer. It is decided to use two ECUs connected using a CAN bus: one low-cost *Low Performance Processor (LPP)* ECU operating at a low clock frequency and one *High Performance Processor (HPP)* ECU operating at a higher clock frequency.

Given these architectures, a mapping between the architectures is negotiated. In case of the power window system, it is decided that the *LPP* ECU holds both debounce components (one for each window) and the control exclusion component. The second *HPP* ECU holds both components implementing the power window logic. The negotiated design parameters are formalized as a set of mapping contracts shown in Figure 3.22 (Section 3.7).

Round-Trip Engineering

The architecture and the negotiated contracts serve as an input of the RTE module to create the control model skeleton in Simulink®, including the timing related deployment information using delay blocks. To this end, the parametrized rule-based model transformation of Figure 4.1 is defined in Simulink®. After initiating the RTE method, the set of model transformation is automatically created in which the parameters characterized by an asterisk in the RHS of the rule (Figure 4.1(b)) are replaced by WCET and schedulability design parameters defined in, or derived from, the contract. The result of automatically executing the set of transformations on the control exclusion component is shown in Figure 4.5. As its contract, shown in Figure 3.22, defines a WCET of 1 ms, the green delay block representing this design parameter is given a value of 1 with a sample time of 0.001 s. The WCRT equals the WCET implying that the control exclusion will be given the highest priority on the embedded system. As such, the scheduling time, represented by the orange delay block, is given a value of 0.

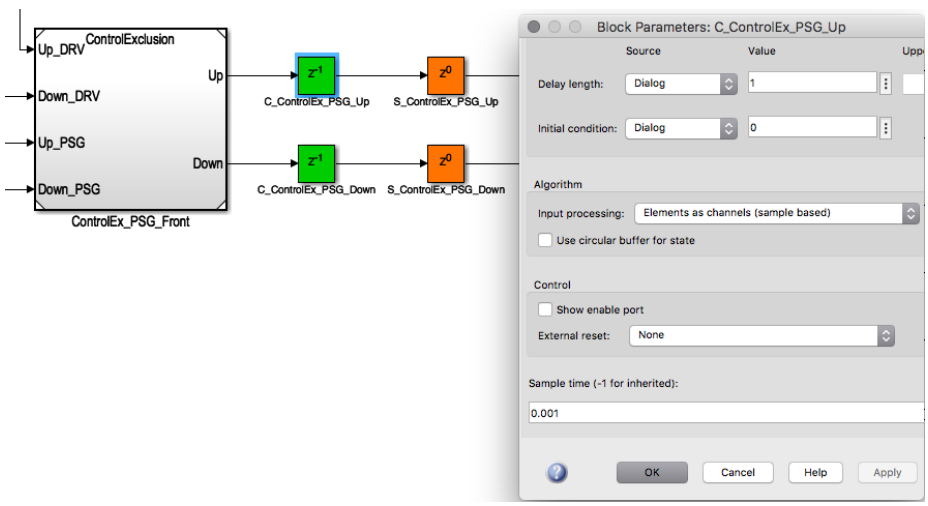


Figure 4.5: Result of a model transformation

Detailed Control Design

Once an architecture is defined and contracts are negotiated, the earlier defined control components (i) through (iv) are implemented by the control engineer, while (v) is modeled by a mechanical engineer. In the following, we briefly describe the control strategy for each of these components.

(i) The environment component emulating the actions of driver and passenger is modeled using Simulink® Signal Builder. For both driver and passenger a set of up and down signals are generated. At some points in time, a simultaneous action from driver and passenger is generated to test the control exclusion requirement.

(ii) Signal debouncing is modeled by the use of Stateflow®. The implementation of the debounce circuit is trivial: a signal must be in its new state for at least 30 ms before it is forwarded.

(iii) By using some basic logic gates, a control exclusion circuit is modeled so that driver priority is obtained when a driver and passenger operate the passenger window simultaneously.

(iv) Based on the work of Prabhu and Mosterman [PM04], which can also be found as a Simulink® tutorial, the behavior of the main power window control logic is modeled using a Stateflow® diagram.

(v) For the plant, the behavior of the motor and window mechanism is modeled using causal block diagrams. The properties of these physical components are explicitly modeled by using elementary control theory. Note that the external pinch force is determined using a sensor that measures the requested current of the motor. Therefore, a feed back loop from plant to control model is present.

Simulation Results

To illustrate the deployment effects on the behavior of the control algorithm, and as such the applicability of our RTE method, we compare the *realistic* simulation results with the ones of the *naive* simulation in which timing related information is not taken into account. The simulation results of this naive approach are shown as a solid blue curve in Figure 4.6 and, more detailed, in the upper part of Figure 4.7, in which the behavior of the passenger window is shown. At certain time stamps, a command from the driver and/or the passenger is issued. Within the scope of this discussion, we elaborate on three of them.

At time stamp 1 s the driver initiates an up-command, whereafter the window responds within 50 ms. During this movement, a force of 100 N is detected at time stamp 3.15 s. This results in a revert movement of the window 33 ms after pinch detection. The driver sends a down-command at time stamp 8 s for a time period longer than 500 ms, resulting in a downward movement of the window 52 ms after the command is issued. At time stamp 10 s both driver and passenger issue a

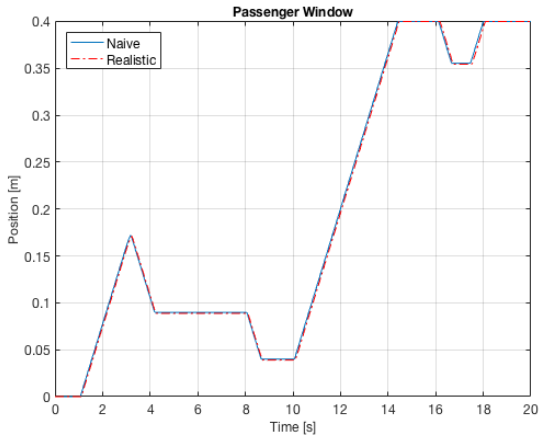


Figure 4.6: Simulation results for operating the passenger window

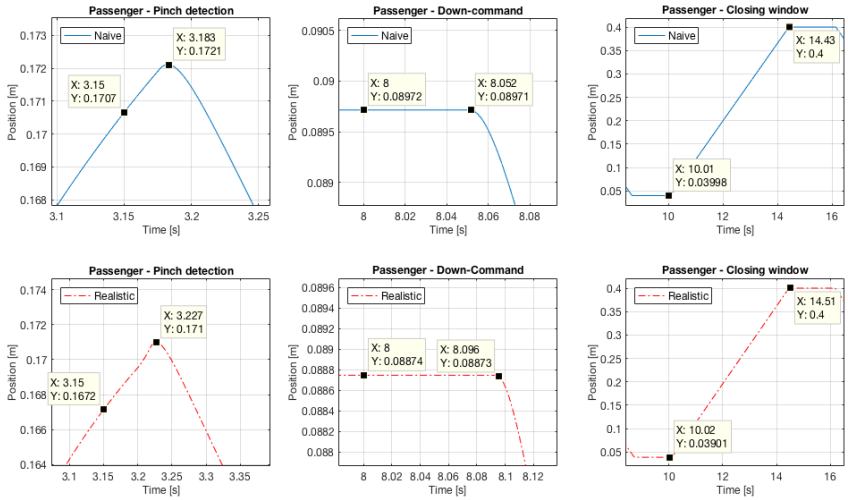


Figure 4.7: Detail of the simulation results—upper: naive simulation; lower: after in-place transformation

window command. However, their commands conflict with each other giving it priority to the driver who issued a short up command. This results in a completely closed window within a time period of 4.43 s.

When evaluating the simulation results of the control algorithm updated by our RTE method, shown as a dash-dotted red curve in Figures 4.6 and 4.7, we notice some remarkable differences compared to the naive simulation results (solid blue curve). For example, at time stamp 8 s a response time of 52 ms was derived for the naive simulation. Taken into account the negotiated timing related deployment information, we notice that the response time is increased to 96 ms, that is, almost doubled. Nevertheless, the requirements listed in Section 1.5.1 do not appear to be violated. However, certain requirements may be further refined to comply with (inter)national safety regulations to prevent safety-critical situations. For example, requirement *4* and *5* 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 2 mm before safety-critical situations occur.
2. Temporal dimension—given the spatial dimensions and the inertia of the system, safety can be guaranteed if the window lowers within 50 ms.

These requirements where validated at time stamp 3.15 s where a force of 100 N was detected. In the naive approach it only took 35 ms to reverse the movement of the window. When taking into account the delays negotiated in the contract, one can notice that the timespan between detection and action is increased to 77 ms. Due to this slower response time, the window closes for an additional 1.4 mm before an action takes place, violating both refinements of the safety requirement.

The simulation results show that our RTE method, using parametrized model transformations, does add essential information to the control model so that engineers are able to evaluate their design while taking into account the impact of the assumed design parameters. Note, however, that the negotiated design parameters may be overestimated. As guarantees may be refined (e.g., computation and scheduling time may be lowered), the behavior of the deployed system might be better than what is simulated during design time. The CBCD method allows these refinement operations during the concurrent design process, as shown in Figure 3.14, so that viewpoint-specific contracts can immediately be updated based on information from other viewpoints. By initiating the RTE method in the framework, refinements in one viewpoint can be pushed to the other viewpoints so that the views are consistent and up-to-date at all times.

4.5 The Round-Trip Engineering Method for Common Design Processes

Although the RTE method supports our CBCD method so that engineers are able to assess the impact of the assumed design parameters in their implementations, the RTE method can be made more generic so that it can be applied in any common design process for CPSs. In these more common design processes for CPSs, a precedence relation [PTQ+13] often exists between control and embedded design activities. This results in a late detection of conflicting views on the system under design, that in turn results in multiple iterations to deploy a single control model. The RTE method attempts to minimize these iterations by augmenting the control algorithm with timing related design parameters from the embedded domain.

We formalize the integration of the RTE method in these sequential design processes using the FTG+PM shown in Figure 4.8. We distinguish between three design phases: (i) Control Design, (ii) Deployment, and (iii) Round-Trip Engineering. Note that each of these phases respectively correspond to one column in both the FTG and PM side of Figure 4.8. As there is no notion of a contract that formalizes the design parameters and relates them to design components, we suggest the use of an Architecture Description Language (ADL) to store design information while executing each design phase and to maintain traceability. For this purpose, formalisms such as Modeling and Analysis of Real-Time and Embedded systems (MARTE) can be used which is a Unified Modeling Language (UML) profile to support the specification, design, and validation of real-time and embedded systems [FBSG07]. In what follows, we elaborate in more detail on the different stages of the RTE method applied in a common design process.

Control Design

Given a set of specifications and an architectural model, formally described and stored in an ADL, a control engineer creates an algorithm to control (part of) the system. A common approach to specify control logic is by using the causal block diagram formalism, supported by well established engineering tools such as Simulink®. Control engineers connect plant models to the control models to verify the behavior of the designed algorithms in the context of the system with respect to the specifications of the system. The created control models are prepared for deployment by the control engineers. This involves the discretization of a continuous-time model to a discrete-time model.

If the output of the control model still meets the predetermined specifications, the model is handed over to the software engineer. They further process the model such as the modularization of the control model with respect to the hardware configuration while maintaining traceability. To this end, the software engineer adds the different components to the component model of the ADL and models the interactions between the new components and the rest of the system.

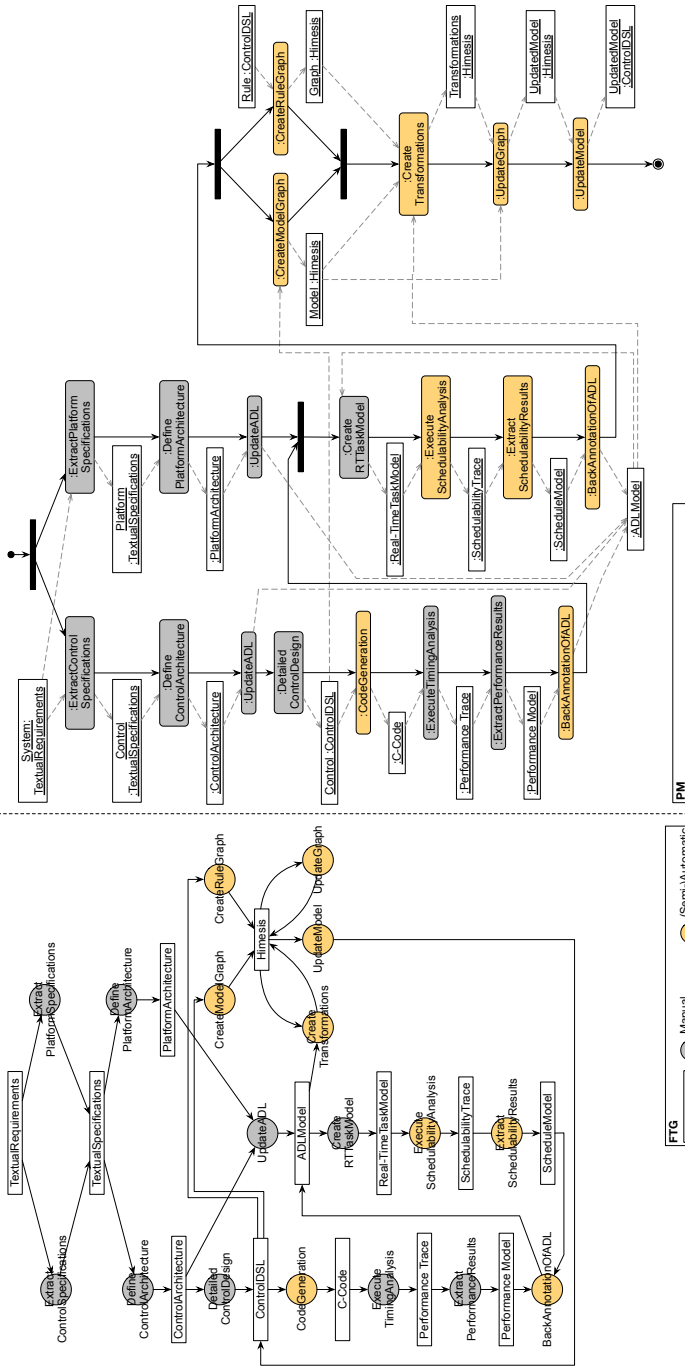


Figure 4.8: The Round Trip Engineering Method applied in a common design process

Traceability links link the ADL model to the behavioral models. From the different components, source code is automatically generated for deployment on the hardware architecture, that is, a set of ECUs connected using a communication medium such as CAN. Widely available tools such as Simulink® Coder™ are typically used for generating source code.

Embedded Design

The source code generated from each component, called software components, must be feasibly and optimally mapped onto an ECU or a set of networked ECUs. Each software component is allocated to an operating system task while task related parameters are set. Signals originating in the software components are packed into bus messages for communication between networked ECUs. To this end, parameters such as message priority are set.

To check whether a configuration is feasible and optimal, the embedded engineer starts by determining the performance of each software component by executing a timing analysis. For this purpose, two different methods exist: static and measurement-based method. The former method makes use of the generated code and a model of the target hardware to analyze the set of different possible control flow paths. The latter method is also known as profiling and executes the generated code on the target hardware or on a low-level simulation model to measure the execution time given a set of inputs. Both methods lead to the (statistical) determination of the WCET. A detailed overview of the tools and methods involved in obtaining the WCET can be found in [WMM+08].

To enable our RTE method, the results of the timing analysis are added to the ADL model, from which a Real-Time Task Model can be derived. This generated model contains software related information (e.g., WCET), information about the target hardware (e.g., number of ECUs), mapping information, and information related to the communication channel. The Real-Time Task Mode enables an embedded engineer to execute a schedulability analysis. Different techniques such as the one described by Tindell and Clark [TC94] or Palencia and Harbour [PG98] can be used. As a result, the schedulability analysis provides the embedded engineer with a trace containing the WCRT for each subsystem. Nowadays, several tools can be invoked to execute a schedulability analysis resulting in a trace containing the WCRT (e.g., MAST). The results of the deployment are fed back to the ADL model.

However, the choices made by the embedded engineer when deploying the system onto the hardware affect the performance of the designed control loop. For example, a signal can be delayed due to its transmission via a bus or a software component may encounter a longer execution time due to task related parameters. For this reason, the deployment process is typically an iterative process in which each iteration attempts to minimize the deployment effects on the behavior of the control algorithm while maintaining schedulability.

This iterative process is not taken into account in the FTG+PM shown in Figure 4.2, but is demonstrated in the work of Mustafiz et al. in [MDLV12].

Round-Trip Engineering

The results of the (iterative) deployment process are used to create new behavioral models by updating the Simulink® control model with extra delay blocks. Similar to the implementation in CBCD driven design processes, these blocks reflect the WCRT, combining the WCET and scheduling time, as a result of the schedulability analysis. It enables control engineers to evaluate the behavior of the deployed control algorithm at their level of abstraction, using methods and techniques they are familiar with. As the WCET depends on the chosen control strategy, the RTE method also enables control engineers to evaluate to what extent the control algorithm influences the deployed system. This might be helpful in case a schedulable implementation can only be achieved by lowering the computation time (i.e., the WCET).

4.6 Conclusion

When designing a system using a contract driven design process, it is often unclear for engineers how the contract assumptions influence the set of parameters to be guaranteed and, as such, the viewpoint-specific implementation. To assist engineers in assessing these dependencies, we proposed a Round-Trip Engineering method that allows control engineers to evaluate the behavior of their algorithms taking into account timing related deployment assumptions. To achieve this, a parametrized rule-based model transformation is defined, enabling a Py-T-Core based implementation to create a set of model transformations based on the negotiated design parameters. These model transformations enable the control engineer to create a (top-level) control architecture in a modeling environment such as Simulink® in which delay blocks are introduced representing negotiated timing related platform design parameters (e.g., WCRT).

