Multi-view design for cyber-physical systems

Hui Zhao

UNIVERSITÉ CÔTE D'AZUR

ÉCOLE DOCTORALE
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

THÈSE DE DOCTORAT

Multi-Vue Design Pour Cyber-Physical Systems

Hui ZHAO

Université Côte d'Azur, INRIA, I3S

Présentée en vue de l'obtention du garde de docteur en

Informatique

Sciences et Technologies de l'Information et de la Communication (STIC) d'Université Côte d'Azur

Thèse dirigée par: Prof. Frédéric Mallet

Et co-dirigée par: Prof. Ludovic Apvrille

Soutenue le: 22 March 2022

Devant le jury, composé de:

Régine Laleau, Professeur
Université Paris-Est Créteil, Rapporteur

Jean-Michel Bruel, Professeur
Université de Toulouse, Rapporteur

Frédéric Mallet, Professeur
Université Côte d'Azur, Directeur de thèse

Ludovic Apvrille, Professeur
Telecom ParisTech, Co-directeur de thèse

i3S sophia antipolis

Inria

cnrs

LABEX UCN User Centric Network @ SOPHIA

Multi-Vue Design Pour Cyber-Physical Systems

Multi-View Design For Cyber-Physical Systems

Jury:

Rapporteurs

Régine Laleau, Professeur Université Paris-Est Créteil
Jean-Michel Bruel, Professeur Université de Toulouse

Examinateurs

Frédéric Mallet, Professeur Université Côte d'Azur
Ludovic Apvrille, Professeur Telecom ParisTech

Remerciements

Si cette thèse a pu voir le jour, c'est avant tout grâce à l'exemple et au soutien des personnes qui m'ont été proches jusqu'aujourd'hui, et tout particulièrement celles qui m'ont accompagné durant ces trois dernières années. Je leur dédie naturellement le premier moment de cet écrit.

Tout d'abord, j'aimerais remercier chaleureusement mon directeur de thèse, Prof. **Frédéric Mallet** et co-directeur Prof. **Ludovic Apvrille**. Je les remercie pour avoir cru en moi, pour m'avoir soutenu dans les moments difficiles et encouragé dans mes regains de dynamisme. Je leur suis surtout reconnaissant pour leur disponibilité, leur attention, leur investissement et leur optimisme. J'ai beaucoup appris à leur contact et surement gagné en ouverture d'esprit! Un grand merci également à toutes les jury membres.

Je souhaite également remercier profondément Prof. **Robert de Simone** et Mme. **Patricia RIVEILL** pour leur accueil enthousiaste au sein de l'équipe *KAIROS*. Je remercier également Dr. **Letitia Li**, Prof. **Rabea Ameur-Boulifa** et tous les membres de l'équipe *LabSoC* et *KAIROS*.

Je remercie particulièrement Dr. **Feng Yu** et M. **Emmanuel Cinneri** qui ont été gentils avec moi et m'ont invité à manger chez eux, c'était mon honneur de les rencontrer. J'ai aussi remercie à mes amis, Dr. **Zihao Wang**, Dr. **Chuan Xu** et les autres!

Finalement, merci à ma famille pour m'avoir porté jusqu'ici. A mes parents, j'arrête de promettre que c'est le dernier examen que je passe. Ça fait 10 ans que je vous fais le coup, je sais ... Merci d'avoir toujours cru en moi et de m'avoir donné les moyens de réussir. je remerecie encore à ma mère **Tang Qiaoqing** pour m'avoir appris à regarder et, à travers un regard sans tache, à voir. À mon père **Zhao Jianguo** pour m'avoir appris à agir et à transformer ce que je voyais. Je remercie à ma femme bien-aimée Dr. **Min Ju**. Je n'ai pas de mots assez forts pour vous remercier.

Acknowledgement

First of all, I would like to warmly thank my thesis director, Prof. Frédéric Mallet and co-director Prof. Ludovic Apvrille. I thank them for having believed in me, for having supported me in difficult times and encouraged in my revival of dynamism. I am especially grateful for their availability, attention, investment and optimism. I learned a lot from them and certainly gained an open mind! I would like to thank all of the jury members.

I would also like to express my gratitude towards the committee members, for their time spent on reading and their constructive comments that enrich this thesis.

I would also like to thank deeply Prof. Robert de Simone and Mme. Patricia RIVEILL for their enthusiastic welcome to the *KAIROS* team, as well as Prof. Marie-Agnes PERALDI-FRATI, Prof. Eric Madelaine and Prof. Julien Deantoni. I would also like to thank all the members of the *LabSoC* team who welcomed and helped me, Dr. Letitia Li, Prof. Rabea Ameur-Boulifa and the others!

I especially thank Dr. Feng Yu et M. Emmanuel Cinneri who were nice to me and invited me to eat at home, it was my honor to meet them. I also thank my friends of INRIA, Dr. Zihao Wang, Dr. Chuan Xu and the others!

Finally, thank you to my family for bringing me here. To my parents, I stop promising that this is my last exam. I've been doing it for 10 years, I know…Thank you for always believing in me and for giving me the means to succeed. I still thank my mother Tang Qiaoqing for teaching me to look and, through a spotless look, to see. To my father Zhao Jianguo for teaching me to act and transform what I saw. I especially thank my beloved, Dr. Min JU. I do not have words strong enough to thank you.

致谢

这篇论文的顺利完成，要感谢所有给过我支持和帮助的可爱的人。在写论文的的第一刻，我要向他们特别致谢，真诚的说一句『感谢有你们和我在一起』。

我首先要感谢 Frédéric Mallet 教授和 Ludovic Apvrille 教授，感谢他们对我的指导，并让我领悟到严谨科学态度和执着的科研精神。感谢他们对我的信任，并在我最艰难的时刻支持了我，并鼓励我重新找到方向。

我还要感谢 *KAIROS* 团队和 *LabSoC* 实验室全体成员。感谢在和他们相处的三年里所有的欢声笑语，他们每一个人的笑容都将永久的镌刻在我的记忆中。特别是 Robert de Simone，他就像一位慈祥的长者，不仅在学术上给予我指导，还时刻用他的快乐感染着我和团队中的每一个人。我还要感谢三年中和我一起的朋友和同事们，回想起三年中的每一天，都有他们的陪伴，岁月也显得不那么难过。这里特别要提到 Feng Yu 博士和 Emmanuel Cinneri 先生，谢谢他们给予我的照顾和关心。

最后也是最重要的，我要深深地感谢我的父亲**赵建国**先生，母亲**唐巧琴**女士，是他们给了我来到这个世界的机会，哺育我长大，让我接受良好的教育，并在成长的过程中，包容我的任性与不羁。与他们在一起的时刻我才能感受到人生最大幸福与温暖。在中国人的传统情感中很难启齿说出爱的情感，特别是与父母之间，在此我要郑重的向他们说，『我爱你们，我的爸爸妈妈！』。我还要特别感谢我的至爱，**巨敏**博士，感谢你在我最困难的论文写作和答辩准备期间一直支持与陪伴。在以后的人生中我们松萝共倚，携手同行！

谨以此将我对他们所有感谢和爱铭刻在论文里，永久保存！

Abstract

Cyber-Physical Systems (CPSs) are large-scale interconnected systems of heterogeneous components that integrate computational parts with physical processes. Unlike pure digital embedded systems, CPS combine both digital and physical parts, and integrate the (very uncertain) environment as part of the model in a closed-loop retroaction fashion. Therefore, while traditional reactive embedded systems are frequently modeled as pure discrete systems, CPS also come with continuous models of physical phenomena and are therefore considered heterogeneous. The variety of subsystems demand to gather different expertises and put together different models and tools going beyond the typical border of software engineering in a field called systems engineering.

Furthermore, each part of CPS consists of a set of independent sub-systems. Different engineer teams develop sub-system independently, and even some sub-systems come from different sub-contractors. Keeping the consistency among those independent (sub)systems at system development phase is a challenge. During the whole development life-cycle, the system designer should consider a wide variety of domains and involve many different experts to handle domain-specific problems. For example, in some safety-critical real-time systems, the system engineers focus on functional issues at first. They also have to consider some non-functional aspects that could affect the system reliability and performances, such as security and timing. Potential attackers can use system vulnerabilities and flaws to hijack the system. To deal with the complexity of CPS, we need an integrated framework able to capture all the different views (models) of such complex systems in a consistent way.

In Model-Based System Engineering (MBSE), CPS designers bring together a wide variety of experts who use different sets of languages and tools, while keeping the model at the center of the process. This variety makes it difficult to combine and

integrate all these models and artefacts. The consistency problem becomes an essential issue. Ensuring consistency is one of the main attention point of this work.

In this thesis, we advocate for a model-centered approach that can combine heterogeneous artefacts (called views) into a sound and consistent system model. Rather than trying to build an universal modeling language to capture all aspects of systems, we elaborate on subsets of existing languages to keep only what is needed to conduct the required analyses.

We take a case study and verify this case study with Capella, an open-source solution used by major integrating companies of critical-systems. Capella covers a wide design flow from functional analysis to component deployment. Even though Capella is already quite expressive, it does not fit all design requirements to perform tasks such as security and safety analysis, or scheduling. It relies on external plugins or some external Domain-Specific Modeling Language (DSML) for that purpose. We explore a solution to combine it with another tool, SysML-Sec, which is an extension of SysML dedicated to security and safety analysis. We extract only the required subsets to conduct functional analysis with Capella, and security analysis with SysML-Sec. We do the same exercise with Architecture Analysis & Design Language (AADL) to do schedulability analysis and show our language is generic enough to extract subsets of languages and combine them to build views for different experts. Moreover, the global model also maintains a global consistency between the different views.

Keywords: Multi-view, Cyber-physical systems, Modeling language, SysML, MDE, AADL, Security and safety

Résumé

Les systèmes cyber-physiques (CPS) sont des systèmes distribués qui combinent des parties numériques avec des sous-systèmes physiques. Contrairement aux systèmes embarqués réactifs traditionnels qui gardent le non-déterminisme en dehors du modèle, les CPS prennent en compte l'incertitude de l'environnement dans un système en boucle fermée dans lequel les systèmes de contrôle impactent l'environnement (par exemple la température) ce qui a son tour produit de nouvelles réactions sur le contrôle (par exemple dans un bâtiment intelligent). Cette hétérogénéité dans les phénomènes modélisés (sytèmes numériques discrets, phénomènes physiques continus) fait appel à différentes expertises qui vont bien au delà des compétences d'un ingénieur logiciel et qui relèvent de l'ingénierie système.

Les concepteurs de CPS doivent prendre en compte de nombreux facteurs en raison de la complexité et de la diversité des systèmes. Ils impliquent de nombreux experts pour gérer les problèmes spécifiques à chaque domaine. Ils s'appuient sur différents modèles et langages, chacun adapté à un sous-domaine particulier, ce qui conduit à des problèmes de cohérence entre ces modèles et ces langages. Comment mettre ensemble ces modèles est le point d'étude central de cette thèse.

Nous explorons une approche basée sur des modèles en composant plusieurs artefacts hétérogènes (vues) dans un modèle intégré du système cohérent. Plutôt que d'essayer de créer un langage de modélisation universel pour capturer tous les aspects, nous rassemblons de petits sous-ensembles de langages de modélisation pour nous concentrer sur des capacités d'analyse spécifiques. Nous avons proposé une approche basée sur un modèle et un langage suffisamment générique pour extraire des sous-ensembles et les combiner pour créer des vues pour les différents experts. Le modèle central maintient également une cohérence globale entre les différentes vues.

Nous prenons une étude du cas de Capella, une solution open-source utilisée par les grandes entreprises d'intégration, qui fournit un large support allant de l'analyse fonctionnelle des exigences au déploiement des composants logiciels ou matériels. Même si Capella est déjà assez complet pour l'analyse fonctionnelle, il ne répond pas à toutes les exigences de conception telles que l'analyse de la sécurité et de la sûreté, ou la planification. Nous montrons comment le combiner avec d'autres langages dédiés permet d'augmenter l'expressivité globale et la capacité d'analyse. Nous le combinons, dans un premier temps à SysML-Sec, une extension de SysML dédiée à l'analyse de la sécurité et de la sûreté. Nous extrayons les sous-ensembles des deux langages pour construire une vue cohérente et puis, effectuons une analyse fonctionnelle avec Capella et une analyse de sécurité avec SysML-Sec. Nous utilisons le même technique avec AADL pour effectuer une analyse d'ordonnançabilité. Ces deux cas d'études montrent que notre langage est suffisamment générique pour extraire des sous-ensembles de langages et maintenir leur cohérence.

Mots clés: Multi-vue, Cyber-physical systems, Langage de modélisation, SysML, MDE, AADL, sécurité et sûreté

Publications

1. Zhao, Hui and Apvrille, Ludovic and Mallet, Frédéric (2017). **"Multi-View Design for Cyber-Physical Systems"**. In: *Proceedings of PhD Symposium at 13th International Conference on ICT in Education, Research, and Industrial Applications*, Ukraine, p. 22–28.

2. Zhao, Hui and Apvrille, Ludovic and Mallet, Frédéric (2019). **"Meta-models Combination for Reusing Verification Techniques"**. In: *Proceedings of 7th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, Czech Republic, p. 39–50.

3. Zhao, Hui and Mallet, Frédéric and Apvrille, Ludovic (2019). **"A language-based multi-view approach for combining functional and security models"**. *Accepted by Asia-Pacific Software Engineering Conference (APSEC)*, Malaysia, p. 426–433.

4. Zhao, Hui and Apvrille, Ludovic and Mallet, Frédéric (2020). **"A Model-Based Combination Language for Scheduling Verification"**. *Accepted by book Communications in Computer and Information Science (CCIS)*, Vol. 1161, Springer, p. 27–49.

Table of contents

List of Figures

List of Tables

1 Introduction

> *Success is not final; failure is not fatal: It is the courage to continue that counts.*
>
> **— Winston S. Churchill**
> (British statesman, army officer, and writer)

Digital systems are pervasive and are present in many aspects of our lives (from online purchasing and payment to high-speed train and aircraft control systems, as well as within autonomous cars, smart building or smart cities). While some systems are designed and deployed independently of each other, others are devised for being integrated. System engineering attempts to capture and model a set of heterogeneous sub-systems working together in a bid to understand, predict and then improve the global behavior emerging from their multiple interactions. It goes beyond the considerations of pure software engineering as it includes some description of all systems whether digital or not. When it includes a description of physical phenomenon (e.g., law of motion, thermodynamics) then it is called Cyber-Physical systems to emphasize the mix of both discrete and continuous, cyber and physical sub-systems. Simply capturing the various models and keeping the consistency between them is a challenge in itself that is addressed in this work.

1.1 A brief introduction to Cyber-Physical Systems

The term Cyber-Physical System (CPS) emerged around 2006 when it was coined by Helen Gill at the National Science Foundation in the United States [1]. CPS concerns go beyond the one of embedded control systems [2] as they bring together digital computational systems such as embedded systems and communication networks (called cyber systems), with surrounding physical processes (e.g., chemical, bio-medical, civil, and electromechanical systems). Computations are meant to control and monitor the physical environment with feedback loops. Physical control process affect computations and vice versa [3, 4, 5].

Applications of CPS are affecting everyone's life. Looking around people's daily life, medical devices assist surgeries, intelligent traffic control systems mitigate traffic jams and save energy, high-speed train reduces the distance between cities and makes the Metropolitan Region within 1 hour. A pretty convincing case is the Rough-Terrain Quadruped Robot – BigDog, which was made by Boston Dynamic. It is equipped with four legs for movement, allowing it to move across surfaces. Instead of wheels or treads, the legs contain a variety of sensors which are controlled by high-performance embedded systems, including joint position, ground contact, laser gyroscope, and stereo vision system [6, 7]. It goes beyond standard robotic systems when integrated with the system infrastructure of a smart cities, in constant interactions with other devices, whether digital or not, whether fully autonomous or not.

It is easy to envision new capabilities. A better-embedded intelligence automobile improves safety and efficiency for transportation systems. Networked building systems significantly improve energy efficiency, reduce greenhouse gas emissions and our dependence on fossil fuels by better controlling household electrical appliances, as well as air-conditioners and lighting systems. Networked autonomous vehicles could dramatically enhance our automobile's effectiveness and offer substantially more effective disaster recovery techniques [8].

However, the CPS have been held to a higher reliability and predictability standard than general-purpose computing [2]. In a general-purpose embedded system, time

is considered as a factor to evaluate the performance of the system. Taking longer time to perform tasks is not a critical issue. It is merely less convenient and less valuable, yet in the CPS, timing is an issue of vital importance, as the system must react too late or too early as what the environment, often uncertain, expects.

For instance, the high-speed train system must be a high confidence transportation system. The so-called signaling system is a safety-critical and real-time system. It is an essential system to ensure and assist the automatic operation of high-speed trains. In the scenario of automatic train operation, a train moves into a speed-restricted zone. The signaling system should send a set of commands to the locomotive subsystem (including mechanical components) for reducing the speed to an expected safe interval. The commands must be received and effected in an expected time bound. In this case, sending a command to slow the train down is a function. The time of execution is a critical factor in evaluating the system's risks. Any delay (in both computations or mechanism) may lead to safety problems and possibly accidents.

In addition, suppose attackers try to hack a system by using the system's flaws and vulnerabilities. Attacking may lead to the system's jitters and delays, then the train's speed probably has not been reduced to a speed value as slow as expected. It would also lead to accidents (e.g., derailment or collision) which can injure people, even cause death. Therefore, the system designers have to consider safety and security and timing properties throughout the whole design of the system.

The typical characteristics and challenges of CPS are well-known [2, 4], yet global solutions do not exist:

heterogeneity, in the sense that they capture the different aspects and views relay on various models, discrete or continuous, state-based or flow-related, digital or physical.

platform-aware and resource-constrained, embedded system design depends a lot on the execution platform on which the system should execute, and thus the program depends on various non-functional properties imposed by the platform.

time-sensitive and often safety-critical, the time of execution is a key factor of the system. The programs and data are allocated to computing resources and data memory, and there is a distance between them. This spatial distribution requires performing the temporal scheduling of the execution of programs and loading data to computing resources. The logical concurrency comes from hardware and data, and controls dependencies of the applications.

widely distributed with heterogeneous interconnects, simple embedded systems rely on homogenous interconnects. Compared to traditional embedded systems, CPS usually contain multiple interconnected embedded subsystems, some of which are computing devices and some are physical devices. This requires heterogeneous interconnects.

Since CPS development is extremely complex, the design of CPS requires modeling methods and frameworks to describe each part of the system. Logical imperative programs and discrete event models are used to describe the cyber part e.g., deterministic modeling frameworks Ptolemy II[1]. The physical environment is often understood by models of physics and motions that can be can be characterized as Partial Differential Equation (PDE) [9].

1.2 Motivation and objective

To deal with the heterogeneity and complexity of CPS, one needs an integrated framework able to capture all the different views of such complex systems in a consistent way.

The aim of our research is to propose and study a technique which can build a bridge between different models (in an horizontal way) while building large system, but also among inner models at different system levels (in a vertical way). Based on the idea of refinement, designer can use systematic approach to construct models gradually and to facilitate a systematic reasoning method by means of proofs. The vertical axis are different abstraction levels for one single system; lower levels are refined versions from the above levels and must conform to the above levels. Re-

[1] http://ptolemy.org/books/Systems

finement mechanism usually contains formalized constrains to maintain the consistency of the system [10]. The *B* method [11, 12] is a frequently mentioned formal method that supports whole life cycle of the development, throughout requirement, specification, refinement and implementation [13, 14, 15].

Complex systems should be constructed to be correct according to the standards of engineering. The discrete technique decomposition allows designer modeling the complex systems as a set of subsystems [10]. The horizontal axis are different systems which have compact interactions among system parties (components), i.e., a set of components whose interaction semantics are usually informal, and the heterogeneous components that are expected to satisfy some of the system properties. By leveraging some of the properties obtained on the component level, we hope to offer useful mechanisms for the integration stage: verify that components satisfy system requirements, allow substitution of components and exploration of alternative costs with regards to both their functional and non-functional properties. Meanwhile, we intend to conduct execution, verification, and validation activities at the system level. The two dimensions are shown in Fig.1.1.

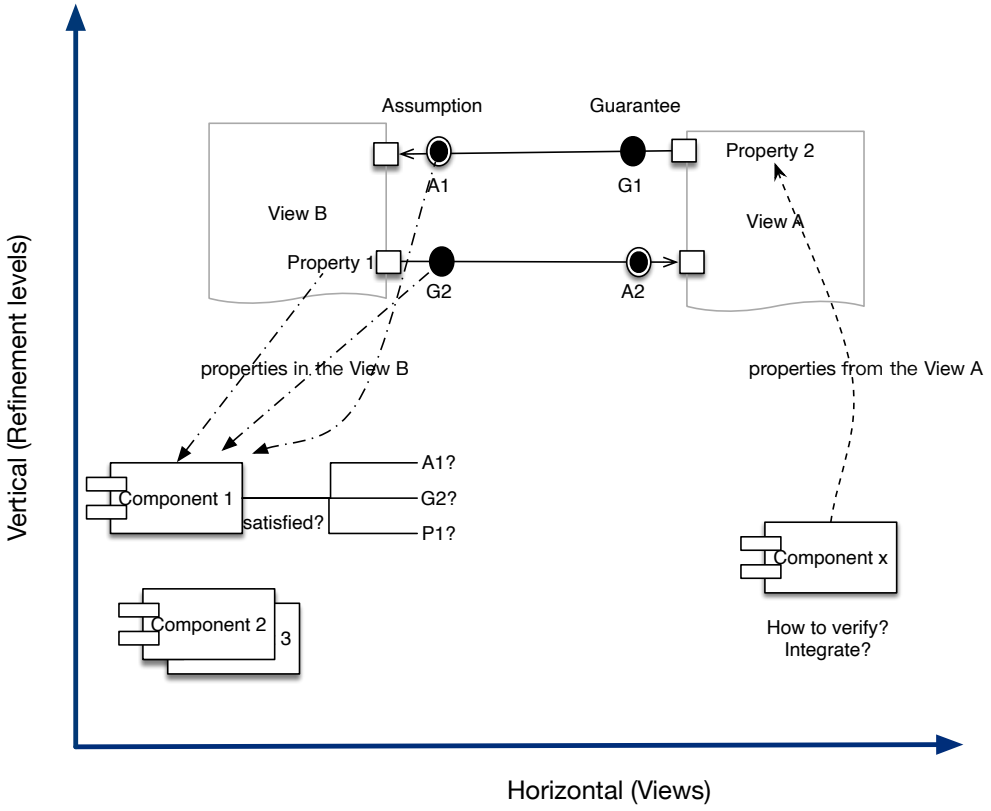


Fig. 1.1 Horizontal and vertical system views

Lots of scientists have contributed to this field for some years and made remarkable achievements. That is inspired by existing Model-Driven Engineering (MDE) methodologies and approaches (Arcadia/Capella). Existing MDE frameworks, e.g., Eclipse Modeling Tool [2], integrate various analysis techniques supporting the engineering process within a common environment. The Eclipse Modeling Framework (EMF) is used to capture meta-models as a high-level abstract model. Moreover, we chose TTool [3] as a target for security and safety design purposes. We rely on TTool to model the security and safety properties and perform formal proofs and simulation. TTool is a free and open-source support toolkit which supports UML profiles such as SysML-Sec [16]. TTool offers diagrams for capturing system requirements, modeling software/hardware partitioning, and performing performance/security/safety proofs, supporting Model Transformation (MT) techniques. For security and safety proofs, TTool relies on *ProVerif* and *UPPAAL*, respectively. For the purpose of furthermore validating that our language-based solution is able to be compatible with other modeling method. We also practice with Architecture Analysis & Design Language (AADL) and its support environment (such as OSATE) to verify scheduling design.

Furthermore, by contrast with the existing model combination technology, we pursue a generic enough method which is easy to use. Based on this method, we provide a friendly tool that can afford the design engineer full facilities for using this method. After reviewing and using existing tools, we hope that our tool will have less learning time and higher execution efficiency.

1.3 Problem statement

As presented in this thesis, the design of CPS spans several domains of engineering. Each domain relies on specific expertise (mechanisms, aerodynamics, software, security, hardware, power), tools, and models. Integration and putting together a variety of properties and models in a semantically correct way is a significant issue for CPS design and modeling frameworks.

[2] http://www.eclipse.org
[3] http://ttool.telecom-paristech.fr/index.html

My thesis was partially sponsored by the CLARITY project [17, 18]. The CLARITY project is based on the MBSE solution Capella and its extensions. It aims to construct an ecosystem for modeling a large system and helps the engineer to design a system model.

Capella is a key technology to reduce system design complexity. It provides methodological support and guidance for systems engineers. Capella is a disruptive technology of MBSE [19, 20, 21]. My work is mainly based on Capella. Capella is further discussed in the background and technical contribution chapters.

Although Capella is a powerful modeling framework, it is still somewhat limited. Capella could not work together with other tools to design security models, as well as scheduling models. In this thesis, we advocate for a language-base modeling approach which can combine heterogeneous artefacts (called views) into a sound and consistent system model. Rather than trying to build a universal language to combine all the expressiveness of all the sublanguages, we elaborate on subsets of existing languages to keep only what is needed to conduct the required analysis.

1.4 Contributions of this thesis

I devote my efforts to combine a variety of models for CPS design and improve the productivity for modeling CPS. I proposed a modeling language used to establish a set of relationships among (meta) models. A support tool serves as a parser for languages. This tool can manipulate the (meta-) models at the abstract level to assemble an produce a new model to enable further designs. It is also able to evaluate the properties to determine whether the models satisfy the requirements or not. We elaborate later on how to combine (meta-) models. We also demonstrate the combination modeling language applications with two scenarios, the scheduling and security & safety models (views). More specifically, the contributions of my thesis are as follows:

1.4.1 Combination Modeling Language (CML)

The proposed Combination Modeling Language is a dedicated (meta) language to extend and enrich one DSML capabilities by combining with other DSMLs. By using this language, system experts can explicitly capture a set of scenarios and co-work with different domain experts at the language level. To do that, the syntax and semantics must be strictly defined, respectively.

For syntax part, Extended Backus Naur Form (EBNF) is used to define context-free grammar formally. For the semantics part, the combination pattern is used to specify different combination relationships. Specific operators are provided to build up Transformation Rule Expression (TRE). A set of TREs defines a Transformation Rule Library (TRL) which specifies how to combine different (meta) model elements. Once the TRL is completed, it can be parsed by an automatic tool.