Besides enabling control engineers to evaluate the virtually deployed control algorithm at their level of abstraction, the RTE method can also be used to support control and embedded engineers in the tradeoff analysis of design parameters in the contract negotiation process. In particular, it enables engineers to evaluate the control and deployment strategy. The result of the tradeoff analysis might, for example, indicate that the estimated complexity of the control algorithm is chosen too high for the amount of slack time available on the processor. This requires that a (partial) implementation already exists when negotiating the contract(s).

As negotiated parameters may be overestimated during contract negotiation, because of incomplete or incorrect design estimations, design parameters may be refined while (concurrently) detailing the design. The RTE method may support engineers in this refinement process by pushing changes

made in one viewpoint to the other (related) viewpoints so that behavioral simulations during design represent the integrated system at all times.

Although focusing on how the RTE method supports the CBCD method, we discussed a generic RTE approach so that the method can be applied in any design process. The generic approach consists of three design phases: (i) control design, (ii) deployment, and (iii) round-trip engineering.

To summarize this chapter, we return to the research question formulated in Section 1.3:

RQ4 *How should contracts syntactically and semantically be interpreted by different engineering domains?*

Using a Round-Trip Engineering method, the control domain, and its related viewpoint, is provided with a (top-level) architecture in which contract assumptions from the embedded domain (and its related viewpoints) are modeled at the most appropriate level of abstraction. While the RTE method currently focuses on supporting the control domain, the method can easily be extended to assist embedded engineers in estimating the impact of their design decisions on the behavior of the control algorithm at a higher level of abstraction. As such, we conclude that the RTE method enables engineers to assess the impact of assumed design parameters on their viewpoint-specific implementations, using methods and techniques they are familiar with.

CHAPTER 5

Design-Space Exploration Supporting Contract-Based Co-Design Driven Processes

Abstract. A designer often has to evaluate alternative designs during the development of a system. A multitude of Design-Space Exploration (DSE) techniques exist in the literature. Integration of these techniques into the modeling paradigm is needed when a model-driven engineering approach is used for designing systems. To a greater or lesser extent, the integration of those different design-space exploration techniques share characteristics with each other. Inspired by software design patterns, we introduce an initial pattern catalog to categorize the embedding of different design-space exploration techniques in a model-driven engineering context. We elaborate on their use by a literature survey, discuss the consequences of each pattern and illustrate their applicability on two distinct examples. Finally, we demonstrate how design-space exploration techniques can be integrated into a contract-based co-design driven (multi-viewpoint) design process.

5.1 Introduction

While designing a system, the need often arises to explore different design alternatives for a specific problem. Design-space exploration is a (semi-)automatic process where possible alternatives of a particular design problem are explored. The exploration is guided with imposed constraints and optimality criteria on the different candidate solutions. Design-space exploration techniques may also be used in the context of CBCD driven design processes to support engineers in their viewpoint-specific implementations. The degree to which DSE techniques may assist them highly depends on the maturity of the design team and/or the project. If this is considered low, DSE techniques can be used to (semi-)automatically define architectures for both the control and embedded domain. It goes without saying that a library of (validated) components should already exist to enable these explorations. On the contrary, if the maturity is considered low such that engineers have to define both contracts and architectures, DSE techniques may still be useful for exploring an optimal mapping of both architectures. In that respect, the software viewpoint considers subsystem and model reference blocks defined in the control architecture as software components. For the latter case, we demonstrate how DSE techniques can be integrated into the CBCD framework to support engineers in exploring an optimal mapping of control and embedded architectures, often referred to as a control-embedded allocation problem, for which the processor load and bus communication should be optimized.

In the literature a multitude of DSE techniques are available to explore a design space, for example evolutionary algorithms, constraint satisfaction and (Mixed Integer) Linear Programming. In our experience with embedding DSE in a model-driven engineering context and with a survey of the literature, we observed the use of different models, expressed using different formalisms, for both design, exploration, and the modeling of goal functions. Combining the different models, using transformations, with the multitude of techniques available for searching design spaces revealed similarities between the models and transformations of the different exploration techniques. To consolidate this knowledge, we organize these techniques into an *initial pattern catalog*, inspired by software design patterns. Each pattern is supported by a well-defined description, including its intended use. The goal of this effort is to create a more complete pattern catalog for model-driven engineering approaches for design-space exploration with the support of the community.

5.2 Related Work

The concept of patterns is widely used in Software Engineering. They provide generalized solutions to common software problems in the form of templates. The templates can be used by software developers to tackle the complexity in a larger software problem. One of the most highly cited contributions to pattern catalog in the field of software is the work of the "Gang of Four" [GHJV95], that presents various design patterns with respect to object-oriented program-

ming. Inspired by the Gang of Four, Amrani et al. [ADL+12] presents a model transformation intent catalog that identifies and describes the intents and properties that the cataloged transformations may or must possess. Their catalog can be used for several purposes such as requirements analysis for transformations, identification of transformation properties, and model transformation language design. Their presented catalog is a first attempt to introduce the concept of patterns in MDE.

A more in-depth literature study is integrated in Section 5.3 such that each pattern is illustrated by known uses. This motivates one to the application of the introduced patterns.

5.3 An Initial Pattern Catalog for Design-Space Exploration

By definition design patterns are used to formalize problems that recur repeatedly. They help a designer to evaluate alternatives for a given design problem in order to choose the most appropriate design. The usefulness of such patterns has already been proven in the Software Engineering domain where the "Gang of Four" [GHJV95] gave impetus to the creation of a widely accepted software design patterns catalog. The successful impact of its widespread use is undoubtedly the well defined structure of each pattern. More specifically, each pattern is typed by: (1) Pattern Name and Classification, (2) Intent, (3) Also Known as, (4) Motivation, (5) Applicability, (6) Structure, (7) Participants, (8) Collaborations, (9) Consequences, (10) Implementation, (11) Sample Code, (12) Known Uses, and (13) Related Patterns. Each of these sections is textually described and where necessary graphically supported using Class Diagrams, describing structure, and/or Activity Diagrams, describing the workflow of the pattern. At least one case study demonstrates how the patterns can be applied in practice.

In accordance to software design patterns, we define a pattern catalog specific to the design-space exploration domain for which the format of each proposed pattern is as follows:

— **Intent:** gives a short explanation of the intent of the pattern.

— **Structure:** describes the general structure of the pattern.

— **Consequences:** describes the tradeoffs in using the pattern.

— **Known Uses:** Lists the applications of the pattern in the literature. While this is not intended to be an exhaustive literature review of all the applications of the pattern, one can draw inspiration from these examples to apply the pattern.

— **Application:** gives a short description in which cases this pattern can be useful and how it can be implemented.

The ***Structure*** is graphically supported by the FTG+PM. The reason for using this supported formalism is threefold. First, it clearly represents the structure of the approach by connecting the different formalisms with transformations on the left-hand side of the FTG+PM. The FTG+PM also shows the workflow of combining the different models and transformations in a process on the right-hand side. Second, the FTG+PM can be used to (semi-)automatically execute the defined transformation chains (yellow colored). Manual operations are also possible that allow for experience based optimization and design (gray colored). Third, different patterns described in this formalisms are easily connected to each other. This enables the embedding of DSE within the MDE design of systems.

Executing design-space exploration in a model-driven engineering context can be abstracted in some steps:

1. A metamodel defines the structural constraints of a valid solution.
2. A DSE tool generates valid candidate solutions conforming to the metamodel. An initial model adds other structural constraints to the set of candidate solutions.
3. A transformation transforms the set of candidate solutions to an analysis formalism to check the feasibility of the solution with respect to a set of constraints.
4. If necessary, a second transformation generates a model in a performance formalism to check the optimality of the solution with respect to certain optimization goals.
5. Depending on the optimization technique, the process is iterated multiple times. Information from feasibility and performance models is used to guide the exploration.

Depending on the exploration technique, we classify different model-driven engineering approaches to solve this generic design-space exploration strategy.

5.3.1 Model Generation Pattern

Intent: This pattern transforms the metamodel of a problem space together with constraints to a constraint satisfaction problem. The exploration of the design consists of the generation of a set of models that satisfy the structural constraints imposed by the metamodel and the other constraints provided using a constraint formalism.

Structure: The pattern, shown in Figure 5.1, starts with a metamodel and some constraints. A transformation transforms these models into a constraint satisfaction problem. By invoking a solver, an exploration of the design space generates candidate solutions. Each candidate solution is transformed into an analysis representation. The analysis produces traces of each candidate solution. Based on the goal function model, the optimal trace is transformed to a solution model. This solution model can either be expressed in the exploration formalism, the original model formalism, or a specific solution formalism.

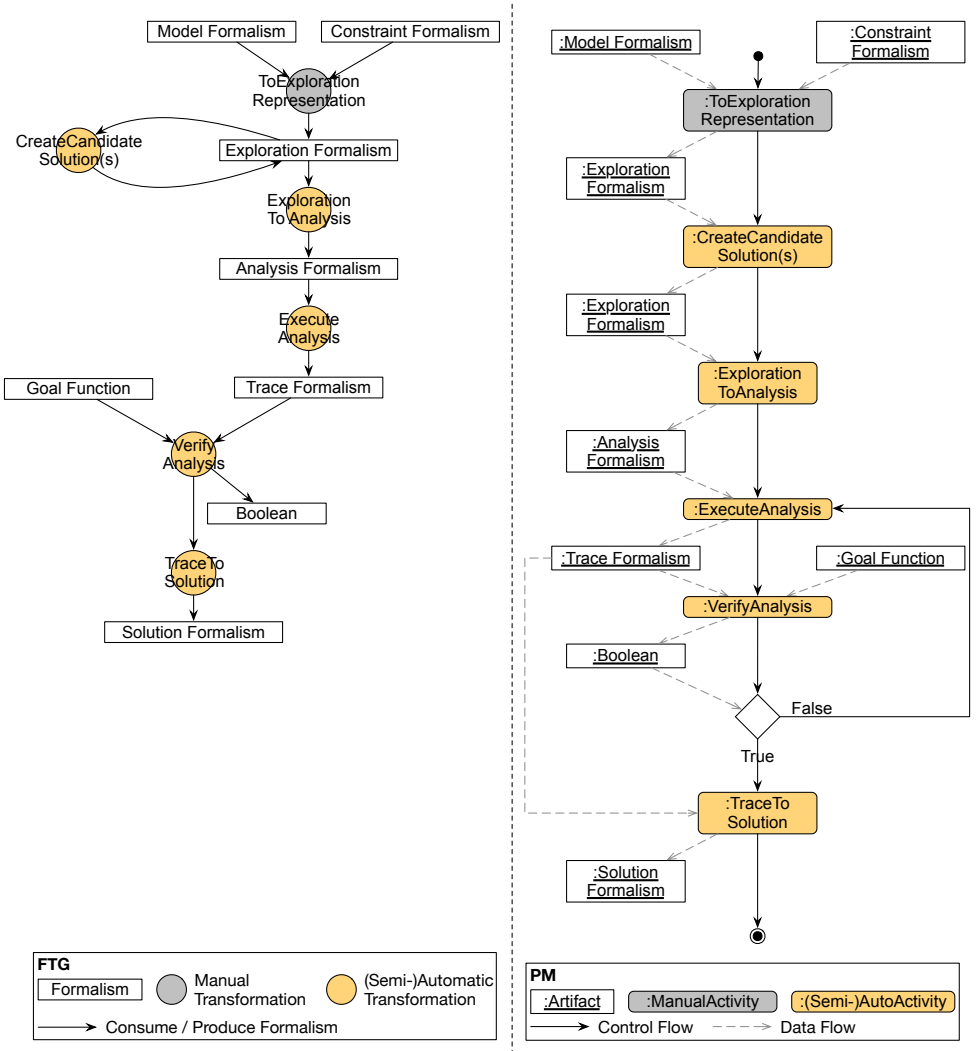


Figure 5.1: Model Generation Pattern

Consequences: Depending on the used solver, this method may be computationally and memory intensive because an exhaustive search of the design space is executed. A transformation is necessary to translate the metamodel with constraints to a model that is usable by the DSE tool. Domain knowledge can be introduced by adding constraints to the metamodel. Note that adding extra constraints helps the search for a solution. An initial model, where some choices are predetermined, adds extra constraints. A less generic alternative is to add the initial model when evaluating candidate solutions.

Known Uses: Neema et al. [NSK03] present the DESERT framework used for model-driven constraint-based DSE. It implements an automated tool that abstracts the Simulink® design space to generate candidate solutions. In [JKD+10] the FORMULA tool is presented, where candidate solutions are generated from a metamodel. A similar tool called Alloy is used by Sen et al. [SBM08] to automatically generate test models. Saxena and Karsai [SK10] present an MDE framework for generalized design-space exploration. A DSE problem is constructed of a generalized constraint metamodel combined with a domain specific metamodel.

Application: Tools implementing this pattern will solve a constraint satisfaction problem that results in an explosion of the design space. Therefore, this pattern is not recommended when one searches for an optimal solution out of a large search space without a lot of constraints. On the other hand, this pattern is very useful to rapidly obtain candidate solutions conforming to the metamodel.

5.3.2 Model Adaptation Pattern

Intent: This pattern transforms the model or a population of models to a generic search model used in (meta-)heuristic searches. Depending on the problem and search algorithm, different search representations can be used.

Structure: As depicted in Figure 5.2, a model or population of models expressed in a certain formalism is transformed to a specific exploration formalism. Based on the guidance of a goal function, an algorithm creates new candidate solutions. A (set of) candidate solution(s) is transformed to an analysis model in order to evaluate. Finally, the result is transformed to a solution model. This solution model can either be expressed in the exploration formalism, the original model formalism, or a specific solution formalism.

Consequences: A dedicated search representation must be created as well as manipulation functions to create alternative designs. This requires an adequate understanding of the problem and domain knowledge. A translation from the problem domain to the search representation and vice-versa is required. An initial model, as a constraint, can be added by fixing the generated solution or by rewriting the functions to create new solutions (cross-over, mutation, and so forth).

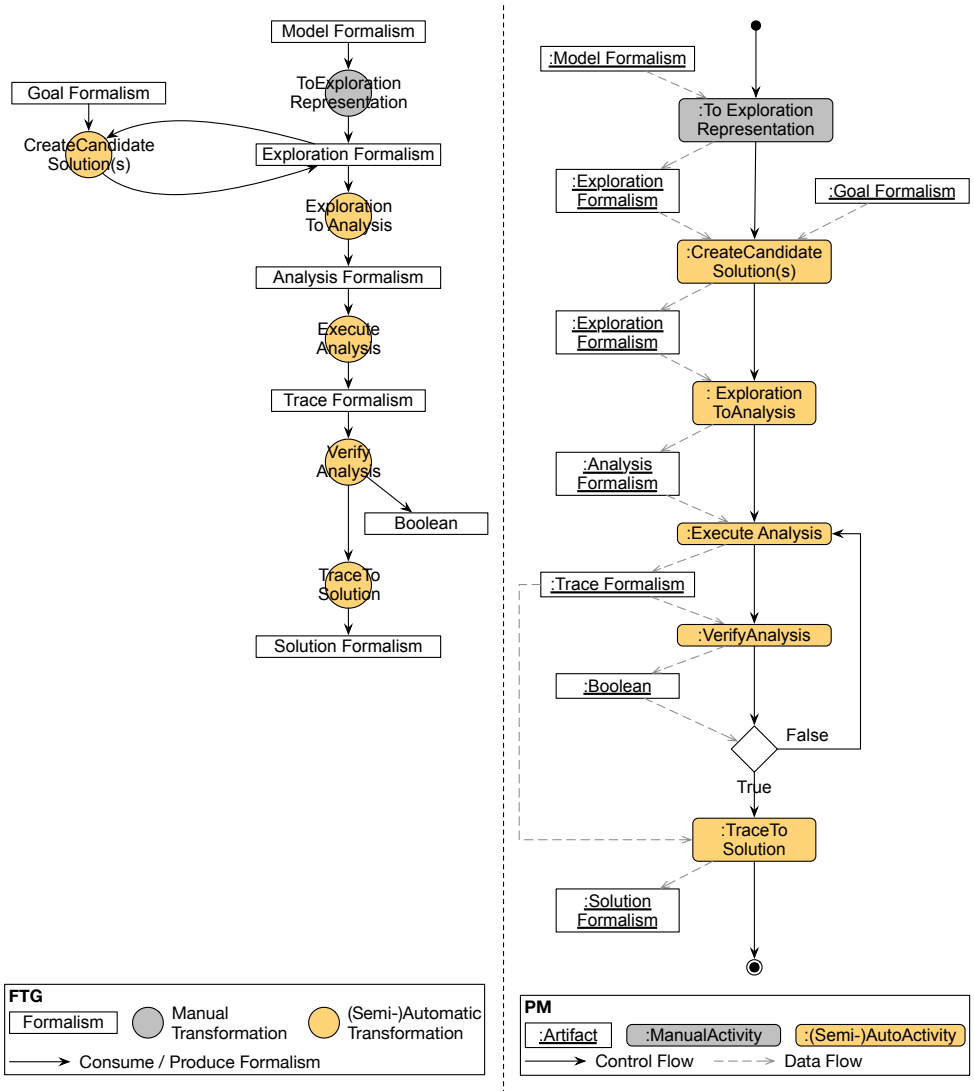


Figure 5.2: Model Adaptation Pattern

Known Uses: Williams et al. searched for game character behavior using a mapping to a genetic algorithm [WPR+11]. Burton et al. solve acquisition problems using MDE [BP13]. Genetic algorithms are used to create a Pareto front of solutions. A stochastic model transformation creates an initial population. In [KLW13] Kessentini and Wimmer propose a generic approach for searching models using Genetic Algorithms. The proposed method is very similar to the described pattern. It served as an inspiration for combining search-based optimization techniques with rule-based model transformations in [AVS+14, FTW15].