This CML enables several modeling views which can be considered and designed at the same abstract level, and it allows that different modeling frameworks to reuse each other's artefacts. It largely augments the system design efficiency, reduces the complexity, and ensures the consistency of the system.

Support tool for CML. According to TRL, the integration engineers can take some parts of two (meta) models and combine them together, and then export a new (meta) model. The manual combination of models is error-prone, and wastes a lot of time, because a TRL may include many TREs. As each TRE involves different elements with a set of parameters. The integration engineers have to pay much more attention to building a new model according to each TRE. Any mistake can lead to unpredictable results, and it is difficult to detect those mistakes.

Instead of doing this manually, a support tool is designed to accomplish the process automatically. It can ensure the correctness of generating a new combined (meta) model and export the new (meta) model in an easy way. A Graphical User Interface (GUI) allows integration engineers to import two original (meta) models, respectively. The relations and elements of the model are shown in the original model areas. The integration engineers write the TRL to indicating how to transfer the elements of models. Once the TRL gets ready, then the tool runs in accordance

to each TRE. Finally, the new combined (meta) model and internal relationships are built. All of those manipulations are with graphic interface support built-in, and transforming processes are executed automatically. By using this tool, the correctness of the combination can be ensured to a good level.

1.4.2 Combining AADL for scheduling verification

As we mentioned, one of the CPS characteristics is time-sensitiveness. The time of execution is a critical factor of the system. The data and programs are allocated to computing resources according to the architecture of system. AADL is a modeling language dedicated to describing the architecture, and it is also able to conduct a scheduling verification. In order to avoid redundancy, we require to reuse functional models with architecture models for verifying system properties. To this end, we rely on a new modeling language, CML, a DSML that combines two modeling languages by defining how to link two (sub-) metamodels. Using the proposed language and approach, two models m_1 and m_2 of two different modeling languages, respectively: m_2 can automatically be augmented with some information of m_1 to perform verification on the enriched model (e.g., scheduling, timing, safety), and then verification results can be traced to m_1.

To validate this contribution, SysML and AADL are selected as two target languages, and their support environments (tool) Capella/Arcadia and OSATE2 [4] are used to show the design of the example system.

1.4.3 Safety and security design

The Safety& Security issues take a vital role in the CPS, especially in some industrial critical systems, such as automotive and aeronautic areas. While, the Safety& Security may affect other aspects or be inflected by other aspects, for example, functional aspect and performance aspect. Hence, the Safety& Security issues must be considered with others aspects (views).

[4] http://osate.org/index.html

In practice, Safety& Security design is very complex to link with the functional model as it includes a variety of contents and involves a lot of approaches. In order to accomplish all the functions provided by security and safety, people need to consider each aspect of the system independently. Therefore, we conduct demonstration to guide the integration through broken-down the elements and relevant properties. We detail several examples of TREs.

1.5 Organization of this thesis

This thesis starts with an introductory chapter that presents essential concepts of this thesis and explains the motivation and objective of our work, and briefly summarizes the technical contributions during my research life. At the end of this chapter, we illustrate the plan of the whole thesis.

After a brief introduction to the scientific context of the research work. In the chapter background 2, we introduce the CPS concepts and relevant applications. Then, we present several modeling technologies, including modeling languages and frameworks. We present related works, regarding methodologies and toolkits related to design of CPS, Multi-View (MV) design, DSML/Domain-Specific Language (DSL), MT and model weaving techniques in the chapter state-of-the-art 3.

Next, we systemically present the detail of the technical contributions. Chapter 4 introduces the main contribution, a DSML for combining different (meta) models smoothly. This modeling language can coordinate different modeling phases and (meta) models with a multi-view approach. We also present the overall objectives and systematic syntax and semantics of Combination Modeling Language (CML). Then, we show the model fusion tool which supports environment of our proposed modeling language in the tool chapter. This tool can play two (meta) models with the rule library. We also hint-light the strengths of this tool in contrast with other ones.

Chapter 6 presents my contributions of practice to enable Capella co-working with AADL to design a timing-critical system, applied to perform a unified verification for tasks scheduling. Chapter 7 presents the methodology of combining different

domains using proposed operators. Moreover, we illustrate analysis of safety and security properties with the industrial functional design framework Capella and TTool framework.

Chapter 8 concludes and discusses potential future work.

2 Context: CPS and model-based design

> *Your time is limited, so don't waste it living someone else's life. Don't be trapped by dogma – which is living with the results of other people's thinking.*
>
> **— Steve Jobs**
> (CEO, and co-founder of Apple Inc.)

This chapter introduces the technical background and main concepts used in the remainders of the thesis. We discuss the challenges of the CPS and some related terms which are mentioned with CPS frequently. Then, we present Model-Driven Engineering, and the main principles of modeling approaches for CPS design in section 2.2. We also present several modeling languages and their supporting frameworks in section 2.3. Especially, we put a particular emphasis on Domain-Specific Languages and associated workbenches. Along the presentation of these concepts, we draw the boundaries of our contributions.

2.1 Introduction

Cyber-Physical Systems are concerned with collaborative and interactive activities between cyber and physical components through sensing and actuation. Recent new manufacturing and upward trend of smart things (such as smart cities, autonomous cars) have paved the way for a massive deployment of CPS. Especially, wide requirements of the new generation of manufacturing industry boost CPS development and applications, the information from all related perspectives is closely monitored and synchronized between the physical factory level and the cyber computational space. Networked machines are able to perform more efficiently, collaboratively, and resiliently [22]. This evolution trend also has a significant impact on development issues to adapt and satisfy new requirements. With recent developments that have resulted in higher availability and affordability of sensors, data acquisition systems, and computer networks. The competitive nature of today's industry forces more factories to move towards implementing high-tech methodologies. Consequently, the ever-growing use of sensors, networked machines, and embedded control systems has resulted in the continuously increasing complexity, which is known as the challenge of consistency among related systems.

Furthermore, integrating functional models with non-functional models would bring more applications to improve industrial processes and enhance people's life quality in current industrial practices. For example, the safety and security models include some key properties of the system that are used to help engineers enhance the system robustness. In this chapter, we involve and introduce some of CPS related terms and their technical background, as well as modeling methods of CPS.

2.1.1 CPS and IoT

CPS are frequently mentioned along with the popular terms Internet-of-Things (IoT) and Industry 4.0. The new industrial revolution is known as the fourth industrial revolution or Industry 4.0 [22, 23, 24]. Multidisciplinary areas, such as CPS and mechatronics, Internet-of-Things, huge sensor network TSensors (Trillion sensors) [25, 1] and the cloud computing are playing essential roles in this industrial revolution.

CPS are considered as a global network infrastructure, and it can provide the foundations for integrating the physical manufacturing facilities and machines with the cyber world of Internet and computer applications into single exploited and explored system that rely on sensory, communication, networking, and information processing technologies [26, 27]. Cost-saving and real-time deployment are the two dominant features of CPS, and these two features are also the major drivers of Industry 4.0 [28]. The term "CPS" does not only refer to either implementation approaches (e.g., the "Internet" in Internet-of-Things) or particular applications (e.g., "Industry" in Industry 4.0), but rather CPS focus on the fundamental scientific problems of combining the traditional engineering of the cyber and the physical worlds.

Industry 4.0 is more used to describe a production-oriented CPS that integrates production facilities, warehousing systems, logistics, and even social requirements to establish the global value creation networks [29]. It gives a vision of a technology that deeply connects the physical world with the information world.

The IoT is based on connections between physical assets and data. The connections are made possible by the secure implementation of computer networks, internet, and communication protocols. This communication is based on typical internet protocols or dedicated narrowband, low-power network technologies such as NB-IoT, Zigbee [30].

The similarities of IoT and CPS definitions in using networking, computational system, and sensors might lead to wonder whether these two terms are different definitions of the same concept. However, there are similarities, CPS are not the same thing as IoT.

In the physical world, the machines are connected, and the data would share among the machine network. In the cyber world, the digitalized object is abstracted to interact with the human through Human-Machine Interface. In fact, the digitalized object in the cyber world is highly similar to the machine in the physical world. Thus we call them Digital Twins (DTs). The digitalized object is shown as the data model or other models (function, behavior), which are images of relative physical objects. Digital twin is one of the most promising enabling technologies for realizing smart manufacturing and Industry 4.0. Digital twins are characterized by the

seamless integration between the cyber and physical world [22, 31]. Fig 2.1 shows the relationship of IoT, CPS and DT. The CPS are characterized by a physical asset, a software model that mimics the behavior of the physical asset.

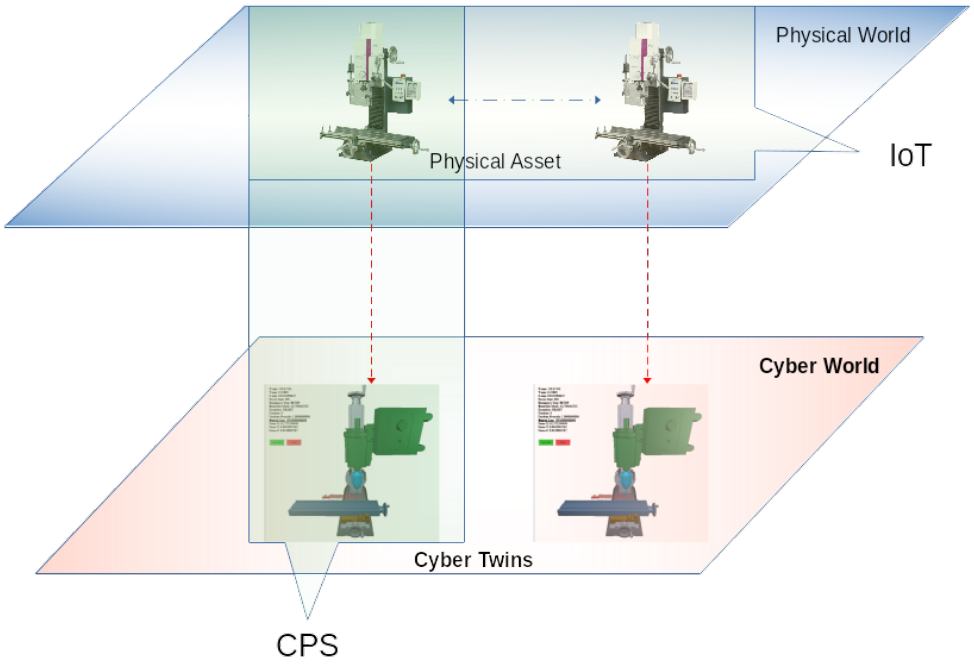


Fig. 2.1 Software model mimics the behavior of physical asset

2.1.2 Industrial applications

Due to unique features, CPS and its design approach have been used in many domains. In what follows, we enumerated some applications of CPS in the table below (Tab. 2.1).

Let us look at a typical application of manufacturing. A modern factory equips a digital machine with sensors. The machine units are in different geographical locations. Sensors measured their status, such as pressure, vibration, and temperature. The CPS also collects signals such as feed rate and size of the material. On-site industrial computers (upper monitor) perform the preliminary data-to-information conversion and provide a low-level interactive interface. More complex adaptive use-based health and data analysis methods assess the performances and make prediction. Analysis results appear through Human-Machine Interface (HMI) applica-

Applications	References
Aeronautic systems	[32, 33, 34, 35]
Automotive systems	[36, 8]
Public transportation systems (e.g., Railway)	[37, 38]
Manufacturing systems	[31, 39, 40, 41]
Medical devices	[42, 43, 44]
Military systems	[5]
Assisted living	[24]
Intelligence power generation and distribution (so-called smart grid)	[45, 34]
Heating, Ventilation and Air Conditioning (HVAC)	[2, 46]
Physical security (access control and monitoring)	[42, 47, 48, 1]
Asset management and distributed robotics (telepresence, telemedicine)	[49, 50]

Tab. 2.1 Applications of Cyber-Physical Systems

tions, and the user also can be in the loop and send a control command to operate the machine unit at any time, so-called human-in-loop decision (see Fig. 2.2).

This case also reveals many technologies, such as data analysis, sensor networks, communication protocols, and cyber-security [39, 40, 41]. All of those systems are considered as CPS which are designed with model-based approaches. Using model-based approaches can also to test and verify systems before industrial applications [51, 52].

2.1.3 Challenges for CPS

CPS are considered as a new theory that explicitly addresses the interaction between physical and cyber subsystems. This scientific foundation must provide the basis for an overall understanding of the system development, design, evolution of CPS, as well as qualification (certification). It must integrate models of computing and communication systems, sensing networks, control of physical systems, and the interactions between humans and CPS. We then introduce some research challenges for CPS:

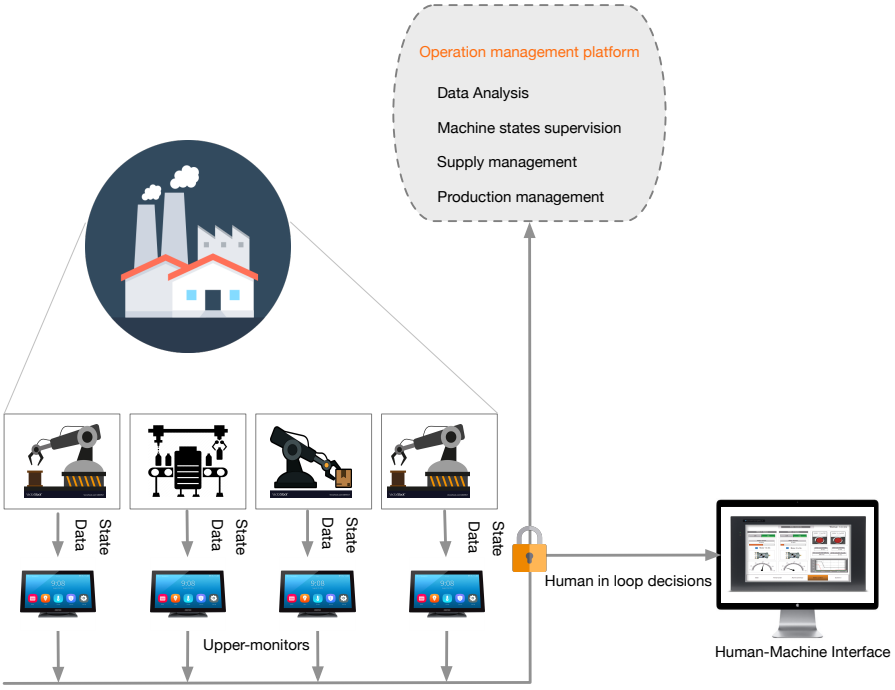


Fig. 2.2 Operators and managers will interact with CPS through a variety of interfaces.