Application: This pattern is recommended when a design problem can easily be transformed to an optimal search representation, for example, a list or tree representation. Different operations on this new representation are implemented in the solution space (usually a generic programming language). Well-known algorithms, such as genetic algorithms and hill-climbing, implement the search.

5.3.3 Model Transformation Pattern

Intent: This pattern uses the original model to explore a design space. Model transformations encode the knowledge to create alternative models. Guidance to the search can be given by selecting the most appropriate next transformation or by adding (meta-)heuristics to the model transformation scheduling language.

Structure: Figure 5.3 outlines the structure of this pattern. A model combined with a goal function is used to create a set of candidate solutions that are expressed in the original model formalism. These are transformed to an analysis representation to gather some metrics that are expressed by a trace. Using (meta-)heuristics, a new set of candidate solutions can be generated according to a goal function. Finally, if required, the optimal solution or set of solutions can be transformed into a solution model.

Consequences: A high degree of domain knowledge about the problem is required to design the transformation rules. On the other hand, the rules encode domain knowledge to guide the exploration. Model-to-model or model-to-text transformations are required to evaluate a candidate solution. An initial model, as a constraint, can be added by adjusting the metamodel with variation tags. Similarly to the *Model Adaptation Pattern*, the initial conditions can also be implemented as fix operations using model transformations. Model transformations to create new candidate solutions are computationally expensive because of the subgraph isomorphism problem.

Known Uses: In [HHRV11] a model-driven framework is presented for guided design space exploration using graph transformations. The exploration is characterized by a so called exploration strategy that uses hints to identify dead-end states and to order exploration rules. As such, the number of invalid alternatives is reduced. Denil et al. [DMV14] demonstrates how Search-Based Optimization (SBO) techniques can be included in rule-based model transformations.

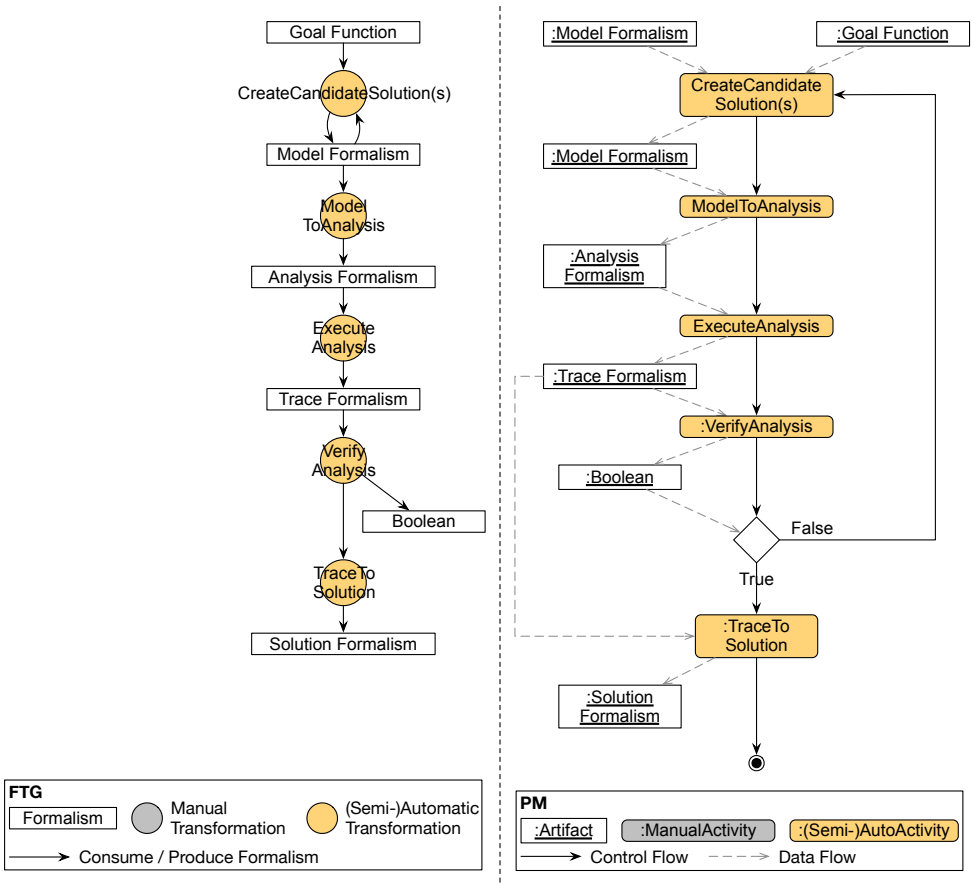


Figure 5.3: Model Transformation Pattern

Application: The pattern is used when it is challenging to obtain a generic search representation. Model transformation rules, expressed in the natural language of the engineer, are implemented using current model transformation tools. Guidance is implemented through the scheduling of the model transformation rules.

5.3.4 Exploration Chaining Pattern

In order to prune the design space more efficiently, multiple of the proposed patterns can be chained. This technique is called "Divide and Conquer" and may as well be described by a pattern. To represent the chaining of multiple FTG+PMs, this pattern is graphically supported by means of a principle representation.

Intent: This pattern adds multiple abstraction layers in the exploration problem where candidate solutions can be pruned. High-level estimators are used to evaluate the candidate solutions and prune out non-feasible solutions and solutions that can never become optimal with respect to the evaluated properties. Figure 5.4 shows the overall approach of this pattern.

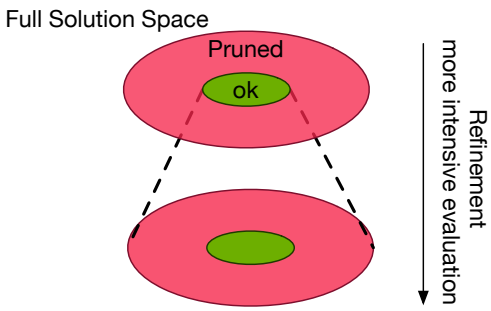


Figure 5.4: Exploration Chaining Pattern

Structure: At each of the abstraction layers an exploration pattern is used to create and evaluate candidate solutions. Non-pruned solutions are explored further in the next exploration step.

Consequences: Domain knowledge about the problem is required to add levels of abstraction. High-level estimators are needed at each of the abstraction layers to evaluate a candidate solution. Because more information is introduced at each of the abstraction layers, the evaluation of a single candidate solution becomes more complex and usually more computationally intensive. Finally, a pruning strategy is required to decide what solutions must be pruned at each of the abstraction layers.

Known Uses: Sen and Vangheluwe add different levels of abstraction in the design of a multi-domain physics model [SV06]. This numerically constraints the modeler to create only

valid models. Kerzhener and Paredis introduce multiple levels of fidelity in [KP10]. Finally, multiple levels of abstractions for an automotive allocation and scheduling problem are introduced in [DCB+11].

Application: This pattern provides a solution when memory and time complexity are an issue during the exploration of the design space. It tackles the complexity by its layered pruning approach. Therefore, this pattern is preferred when searching for an (set of) optimal solution(s) in a large search space. Different exploration patterns are chained to create solutions while the required domain knowledge is often added by the designer. Alternatively, when moving between abstraction levels (i.e., exploration patterns) the domain expert may guide the search by selecting an (set of) optimal solution(s).

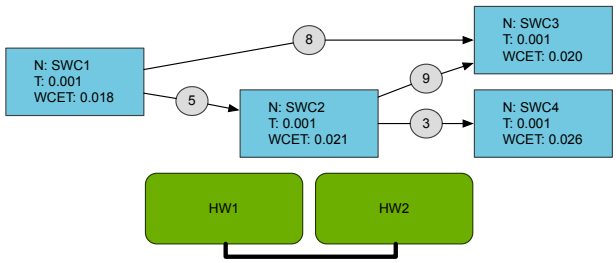
5.4 The Pattern Catalog Applied in Contract-Based Co-Design Driven Design Processes

The aforementioned DSE patterns and their corresponding techniques may also be used in the context of CBCD driven design processes. In particular, they can assist both the control and embedded domain in detailing their design. Within the scope of this dissertation we implemented a DSE technique in the integrated CBCD framework that supports the embedded domain in defining a schedulable system. Nevertheless, this section will also detail how the control domain may be supported in detailing the control architecture.

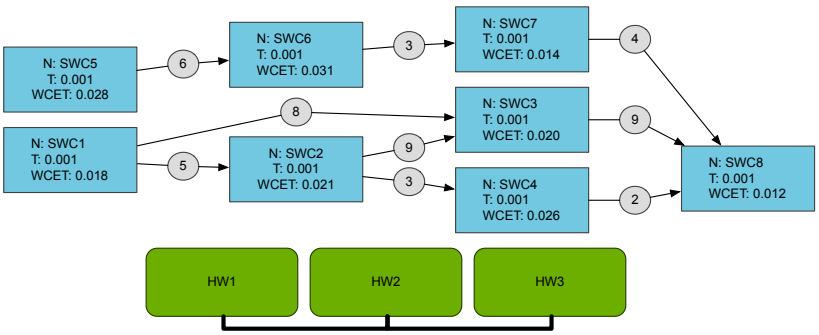
5.4.1 Design-Space Exploration Supporting the Embedded Domain

As already mentioned in Section 3.7, the integrated framework allows embedded engineers to manually allocate components of the control architecture to the runnables defined in the embedded architecture, consisting of multiple ECUs connected using a communication bus. In that respect, the software viewpoint considers subsystem and model reference blocks defined in the control architecture as Software Components (SWCs). This allocation process can be automated by using an appropriate DSE pattern and corresponding technique.

Let us consider two examples of an allocation problem, shown in Figure 5.5, to examine how DSE may assist engineers in defining and optimal mapping. The rectangles in Figure 5.5 denote the SWCs for which a name (N), period (T), and WCET is defined. Arrows between two SWCs indicate that there exists some communication between them for which the number in the circle denotes the communication size (e.g., in bits). The rectangles with rounded corners in the lower parts of both examples represent the ECUs. The line between them represents the communication bus.



(a) Basic Allocation Problem



(b) Extended Allocation Problem

Figure 5.5: Examples of an allocation problem

The corresponding metamodel for the *Allocation* problem is shown in Figure 5.6. It contains multiple *SWC* typed by a name, a period and a wcet attribute. Two *SWC* can communicate to each other using *Message*s. The size attribute indicates the bitsize of a message. Each *SWC* can be mapped onto a single *ECU*. When a *SWC* is mapped onto an *ECU* its load attribute is increased by the equation $wcet/period$, in which the values are substituted by the ones defined for the *SWC*. Given the size attribute of the communication *Message* and the period of the sending software function, a similar calculation is used for determining the load attribute of the *Bus*. The load on the *Bus* only increases when the sending and receiving *SWC* are mapped onto a different *ECU*.

Zheng et al. [ZZDNSV07] approach the problem by searching for a mapping where the load of the different ECUs is below a threshold of 69 % (the schedulability test for rate-monotonic systems [LL73]). As communication on the bus introduces delays that impact the timing behavior of the final solution, the goal is to find an optimal mapping for which the communication between the different ECUs is minimized. In that respect, Equation 5.1 defines the goal-function for the allocation example. If the threshold of 69 % is not exceeded for an ECU than the sum of the communication cost between software components mapped onto different ECUs will determine the score. In the other case, a penalty for infeasibility is added to the score. This ensures that the optimal solution of our design problem is the one where no ECU exceeds the threshold of 69 %

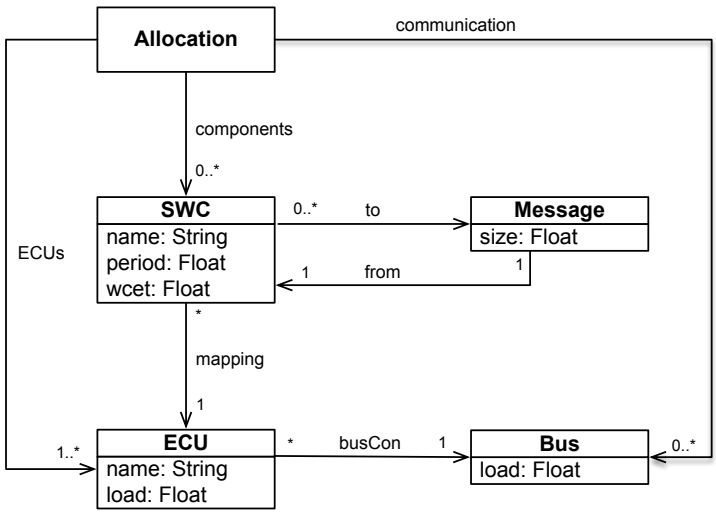


Figure 5.6: Metamodel used for allocating software components on a hardware architecture

and in which there is as little as possible communication between ECUs, resulting in the lowest possible score.

$$\text{Score} = \text{Effective Communication Cost} + \text{Penalty} \tag{5.1}$$

$$\text{Penalty} = (\text{Total Communication Cost} * 3) \quad \text{if threshold } > 69,3\,\% \tag{5.2}$$

We applied each pattern of the catalog introduced in Section 5.3 using different available tools: Alloy [Jac06] for the *Model Generation Pattern*, DEAP [FDG+12] for the *Model Adaptation Pattern* and Rule-Based [SVL15] (with a Hill Climbing meta-heuristic) for the *Model Transformation Pattern*. Each experiment is executed three times on a laptop with Intel Core i7 @ 2.80 GHz processor and 6 GB memory, running a 64-bit Windows 7 Enterprise operating system. The experiment results, for both the basic (Figure 5.5.(a)) and extended (Figure 5.5.(b)) allocation problem, can be found in Table 5.1. We recorded the average (Avg.) exploration time and score with their corresponding standard deviation (Std. Dev.) and the total number of generated candidate solutions (# Candidates).

As can be seen from the simulation results, this particular mapping problem does not lend itself for using the *Model Generation Pattern*. Although Alloy is able to find a solution in both experiments, time and memory complexity are increasing when adding more variables, resulting in a uncompleted exploration of the design space and thus a non-optimal solution. While the results obtained with the *Model Transformation Pattern* and the *Model Adaptation Pattern* are

Tool	Exp.	Time (s) Avg.	Time (s) Std. Dev.	Score Avg.	Score Std. Dev	# Candidates
Alloy	Basic	2.21	0.24	3	0	30
	Extended	986.54	6.79	15	0	27228
DEAP	Basic	0.16	0.02	3	0	1780 (avg)
	Extended	1.48	0.23	16.66	2.88	23055 (avg)
Rule-Based	Basic	0.33	0.04	3.0	0	23 (avg)
	Extended	4.31	0.748	16.66	2.88	53 (avg)

Table 5.1: Results *Allocation Problem*

comparable, the *Model Adaptation Pattern* outperforms the other patterns, and corresponding techniques, in optimization time. Although, it does not necessarily find the optimal solution. This is because the large influence of the different parameters such as population size, cross-over, selection, and mutation rate.

As such, we might conclude that this design problem lends itself for using the *Exploration Chaining Pattern*, in which the *Model Generation Pattern* is followed by the *Model Adaptation Pattern*. This conclusion is well-founded when exploring an optimal mapping in more common design processes. However, CBCD driven design processes impose a negotiation phase in which contracts are defined. As they specify (ranges of) values for different design parameters, contracts limit the design phase and, as such, provide an initial population for the *Model Adaptation Pattern*, thereby eliminating the need for prior use of the *Model Generation Pattern*.

Our CBCD framework currently integrates the *Model Adaptation Pattern* for exploring an optimal mapping of the control on the embedded architecture. To this end, the CBCD framework integrates the Multi-Objective Evolutionary Algorithm (MOEA) framework [Had], illustrated as a DSE module in Figure 5.7. A stated in [Had], the MOEA framework supports genetic algorithms, differential evolution, particle swarm optimization, genetic programming, grammatical evolution, and more. Although the MOEA framework includes some predefined algorithms, users can easily implement their own algorithms. Within the scope of this dissertation, we opted to use the predefined Non-dominated Sorting Genetic Algorithm (NSGA)-II algorithm [DPAM02] enabling one to search for an allocation in which processor load allocation is optimized. As the NSGA-II algorithm supports multi-objective optimization, the CBCD framework also provides support for optimizing the processor cost.

To enable the exploration, both control and embedded architecture are transformed to a list representation. Both lists are traversed so that additional component information can be extracted; that is, the period for each SWC and the WCET, current load, and (optional) cost for each processor. Depending on where in the design process the optimization is initiated, this information

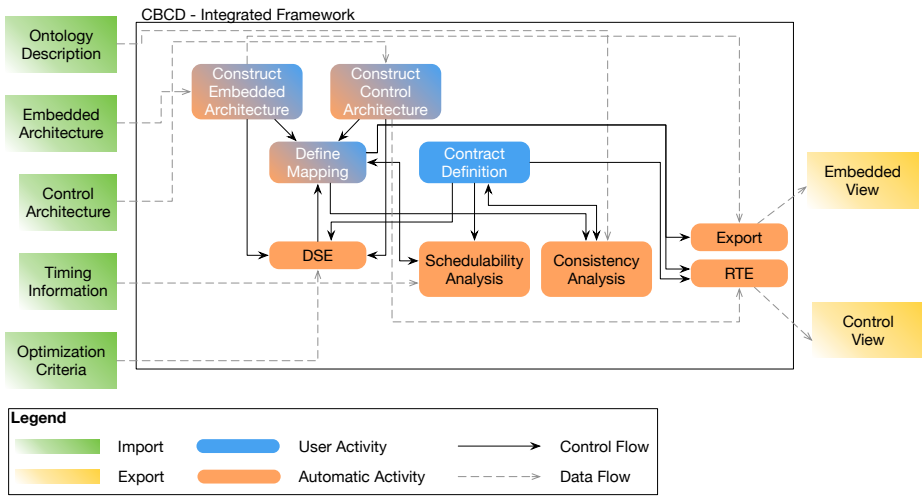


Figure 5.7: Architectural overview of the CBCD framework, including DSE support

can be obtained from the defined contracts or from data obtained throughout the detailed design phase. In the latter case, engineers refine the negotiated design parameters by detailing the architectures. For example, by executing a timing analysis the exact WCET can be determined. The extracted component information is again structured as a list. Combined with the *architecture* lists, the genetic algorithm uses an evolutionary strategy to determine a Pareto-optimal front for the allocation problem. One of these solutions is reflected to the engineers by connecting allocated components using a *mappedOnto* relation in the viewpoint of the embedded engineer.

5.4.2 Design-Space Exploration Supporting the Control Domain

Design-space exploration can also support the control domain to search for the optimal implementation of a particular design problem. We demonstrate this by the design of an electronic filtering circuit, conceptually shown in Figure 5.8. The surrounding rectangle with rounded corners represents the filter, and has three nodes located to the left (the input node), to the right (the output node), and at the bottom (the ground node). The filter connects the input node to the output node and ground node through a topology of resistors, capacitors, and inductors. The *minus* and *plus* signs on the connections denote the direction of the electric flow. It is assumed that the input connects an electric source to the system, but it may as well be an output signal of a preceding electric or electronic circuit. The ground port leads directly to a ground. The filter's behavior is measured by inspecting the potential difference at the output port and ground port.

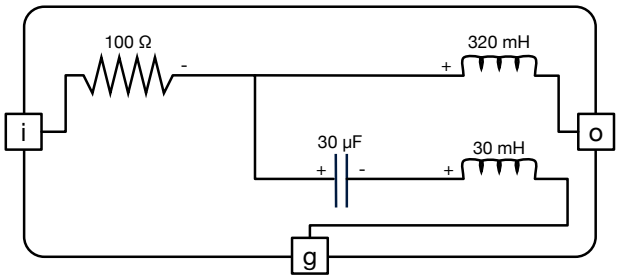


Figure 5.8: Example of an electronic filter

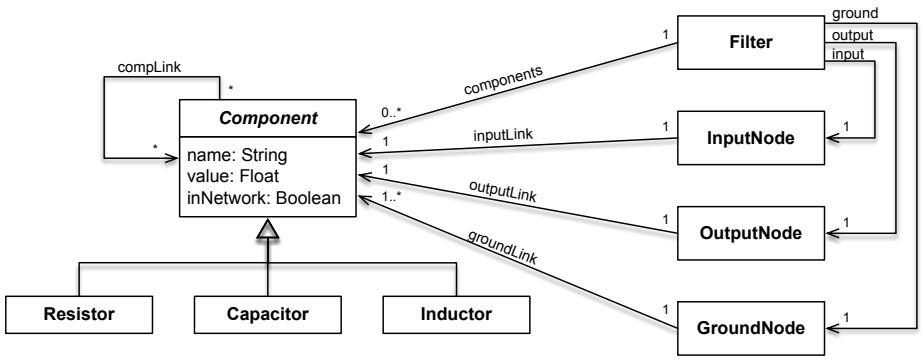


Figure 5.9: Metamodel used for exploring a passive electronic filter

We abstract from the design problem by regarding the filter design as a black box with an input, output, and ground node with some passive electrical components in between that are connected to each other. The corresponding metamodel is shown in Figure 5.9. A *Filter* may consist of multiple components that can be a *Resistor*, *Capacitor*, or *Inductor*. Each *Component* has a name, a value (resistance in Ohm, capacitance in Farad, and inductance in Henry) and a boolean attribute to indicate whether the component is part of the network or whether it can be used in the search problem. Note that the *Component* class cannot be instantiated as it is an abstract class, denoted by the italic font. A filter has exactly one *InputNode*, one *OutputNode* and one *GroundNode*. Only one *Component* can be connected to an *InputPort* and an *OutputPort*, but more than one *Component* can be connected to the ground.

Various configurations of those components may lead to the construction of different types of filters, for example, a Low Pass Filter (LPF) that passes low-frequency signals and (ideally) attenuates signals with frequencies higher than the cut-off frequency (ω_c). On the contrary, a High Pass Filter (HPF) (ideally) attenuates signals with a frequency lower than the cut-off frequency and passes high-frequency signals. Other types of passive filters include Band Pass Filter, Band Stop or Notch Filter, and All Pass Filter. A filter's expected (first order) behavior is specified

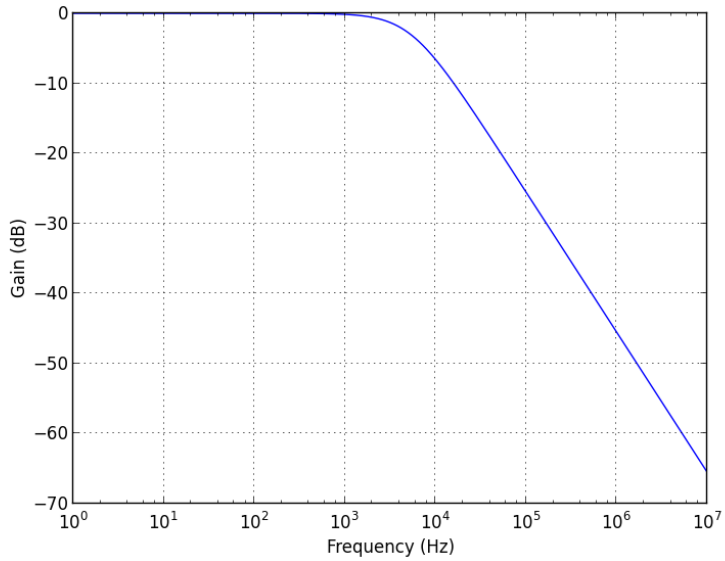


Figure 5.10: LPF Bode plot

using a gain-magnitude frequency response, also called Bode plot. The Bode plot of a LPF with a cut-off frequency of 5 kHz is shown in Figure 5.10. At this cut-off frequency, the filter will attenuate the signal by half of its original power. This corresponds to an attenuation of 3 dB, that increases by 20 dB each decade. In a CBCD driven design process, these specifications would be part of the guarantees of a contract. Note, however, that the current implementation does not allow to specify these more complex (behavioral) constructs.

The goal of the exploration is to find a filter where the Bode plot has a minimal deviation compared to the expected one shown in Figure 5.10. Therefore, we see the deviation as a difference value between the expected and measured gain for each frequency point. In that respect, the goal-function or fitness-function can be formulated by Equation 5.3. A larger deviation will result in a higher score, while a solution containing fewer components will result in a lower score. This latter constraint may be needed to reduce the complexity of the implementation and, as such, its WCET specified in the negotiated contract.

$$\text{Score} = (\text{Deviation} * 20) + \text{Number of components} \tag{5.3}$$

As with the allocation problem, the design space is constrained by the negotiated contract in which the possible values of each component are specified. Table 5.2 lists the constraints we have imposed while evaluating each pattern of the introduced catalog of Section 5.3. Again, each

	Resistor (R)	Capacitor (C)	Inductor (L)
Value 1	10 Ω	3 μF	30 mH
Value 2	100 Ω	33 μF	320 mH
Value 3	1000 Ω	330 μF	3500 mH

Table 5.2: Predefined component values when exploring using the *Model Generation Pattern*

			Time (s)		Score		
Tool	Exp.	# Comp.	Avg.	Std. Dev.	Avg.	Std. Dev	# Candidates
Alloy	1	2	168.27	8.55	-36.99	0	402
	2	3	3344.91	157.86	-36.99	0	5352
DEAP	1	18	26.43	0.62	-67.82	53.42	400 (avg)
	2	36	27.15	0.78	-174.28	116.8	400 (avg)
Rule-Based	1	18	66.7	52.13	-36.99	0	430 (avg)
	2	36	105.89	70.6	-41.72	4.11	675 (avg)

Table 5.3: Results *Electronic Filter*

experiment is executed three times on the same laptop used to explore the design space for the allocation problem. Table 5.3 show the results for exploring a LPF satisfying the Bode plot of Figure 5.10. We recorded the maximum number of predefined components present in the network (# Comp.), the average (Avg.) exploration time and score with their corresponding standard deviation (Std. Dev.) and the total number of generated candidate solutions (# Candidates).

Again, the results clearly demonstrates that the *Model Adaptation Pattern* finds an optimal solution in a considerable shorter time span. However, when choosing this pattern one will notice we are dealing with a design problem that is difficult to transform to a generic search model such as a list or tree. The most appropriate pattern to implement these types of design problems is by using the *Model Transformation Pattern*. It provides average simulations results for exploring the design space while designers are able to encode their domain knowledge (to guide the search) by creating a set of model transformation rules at their level of abstraction. These model transformations rules apply mutation operations on the initial model. Example of such search rules are: adding or removing serial or parallel connections, adding or removing components, and so forth.

5.5 Conclusion

Resulting from our own experiences with DSE and a literature survey, we presented an initial pattern catalog that categorizes different approaches of Model-Driven Design-Space Exploration. Inspired by software design patterns, we described each pattern using a well-defined structure in which its intent, structure, consequences, known uses, and application are described. With respect to the structure, we made use of an FTG+PM to graphically support the involved formalisms and their relations using model transformations.

The introduced pattern catalog should not be considered complete. With the support of the community, it is our ambition to extend this towards a more complete pattern catalog, similar to the widely available software design patterns used in software engineering. Techniques that potentially could become a pattern in a new version of the catalog are:

Multiple Objective Multi-objective optimization deals with the decision making process in the presence of tradeoffs between multiple goal functions. Certain DSE and search algorithms can deal with multi-objective functions by construction. However, some techniques do not have this features. Here we give two approaches of dealing with the problem.

- *Scalarize the Objective-Function:* When scalarizing a multi-objective optimization problem, the problem is reformulated as a single-objective function. The goal function model becomes a combination of individual objective functions. A model defines how the different individual goal function models are combined, for example, in a linear fashion or other more complex functions.
- *Create Variants:* In certain cases the designer would like to compare the different tradeoffs using a Pareto curve. We use the scalarizing pattern to create multiple variants of the combined objective function. Intermediate results of the exploration are used to select an appropriate recombination that could potentially add a new Pareto solution.

Metamodel reduction By using sensitivity analysis of the involved modeling elements and parameters, the metamodel can be reduced with the elements and parameters that have a small influence on the result of the goal function. An example of this technique can be found in [EON+12].

We elaborated on how the pattern catalog can support engineers while defining their detailed designs in CBCD driven design processes. Embedded engineers are often charged with a bin packing design problem, that is, they need to find an optimal mapping between the control architecture and the embedded architecture for which they should optimize processor load and cost. In common design processes, one would chain different exploration techniques to cover the design space. As the CBCD driven design processes impose the negotiation of a (set of) contract(s), the design space is limited so that exploration techniques belonging to the *Model*

Adaptation Pattern are highly suitable to find an optimal solution in a reasonable amount of time. Control engineers search for the optimal implementation of a component with respect to the expected behavior of a control component and its complexity (expressed in terms of WCET) defined in the negotiated (set of) contract(s). We demonstrated how contracts may constrain the design space for these design problems by means of an electronic filtering circuit. As encoding domain knowledge is of great importance in exploring different control strategies, we concluded that exploration techniques belonging to the *Model Transformation Pattern* are highly suitable to support control engineers in their design decisions.

Finally, we return to the research question formulated in Section 1.3:

RQ5 *Can contracts be used as an input for design and/or deployment optimization methods?*

A CBCD driven design process imposes a negotiation phase in which contracts are defined. As they specify (ranges of) values for different design parameters, contracts limit the design phase and, as such the exploration space. As contracts are associated with architectural components, genetic algorithms can be used to explore an optimal mapping of the control on the embedded architecture. In that respect, contracts provide an initial population for the *Model Adaptation Pattern*, thereby eliminating the need for chaining the *Model Generation Pattern*. When exploring implementation of a control algorithm exploration techniques belonging to the *Model Transformation Pattern* are more suitable. They allow the control engineers (and their respective viewpoints) to encode their domain knowledge and guide the search so that feasible solutions are explored.

RQ6 *How can we structurally organize the plethora of DSE methods to assist engineers in the evaluation of the most appropriate method for a given optimization problem?*

We have introduced an initial pattern catalog that categorizes different DSE techniques used in a MDE context. Inspired by software design patterns, we described each pattern using a well-defined structure in which its intent, structure, consequences, known uses, and application are described.

CHAPTER 6

The Integrated Framework Applied in a Contract-Based Co-Design Driven Development Process

Abstract. Contract-based co-design as a method enables engineers to consistently design a cyber-physical system in a concurrent setting. To that end, an upper ontology models the implicit engineering knowledge of each domain so that syntactically and semantically different design parameters can be related to each other. Round-trip engineering methods and design-space exploration techniques can assist engineers while enacting these contract-based co-design driven design processes. The round-trip engineering method annotates domain-specific models with design parameters of other domains that may be of influence. Given the formulated constraints of the design space in a contract, design-space exploration techniques can be efficiently employed to semi-automatically search for an optimal design choice. These novel methods and techniques can only be successfully applied by an engineering team if they are supported by an integrated framework that incorporates these methods. The integrated contract-based co-design framework is proposed to fulfill this requirement. Using the development of a hybrid hydraulic vehicle we demonstrate how the integrated contract-based co-design framework supports engineers throughout their contract-based co-design driven design processes.

6.1 Introduction

In the previous chapters we discussed a set of methods and techniques to enable consistent concurrent design of CPSs and demonstrated their applicability using an academic case study. Throughout the discussion, we also presented the integrated CBCD framework that incorporates both the CBCD and RTE method, and a DSE technique for optimal deployment given a set of negotiated contracts.

This chapter intents to detail a holistic view on the concurrent design of CPSs using the overall integrated CBCD framework, shown in Figure 5.7. In particular, we elaborate on the design process of the HHV, introduced in Section 1.5.2, going from architectural design down to detailed design, and discuss where in this process the integrated CBCD framework is able to support engineers in their decisions. For this, a team covering four disciplines in the design of a CPS is considered: systems engineering, mechanical engineering, control engineering, and embedded engineering. The latter discipline covers both the software and hardware design of the system. Although the mechanical engineer is necessary to model the physical system, we continue to focus on the interactions between control and embedded engineering. The team is responsible for the concurrent design of the HHV following industry proven design guidelines and using industry accepted tools. In this respect, the control engineer uses a MDE approach to design the control algorithm, while the embedded engineer relies on schedulability analysis techniques to configure a RTOS so that real-time behavior can be guaranteed.

6.2 Related Work

In the current literature, some frameworks have already been proposed that enable the concurrent design of CPSs. Being a general-purpose modeling language based on *UML* [RJB04], *Systems Modeling Language (SysML)* [FMS11] provides such a framework by enabling engineers to explicitly model and relate the structure, composition, interfacing, and behavior of a system. As such, a common model can be designed, acting as a single point of truth, to ensure consistency among viewpoints. With respect to real-time embedded systems, more specific UML profiles such as MARTE [Obj11] and Papyrus [Thec] exist to support engineers in designing complex systems.

In [MZL+13], a more holistic framework is proposed for the concurrent design of Heating, Ventilation and Air Conditioning (HVAC) systems. In particular, Maasoumy et al. propose a framework for the concurrent design of the control algorithm with that part of the hardware platform that highly affects the sensing accuracy. Therefore, a set of interface variables are determined and is used by a DSE algorithm to define the optimal design for both the control algorithm and hardware platform. Based on the FOCUS theory [BS01], Hölzl et al. [HST10,

HF11] and Aravantinos et al. [AVT+15] introduce the *AutoFOCUS 3* framework that aims to support engineers in the complete design process from requirements to code generation using a MDE approach. Using traces between the models of the various viewpoints, different analysis can be executed leading to a consistent integrated system. In particular, they allow assume/guarantee reasoning at the component level for which it is verified whether the guarantees of one component fulfill the assumptions of the connected components. Note that the CBCD extends this reasoning throughout the entire design process, going from contract negotiation to verification while detailing the design. Similar to the *AutoFOCUS 3* framework, Ringert et al. present in [RRW12, RRW13] the *MontiArcAutomaton* framework. Focusing on robotic applications, it extends the *AutoFOCUS 3* framework so that models of both the architecture and the behavior can be specified. Using the implemented transformations, engineers are able to analyze their designed system using formal methods and to generate (deployment) code.

6.3 Designing a Hybrid Hydraulic Vehicle Using a CBCD Driven Design Process

As already mentioned in Section 2.2.1, engineers follow a well-defined process when designing complex systems such as a CPS. In Section 3.6.2 a rather generic enactment of a CBCD driven design process is shown in Figure 3.14. The design process of the HHV extends the generic enactment and is shown in Figure 6.1. For the sake of simplicity, we limit ourselves to detailing the process model. In what follows, we discuss in more detail the followed process and elaborate on where in the process our methods and techniques, and the supporting integrated CBCD framework, is able to support the engineers.