- **Safety, Security and Robustness of CPS:** Uncertainty in the environment [53], system flaws [54], and errors in physical devices make a critical challenge to ensure overall system robustness, security, and safety. Security and safety increase the complexity of CPS design [48], i.e., the engineer must consider security and safety countermeasures and integrate them into functional design.

- **Real-Time Embedded Systems:** Embedded systems must be able to respond to the requests in time with limited resources, for example, real-time resource allocation, data aggregation, decision making. All the task execution times have to be estimated and simulated, and the scheduling has to be arranged in a proper way.

- **Control and Hybrid Systems:** CPS process must merge discrete and continuous variables for feedback control. This process must be applied to hierarchies involving asynchronous dynamics.

- **Architectural Consistency:** CPS architecture must be consistent across the whole system. Architectures capture a variety of physical information and software parameters.

- **Sensor and Mobile Networks:** The need for increasing system autonomy in practice requires self-organizing (and re-organizing) mobile networks and ad-hoc CPS networks. It is essential to collect the knowledge from the vast amount of raw data.

- **Model-based Development of CPS:** Models are used today to generate and test software implementations of control logic. Abstractions that cover the whole CPS design space must be developed, modified, and integrated. Communications, computing, and physical dynamics must be abstracted and modeled at different levels of scale and time granularity.

- **Verification, Validation, and Certification of CPS:** Verification technologies are often used to mitigate the complexity of all the interactions between functional and non-functional requirements throughout a full development life-cycle. The gap between formal methods and verification needs to be bridged. Compositional verification and testing methods that explore the heterogeneous nature of CPS models are essential. Verification and Validation (V&V) must also be incorporated into certification regimes [52].

- **Education and Training:** Design engineers (development and testing) and system integrators who are properly trained in the fundamentals of computation, control, networking, and software engineering are critically needed. All of the people who are involved in system design must spend much time to learn numerous design platforms and domain-specific modeling languages. Creative trade-offs between depth and breadth may need to be adopted.

2.2 Modeling approaches for CPS design

This section focuses on modeling approaches and their challenges to the realization of CPS. Models can have formal properties. We can thus say definitive things

about models. The use of models emphasizes understanding the distinction between a model and the thing being modeled. We call the thing being modeled the *target* of the model. A target could have a set of useful models. For example, a microprocessor chip may be modeled as a three-dimensional geometry of doped silicon (model *A*). The differential equations can describe the semiconductor physics (model *B*) and the logical program specifying an embedded system for the chip to run (model *C*). The relations network describes the relationship between programs and chips (model *D*). Those models are all abstractions of the system, and they consist of physical aspects of the chip, logical programs, and their relationships.

Every model is described by some modeling language that provides the *syntax* (how it is written down) by which the model is specified and the *semantics* (what is the given means). For example, a three-dimensional describing language is used to describe Model *A*. Model *B* is given in the mathematical language of the calculus of ordinary and partial differential equations. Model *C* can be specified within a hardware description language such as Verilog [1] and VHDL [2]. A high-level modeling language (e.g., UML-like languages) can be used to illustrate model *D*. Each language is supported within some modeling frameworks, which provides Graphical User Interface (GUI) and assistance. In this context, the methodology for Multi-Paradigm Modeling (MPM) of CPS have to be established and standardized. The precise definition of MPM are provided by the work of working group during the COST action IC1404 [55, 56]. Reusing multiple existing formalisms and their associated paradigms is a tendency [57].

2.2.1 Model-based system engineering

Model-based design and Model-Driven Engineering play an essential role in the full life cycle of CPS development [28, 58, 59]. They are various approaches to handle and analyze complex systems on different levels and diverse views [60, 48, 61].

[1] http://www.verilog.com
[2] http://www.vhdl.org

The main drivers for the development and evolution of CPS are not only for satisfying the system requirements but also for the reduction of development costs and time. This involves a number of specific domains to construct a comprehensive system, to support verification and validation, and to enhance its value. Each domain specification has different characteristics while the developer considers the domain as a view separately.

A model-based engineering solution Capella that has been successfully deployed in a wide variety of industrial contexts [20]. Capella can ensure engineering-wide collaboration by sharing the same reference architecture, and mastering different engineering levels and traceability with automated transition and information refinement. In fact, system designer benefits from the top-down model-based engineering, it allows the designer to consider much more aspects at the abstraction level than code level. For example, security concerns (e.g., confidentiality, integrity, availability, and authenticity) can be considered together with the functional logic (and other quality attributes like performance) at a very early stage, which is crucial in engineering secure systems. SysML-Sec [62] is a DSL that extends UML to perform security analyses. In other words, a DSL that is tailored for specifying a specific security aspect (e.g., access control) should be more expressive than a general modeling language like UML. However, the UML profile mechanism can be used for the definition of security-oriented DSLs as surveyed in [63].

The design of systems at the model level enables model-based verification and validation methods with tool support, which are important for detecting system design flaws at early stages. If transforming security models into inputs for formal methods is feasible [64], formal methods such as model checking can be employed for verifying security properties. Model-based security testing methods can be employed for validating the resulting secure systems (especially when formal methods would not be applicable).

Model-based engineering enables automation provided by automated Model to Model (M2M) transformations [65] and Model to Text (M2T) transformations. M2M can take part in the key steps of the engineering process, e.g., composing security models into functional models or transforming models between different DSLs. M2T

can be used for generating code, including security mechanisms, e.g., a configured access control mechanism. The automation would make the development process more productive with higher quality compared to a hand-written code development process.

2.2.2 Multi-view modeling approach

Multi-View Modeling (MVM) is not a new topic, and terms such as "view" and "viewpoint" often appear in system engineering literature, including standards such as ISO 42010 [66]. Because modeling all aspects of a complex system within a single model is a difficult task. Multi-view modeling is a methodology where different models or views capture different aspects of the system (a concept of Multi-view design [35] is shown as Fig 2.3). The whole production system is built by different aspects. In this figure, we can see an instance of a car. The car is a product, and it contains numerous views and models, e.g., the design of the engine may rely on functional view, it is also related to interconnection view and behavior view. The mechanical parties are split into several physical views. And the hybrid view may help the engineer to analyze the relationships between control systems and physical attributes.

As mentioned in section 2.1.3, one of the challenges is consistency, e.g., different views of a system have some degree of overlap, and we must guarantee that the aspects (views) do not contradict each other (i.e., they are consistent). Therefore, MVM is a crucial concern in system design.

Implicitly, MVM is supported by multi-modeling languages such as UML, SysML, and AADL. For instance, AADL defines separate "behavior and hybrid annexes" and having separate models in these annexes can result in inconsistencies. But capabilities such as conformance or consistency checking are typically not provided by the tools when implementing these standards. Architectural consistency is studied in our work.

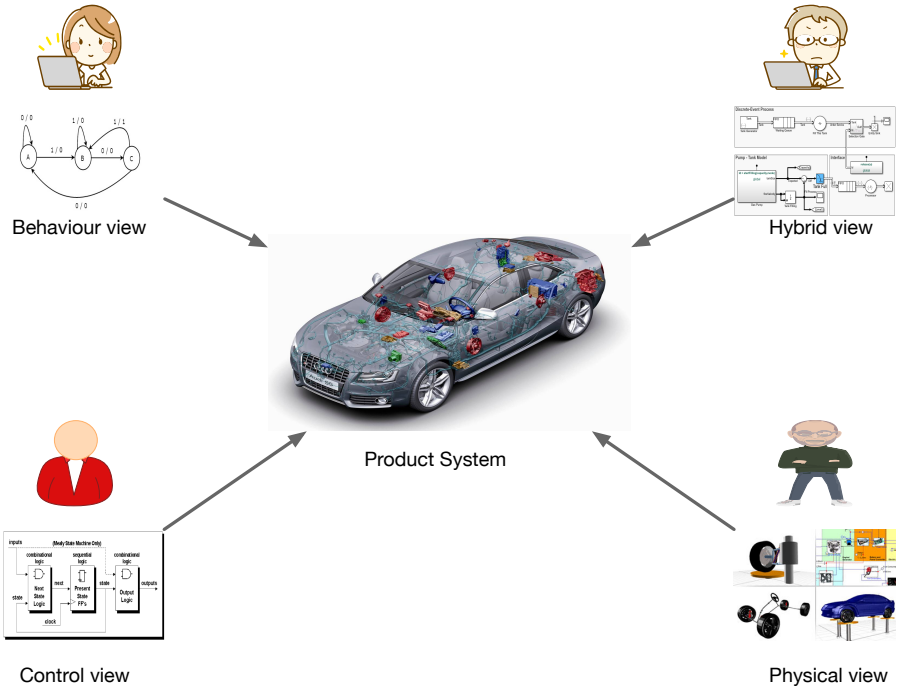


Fig. 2.3 Concept of Multi-View Design

2.2.3 Challenges for modeling CPS

A model of CPS comprises models of physical process as well as models of the software, computational platforms, and networks. The feedback loop between physical process and computations encompasses sensors, actuators, physical quantities (temperature, humidity, pressure), task scheduling, and networks with contention and communication delays. Modeling such systems with reasonable fidelity and maintain the consistency among models is challenging as it requires the inclusion of control engineering, software engineering, electromechanical, sensor networks. Specifically, the models involve numerous heterogeneous components and different aspects. Hence, the CPS design must connect to a large number of domain experts. To effectively work together and smoothly integrate their artefacts with others, establishing a mono-language with composition semantics becomes a key factor. However, the inherent heterogeneity and increasingly complexity pose new challenges which not only exist at the design stage but are also effective in verification & validation and integration stages.

Challenge 1: *Complexity of System and Heterogeneous Subsystems*: CPS may be modeled as a hybrid system where continuous time models of dynamics are used to

represent physical process and computations are described using data-flow models, state machines, and/or Discrete Event (DE) models. Continuous time models work with solvers that numerically approximate the solutions to Differential Equation (DE) or Partial Differential Equation (PDE). Integrating heterogeneous models is a big issue, and that persisted in many available tools. In the paper of *Patricia Derler et al.*, they mentioned this challenge as "*Solver-Dependent, Nondeterminate, or Zeno Behavior*" [5].

One of the problems is that the behavior defined by a model which may be non-determinate even if the models of the underlying system are determinate. It means that the model defines a variety of behaviors, rather than a single behavior. This can occur, for example, when DE models are simultaneous, and the semantics of the modeling language fails to specify a single behavior. Hence Larsen et al. [67] proposed a behavior coordination modeling language to specify the coordination among events. Another problem is that numerical solvers typically dynamically adjust the step size that they use to increment time, and the behavior of the model depends on the selected step sizes. The other problem is that some models exposes Zeno behavior, where infinite events occur in a finite time interval. Such behavior from a model may reflect physical phenomena, such as stuttering, but Zeno behavior can also arise as an artefact of modeling [68].

Challenge 2: *Keeping Model Components Consistent*: People can consider a set of homogeneous (in contrast to heterogeneous) models as the simple and uniform objects at the same level of design (the same refinement level). The problem arises as a simple model evolves into a complex one, where the uniform and homogeneous component in the simple model becomes multiple and heterogeneous components in the complex one, even the simple model is refined and further designed. How can we ensure that the components evolve together? We consider the problem of evolving multiple models with multiple variants of components, all of them while ensuring some measure of consistency across the design levels and models [28]. In a modeling environment, one element of the model can be copied and reused in various parts of the model. However, if later a change in the original model becomes necessary, the same change has to be applied to all other models that are

copied. This procedure is error-prone because there is no way to ensure that all copies are updated accordingly.

2.3 Modeling languages and frameworks

The development of CPS software applications for specific domains via modeling become an arduous task: it requires a full understanding of both the domain space (e.g., software/hardware systems, mechatronics, production system) and the solution/implementation space (e.g., modeling/programming language, platform). To span some of the domain to design a synthesis system, people usually involve domain experts to handle the professional design problem. In recent years, there has been a proliferation of modeling languaged for describing embedded (also adapted to CPS) systems. Some of these languages have emerged from domain-specific frameworks, and others are adoptions or extensions of more general purpose languages. We describe several widely used standard modeling languages:

Unified Modeling Language (UML) is a historical and general visual modeling language with a graphical syntax developed for specification, visualization, documenting and constructing entities of a system. UML is currently the standard [69, 70] for representing the structure of object-oriented programs, sequence diagrams and requirement of systems. Object Constraint Language (OCL) is a formal expression language for specifying UML constraints unambiguously [71]. It is pure expression language, and does not have side effects and cannot change anything in the UML model.