6.3.1 Preliminary Design

Functional Specifications

Given the set of requirements listed in Section 1.5.2, the design team considers the system to be designed as a black box and derives the technical requirements called *specifications*. For the HHV case study, this includes (i) determining the conditions for which a driving mode is operational, (ii) exploring the component dimensioning of the drivetrain configuration, (iii) estimating the prerequisites of the embedded hardware and (iv) determining the environmental conditions under which the system should operate. The latter is of importance to allow for signal conditioning; that is, determining the inputs and outputs of the system, their data values, type, and precision (i.e., resolution). As a running example, we highlight the translation of one requirement to a set of specifications. System requirement 1.(d), for example, is refined by each domain as follows:

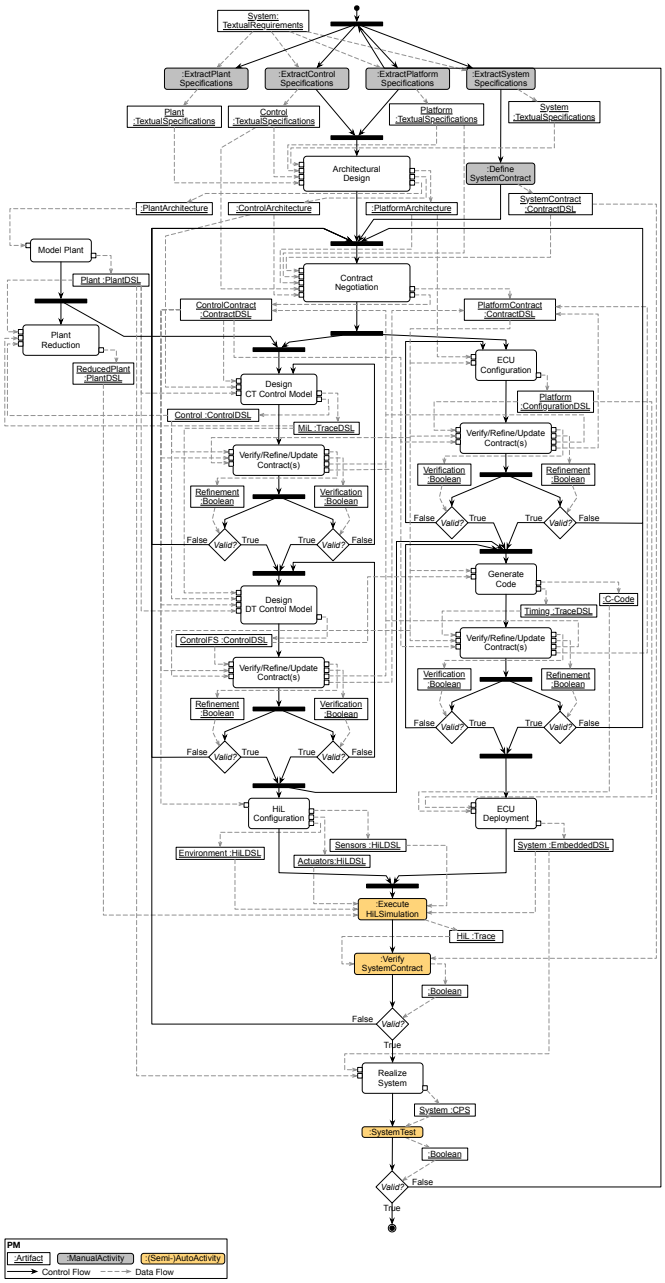


Figure 6.1: Process model for designing a Hybrid Hydraulic Vehicle using a CBCD driven design process

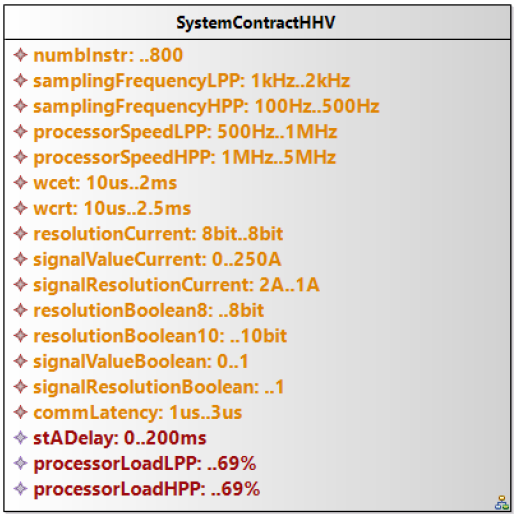


Figure 6.2: System contract for the HHV case study

— Mechanical domain—the braking torque on both axles should be at least 3500 Nm.

— Embedded domain—braking signal must be processed within 20 ms.

— Control domain—the controller must switch from Drive to Brake (in Forward operation) when brake is engaged and the actual speed is larger than 0 km/h.

The system to be designed may also be part of a larger system. In that respect, a systems engineer ensures that possible imposed design constraints are met. The integrated CBCD framework supports the systems engineer by providing the ability to formalize the imposed design constraints (i.e., specifications) by means of a system contract. Figure 6.2 depicts such a contract for the HHV case study, in which a systems engineer imposes a set of design constraints (in terms of design parameters) for the control and embedded domain. Orange contract entries denote contract assumptions, while red ones denote contract guarantees. The latter ones are derived from the system requirements, that is, the ones that are imposed by, for example, an Original Equipment Manufacturer (OEM). The system contract's assumptions are estimates made by the systems engineer, for each of the domains involved, to fulfill the guarantees. Each entry is characterized by a (range of) possible value(s). Note that the activity of defining a contract conforms to the blue user activity block in Figure 5.7.

Architectural Design

Once the specifications for the domain (and their respective viewpoints) are derived, each viewpoint defines a high-level architecture describing at the structure of their implementation in terms

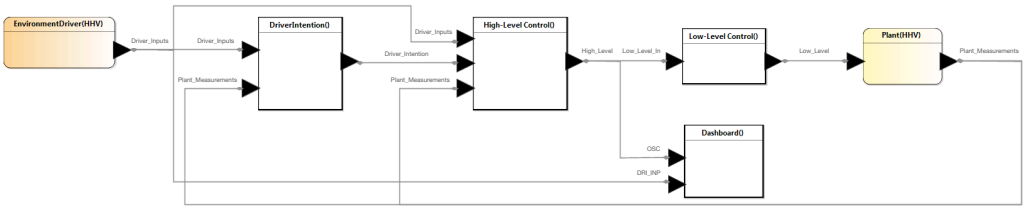


Figure 6.3: Architecture of the control viewpoint for the HHV case study

of elements/components, and how they are related to each other (within the same viewpoint). In order to be able to define such a framework, we presume engineers are able to reason about a possible (set of) implementations given the set of requirements and specifications.

With respect to the control viewpoint, for example, control engineers are able to determine which drivetrain elements (Figure 1.3) should be controlled in which driving mode (Figure 1.4). Based on this reasoning, control engineers decide upon a high-level model architecture enabling them to reason about: (i) the high-level control strategy consisting of several (reusable) components implementing a certain functionality and (ii) the interfaces between those functional components. The integrated CBCD framework supports engineers in this process by providing a library enabling them to graphically construct their viewpoint-specific architectures within the framework. As an example, Figure 6.3 depicts the architecture defined by the control viewpoint for the HHV. However, often engineers opt to construct the high-level architecture using viewpoint-specific tools they are familiar with (e.g., Simulink® or SystemDesk®). As such, model-to-model transformations enable the engineers to import these architectures in the framework. Note that defining functional components as part of an architecture is characteristic for MBSE, that aims the reuse of models.

For the embedded domain, the integrated CBCD framework provides a library to construct the architecture of the platform viewpoint, combining both the hardware and software viewpoint of the system. In that respect, several ECUs can be connected to each other using a CAN communication medium. Each ECU consists of one or more processing units running a RTOS. In the example of the HHV, a distributed hardware architecture is decided connecting two single-core ECUs, a Low Performant Processor and a High Performant Processor, using a CAN communication channel.

6.3.2 Contract Negotiation

The viewpoint-specific architectures and specifications combined with the system contract serve as an input of the contract negotiation phase. In this design phase, that is obligatory to ensure consistent concurrent design, the control and embedded engineers reason on how their respective (high-level) architectures can possibly be mapped onto each other and how this mapping restricts

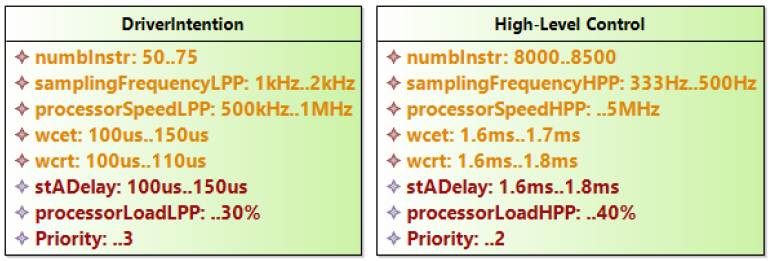


Figure 6.4: Negotiated *DriverIntention* and *High-Level Control* mapping contract for the HHV case study

the viewpoint-specific design parameters. Throughout this process it is recommended to consider the functional components of the control architecture as SWCs for the software viewpoint. Each SWC can then be mapped onto a processing unit of the hardware architecture. For each of these mappings, the integrated CBCD framework enables the engineers to formalize the imposed parameter restrictions using a (set of) mapping contract(s).

With respect to the HHV case study, four so called mapping contracts are negotiated corresponding to the number of functional components defined in the control architecture: *DriverIntention*, *Dashboard*, *High-Level Control* and *Low-Level Control*. As an example, Figure 6.4 depicts the negotiated *DriverIntention* and *High-Level Control* mapping contract. The integrated CBCD framework assists engineers in the negotiation phase by evaluating the imposed restrictions on consistency and provides textual recommendations to resolve detected inconsistencies. As discussed in Section 3.6, the domain knowledge stored in the ontology is therefore parsed such that design parameters can be related to each other. If a (chain of) $\mathbb{L}3$ relationship(s) exists, the symbolic execution engine substitutes the parameters with the negotiated values. The results of these calculations are used to evaluate the consistency of the negotiated mapping contract(s). For the *DriverIntention* contract in Figure 6.4, for example, the consistency analysis module, shown in Figure 5.7, detects two inconsistencies. One relates to a mismatching maximum values for the *wcrt* and *stADelay* parameter. The second inconsistency refers to the maximum value of the *samplingFrequency* parameter. Because of the $\mathbb{L}3$ relation $processorLoad = wcrt * samplingFrequency$ that exists in the ontology, it is determined by our CBCD framework that the *samplingFrequency* parameter is defined too low considering the maximum defined *processorLoad* and the *wcrt*.

Similar to the A/G reasoning in *AutoFOCUS 3* [HST10, HF11, AVT+15], the integrated CBCD framework allows engineers to complement the mapping with interface contracts (that are also defined during the negotiation phase). These contracts restrict design parameters related to the interfaces between components: signal resolution, minimum and maximum signal value, etc.

Note that the negotiated contracts are strict refinements of the earlier defined system contract. The integrated CBCD framework is able to verify whether the set of interface and mapping contracts refine the system contract. Again, textual notifications are provided to the engineers if inconsistencies are detected. It is up to the engineers, however, to resolve or even tolerate these inconsistencies. In other words, the framework only provides recommendations to resolve the detected inconsistencies but does not restrict further system development.

6.3.3 Concurrent Detailed Design

Once the contracts are negotiated, low-level details are implemented concurrently to each other using domain-specific tools and methods. If and only if the detailed design (i.e., the design parameters) satisfy the negotiated contracts, consistency across domains can be guaranteed. Each parallel design activity must be considered as a further refinement of the viewpoint-specific architectures. For each of the domains involved in the design of the HHV, we detail the design activity and elaborate on how the integrated CBCD framework assists engineers in preserving consistency. Note that the described concurrent design process is depicted in Figure 6.1.

Mechanical Design

Although we will not focus on the mechanical design, we dedicate a brief paragraph to the design of the plant model as it is used for various validation purposes, that is, MiL, SiL, and HiL simulations.

Given the viewpoint-specific architectural description, the mechanical engineer defines the necessary components and their dimensioning using mechanical modeling tools such as LMS AMESim™. Plant modeling enables the mechanical engineer to gain insights in the mechanical behavior of the system prior to the construction of the real system. Furthermore, the plant model is used by the other domains to verify the behavior of the control algorithm using MiL simulations and to execute HiL simulations to verify the integration of the control and embedded domain. The plant model for the HHV is an extension of the validated plant model of the conventional Citroën C3 plant model. As such, the hydraulic components of Figure 1.3 are added to the conventional plant model.

Control Design

As already stated in Section 3.7, the integrated CBCD framework does not explicitly provide viewpoint-specific contracts to the engineers. Instead, the framework exports the viewpoint-specific architectures to domain-specific tools (e.g., Simulink®) while incorporating relevant design parameters negotiated in the contracts. The framework therefore relies on the RTE method

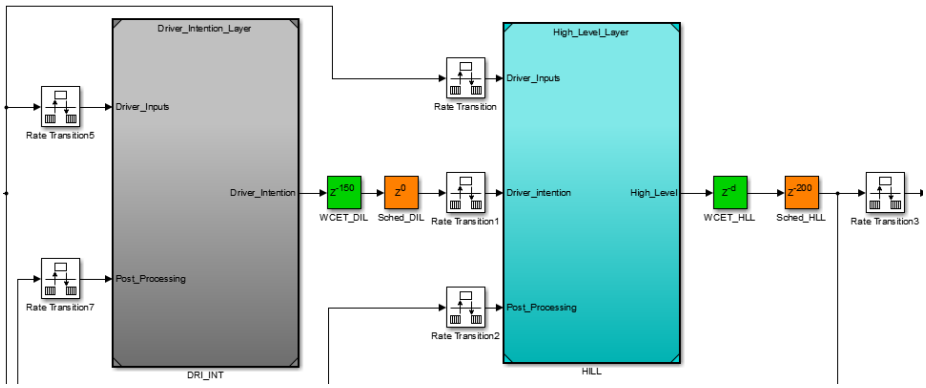


Figure 6.5: Exported control viewpoint for the HHV case study

discussed in Chapter 4. To enable this, however, the control engineer first needs to associate a mapping and interface contract to one or more functional components and interfaces, respectively, of the control architecture defined in the framework. As an example, Figure 6.5 shows the exported Simulink® representation for the HHV case study, that incorporates the negotiated timing information of Figure 6.4. Note that the timing information is expressed in μs. The green and orange delay blocks represent the WCET and the scheduling time, respectively. Furthermore, the RTE method updates the viewpoint-specific parameters of the control architecture and, related to that, the implementation in Simulink® when (in parallel) the platform is detailed. As such, the views remain consistent during the detailed design phase.

Given this high-level model framework, the control engineer models the behavior of the control algorithm. For this, they may rely on a library containing a number of functional units, designed within the scope of other projects, that can be reused in the current project context. Functional units designed or modified within the scope of the running project are added to the library for future reuse. Note that the context in which these functional units are designed is preferably added to the library as well. This context related information can be used by design-space exploration techniques to (semi-)automatically search for an optimal implementation. Each selected or designed functional unit is tested under the environmental conditions derived from the requirements. Such a simulation is called a unit test.

Once the control algorithm is modeled, a synchronization between the mechanical and control engineer takes place as the behavior of the control algorithm can only be tested by executing a MiL simulation. Hence the need for a plant model representing the mechanical behavior. As the relevant information of the embedded platform is incorporated in the MiL simulation model (see Figure 6.5), the control viewpoint is able to evaluate the behavior of the control model as if it were deployed. Table 6.1 gives an overview of the MiL simulation results for both a naive behavioral simulation and a simulation incorporating platform information. The simulation evaluates whether

	Naive Model	Annotated Model
Brake Activation Time [ms]	650	1980
Brake Distance [m]	53.3	278

Table 6.1: MiL simulation result for the HHV case study

the control algorithm meets the specifications and requirements related to the maximum brake distance when a brake command is issued at a speed of 100 km/h. When evaluating using the naive implementation the required maximum brake distance of 54 m is being met. When incorporating the timing information, however, the brake distance increases to 278 m so that the requirement is violated. Note that the brake activation time also incorporates the delay due to mechanical factors (e.g., inertia). In the naive case, this amounts to 650 ms.

Typically, control engineers use continuous-time, floating-point, models while designing their control algorithms. Depending on the chosen hardware platform, however, the processor may not be able to deal with floating-point numbers. As such, control engineers need to transform the floating-point numbers to a fixed-point representation in their control model.

Whether or not the hardware platform supports floating numbers, the algorithm is executed at a certain clock rate once deployed. For this reason, the continuous-time control model is transformed to a discrete-time (also called fixed-step) control model by selecting an appropriate (fixed-step) solver and step size. After every conversion, the control engineer executes a Back-to-Back (B2B) MiL simulation to verify the transformation/conversion such that the distance (i.e., the error) between the continuous-time floating-point representation and the discrete-time (fixed-point) representation is within certain margins.

As shown in Figure 6.1, the integrated CBCD framework also supports control engineers throughout their design process by verifying whether the control model satisfies the negotiated mapping contracts. For example, after discretizing the control model the framework verifies whether the chosen step size corresponds to the *samplingFrequency* defined in the mapping contract. Control engineers may also decide to update the negotiated mapping contracts. After all, negotiated design parameters may be misjudged, which may become clear throughout the design process. When updating design parameters is considered as a strict refinement of the negotiated contract, then the design parameters of the other viewpoints can be automatically updated. If not, the mapping contracts need to be renegotiated. Note that automatically updating and refining a mapping contract from the control viewpoint perspective is currently not supported in the framework.

Platform Design

For the embedded domain, the integrated CBCD framework enables the hardware and software viewpoint to further detail the platform architecture defined in the preliminary design phase. With

respect to configuring the earlier defined ECUs, the software viewpoint is able to allocate a number of tasks to the RTOS for which a set of design parameters can be defined by the engineer; that is, period, deadline, priority, and so forth. Each task executes one or more runnables holding exactly one SWC (i.e., a functional component of the control architecture).