System Modeling Language (SysML) is a modeling language with a graphical syntax developed and standardized by the Object Management Group (OMG). SysML was designed to describe system, capture the interactions of software with physical entities. SysML is widely used for systems engineering [72]. In contrast to UML, SysML has added some support for systems engineering (e.g. requirements engineering, and quantitative analysis of physical aspects of the system), meanwhile removing some UML constructs.

SysML-Sec is SysML support environment with a more holistic approach, which introduces both customized SysML diagrams for security matters and an associated methodology. SysML-Sec aims at helping security experts to intervene on the design and development of an embedded system together with system designers [73]. A key point of SysML-Sec is its partitioning stage during which safety & security-related functions are explored jointly and iteratively with regards to requirements and attacks. Once partitioned, the system is designed in terms of system functions and security mechanisms, and formally verified from both the safety and the security perspectives. The SysML-Sec methodology and diagrams have been developed and experimented in the European project EVITA [74, 75]. The support environment, called TTool, provides design space exploration for SysML-Sec, and integrates dozen of use cases.

Domain-Specific Languages (DSLs) have been briefly mentioned many times in previous sections. DSLs are an integral part of CPS development. They are related to both software and hardware. DSLs have many uses, they are used as an intermediary step from requirements towards final implementation. They are used for modeling specific aspect of system, so-called Domain-Specific Modeling Language (DSML). They are used to verify critical properties of complex systems such as safety & security and liveness, and they may be used for automatic code generation, performance evaluation, and test-case generation.

2.3.1 Capella and Arcadia methodology

The group Alenia Space of Thales, focuses on system engineering, which covers most areas of its activity spectrum, covering Observation, Navigation, Space Exploration and Science and Telecommunications. Besides actively sponsoring the achievement of INCOSE CSEP (Certified System Engineering Professional) among its employees and with the goal of fostering a common tooled up approach and use of the same reference architectures, Thales has conceived a solution based on these core elements:

- a system engineering methodology, called Sys-EM, which defines the successive stages of the overall engineering process

- a model-based engineering method for systems, hardware and software architectural design, called ARCADIA

- a THALES internal system modeling tool, Melody Advance, now released in the Open Source as Capella [3]

2.3.1.1 Capella project

Capella project consists of Model-Based System Engineering methods and tool suites for designing systems from a high level of abstractions. Capella also adopts a multi-view point description to illustrate different specifications, such as physical part, logical part, and allocation relationships. Capella has been successfully deployed in a wide variety of industrial contexts.

2.3.1.2 The ARCADIA engineering methodology

ARCADIA (ARChitecture Analysis and Design Integrated Approach) is a model-based engineering methodology for systems, hardware and software architectural design. It has been integrated in Capella project and developed by Thales since 2005 [76] through an iterative process involving operational architects from all the Thales business domains (transportation, avionics, space, radar). ARCADIA enforces an approach structured on successive engineering phases which establishes clear separation between needs (operational need analysis and system need analysis) and solutions (logical and physical architectures) in accordance with the ISO 42001 standard [77].

According to ARCADIA methodology, we give the definition of each phase, and sketch meta-models using the Eclipse Modeling Framework (EFM)[5]. Fig 2.4 shows a global view of ARCADIA methodology from operational phase to the final product breakdown phase, red rectangle represents the operational activities, green rectangle represents functions and yellow rectangle represents physical components.

[3] https://www.polarsys.org/capella
[4] https://www.polarsys.org/capella
[5] https://www.eclipse.org/modeling/emf/

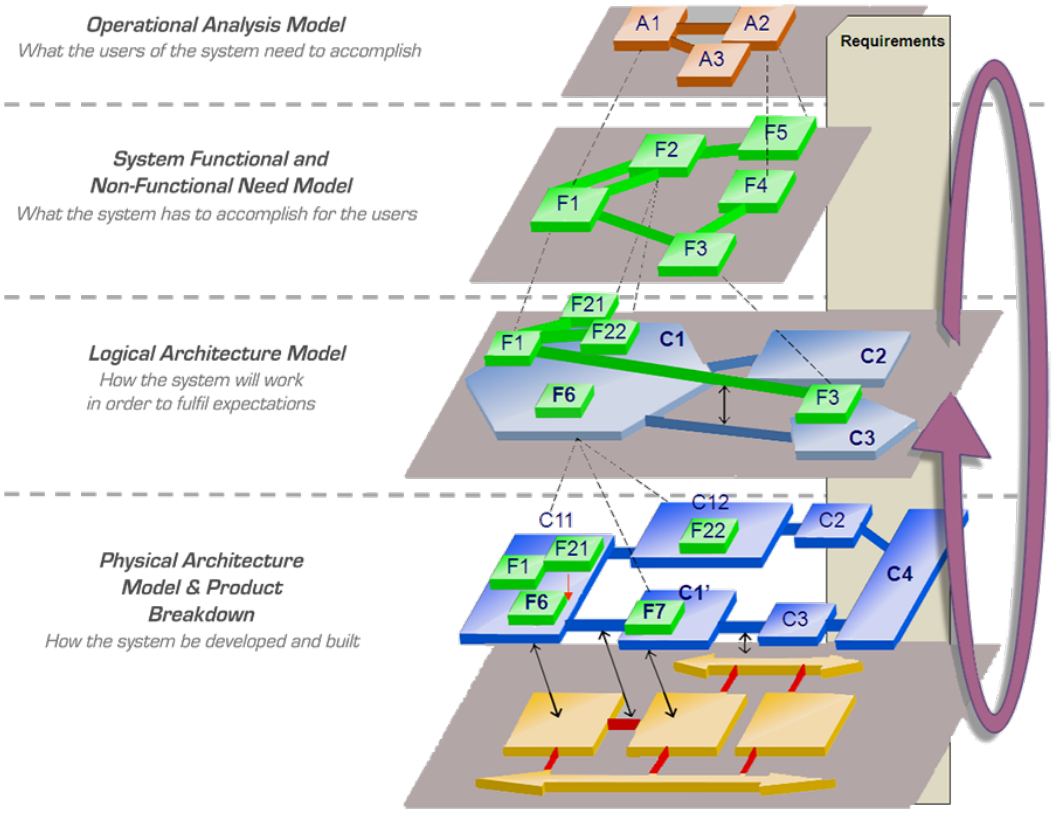


Fig. 2.4 Global view of ARCADIA methodology[4]

Operational analysis

At the Operational Analysis phase, we should capture the Operational Activities and Operational Entities and the interactions between them. The activities include functional and non-functional properties such as partitioning, safety, security. Finally, it can describe and structure the needs and the goals of the customer. The meta-model of our approach is shown in Fig.2.5

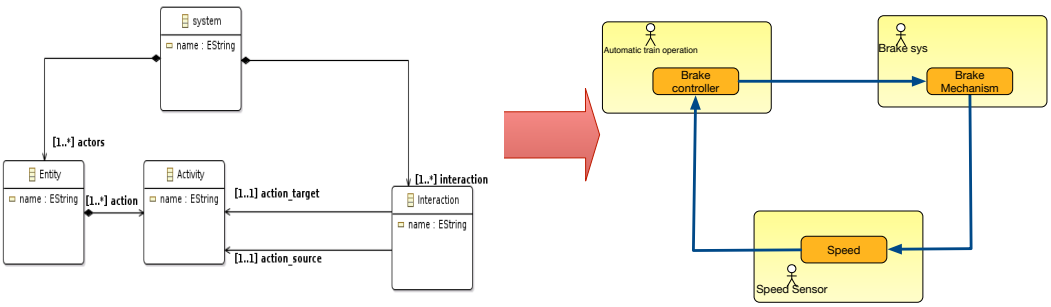


Fig. 2.5 Meta-Model of Operational Analysis

System analysis

At the System Analysis phase, we focus on the system level. An architecture is intended to illustrate allocations (Fig.2.6) of functions onto components so as to com-

ply with system needs. Meanwhile, the architecture diagram is also used to check the feasibility of the customer requirements with a multi-view approach (safety, cost, consumption).

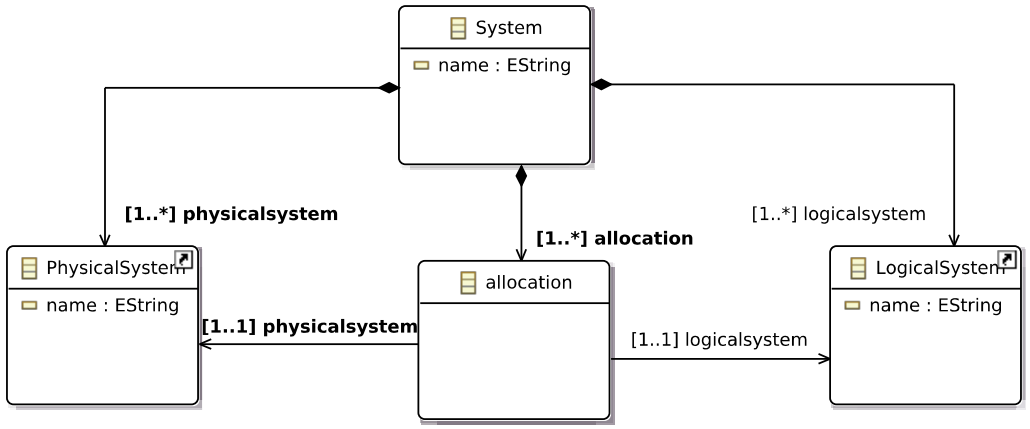


Fig. 2.6 Allocation on system level

Logical architecture

This phase aims at breaking down the functional design of system. All the functional and non-functional constraints (safety, security, performance, cost, non-technical) are taken into account, starting from previous functional and non-functional analysis refined results (functions, interfaces, data flows, behaviors), building one or several decompositions of the system into logical components.

Physical architecture

The Physical Architecture phase is similar to logical architecture design procedure, yet it focus more on Physical object. It consists of the selected physical architecture which includes components to be produced, formalization of all viewpoints and how take them into account at the components design. Once the model has been finished, a more classical development stage can start. The same viewpoint-driven approach as for logical architecture design is used.

2.3.2 TTool – A SysML-Sec support toolkit

TTool is a SysML-Sec based support toolkit [16, 62], which can capture system requirements, model software/hardware partitioning. Fig 2.7 shows the partial con-

cept for securing the system in TTool. Once the security goals are assigned, the security engineers conduct risk analysis. They can next set up the security configurations, e.g., using a key-based method to create a function for authenticating purposes. This is an iteration process which used to help security engineers to achieve the security goals step by step.

TTool is also proposed to improve both partitioning and prototyping development stages for security and safety issues. In fact, prototyping can rely on software and hardware elements that are formally evaluated at partitioning. Partitioning models can be enhanced using precise parameters that can be obtained during the simulation at the prototyping level.

The design with security strategies can be quickly validated and iterated. It can help engineers find an appropriate design solution timely.

TTool furnishes a press-button approach to evaluate the design at a given stage, and to propagate the results to enhance the system at another stage. Relying on internal (simulator, model-checkers) and external tools (e.g., ProVerif and UPPAAL), TTool can perform simulation and formal verification for safety, security and performance [78, 60]. Results can help engineer decide whether safety, performance and security requirements are fulfilled [79, 80]. Especially, in TTool, it translates the SysML models into an intermediate form that is sent into the underlying simulation and formal verification utilities. Backtracking to models is then performed to better inform the users about the verification results. Proofs of safety involve UPPAAL semantics [81], and security proofs use ProVerif [82].