Mapping a SWC onto an ECU and defining the *holds* relation between a runnable and a SWC, as shown in Figure 6.6, can either be done manually or automatically using the DSE capabilities of the integrated CBCD framework. In the former case, it is required that the hardware platform is configured as described earlier and, more precisely, that the design parameters previously listed are defined. As such, a schedulability analysis can be executed by the framework that in turn annotates both control and platform architecture with the results of the analysis. For example, after analysis each task is annotated with the WCRT information. In case an automatic allocation is requested by the engineers, it is sufficient to assign a (set of) negotiated mapping contract(s) to the tasks defined in the platform architecture. Using the Tindell equations [TC94], an optimal mapping is suggested given the worst case conditions negotiated in the mapping contracts.

As with the control design workflow, the integrated CBCD framework supports embedded engineers throughout their design process by verifying whether the platform configuration satisfies the negotiated mapping contracts. Note that to enable these consistency analyses it is required to relate platform components, such as tasks, to a (set of) negotiated mapping contract(s). For example, the consistency analysis evaluates if the WCRT determined by the schedulability analysis is within the range specified in the negotiated mapping contract. Using the RTE capabilities, the framework is able to update the viewpoint-specific parameters of the embedded architecture when (concurrently) the control architecture (and its implementation) is further detailed. As such, the viewpoints remain consistent during the concurrent design phase.

The design process of the embedded domain continues by generating code from the discretized, fixed-point, control algorithm. Therefore, the software engineers typically rely on industry accepted tools such as Simulink® Coder™. Using PiL and SiL simulations, the generated code is compared against the trace obtained from the MiL simulation (using the continuous-time model)

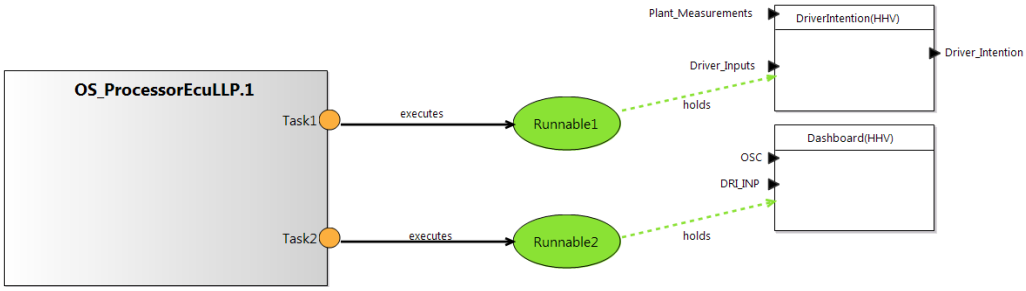


Figure 6.6: Software configuration of the platform for the HHV case study

to examine errors introduced by the compiler or linker. Although not supported by the integrated CBCD framework at the time of writing, the trace obtained from the MiL and SiL can be evaluated for consistency with (behavioral) design parameters defined in the mapping contracts.

Integration

If the PiL and SiL simulation traces satisfies the specifications (i.e., the contracts), the generated code scheduled by a RTOS is deployed on the platform by the embedded engineers. It goes without saying that they rely on the earlier defined and analyzed platform architecture. A final HiL simulation enables one to verify whether the system specifications, and the related system contract, are satisfied. If the suggested CBCD driven design process is strictly followed such that the domain-specific implementations satisfy, or refine, the negotiated contracts, system specifications should be met so that the final system can be realized and validated. However, if the HiL simulation reveals that the integration has failed, the design process must be repeated starting with renegotiating the contracts.

6.4 Conclusion

Given the novel methods and techniques presented in Chapter 3 through 5, we demonstrated how the integrated CBCD framework, combining these methods and techniques, can be used to support engineers through a design process that incorporates contracts to enable consistent concurrent design. Although the CBCD method and its supported framework does not impose a design process to the engineers, we explicitly applied our contributions to the concurrent design process of hybrid hydraulic vehicle. From this case study, we demonstrated that, at the time of writing, the integrated CBCD framework supports engineers:

— In defining their viewpoint-specific architectures. Engineers are free to specify these architectures in the framework or in tool of their choice.

— In defining system, mapping, and interface contracts. Using consistency analysis techniques, that rely on the viewpoint-specific knowledge modeled in the upper ontology, it is verified whether (i) the design parameters defined in a single contract are consistent and (ii) mapping and interface contracts are correct refinements of the system contract.

— In detailing their architectures. In particular the embedded architecture for which DSE techniques are provided to determine an optimal mapping and schedulability analysis techniques to verify the feasibility of the mapping.

— In providing a domain-specific view on the system to be developed. In particular, the control architecture is exported to a Simulink® representation incorporating relevant platform information at the level of abstraction of the control engineer.

— In verifying whether the design satisfies the viewpoint-specific contract. Such a viewpoint-specific contract is automatically deduced from the negotiated mapping contract, although, not made explicit to the engineers.

Throughout the enactment of the development process we experienced how the integrated framework accurately indicates the inconsistencies between the different viewpoints, whether deliberately introduced or not, both during contract negotiation and detailed design. Moreover, the framework enabled us to verify whether the mapping contracts correctly refined and implemented the constraints imposed by the systems engineer (and its related contract). The framework is also able to deal with derivatives of a particular design parameter (e.g., *processorSpeed* and *processorSpeed_LPP*) when verifying for consistency. In case the development team uses different terminology, it is sufficient to extend the instances of the ontology with these synonyms, for example, *clockSpeed* as a synonym for *processorSpeed*. This is particularly interesting from a business perspective as the underlying consistency relationships (i.e., the upper ontology) do not require any changes.

However, this case study also revealed a number of shortcomings, in particular for the control viewpoint. We experienced that the control domain ontology is underspecified so that a limited number of consistency relations can be verified. Partly because only $\mathbb{L}3$ relationships are verified, it is currently not possible to verify the behavioral (simulation) traces of an implementation nor to execute sensitivity analysis between viewpoints. In that respect, it is also recommended to extend the expressivity of the contracts with temporal information. Temporal logic seems to be suitable for these purposes, although, at the cost of user friendliness because of a lower level of abstraction.

CHAPTER 7

Conclusion

The design of a Cyber-Physical System (CPS) often requires a multidisciplinary design team to collaborate, for which each discipline (e.g., control logic, embedded system design, and mechanics) adopts a different viewpoint on the system under design. As a consequence, requirements, representing the behavior of the real world system in a certain context, are –often implicitly– shared among viewpoints due to the overlapping concerns, leading to inconsistent decisions on shared design parameters. This results in iterative, time consuming design processes where inconsistencies are resolved, in turn possibly creating new ones.

To avoid inconsistencies between design artifacts (i.e., design parameters) while designing a system, Contract-Based Design (CBD) has been proposed as a method to detect and avoid inconsistencies. Therefore, a contract consisting of a set of assumptions and guarantees is defined between engineers prior to the design phase. The assumptions and guarantees describe the conditions under which a system promises to operate while satisfying desired properties. Multi-viewpoint concurrent design processes supported by the CBD theory are characterized by a common engineering phase in which viewpoint-specific architectures are defined and a common contract is negotiated. As engineers consider the design from a different viewpoint they face difficulties in reasoning how design artifacts, and their corresponding design parameters, originating from different viewpoints are related to each other. As a result, the content of the common contract may be incomplete or, even worse, inconsistent. In addition to this, deriving viewpoint-specific contracts, containing information that is only relevant for a particular viewpoint, is challenging as it requires translation mechanisms from commonly negotiated design parameters to viewpoint-specific design parameters.

We contributed to contract driven co-design processes by proposing a Contract-Based Co-Design (CBCD) method that explicitly models the domain knowledge of each viewpoint using properties in so called domain ontologies. They are combined using general concepts in an upper ontology. Design parameters are related to one or more ontological properties, that in turn can be related to each other using *Require* relationships. This enables ontological reasoning whereby syntactically and semantically different design parameters are related to each other. As a result, consistency of negotiated Assume/Guarantee (A/G) contracts can be validated so that consistent concurrent design is enabled.

To support engineers when dealing with CBCD driven design processes, this work proposed an integrated CBCD framework in which preliminary design architectures and A/G contracts can be defined, while interacting with third-party tools that detail the viewpoint-specific implementations. Given the upper ontology, the framework is able to validate for consistency among negotiated design parameters specified in a (set of) A/G contract(s). In addition, the framework is able to verify whether parts of the implementation satisfy the negotiated (contracts).

To assist engineers in their implementation, the integrated CBCD framework is able to derive viewpoint-specific contracts. They specify what a particular viewpoint should guarantee under a given set of assumptions. The assumptions of these derived viewpoint-specific contracts are typi-

cally design parameters guaranteed by another viewpoint. Although ontological reasoning already syntactically and semantically transforms design parameters originating from a different viewpoint, engineers are often not aware to what extent the assumed design parameters influence the viewpoint-specific implementations and, as such, the contract parameters to be guaranteed.

In this work, we proposed a Round-Trip Engineering (RTE) method that provides the control viewpoint a (top-level) control architecture, in a modeling environment such as Simulink®, in which delay blocks are introduced that represent the assumed timing related design parameters of the platform viewpoint. As such, the RTE method enables control engineers to evaluate the behavior of the control algorithm as if it were deployed on the platform, at their level of abstraction using methods and techniques they are familiar with.

We also discussed how the RTE method can also be used to support control and embedded engineers in the tradeoff analysis of design parameters in the contract negotiation process. As negotiated parameters may be overestimated, refinements in one viewpoint can be pushed to the other viewpoints so that behavioral simulations during design represent the integrated system at all times. As such, consistency between viewpoints is ensured throughout the entire design process.

While detailing the design (in a concurrent setting), the need often arises to explore different design alternatives for a specific problem. For example, embedded engineers are often charged with a bin packing design problem that requires them to find an optimal mapping between the control architecture and the embedded architecture while optimizing processor load and cost. A multitude of Design-Space Exploration (DSE) techniques exist to support engineers in exploring possible alternatives of a particular design problem. To cover the possibly infinite design space, we discussed how different exploration techniques may need to be chained in order to find a (sub-) optimal solution for a particular design problem. However, CBCD driven design processes impose a negotiation phase in which contracts are defined. As they specify (ranges of) values for different design parameters, contracts limit the design phase and, as such the exploration space. As a result, the most suitable exploration technique can be selected to (semi-)automatically search for an optimal implementation. For the bin packing problem, we demonstrated how the integrated CBCD framework can be extended such that genetic algorithms can be used to explore for an optimal mapping between the control and embedded architecture. Furthermore, we discussed how the framework may assist control engineers in exploring the optimal implementation of a component with respect to the expected behavior of a control component and its complexity (expressed in terms of Worst-Case Execution Time) defined in the negotiated (set of) contract(s).

Nevertheless, due to the numerous exploration techniques available, it is often challenging to choose the most appropriate (set of) technique(s) for a particular optimization problem. In this work, we categorized different approaches of Model-Driven Design-Space Exploration techniques in an initial pattern catalog. Inspired by software design patterns, we described each pattern using

a well-defined structure in which its intent, structure, consequences, known uses, and application are described. With respect to the structure, we made use of a Formalism Transformation Graph and Process Model (FTG+PM) to graphically support the involved formalisms and their relations using model transformations. The introduced pattern catalog should not be considered complete. With the support of the community, it is our ambition to extend this towards a more complete pattern catalog, similar to the widely available software design patterns used in software engineering.

Finally, we demonstrated how the integrated CBCD framework, incorporating the methods and techniques presented in this work, can be used throughout the enactment of the development process of a Hybrid Hydraulic Vehicle (HHV). We experienced: (i) how the framework accurately indicated the inconsistencies between the different viewpoints, (ii) how the framework enabled us to verify whether the mapping contracts correctly refined and implemented the constraints imposed by the systems engineer, (iii) how the framework is able to deal with derivatives of a particular design parameter, and (iv) how the (upper) ontology can be tailored for a particular design team (and its used terminology).

It may be clear that this dissertation has led to a number of contributions that enables engineers to reasons about consistency prior and during the design of cyber-physical systems. Nevertheless, many challenges remain of which the following may significantly contribute to the work presented.

Traceability

At the time of writing, the CBCD framework provides support to detect inconsistencies between the prenegotiated mapping contracts and the detailed design or implementation. Therefore, we explicitly link viewpoint-specific architectures with negotiated contracts in the framework. As such, a trace is created between the architectures, the contracts, and the (exported) model artifacts. These tracing capabilities should be further extended such that there is a link between requirements, functional specifications, architectural design, implementations, and the various verification and validation steps. It enables one (i) to verify whether mapping contracts are complete in term of negotiated design parameters and (ii) in case validation fails, to trace back to the cause of the problem. To do so, the requirements and functional specifications need to be translated from a human language into a language that can be processed by a formal analysis engine. Note that therefore the contract language also needs to be redefined in order to achieve traceability throughout the design process.

Validity Frames

More and more companies are investing time and money in applying a Model-Based Systems Engineering (MBSE) approach to deal with the increasing complexity of cyber-physical systems. One of the basic principles of MBSE is the reuse of models, that are an abstraction of the real world and are only valid under certain conditions. In literature, experimental/validity frames are suggested to define these conditions and describe what the model guarantees under these conditions. Note the link with our CBCD framework where contracts characterize a system using a set of preconditions and postconditions. Both the content and the formalization of validity frames are yet unclear to the community. We consider that our contributions can be useful in the formalization of these validity frames. Moreover, combining validity frames and contracts might possibly enable engineers to (i) (semi-)automatically define contracts based on a set of selected models and, vice-versa, (ii) limit the number of usable models stored in a library when contracts are defined.

Control-Embedded Co-Simulation

The CBCD framework currently implements the RTE method that changes the behavior of the control model given the timing information of the deployed model. It enables a control engineer to evaluate the control performance of an algorithm as if it were deployed on the embedded platform. The method should be extended such that other relevant embedded details can be incorporated when evaluating the control algorithm.

A similar co-simulation view should be foreseen for the embedded domain as well, for which the software and hardware viewpoint are more detailed. As for the control domain, this extension enables embedded engineers to evaluate the impact of their decisions on the behavior of the control model. Therefore, a detailed model of the embedded platform should be developed that triggers the model of the control algorithm. As of today, many tools exist that claim to model the embedded hardware (processors, caches, communication buses, etc.) and software (RTOS, AUTOSAR, etc.) at the most appropriate level of abstraction using the most appropriate formalism [CSCS13, CCS13, MD14, CMDN15, HCÅ03, CHL$^+$03, LMMHY16, DMDV17]. In most cases, however, these tools cannot be used to co-simulate the model of the control algorithm and the embedded implementation. Or the tools provide too much/too less detail to the engineers.

Sensitivity Analysis

Having a detailed model of the embedded platform enables the tool to run sensitivity analyses, able to detect which control algorithm parameters are sensitive to changes of the embedded platform. This information can be used to reduce the size of contracts and, related to that, to facilitate

contract management. The latter refers to the fact that embedded engineers often need to manage multiple contracts originating from different control engineers (and as such different control algorithms). It is likely that contracts will be refined or, even worse, need to be renegotiated as the design process evolves. As the embedded architecture, consisting of multiple controllers connected to each other via a communication channel, implements multiple control algorithms (e.g., anti-lock braking, traction control, blind spot detection), embedded engineers need to be supported in the possible relocation of the control algorithm. Using the sensitivity information, embedded engineers are provided an overview on how their decisions may affect the behavior of the control algorithm(s).

Industrial Adoption

The CBCD framework needs to be considered as an academic proof of concept that is used to validate our research by means of two case studies. As such, its Technology Readiness Level (TRL) can be qualified at level 3. To enable our methods to be adopted by industry, a tool should be developed so that our framework becomes (i) scalable, (ii) performant, (iii) extendable, and (iv) customizable. (i) and (ii) relates to the handling of industrial size models and the management of many contracts, while (iii) and (iv) refer to the possible research directions described in this section.

Furthermore, empirical research should reveal whether the upper ontology is mature enough for adoption by industry. In that respect, it is very likely that the ontology of the control domain needs to be revisited. In order to support all the domains involved in the design of CPSs with the CBCD method, the upper ontology also needs to be further extended with those domain ontologies.

Bibliography

[Abb12] Sunitha Abburu. A Survey on Ontology Reasoners and Comparison. *International Journal of Computer Applications*, 57(17):33–39, November 2012.

[ADL+12] Moussa Amrani, Jürgen Dingel, Leen Lambers, Levi Lúcio, Rick Salay, Gehan Selim, Eugene Syriani, and Manuel Wimmer. Towards a Model Transformation Intent Catalog. In *Proceedings of the First Workshop on the Analysis of Model Transformations*, AMT '12, pages 3–8, New York, NY, USA, 2012. ACM.

[Ala13] Emhimed Alatrish. Comparison of Some Ontology Editors. *Management Information Systems*, 8(2):018–024, 2013.

[AUT] AUTOSAR. Automotive open system architecture release 4.3.

[AVS+14] Hani Abdeen, Dániel Varró, Houari Sahraoui, András Szabolcs Nagy, Csaba Debreceni, Ábel Hegedüs, and Ákos Horváth. Multi-objective Optimization in Rule-based Design Space Exploration. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ASE '14, pages 289–300, New York, NY, USA, 2014. ACM.

[AVT+15] Vincent Aravantinos, Sebastian Voss, Sabine Teufl, Florian Hölzl, and Bernhard Schätz. AutoFOCUS 3: Tooling Concepts for Seamless, Model-based Development of Embedded Systems. In *8th International Workshop on Model-based Architecting of Cyber-physical and Embedded Systems*, volume 1508 of *CEUR Workshop Proceedings*, pages 19–26, 2015.

[BBR+06] R.G. Branco, D. Broomfield, V. Rampon, P.C.R. Garcia, and J.P. Piva. Accidental asphyxia due to closing of a motor vehicle power window. *Emergency Medicine Journal: EMJ*, 23(4), 2006.