2.4 MBSE concerns in CPS design

MBSE is a well-known approach that is a key enabler for building large-scale complex cyber-physical control systems [83, 84]. It has features to reduce development complexity, enhanced productivity, efficient change management, and improved time-to-market [85]. Therefore, it has been frequently researched and customized for the development of embedded systems and industrial control systems [23, 86,

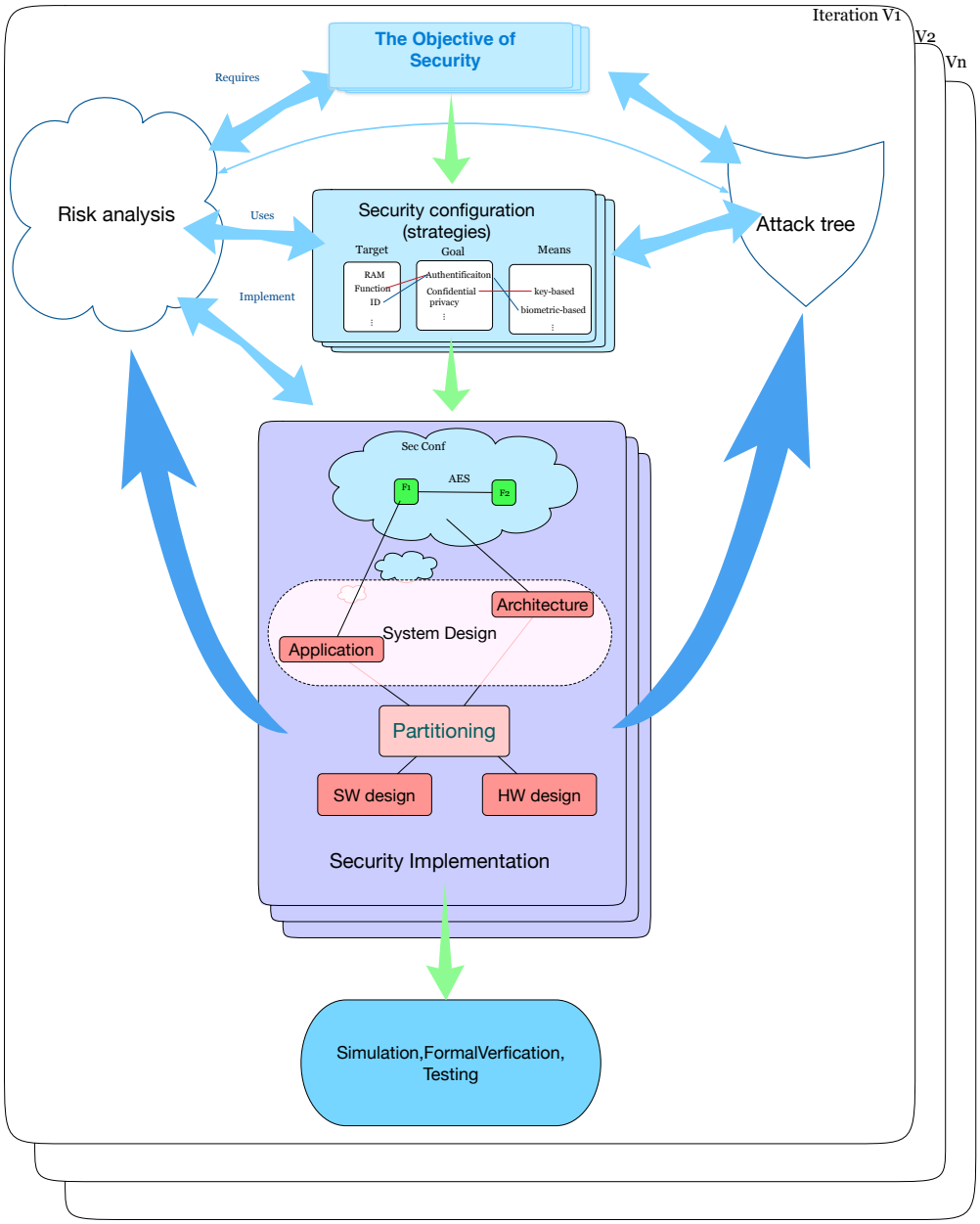


Fig. 2.7 Partial concept for security in TTool

87]. The Conception of MBSE for full-life development of system is shown in Fig 2.8.

Modeling functional and structural aspects of embedded systems are the foremost activities. All other MBSE tasks (i.e., MT, verification, and validation) are centered on the model. Therefore, models are developed by taking into consideration the MT, verification, and simulation requirements. For example, one of the major challenges is to model behavioral/temporal aspects of complex embedded systems for further verification and validation.

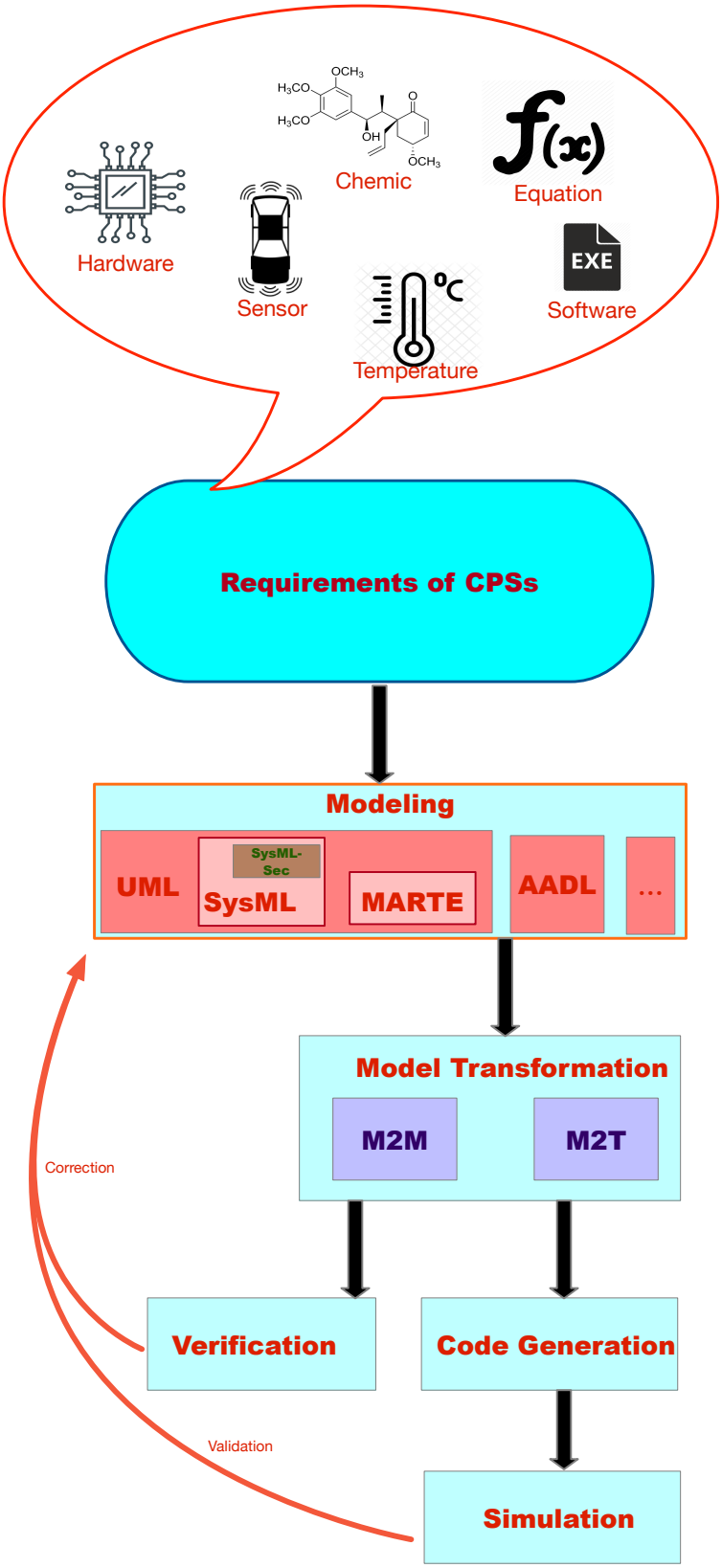


Fig. 2.8 Concept of MBSE for full life-cycle development of CPS

UML [70] and its profiles, SysML [88] and Modeling and Analysis of Real-time and Embedded system (MARTE) [69] are frequently used in contemporary system development practices. They are the key enabler for establishing models of the system, and all of them can also be used in the development of CPS [4, 5, 89] to specify systems requirements and to generically model systems. Furthermore, a number of techniques and languages have been proposed to describe some non-functional properties related to, among other thing, safety, behavior and temporal aspects [67, 62, 90]. Once requirements are modeled, different MT techniques/languages have been applied to develop a platform-specific model and/or source code generation. Two types of transformations are commonly used, i.e., model-to-model (M2M) transformation and model-to-text (M2T) transformation [91].

The verification is performed to ensure the correctness of the model/system, and it is tightly coupled with the modeling technique used to specify non-functional aspects such as safety and security [78, 60, 47, 92, 93]. Various formal verification techniques [94, 95, 96, 97] have been used to verify the safety/security aspects of the system. If the model does not satisfy the verification requirements, then corrections must be made to the model as shown in Fig 2.8. The validation of the model/system can be performed through simulation.

2.5 Conclusion

In this chapter, we have presented the background of CPS and introduce CPS challenges. There are the main challenges to be solved in the remainder of this thesis. We have also briefly presented the relationships between the terms such as CPS, Industry 4.0 and IoT. Then, we have given some application examples to show the potential of them both in academia and industry.

Next, we have presented the modeling languages and frameworks. Modeling languages have a long and rich history in computer science, and many techniques have been proposed for supporting their definition. We have pointed at some of the import languages, such as UML and SysML, for the reader to understand what follows. Then, we have briefly introduced Capella and Arcadia methodology, a widely used framework. Specifically, focusing on the ARCADIA modeling ap-

proach in the Capella framework, including the four levels of Arcadia methodology and Capella project. We have also mentioned SysML-Sec, a SysML's profile and support toolkit–TTool as it is used in the following.

State-Of-The-Art

3

> *The people who get on in this world are the people who get up and look for circumstances they want, and if they cannot find them, make them.*
>
> **— George Bernard Shaw**
> (writer and Novelist)

In the previous chapter, we have described the relevant technologies around CPS, and mentioned the challenges of the CPS design. In this chapter, we review MDE approaches and model transformation related technologies in both theoretical and technological, and their implementation and workbenches. We also review the applications of MDE on specific domains such as security & safety and schedulability.

3.1 Introduction

MDE advocates the use of models during the whole system development process. It refers to systematic use of models as first-class entities throughout the system development life-cycle [98]. By leveraging abstraction and automation, MDE techniques can simplify design activities, and communication, reducing the complexity of the development, increasing compatibility among subsystems and productivity, and boosting development efficiency [65, 23]. MDE can also facilitate a more comprehensive description of the system, since the different viewpoints of the system can be described by using models [99, 61]. Model-Driven Development (MDD) is a special case of MDE. In a model-centric development approach, the models serve as primary artifacts, e.g., fully executable code is generated automatically according to the models [100].

In the CPS design, the system designers use MDE approach to handle different aspects for one whole system, and there are some issues such as complexity that we mentioned in the introduction. Thus, system designers need to reuse the modeling artefacts [101] and exchange informations between various frameworks. We specifically look at methods that support combining two different modeling design frameworks which have different professional design abilities, such as functional and safety & security design. MT approaches can help system engineers to reuse the tools and transforming or sharing the designed models between stakeholders.

3.2 Model transformation

Model transformations (MTs) are at the core of MDE, it is a kind of program used to transform a model or meta-model from one form of representation to another one. The result of a survey shows a tendency towards applying transformations between models and reusing of multiple existing formalisms [57]. MTs are complex pieces of software then reuse mechanisms are important [102] and needed by the community [103]. A lot of MT languages and tools have been proposed over the last few years [104, 65]. MTs must have an input model of transformation which is called source (model) and conforming to a higher level source (meta-model), and an

output model (so-called target model) conforming to a target meta-model in Model to Model (M2M) transformations [102]. When the target is pure text (not a model), then we refer to it as Model to Text (M2T) transformations [91].

A MT uses a language to write the description/specification, defining how one or more source models are transformed to one or more target models. If the transformation description is rule-based, the transformation consists of a set of transformation rules. The transformation engine/tool produces the target model from the source model according to MT expressions. Meanwhile, the models must be expressed in a well-defined notation. Thus transformation specifications use the meta-model to define the appropriate structures, and properties to which a model must conform. There are higher abstraction of models that define meta-models so-called meta-meta-models. Meta-meta-models are often reflexive so that they can be defined based on themselves. While in theory, there is any arbitrary number of meta-modeling levels, the OMG defined a four meta-modeling level architecture from M0 to M3. Figure 3.1 shows the architecture of models (left side) defined by OMG. This figure also shows the MT conception in four levels, each level can find a corresponding level in left side. In OMG standard, level M0 represents the real-world system (with the blue box), next level, M1 represents the modeling level of the system (within the red box) that is an instance of the next level. Level M2 is the meta-modeling level that describes the model in the level M1. The meta-meta-modeling in level M3 shows that meta-model conforms to itself. The relation between a model and its meta-model, and the meta-model with one of its models is shown with conformance and instantiation type respectively.