[BCF+08] Albert Benveniste, Benoît Caillaud, Alberto Ferrari, Leonardo Mangeruca, Roberto Passerone, and Christos Sofronis. Multiple Viewpoint Contract-Based Specification and Design. In *Formal Methods for Components and Objects*,

volume 5382 of *Lecture Notes in Computer Science*, pages 200–225. Springer-Verlag, Berlin, Heidelberg, 2008.

[BCM+03] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, New York, NY, USA, 2003.

[BCN+12] Albert Benveniste, Benoît Caillaud, Dejan Nickovic, Roberto Passerone, Jean-Baptiste Raclet, Philipp Reinkemeier, Alberto L. Sangiovanni-Vincentelli, Werner Damm, Tom Henzinger, and Kim G. Larsen. Contracts for Systems Design. Technical Report RR-8147, INRIA, 2012.

[BCN+15a] Albert Benveniste, Benoît Caillaud, Dejan Nickovic, Roberto Passerone, Jean-Baptiste Raclet, Philipp Reinkemeier, Alberto L. Sangiovanni-Vincentelli, Werner Damm, Tom Henzinger, and Kim G. Larsen. Contracts for Systems Design : Theory. Technical Report RR-8759, INRIA, 2015.

[BCN+15b] Albert Benveniste, Benoît Caillaud, Dejan Nickovic, Roberto Passerone, Jean-Baptiste Raclet, Philipp Reinkemeier, Alberto L. Sangiovanni-Vincentelli, Werner Damm, Tom Henzinger, and Kim G. Larsen. Contracts for Systems Design: Methodology and Application cases. Technical Report RR-8760, INRIA, 2015.

[BDH+12] Sebastian S. Bauer, Alexandre David, Rolf Hennicker, Kim Guldstrand Larsen, Axel Legay, Ulrik Nyman, and Andrzej Wąsowski. Moving from Specifications to Contracts in Component-Based Design. In *Fundamental Approaches to Software Engineering*, volume 7212 of *Lecture Notes in Computer Science*, pages 43–58, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[Bec09] Sean Bechhofer. *OWL: Web Ontology Language*, pages 2008–2009. Springer US, Boston, MA, 2009.

[BFMSV08] Luca Benvenuti, Alberto Ferrari, Emanuele Mazzi, and Alberto L. Sangiovanni-Vincentelli. Contract-Based Design for Computation and Verification of a Closed-Loop Hybrid System. In *Hybrid Systems: Computation and Control*, volume 4981 of *Lecture Notes in Computer Science*, pages 58–71. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[Bha11] Ajinkya Y. Bhave. *Multi-View Consistency in Architectures for Cyber-Physical Systems*. PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, 2011.

[BKV14] Bruno Barroca, Thomas Kühne, and Hans Vangheluwe. Integrating Language and Ontology Engineering. In *Proceedings of the 8th Workshop on Multi-Paradigm*

Modeling (MPM), volume 1237 of *CEUR Workshop Proceedings*, pages 77–86, 2014.

[BLTT12] David Broman, Edward A. Lee, Stavros Tripakis, and Martin Törngren. Viewpoints, Formalisms, Languages, and Tools for Cyber-physical Systems. In *Proceedings of the 6th International Workshop on Multi-Paradigm Modeling*, MPM '12, pages 49–54, New York, NY, USA, 2012. ACM.

[BP13] Frank R. Burton and Simon Poulding. Complementing Metaheuristic Search with Higher Abstraction Techniques. In *Proceedings of the 1st International Workshop on Combining Modelling and Search-Based Software Engineering (CMSBSE)*, number 4, pages 45–48, Piscataway, NJ, USA, May 2013. IEEE Press.

[BS01] Manfred Broy and Ketil Stolen. *Specification and Development of Interactive Systems*. MCS. Springer-Verlag New York, 2001.

[Can74] Georg Cantor. Ueber eine Eigenschaft des Inbegriffs aller reellen algebraischen Zahlen. *Journal für die reine und angewandte Mathematik (Crelle's Journal)*, 77(258–262), 1874.

[CCS13] Federico Ciccozzi, Antonio Cicchetti, and Mikael Sjödin. Round-trip support for extra-functional property management in model-driven engineering of embedded systems. *Information and Software Technology*, 55(6):1085–1100, June 2013.

[CDLOP02] Gennaro Costagliola, Andrea De Lucia, Sergio Orefice, and Giuseppe Polese. A classification framework to support the design of visual languages. *Journal of Visual Languages & Computing*, 13(6):573–600, 2002.

[CH06] Krzysztof Czarnecki and Simon Helsen. Feature-Based Survey of Model Transformation Approaches. *IBM Systems Journal*, 45(3):621–645, 2006.

[CHL+03] Anton Cervin, Dan Henriksson, Bo Lincoln, Johan Eker, and Karl-Erik Årzén. How does control timing affect performance? Analysis and simulation of timing using Jitterbug and TrueTime. *IEEE Control Systems*, 23(3):16–30, 2003.

[CMDN15] Fabio Cremona, Matteo Morelli, and Marco Di Natale. TRES: A Modular Representation of Schedulers, Tasks, and Messages to Control Simulations in Simulink. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, SAC '15, pages 1940–1947, New York, NY, USA, 2015. ACM.

[CSCS13] Federico Ciccozzi, Mehrdad Saadatmand, Antonio Cicchetti, and Mikael Sjödin. An automated round-trip support towards deployment assessment in component-based embedded systems. In *Proceedings of the 16th International ACM Sigsoft symposium on Component-based software engineering*, CBSE '13, pages 179–188, New York, NY, USA, 2013. ACM.

[dAH01] Luca de Alfaro and Thomas A. Henzinger. Interface Automata. *SIGSOFT Softw. Eng. Notes*, 26(5):109–120, September 2001.

[Dam05] Werner Damm. Controlling speculative design processes using rich component models. In *Fifth International Conference on Application of Concurrency to System Design (ACSD'05)*, pages 118–119, June 2005.

[DCB+11] Joachim Denil, Antonio Cicchetti, Matthias Biehl, Paul De Meulenaere, Romina Eramo, Serge Demeyer, and Hans Vangheluwe. Automatic Deployment Space Exploration Using Refinement Transformations. *Electronic Communications of the EASST - Recent Advances in Multi-paradigm Modeling*, 50, 2011.

[DCtTdK11] Kathrin Dentler, Ronald Cornet, Annette ten Teije, and Nicolette de Keizer. Comparison of reasoners for large ontologies in the OWL 2 EL profile. *Semantic Web*, 2(2):71–87, 2011.

[DDGV16] István Dávid, Joachim Denil, Klaas Gadeye, and Hans Vangheluwe. Engineering Process Transformation to Manage (In)consistency. In *Proceedings of the 1st International Workshop on Collaborative Modelling in MDE (COMMitMDE)*, volume 1717 of *CEUR Workshop Proceedings*, pages 7–16, 2016.

[DDV15] István Dávid, Joachim Denil, and Hans Vangheluwe. Towards Inconsistency Management by Process-Oriented Dependency Modeling. In *Joint Proceedings of the 3rd International Workshop on the Globalization Of Modeling Languages (GEMOC) and the 9th International Workshop on Multi-Paradigm Modeling (MPM)*, volume 1511, pages 32–41. CEUR Workshop Proceedings, 2015.

[Den13] Joachim Denil. *Design, Verification and Deployment of Software Intensive Systems - A Multi-Paradigm Modelling Approach*. PhD thesis, University of Antwerp, 2013.

[Dep09] Department of Transportation (DOT). Proposed Rule: Power-Operated Window, Partition, and Roof Panel Systems. In *Federal Motor Vehicle Safety Standards*, number 74 FR 45143. National Highway Traffic Safety Administration (NHTSA), 2009.

[Dep11] Department of Transportation (DOT). Power-Operated Window, Partition, and Roof Panel Systems. In *Code of Federal Regulations*, number 49 CFR 571.118. National Highway Traffic Safety Administration (NHTSA), 2011.

[Dij59] Edsger Wybe Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, Dec 1959.

[DLFPF18] Loris Dal Lago, Orlando Ferrante, Roberto Passerone, and Alberto Ferrari. Dependability Assessment of SOA-based CPS with Contracts and Model-Based

Fault Injection. *IEEE Transactions on Industrial Informatics*, 14(1):360–369, 2018.

[DLTT13] Patricia Derler, Edward A. Lee, Stavros Tripakis, and Martin Törngren. Cyber-Physical System Design Contracts. In *Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems*, ICCPS '13, pages 109–118, New York, NY, USA, 2013. ACM.

[DMDV17] Joachim Denil, Paul De Meulenaere, Serge Demeyer, and Hans Vangheluwe. DEVS for AUTOSAR-based system deployment modeling and simulation. *SIMULATION*, 93(6):489–513, 2017.

[DMV14] Joachim Denil, Pieter J. Mosterman, and Hans Vangheluwe. Rule-Based Model Transformation For, and In Simulink. In *Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative*, number 4 in DEVS '14, pages 1–8, San Diego, CA, USA, 2014. Society for Computer Simulation International.

[DMV+17] István Dávid, Bart Meyers, Ken Vanherpen, Yentl Van Tendeloo, Kristof Berx, and Hans Vangheluwe. Modeling and Enactment Support For Early Detection of Inconsistencies in Engineering Processes. In *Proceedings of MODELS 2017 Satellite Event: Workshops (ModComp, ME, EXE, COMMitMDE, MRT, MULTI, GEMOC, MoDeVVa, MDETools, FlexMDE, MDEbug), Posters, Doctoral Symposium, Educator Symposium, ACM Student Research Competition, and Tools and Demonstrations*, volume 2019 of *CEUR Workshop Proceedings*, pages 145–154, 2017.

[DPAM02] Kalyanmoy Deb, Armrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, Apr 2002.

[DV05] Werner Damm and Angelika Votintseva. Boosting Re-use of Embedded Automotive Applications Through Rich Components. In *Proceedings of Foundations of Interface Technologies*, FIT'05, pages 1–18, 2005.

[EkT01] Jad El-khoury and Martin Törngren. Towards a toolset for architectural design of distributed real-time control systems. In *Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001) (Cat. No.01PR1420)*, pages 267–276, 2001.

[ELM+12] John C. Eidson, Edward A. Lee, Slobodan Matic, Sanjit A Seshia, and Jia Zou. Distributed Real-Time Software for Cyber-Physical Systems. *Proceedings of the IEEE*, 100(1):45–59, January 2012.

[EON+12] Bryan Eisenhower, Zheng O'Neill, Satish Narayanan, Vladimir a. Fonoberov, and Igor Mezić. A methodology for meta-model based optimization in building energy models. *Energy and Buildings*, 47:292–301, April 2012.

[Est08] Jeff A. Estefan. Survey of Model-Based Systems Engineering (MBSE) Methodologies. Technical Report Rev. B, INCOSE MBSE Initiative, May 2008.

[FBSG07] Madeleine Faugere, Thimothee Bourbeau, Robert De Simone, and Sebastien Gerard. MARTE: Also an UML Profile for Modeling AADL Applications. In *12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS)*, pages 359–364. IEEE, 2007.

[FDG+12] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, July 2012.

[FMS11] Sanford Friedenthal, Alan Moore, and Rick Steiner. *A Practical Guide to SysML: The Systems Modeling Language*. Elsevier Science, 2011.

[FNS+16] Felipe R. Franco, João H. Neme, Max M. Santos, João N.H. da Rosa, and Inácio M. Dal Fabbro. Workflow and toolchain for developing the automotive software according AUTOSAR standard at a Virtual-ECU. In *25th International Symposium on Industrial Electronics (ISIE)*, pages 869–875. IEEE, June 2016.

[FTW15] Martin Fleck, Javier Troya, and Manuel Wimmer. Marrying Search-based Optimization and Model Transformation Technology. In *Proceedings of the First North American Search Based Software Engineering Symposium*. Elsevier, 2015.

[GGPD01] Michael González Harbour, Jose Javier Gutiérrez Garcia, Jose Carlos Palencia Gutiérrez, and Jose Maria Drake Moyano. MAST: Modeling and analysis suite for real time applications. In *Proceedings 13th Euromicro Conference on Real-Time Systems*, pages 125–134. IEEE, 2001.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[GHM+14] Birte Glimm, Ian Horrocks, Boris Motik, Giorgos Stoilos, and Zhe Wang. Hermit: An owl 2 reasoner. *Journal of Automated Reasoning*, 53(3):245–269, Oct 2014.

[GM03] Jürgen Gausemeier and Stefan Moehringer. New Guideline VDI 2206 - A Flexible Procedure Model for the Design of Mechatronic Systems. In *14th International Conference on Engineering Design*, ICED'03, 2003.

[GNNS10] Holger Giese, Stefan Neumann, Oliver Niggemann, and Bernhard Schätz. *Model-Based Integration*, volume 6100 of *Lecture Notes in Computer Science*, pages 17–54. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[Gon] Michael González Harbour. Modeling and Analysis Suite for Real-Time Applications (MAST). https://mast.unican.es.

[GPQ14] Susanne Graf, Roberto Passerone, and Sophie Quinton. *Contract-Based Reasoning for Component Systems with Rich Interactions*, volume 20 of *Embedded Systems*, chapter 8, pages 139–154. Springer New York, New York, NY, 2014.

[GR16] Johannes Gross and Stephan Rudolph. Rule-based spacecraft design space exploration and sensitivity analysis. *Aerospace Science and Technology*, 59:162–171, 2016.

[Gru93] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[GSDA07] Esther Guerra, Daniel Sanz, Paloma Diaz, and Ignacio Aedo. A Transformation-Driven Approach to the Verification of Security Policies in Web Designs. In *Web Engineering*, volume 4607 of *Lecture Notes in Computer Science*, pages 269–284, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[GY14] Frederic Gourves and Andres Yarce. Powertrain of a Hydraulic Hybrid Vehicle, Comprising a Free Wheel and a Planetary Gear Train. Patent WO 2014/199064 A1, 2014.

[Had] David Hadka. MOEA Framework: A Free and Open Source Java Framework for Multiobjective Optimization, http://www.moeaframework.org.

[HCÅ03] Dan Henriksson, Anton Cervin, and Karl-Erik Årzén. TrueTime : Real-time Control System Simulation with MATLAB / Simulink. In *Proceedings of the Nordic MATLAB Conference*, 2003.

[HF11] Florian Hölzl and Martin Feilkas. *AutoFocus 3 - A Scientific Tool Prototype for Model-Based Development of Component-Based, Reactive, Distributed Systems*, volume 6100 of *Lecture Notes in Computer Science*, pages 317–322. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[HHRV11] Ábel Hegedüs, Ákos Horváth, István Ráth, and Dániel Varró. A Model-driven Framework for Guided Design Space Exploration. In *26th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, number ii, pages 173–182. IEEE, 2011.

[Hoa69] Charles Antony Richard Hoare. An Axiomatic Basis for Computer Programming. *Commun. ACM*, 12(10):576–580, October 1969.

[HPSvH03] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: the making of a Web Ontology Language. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(1):7–26, 2003.

[HR04] David Harel and Bernhard Rumpe. Meaningful modeling: what's the semantics of "semantics"? *Computer*, 37(10):64–72, 2004.

[HSB+] Ábel Hegedüs, Rodrigo Rizzi Starr, Márton Búr, Lincoln Nascimento, Róbert Dóczi, Samoel Mirachi, István Ráth, and Ákos Horváth. Massif: Matlab simulink integration framework for eclipse. https://github.com/viatra/massif.

[HST10] Florian Hölzl, Maria Spichkova, and David Trachtenherz. AutoFocus Tool Chain. Technical report, Institut für Informatik der Technischen Universität München, 2010.

[INC07] INCOSE. Systems Engineering Vision 2020. Technical Report INCOSE-TP-2004-004-02, International Council on Systems Engineering (INCOSE), 2007.

[Ise08] Rolf Isermann. Mechatronic systems–Innovative products with embedded control. *Control Engineering Practice*, 16(1):14–29, 2008.

[ISO11] ISO/IEC/IEEE 42010:2011. Systems and software engineering - Architecture description. Technical report, ISO and IEEE, 2011.

[Jac06] Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis*. MIT Press, 2006.

[JČMG12] David Joyner, Ondřej Čertík, Aaron Meurer, and Brian E. Granger. Open Source Computer Algebra Systems: SymPy. *ACM Commun. Comput. Algebra*, 45(3/4):225–234, January 2012.

[JKD+10] Ethan K. Jackson, Eunsuk Kang, Markus Dahlweid, Dirk Seifert, and Thomas Santen. Components, Platforms and Possibilities: Towards Generic Automation for MDA. In *Proceedings of the Tenth ACM International Conference on Embedded Software*, EMSOFT'10, pages 39–48, New York, NY, USA, 2010. ACM.

[JMM08] Bernhard Josko, Qin Ma, and Alexander Metzner. Designing Embedded Systems using Heterogeneous Rich Components. *INCOSE International Symposium*, 18(1):558–576, 2008.

[Joh13] Johnson, Dave, and Speicher, Steve. Open Services for Lifecycle Collaboration Core Specification Version 2.0. Technical report, OSLC Core Specification Workgroup, 2013.

[KL89a] Michael Kifer and Georg Lausen. F-logic: A Higher-order Language for Reasoning About Objects, Inheritance, and Scheme. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, SIGMOD '89, pages 134–146, New York, NY, USA, 1989. ACM.

[KL89b] Michael Kifer and Georg Lausen. F-logic: A Higher-order Language for Reasoning About Objects, Inheritance, and Scheme. *SIGMOD Rec.*, 18(2):134–146, June 1989.

[Kle07] Anneke G. Kleppe. A Language Description is More than a Metamodel. In *Fourth International Workshop on Software Language Engineering*, number 1. megaplanet.org, 2007.