3.2.1 Classification and tools

With MDE becoming more prevalent in software development, the number of model transformation techniques/tools has increased rapidly. Several MT approaches have been discussed over the last decade for MT reuse, such as *"Melange"* proposed by Degueule et al. [105] and the discussion from Dániel Varró and András Pataricza [106]. These approaches can be divided into two categories: approaches for MT reuse without adaptation (i.e., reuse between isomorphic metamodels) and approaches allowing adaptations (i.e., structural heterogeneities). A example of MT

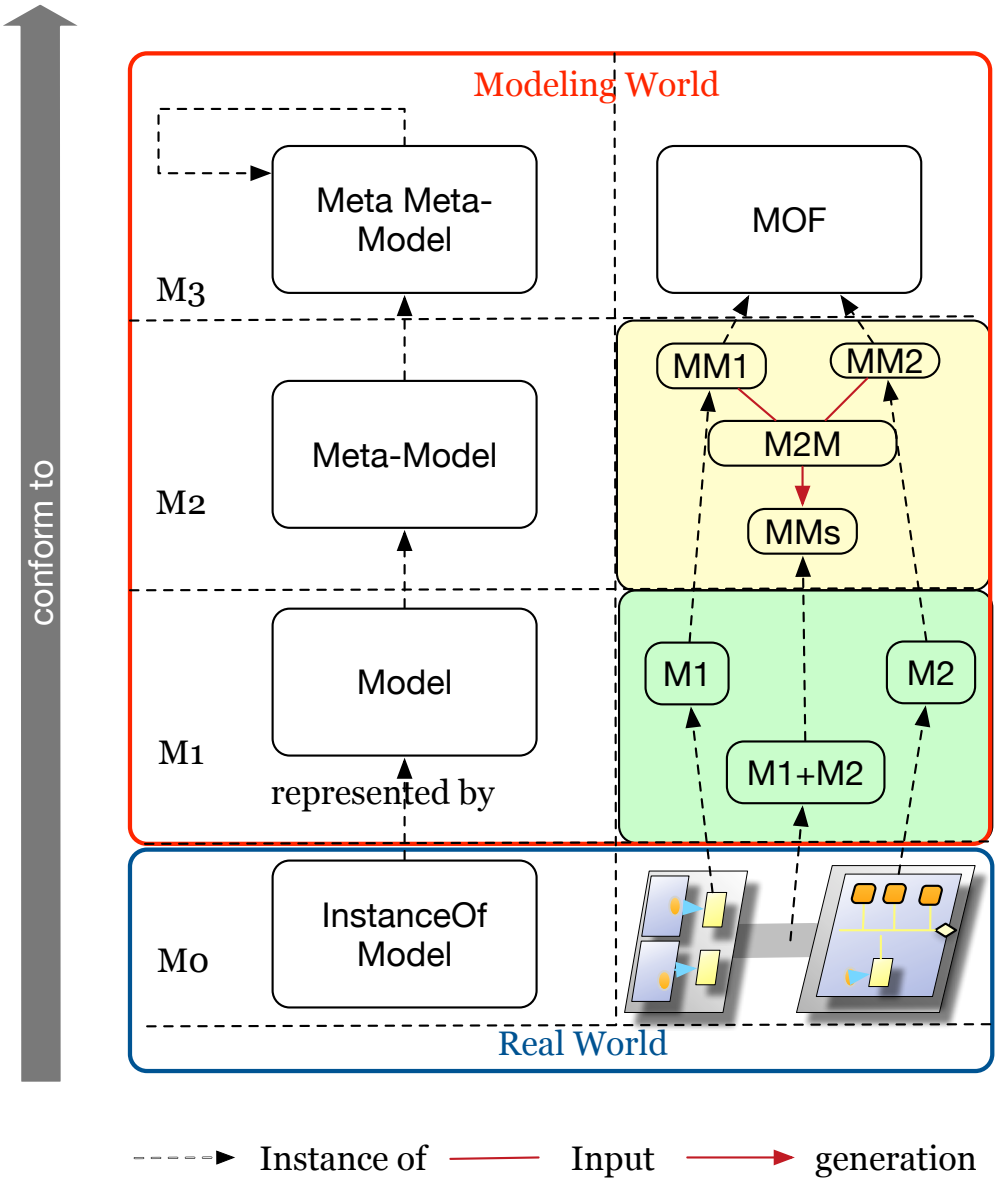


Fig. 3.1 Excerpt of MT and corresponding levels defined by OMG

reuse without adaptation was proposed by Varró and Pataricza who introduced variable entities in patterns for declarative transformation rules [106]. These entities only express the concepts (types, attributes…) required to apply the rules. This allows tokens with these concepts to match the pattern and be processed by rules. Semantic Variation points can be specified through abstract classes defining a template [107]. Metamodels can fix these variation points by binding them to classes extending the abstract classes. Such patterns can be viewed as model types whose variability has to be explicitly expressed.

In fact, there have been a number of publications [108, 109, 110, 111, 112, 113, 103] systematically classifying and comparing model transformation approaches and tools over different features. One of them, Bruel al. [114, 103] analyzed the design space of MT reuse approaches. By the feature model, they classified the alternatives for MT reuse across metamodels into six categories, such as *Strategy, Mappings, Reuse by, Reuse interface, Correctness checking and Properties of reused transformation*.

However, other evaluations and classifications are from MT tool views. Based on the kind of target of MT, its tools can be classified into three main categories namely, M2M, M2T and its inverse, text-to-model (T2M) transformations. T2M transformation tools MoDisco [115], a text-based description as input and models as output of transformations. Because of T2M tools are usually used for reverse engineering, we do not consider T2M transformation in our work.

As one of my contribution is an MT tool which can be classified into M2M, so we mainly study on M2M field. M2M tools server as convertor for transform one or more source models into one or more targets. Transformation languages provide a set of constructs or mechanisms to conduct transformations. In the paper of Kahani et al. [91], they classified the different M2M approaches into three types, relational, imperative, graph-based. They are different types of approaches to implement M2M MT tools.

Tables 3.1 and 3.2 list a high-level overview of the tools based on a taxonomy of their transformation language. The first column in the table is the type of tools, and the last column gives a simple description of the tool.

3.2.2 Relational M2M

Relational/Declarative Approaches focus on what should be transformed into others, without specifying a sequence of execution order. Relational approaches have to define relationships between the elements in the source and target models. These relations are defined with mathematical method in a formal way, they can be specified by predicates and constraints.

Type	Tool	Description
Relational/Declarative	UML-RSDS	A UML based tool with verification support to construct software systems [118]
	JTL	Specifically focuses on synchronization and change propagation models
	Tefkat	A rule and template-based engine implementation of Tefkat language [116]
	PTL	ATL-style rules are combined with logic rules to define transformations [117]
	mediniQVT	Uses QVTR language in the textual concrete syntax
	QVTR-XSLT	Based on the graphical notation of QVTR and XSLT [120]
	Echo	Used for model repair and transformations with the model finder Alloy [121]
	TXL	A grammar-based tool that can be used for MTs

Tab. 3.1 Relational/Declarative model transformation tools

Type	Tool	Description
Imperative/Op/Co	Xtend	A statically-typed high-level programming tool for JVM
	QVTo-Eclipse	An Eclipse implementation based on QVTo [122]
	MetaEdit+	A tool for domain-specific modeling and development [123]
	Kermeta2	Based on a model-oriented language optimized for meta-models and DSLs [124]
	Melange	Kermeta2 supports the semantics of the modeling languages [67]
	JQVT	Based on a compiled QVT engine for Java
	Together	A set of Eclipse plugins which partially implements the QVTo language

Tab. 3.2 Imperative/Operational/Constructive model transformation tools

Relational approaches include functional programming, and logic programming. In functional languages, a transformation function can transform the input model into the output. Object-oriented (OO) languages is a straight approach for MTs. However, functional language has the strength that the developer does not need to deal with non-trivial task of writing code for model traversing. Tools such as Tefkat [116], PTL [117], UML-RSDS [118], JTL [119] are examples of relational approaches. A special type of high-level relational MT approach is QVT Relation. In QVT relations, a relation is specified by two or more domains with a pair of when and where clauses, e.g., mediniQVT, QVTR-XSLT [120], Echo [121]. We listed a high-level overview of those tools, see Table.3.1.

3.2.3 Imperative M2M

Imperative/Operational/Constructive Approaches are based on imperative languages that focus on how and when the transformation should be executed, without considering the relations that must hold between source and target elements. The language specifies a transformation as sequential actions/rules. The Behavioral Coordination Operator Language (BCOoL) [67] injects events in the metamodel to observe dans coordinate the execution of two models. MetaEdit+ is imperative and use procedures as a decomposition mechanism to combine a set of elements [123].

There are also languages such as QVT Operational language (e.g., QVTo-Eclipse [122], Together[1], JVQT[2]) where transformations are defined using mappings. Each mapping can transform one or more elements of a source model to the corresponding target elements. QVTo mappings, similar to relations in QVTr, may contain when and where clauses. Examples of imperative tools are Mitra2, JQVT, ModelAnt, Kermeta2, Modelio, Xtend, Umple, MDWorkbench, Melange, WebRatio, Merlin, Enterprise Architect (EA), and MOFScript. We listed some of those tools, see Tab.3.2.

There is also a mixed approach that can manipulate the models directly with low-level constructs and language concepts to support MTs. In this kind of approach, general-purpose programming languages can be used to implement the core of MTs, so-called parsers, which take in charge the interpretation around models. Our proposed approach can be classified into this kind. It does not request the engineer to spend a lot of time to learn a new language to write transformations. However, these languages were not primarily designed for direct model manipulation, so users have to manually implement many required features of MTs, such as traceability or model exploration.

3.2.4 Graph-based M2M

Graph-based Approaches or Graph-based languages are based on algebraic graph grammars and represent the source and target models using various graphs, such as typed graphs and labeled graphs. The transformation based on the graph consists of a set of graph transformation rules (also called rewiring or production rules [91]). A source graph of the model applies the rules to create a new target graph of the model. Each rule consists of a rule graph. The execution of a graph transformation rule involves the related elements that can be detected by the graph algorithm. All elements that are in the rule graph but not appearing in the source graph are combined with original elements, and all the left elements that exist in the source graph remain unchanged, see Figure 3.2, as an example of graph-based MT. And

[1]Together. URL: :http://www.borland.com/Products/Requirements-Management/Together. Developed by: Borland.
[2]JQVT http://sourceforge.net/p/jqvt/wiki/Home/.

also see some examples of tools, AToMPM [125], MOMoT [126], GROOVE [127], AGG [128], BOTL [129] and GRoundTram [130].

The major drawbacks of the graphical notation is the complexity and verbosity of representing the graph transformation rules. Furthermore, they suffer from traceability difficulty between input and output graph instance elements. Triple Graph Grammars (TGG) [131] were proposed to overcome this weakness by using correspondence graphs or meta-models that maintain *N*-*M* relation between source and target transformed elements. Thus, they can be used to synchronize two different models and check whether they are consistent. Examples of TGGs tools are Henshin [132], TGG Interpreter, and EMorF [133].

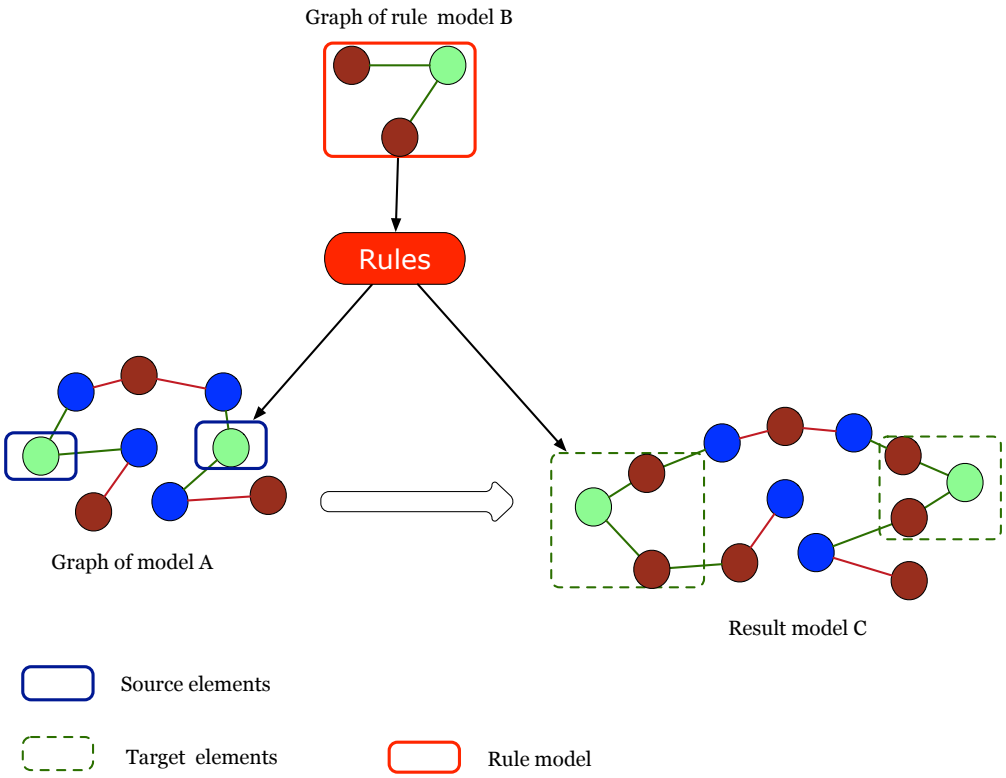


Fig. 3.2 Example of Graph-based Transformation

3.3 Modeling languages

3.3.1 DSML

The Object Management Group (OMG) proposes to specify models by relying on a language that has a well-defined form (syntax), meaning (semantics) and possible rules of analysis, inferences or proof for its constructs [134]. Thus, MDE proposes DSML to build models. As a result, a DSML is defined with a (rigorous) syntax and clear semantics. The syntax is described by a metamodel that defines the concepts and relations that the language is made up. A metamodel is a model that is developed by using a "meta-meta language", e.g., MOF, ECORE. We distinguish three types of approaches for the semantic definition: Operational [135], Axiomatic [136] and Translational [137]. Other researchers have also proposed other ways to define the CPS models [138].

MDE emerged to allow the development of applications based on the definition of models closer to the problem domain than to the implementation domain, reducing the complexity of platforms. To do so, MDE makes uses of Domain-Specific Modeling Languages (DSMLs), which are modeling languages defined for application requirements, behavior, and structure within specific domains. A DSML follows the domain abstractions and semantics, allowing design engineers to perceive themselves as working directly with domain concepts. The definition of a DSML involves at least three aspects: the abstract syntax that may be domain concepts and rules; the concrete syntax is the notation used to represent these concepts in textual or graphical; and the semantics of the language.