[KLFP02] Raimung Kirner, Roland Lang, Gerald Freiberger, and Peter P. Puschner. Fully automatic worst-case execution time analysis for MATLAB/Simulink models. In *Proceedings 14th Euromicro Conference on Real-Time Systems. Euromicro RTS 2002*, pages 31–40, 2002.

[KLW13] Marouane Kessentini, Philip Langer, and Manuel Wimmer. Searching Models, Modeling Search: On the Synergies of SBSE and MDE. In *Proceedings of the 1st Workshop on Combining Modelling with Search-Based Software Engineering*, pages 51–54, 2013.

[KP10] Aleksandr A. Kerzhener and Christiaan J.J. Paredis. Combining SysML and Model Transformations to Support Systems Engineering Analysis. *Electronic Communications of the EASST - 4th International Workshop on Multi-Paradigm Modeling*, 42, 2010.

[KPP08] Dimitrios S. Kolovos, Richard F. Paige, and Fiona A. C. Polack. The Epsilon Transformation Language. In *Theory and Practice of Model Transformations*, volume 5063 of *Lecture Notes in Computer Science*, pages 46–60, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[Küh06] Thomas Kühne. Matters of (Meta-) Modeling. *Software & Systems Modeling*, 5(4):369–385, 2006.

[LDMH17] Haoxuan Li, Paul De Meulenaere, and Peter Hellinckx. Powerwindow: a Multi-component TACLeBench Benchmark for Timing Analysis. In *Advances on P2P, Parallel, Grid, Cloud and Internet Computing*, pages 779–788, Cham, 2017. Springer International Publishing.

[LFK+14] Heiner Lasi, Peter Fettke, Hans-Georg Kemper, Thomas Feld, and Michael Hoffmann. Industry 4.0. *Business & Information Systems Engineering*, 6(4):239–242, 2014.

[LL73] Chung Laung Liu and James W. Layland. Scheduling Algorithms for Multiprogramming in a Hard- Real-Time Environment. *Journal of the ACM*, 20(1):46–61, 1973.

[LMD+12] Levi Lúcio, Sadaf Mustafiz, Joachim Denil, Bart Meyers, and Hans Vangheluwe. The Formalism Transformation Graph as a Guide to Model Driven Engineering. SOCS-TR-2012.1, McGill University, 2012.

[LMD+13] Levi Lúcio, Sadaf Mustafiz, Joachim Denil, Hans Vangheluwe, and Maris Jukss. FTG+PM: An Integrated Framework for Investigating Model Transformation

Chains. In *SDL 2013: Model-Driven Dependability Engineering*, volume 7916 of *Lecture Notes in Computer Science*, pages 182–202, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[LMMHY16] Wei Li, Ramamurthy Mani, Pieter J Mosterman, and Teresa Hubscher-Younger. Simulating a Multicore Scheduler of Real-time Control Systems in Simulink. In *Proceedings of the Summer Computer Simulation Conference*, SCSC '16, pages 11:1–11:7, San Diego, CA, USA, 2016. Society for Computer Simulation International.

[Man95] John C. Mankins. Technology readiness levels: A White Paper. Technical report, NASA, Washington, DC, 1995.

[Man09] John C. Mankins. Technology readiness assessments: A retrospective. *Acta Astronautica*, 65(9):1216 – 1223, 2009.

[Mat] MathWorks. Simulink Website. https://www.mathworks.com/products/simulink.

[MD14] Matteo Morelli and Marco Di Natale. Control and Scheduling Co-design for a Simulated Quadcopter Robot : A Model-Driven Approach. In *Simulation, Modeling, and Programming for Autonomous Robots*, volume 8810 of *Lecture Notes in Computer Science*, pages 49–61, Cham, 2014. Springer International Publishing.

[MDLV12] Sadaf Mustafiz, Joachim Denil, Lúcio Levi, and Hans Vangheluwe. The FTG+PM Framework for Multi-paradigm Modelling: An Automotive Case Study. In *Proceedings of the 6th International Workshop on Multi-Paradigm Modeling*, MPM'12, pages 13–18, New York, NY, USA, 2012. ACM.

[Mey88] Bertrand Meyer. Eiffel: A language and Environment for Software Engineering. *Journal of Systems and Software*, 8(3):199–246, 1988.

[Mey92] Bertrand Meyer. Applying 'Design by Contract'. *Computer*, 25(10):40–51, October 1992.

[MG06] Tom Mens and Pieter Van Gorp. A Taxonomy of Model Transformation. *Electronic Notes in Theoretical Computer Science*, 152:125 – 142, 2006. Proceedings of the International Workshop on Graph and Model Transformation (GraMoT 2005).

[Mod11] Modeling and Simulation Committee. Final report of the Model Based Engineering (MBE) Subcommittee. Technical report, National Defense Industrial Association (NDIA) Systems Engineering Division, 2011.

[MPE04] Pieter J. Mosterman, Sameer Prabhu, and Tom Erkkinen. An Industrial Embedded Control System Design Process. In *Proceedings of The Inaugural CDEN*

Design Conference (CDEN'04), pages 02B6–1 – 02B6–11, Montreal, Quebec, Canada, 2004.

[MSP+17] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, Amit Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Francesco Muller, Richard P. Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. SymPy: symbolic computing in Python. *PeerJ Computer Science*, 3(e103), 2017.

[MV04] Pieter J. Mosterman and Hans Vangheluwe. Computer Automated Multi-Paradigm Modeling: An Introduction. *SIMULATION*, 80(9):433–450, 2004.

[MZL+13] Mehdi Maasoumy, Qi Zhu, Cheng Li, Forrest Meggers, and Alberto L. Sangiovanni-Vincentelli. Co-design of Control Algorithm and Embedded Platform for Building HVAC Systems. In *Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems*, ICCPS '13, pages 61–70, New York, NY, USA, April 2013. ACM.

[Nad13] Andreas Naderlinger. Multiple Real-Time Semantics on top of Synchronous Block Diagrams. In *Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative M&S Symposium*, DEVS'13, pages 6:1–6:7, San Diego, CA, USA, 2013. Society for Computer Simulation International.

[Nas14] Leonardo Nascimento. Hybrid Air : Gestion énergétique optimisée et impact environnemental de la technologie. Technical Report 2014-CNAM-04, Société des Ingénieurs de l'Automobile (SIA), April 2014.

[NM01] Natalya F. Noy and Deborah L. Mcguinness. Ontology Development 101: A Guide to Creating Your First Ontology. Technical Report KSL-01-05 and SMI-2001-0880, Stanford Knowledge Systems Laboratory and Stanford Medical Informatics, 2001.

[NSK03] Sandeep Neema, Janos Sztipanovits, and Gabor Karsai. Constraint-Based Design-Space Exploration and Model Synthesis. pages 290–305, 2003.

[NSVB+15] Pierluigi Nuzzo, Alberto L. Sangiovanni-Vincentelli, Davide Bresolin, Luca Geretti, and Tiziano Villa. A Platform-Based Design Methodology With Contracts and Related Tools for the Design of Cyber-Physical Systems. *Proceedings of the IEEE*, 103(11):2104–2132, 2015.

[NXO+14] Pierluigi Nuzzo, Huan Xu, Necmiye Ozay, John B. Finn, Alberto L. Sangiovanni-Vincentelli, Richard M. Murray, Alexandre Donzé, and Sanjit A. Seshia. A

Contract-Based Methodology for Aircraft Electric Power System Design. *IEEE Access*, 2:1–25, 2014.

[Obj11] Object Management Group. *The UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) - Specification (version 1.1)*, June 2011.

[PDH+09] Roberto Passerone, Werner Damm, Imene Ben Hafaiedh, Susanne Graf, Alberto Ferrari, Leonardo Mangeruca, Albert Benveniste, Bernhard Josko, Thomas Peikenkamp, Daniela Cancila, Arnaud Cuccuru, Sébastien Gérard, Francois Terrier, and Alberto L. Sangiovanni-Vincentelli. Metamodels in Europe: Languages, tools, and applications. *IEEE Design and Test of Computers*, 26(3):38–53, 2009.

[PG98] Jose Carlos Palencia Gutiérrez and Michael González Harbour. Schedulability analysis for tasks with static and dynamic offsets. *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No.98CB36279)*, pages 26–37, 1998.

[PM04] Sameer M Prabhu and Pieter J Mosterman. Model-Based Design of a Power Window System: Modeling, Simulation, and Validation. In *Proceedings of the Society for Experimental Mechanics IMAC XXII*, 2004.

[Pro05] Marc Provost. Himesis: A Hierarchical Subgraph Matching Kernel for Model Driven Development. Master's thesis, McGill University, Montreal, Quebec, Canada, 2005.

[PTQ+13] Magnus Persson, Martin Törngren, Ahsan Qamar, Jonas Westman, Matthias Biehl, Stavros Tripakis, Hans Vangheluwe, and Joachim Denil. A Characterization of Integrated Multi-View Modeling in the Context of Embedded and Cyber-Physical Systems. In *Proceedings of the International Conference on Embedded Software (EMSOFT)*, pages 1–10, 2013.

[Qam13] Ahsan Qamar. *Model and Dependency Management in Mechatronic Design*. PhD thesis, KTH - Royal Institute of Technology, 2013.

[Raj13] Akshay Rajhans. *Multi-Model Heterogeneous Verification of Cyber-Physical Systems*. PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, 2013.

[RBR+14] Akshay Rajhans, Ajinkya Y. Bhave, Ivan Ruchkin, Bruce Krogh, David Garlan, Andre Platzer, and Bradley Schmerl. Supporting Heterogeneity in Cyber-Physical Systems Architectures. *IEEE Transactions on Automatic Control*, 59(12):3178–3193, December 2014.

[RG14] Franck Roy and Marc Giaonnoni. Powertrain for a Hydraulic Hybrid Vehicle, Control Method and Hybrid Motor Vehicle. Patent WO 2014/174167 A1, 2014.

[RJB04] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual (2nd Edition)*. Pearson Higher Education, 2004.

[RR13] Franck Roy and Vicky Rouss. Chaine de Traction d'un Vehicule Hybride. Patent FR 2 977 533 A1, 2013.

[RRW12] Jan Oliver Ringert, Bernhard Rumpe, and Andreas Wortmann. A Requirements Modeling Language for the Component Behavior of Cyber Physical Robotics Systems. In *Modelling and Quality in Requirements Engineering: Essays Dedicated to Martin Glinz on the Occasion of His 60th Birthday*, pages 133–146. Monsenstein und Vannerdat Münster, 2012.

[RRW13] Jan Oliver Ringert, Bernhard Rumpe, and Andreas Wortmann. From Software Architecture Structure and Behavior Modeling to Implementations of Cyber-Physical Systems. *Software Engineering 2013 Workshopband*, pages 155–170, 2013.

[RSI+18] Ivan Ruchkin, Joshua Sunshine, Grant Iraci, Bradley Schmerl, and David Garlan. IPL: An Integration Property Language for Multi-Model Cyber-Physical Systems. In *22nd International Symposium on Formal Methods (FM2018)*, July 2018.

[Rup10] Nayan B. Ruparelia. Software Development Lifecycle Models. *SIGSOFT Softw. Eng. Notes*, 35(3):8–13, May 2010.

[SBM08] Sagar Sen, Benoit Baudry, and Jean-Marie Mottu. On Combining Multi-formalism Knowledge to Select Models for Model Transformation Testing. In *1st International Conference on Software Testing, Verification, and Validation*, pages 328–337. IEEE, April 2008.

[Sch95] Andy Schürr. Specification of graph translators with triple graph grammars. In *Graph-Theoretic Concepts in Computer Science*, Lecture Notes in Computer Science, pages 151–163, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[SI02] Xiaomeng Su and Lars Ilebrekke. A Comparative Study of Ontology Languages and Tools. In *Advanced Information Systems Engineering*, volume 2348 of *Lecture Notes in Computer Science*, pages 761–765, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[SK03] Shane Sendall and Wojtek Kozaczynski. Model transformation: the heart and soul of model-driven software development. *IEEE Software*, 20(5):42–45, September 2003.

[SK04] Shane Sendall and Jochen Küster. Taming Model Round-Trip Engineering. In *Proceedings of Workshop on Best Practices for Model-Driven Software Development (part of 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications)*, 2004.

[SK10] Tripti Saxena and Gabor Karsai. MDE-Based Approach for Generalizing Design. In *Model Driven Engineering Languages and Systems*, volume 6394 of *Lecture Notes in Computer Science*, pages 46–60, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[SM12] Frederic Gourves Stephane Maurel. Hybrid Vehicle using Hydraulic Power and Associated Managment Method. Patent WO 2012/160284 A1, 2012.

[SMH08] Rob Shearer, Boris Motik, and Ian Horrocks. HermiT: A Highly-Efficient OWL Reasoner. In *Proceedings of the Fifth OWLED Workshop on OWL: Experiences and Directions*, volume 432 of *CEUR Workshop Proceedings*, 2008.

[Smu68] Raymond R. Smullyan. *First-Order Logic*, volume 43 of *MATHE2*. Springer-Verlag Berlin Heidelberg, 1968.

[SPG+07] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):51–53, 2007. Software Engineering and the Semantic Web.

[SPHP02] Bernhard Schätz, Alexander Pretschner, Franz Huber, and Jan Philipps. Model-Based Development of Embedded Systems. In *Advances in Object-Oriented Information Systems*, volume 2426 of *Lecture Notes in Computer Science*, pages 298–311, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[Sta16] Stanford Center for Biomedical Informatics Research. Protégé Website. https://protege.stanford.edu, 2016.

[SV06] Sagar Sen and Hans Vangheluwe. Multi-Domain Physical System Modeling and Control Based on Meta-Modeling and Graph Rewriting. In *IEEE Conference on Computer-Aided Control Systems Design*, pages 69–75. IEEE, October 2006.

[SV10] Eugene Syriani and Hans Vangheluwe. De-/ Re-constructing Model Transformation Languages. *Electronic Communications of the EASST - Ninth International Workshop on Graph Transformation and Visual Modeling Techniques*, 29, 2010.

[SVDP12] Alberto L. Sangiovanni-Vincentelli, Werner Damm, and Roberto Passerone. Taming Dr. Frankenstein: Contract-Based Design for Cyber-Physical Systems. *European Journal of Control*, 18(3):217–238, 2012.

[SVL15] Eugene Syriani, Hans Vangheluwe, and Brian LaShomb. T-core: a framework for custom-built model transformation engines. *Software & Systems Modeling*, 14(3):1215–1243, Jul 2015.

[Sym] SymPy. SymPy Website. http://www.sympy.org.

[Syr11] Eugene Syriani. *A Multi-paradigm Foundation for Model Transformation Language Engineering*. PhD thesis, McGill University, Montreal, Quebec, Canada, Canada, 2011. AAINR77560.

[TC94] Ken Tindell and John Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and Microprogramming*, 40(2-3):117–134, April 1994.

[TH06] Dmitry Tsarkov and Ian Horrocks. FaCT++ Description Logic Reasoner: System Description. In Ulrich Furbach and Natarajan Shankar, editors, *Automated Reasoning*, pages 292–297, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[Thea] The Eclipse Foundation. Eclipse Epsilon. https://www.eclipse.org/epsilon/.

[Theb] The Eclipse Foundation. Eclipse Modeling Framework. https://www.eclipse.org/modeling/emf/.

[Thec] The Eclipse Foundation. Papyrus UML Modeling tool. http://www.eclipse.org/papyrus/index.php.

[Thed] The Eclipse Foundation. Sirius. http://www.eclipse.org/sirius/.

[TQB+14] Martin Törngren, Ahsan Qamar, Matthias Biehl, Frederic Loiret, and Jad Elkhoury. Integrating viewpoints in the development of mechatronic products. *Mechatronics*, 24(7):745–762, 2014.

[Uni] United Nations Economic Commission for Europe. UN Vehicle Regulations - Addenda to the 1958 Agreement: Regulation No. 101. http://www.unece.org/trans/main/wp29/wp29regs101-120.html.

[VDI04] VDI-Fachbereich Produktentwicklung und Mechatronik. *Design methodology for mechatronical systems*. The Association of German Engineers (VDI), Berlin, 2004.

[VSB04] Hans Vangheluwe, Ximeng Sun, and Eric Bodden. Domain-specific Modelling with AToM3. In *The 4th OOPSLA Workshop on Domain-Specific Modelling*, 2004.

[VTVMMV17] Yentl Van Tendeloo, Simon Van Mierlo, Bart Meyers, and Hans Vangheluwe. Concrete Syntax: A Multi-paradigm Modelling Approach. In *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering*, SLE'17, pages 182–193, New York, NY, USA, 2017. ACM.

[WMM+08] Reinhard Wilhelm, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, Per Stenström, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, and Reinhold Heckmann. The Worst-case Execution-time Problem&Mdash;Overview

of Methods and Survey of Tools. *ACM Transactions on Embedded Computing Systems*, 7(3):36:1–36:53, April 2008.

[Wor] World Wide Web Consortium. Semantic Web Website. https://www.w3.org/standards/semanticweb/.

[WPR+11] James R Williams, Simon Poulding, Louis M Rose, Richard F Paige, and Fiona A C Polack. Identifying Desirable Game Character Behaviours through the Application of Evolutionary Algorithms to Model-Driven Engineering Metamodels. In *Proceedings of the Third International Symposium on Search Based Software Engineering*, pages 112–126, 2011.

[ZZDNSV07] Wei Zheng, Qi Zhu, Marco Di Natale, and Alberto L. Sangiovanni-Vincentelli. Definition of Task Allocation and Priority Assignment in Hard Real-Time Distributed Systems. In *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, pages 161–170. IEEE, December 2007.