A DSML allows developing software for a particular application domain effectively and quickly, generating programs that are easy to understand, reason about, and maintain [139]. There is a significant overhead in creating the infrastructure needed to support a DSL. Numerous works were proposed to create reusable and composable language units to tackle this issue. Methodologies have been proposed for building DSLs embedded within an existing, higher-order, and typed programming language [140]. Then, techniques have been designed for building modular interpreters and tools for such embedded DSLs. Different techniques have been studied for addressing the challenge of language extension and composition, such as projectional editing [141]. Spoofax is used to define syntaxes and semantics, which

rely on meta-languages. They are inherently modular and composable [142]. Although basic import mechanisms are supported, they usually lack powerful support for customization. More recently, an overview of the support provided by language workbenches has been provided [143, 144].

In the grammar world, several techniques demonstrated the possibility to create language units using attributed grammars [145, 146]. MontiCore applied modularity concepts for designing new DSLs by extending an existing one or by composing other DSLs [147]. MontiCore reifies as a first-class object the concept of language inheritance to allow language feature reuse. Other works propose to leverage concepts from the component-based software engineering community to modularly develop DSLs [148, 149].

In the MDE domain, several meta-tooling platforms propose mechanisms for improving language design modularity. Ledeczi et al. propose to compose domain-specific design environments using MDE technologies [150].

There are also some frameworks with IDE for building textual DSLs, such as Melusine [151], Xtext [152, 153] and MPS [141, 142].

In both the MDE and grammar domains, the increasing trend to create new DSLs from scratch or by adapting existing ones causes the emergence of families of DSLs. A family of DSLs is a set of DSLs sharing common aspects but specialized for a particular purpose. The emergence of a family of DSLs raises the need to reuse common tools among a given family [154, 102] and the need to create composable language units. To ease the language unit composition, Steel et al. [155] and De Lara et al. [156] propose to define a clear contract and a typing system that can be used for composing language units. De Lara et al. present the concept mechanism, along with model templates and mixin[3] layers leveraged from generic programming to MDE [157]. Concepts are close to model types [155] as they define the requirements a metamodel must fulfill for its models to be processed by a transformation, under the form of a set of classes. Sánchez, Wimmer et al. go further than strict structural mapping by renaming, mapping, and filtering metamodel elements [101, 104]. Erdweg et al. proposed a taxonomy to ease the positioning of approach related to language composition [158]. According to this classification, our algebra

supports the language extension, restriction, and unification operators. Additionally, we do not consider that restriction is only a matter of additional validation rules. Instead, we prune the language from the unwanted parts so that only the necessary concepts are kept.

3.3.2 Extending languages

One of our technical contributions is extending the SysML-based engineering framework Capella to AADL. Then, we can analyze the relationships among Arcadia and AADL models in different views at the metamodel level. Likewise, a considerable number of studies have been proposed on "language extension, modeling languages integration and composable language components". This subsection provides a brief introduction to these works.

The complexity of CPS has been a significant issue that puzzles developers. It is not only from the nature of problems but also from the developed languages. Elaasar et al. have discussed [159] about the limitations of UML, which exacerbate the complexity of development and proposes an approach to reduce the complexity of UML tools by implementing and adapting the ISO 42010 standard on architecture description.

Efficient integration of different heterogeneous modeling languages is essential. Modeling language integration is onerous and requires in-depth conceptual and technical knowledge and effort. Traditional modeling language integration approaches require language engineers to compose monolithic language aggregates for a specific task or project. Adapting these aggregates to different contexts requires vast effort and makes these hardly reusable. Arne Haber et al. [160] presented a method for the engineering of grammar-based language components that can be independently developed, are syntactically composable, and ultimately reusable.

There are also specific attempts either to combine SysML and AADL [161] or to extend SysML with AADL-specific constructs [162]. These approaches differ from

[3] In object-oriented programming languages, a mixin is a class that contains methods for use by other classes without having to be the parent class of those other classes. How those other classes gain access to the mixin's methods depends on the language. Mixins are sometimes described as being "included" rather than "inherited".

our approach that attempts to extract only the relevant subsets of both with a goal-oriented approach. In practice, one could design a global system at a high level and then seamlessly refine the models within AADL and its annex for further analysis such as scheduling. In other words, our approach can properly extend Arcadia's design and analysis capabilities with AADL constructs while trying to keep the two languages independent.

An approach for translating UML/Marte detailed design into AADL design has been proposed by Brun et al. [163]. Their work focuses on the transformation of the thread execution and communication semantics and does not cover the transformation of the embedded system component, such as device parts. Similarly, in [164], Turki et al. proposed a methodology for mapping UML/Marte model elements to AADL components. They focus on the issues related to modeling architecture, and the syntactic differences between AADL and UML/Marte are well handled by the transformation rules provided by ATL tool, yet they did not consider issues related to the mapping of UML/Marte properties to AADL properties. In [165], Ouni et al. presented an approach for transformation of Capella to AADL models target to cover the various levels of abstractions. They take into account the system behavior and the hardware/software mapping. However, the formal definition and rigorous syntactic of transformation rules are missing.

Behjati et al. describe how they combined SysML and AADL in [162] and provided a standard modeling language (in the form of the ExSAM profile) for specifying embedded systems at different abstraction levels. De Saqui-Sannes et al. [161] presented an MBE with TTool and AADL at the software level and demonstrated it with the flight management system. Both these works do not provide the description in a formal way.

In industrial domain applications, Suri et al [166] proposed a model-based approach for complex systems development by separating the behavior model and execution logic of the system. Moreover, they used UML-based languages to model system behavior and connected the behavior models to the external physical API of CPS. It focuses on providing a solution for the modularity and interoperability issues related to Industry 4.0 from a systems integration viewpoint.

S. Apel et al. [167] also studied different model-driven methods for heterogeneous systems for Electric vehicles. They have tried to evaluate how model-driven engineering (MDE) combined with generative frameworks can support the transfer from platform-independent models to deployable solutions within the logistical domain.

The work of Kurtev [168] is used in the x-ray machine. It provided a family of domain-specific languages that integrate existing techniques from formal behavioral and time modeling. F. Scippacercola [169] have explored the application of model-driven engineering on the interlocking system (a subsystem of signaling system of the railway). They discussed how to reduce efforts and costs for development, verification, and validation in a critical system.

The modeling language scientists have proposed some specific methods to weave the models as well as metamodels formally such as [58], Degueule has proposed Melange, a language dedicated to merging languages [105], and similar works like [170].

3.4 Multi-View Modeling

Multi-view modeling is used to separate domains in the development of a system, making it easier to manipulate its complexity. Cicchetti et al. [171, 172] proposed two multi-view modeling approaches: synthetic and projective. Projective contains an essential meta-model where the views are the focused concepts of this meta-model. Boulanger et al. [173]'s work follows this approach. Another example of this approach is the work of Nassar et al. [174], they define a UML profile called VUML to support multi-view modeling in UML. Synthetic considers each view as an independent meta-model that describes a part of the system. To build a complete system, the views must be put together.

Our approach uses UML or UML-like modeling language to describe the multi-view model, therefore it follows the projective approach. On the other hand, we specify the views using the profile mechanism. Such a mechanism allows also following a synthetic approach.

From the system engineering view, Multi-View approach allows developing both software and hardware from different domains by quickly and effectively integrating different domain expert artefacts to build up a sound and consistent system. Numerous works were devoted to providing efficient dedicated (meta) language for integrating issues. For instance, Muller *et al.* [175] proposed using aspect-oriented modeling to build an executable meta-language by composing action metamodels, and Jézéquel worked at model weaving approach [58]. In contrast to their languages or approaches, our approach is dedicated to seamlessly combine different models of views at high-level, it is meant to be easier to use and understand. Other approaches addressed modeling consistencies from constraint-based [51] or from architecture models [176]. On our side, we tackle this problem with an efficient yet simple combination of (meta) models.

Jörg Kienzle *et al* proposed a composition technique which is implemented in a tool called Kompose [177]. Kompose focuses mainly on the merging of class diagrams. In their proposition, the model elements to be composed must be the same syntactic type, that is, they must be instances of the same meta model class.

Degueule *et al.* [105] also provided a so-called "Melange" meta-language. This language can weave two DSLs to produce new DSLs that integrated the syntax and semantics of the two languages. Instead of getting a new language, our approach is meant to take strengths of other tools to complete our needs by combining (meta) models.

In Thramboulidis *et al.* [24] paper, they introduced a UML-based approach adapted to Internet of things (IoT), so-called *uml4Iot* that can automatically generate the process which is required for cyber-physical component to be integrated into the manufacturing environment. Our approach can adapt to other application domains in embedded systems (including industrial control system, IoT and smart manufacturing).

3.5 Modeling for security & safety

CPS considers two types of functions: physical and cyber. Functions interact with each other through flows. Functional modeling naturally leaks information that can be used to attack the system. However, recently, some SysML-based modeling language and toolkit was developed to address this issue. For example, TTool and SysML-Sec toolkit [16, 62] can capture system requirements, model software/hardware partitioning, and capture security concerns. Relying on internal (simulation, model-checkers) and external tools (e.g., ProVerif and UPPAAL), TTool can perform simulations and formal verification for safety, security and performance analysis [78, 60]. Results can help engineers in deciding whether safety performances and security requirements are fulfilled [79, 80]. Especially, in TTool, the tool translates SysML models into an intermediate form that is sent into the underlying simulation and formal verification utilities. Backtracking to models is then performed to better inform the users about the verification results. Proofs of safety involve UPPAAL semantics [81], and security proofs use ProVerif [82].

In most MDE projects, requirements are written in plain text. Laleau et al. [14] present some work to combine SysML requirement diagrams and the B formal specification language for conducting formal proofs. SysML requirement model is extended to represent some concepts in the goal-oriented approach. And, derivation rules are used to translate the SysML goal models into B specifications. By doing so, they narrow the gap between the requirement phase and the formal specification, and a more precise semantics of SysML goal models is given.

Albinet et al. [178] proposed to directly include system requirements in the design process but the separation with the proposed solutions as required by safety standards such as ISO 26262[4] is achieved by isolating the following triplet: requirement models, solution models, and verification and validation models. In the ISO 26262 standard, it imposes a clear distinction between the concepts: the solution has to be developed independently with respect to the requirements as well as to the verification and validation (V&V) part. The separation is important because from the

[4]ISO 26262, an adaptation of the Functional Safety Standard IEC 61508 for automotive electric/-electronic systems

given requirements, various solutions can be defined. Also, as cited in [179], the developed solutions must be evaluated by actors independently of the design process, which will promote a diversity of analysis while increasing the coverage and confidence levels of the safety conclusions. Of course, this is not in contradiction with an integrated framework where the traceability between the solutions and the requirements as well as the safety analysis will be maintained.

SysML is semi-formal modelling approach [180]. The large segments of development life cycle rely on SysML models and more formal models, specification, implementation and verification&validation. Yet, the initial model is derived from the user's text-based requirements, the gap between textual or semi-formal requirements and the formal specification is an obstacle in modeling systems. In addition, verification and validation require the requirements being described in a formal way. Therefore, Laleau et al. have used the B method to bridge this gap [14, 13]. Specifically, they define a set of rules to translate UML concepts and SysML concepts into a B specification.

A SysML profile called requirement profile for MeMVaTEX (RPM) has been developed in [178]. The requirement stereotype of SysML is replaced by the MeMVaTEX requirement, by adding various properties such as verifiable, verification type, derived from, satisfied by, refined by, traced to. So, the traceability is assured between requirement models, between requirement and solution models, and between requirement and V&V models using these properties. These V&V models have also been explored in the work of Guillerm et al. [181].

3.6 Conclusion

In this chapter, we have presented the MT with the architecture of models defined by OMG. Then we have discussed different kinds of transformation approaches, such as Relational/Declarative and Imperative/Operational/Constructive and Graph-based. After that, we have classified the transformation tools related to our work and we have highlighted the benefits and limitations of each tool.

Next, we have presented the multi-view modeling languages and frameworks. Multi-view modeling languages have been widely used in the CPS world, and many techniques have been proposed for supporting their definition. We have identified the main techniques. Then, we have presented solutions for the modeling for security and safety aspects, mentioned toolkits, such as TTool and SysML-Sec. Chapter 7, we use this toolkit for demonstrating the combination of security and safety parts with our method.

4 Combination Modeling Language

> *Precise language is not the problem. Clear language is the problem.*
>
> — **Richard Feynman**

Multi-view modeling approaches are used to separate domains in the development of a system, making it easier to manipulate its complexity and diversity. In the process of development of CPS, engineers also have to combine the separate views (models) into a uniform modeling view to conduct further analyses. Therefore, we propose a domain-specific modeling language to combine views, a *combination modeling language*.

4.1 Introduction

Complex systems made of various and heterogeneous subsystems. They have different aspects and each aspect has its own requirements and properties to be satisfied. MDE can handle complex systems at different levels and with diverse views. A model is an abstraction of the real world. This abstraction aims at facilitating an understanding of what the real world works. In the context of MDE, a software model enables a designer to reduce the non-essential complexity of an application by filtering out 'details'.

Multi-view modeling approaches are used to separate domains in the development of a system, making it easier to manipulate its complexity. Yet, in the process of development of CPS, engineers also have to combine the separate views into a uniform modeling view to conduct analyses. Besides, each view of the system is captured by a domain-specific modeling language. However, it is rare to see "one-size-fits-all" modeling language and/or design tools. So we look at the integration of multiple views into a single consistent one.

In this chapter, we explore a model-based approach for systems engineering that advocates the composition of several heterogeneous artifacts (also called views) into a sound and consistent system view model. Rather than trying to build the universal language to capture all aspects of systems, we bring together small subsets of languages to augment specific analysis capabilities while keeping a global consistency of all these small pieces of languages. Thus, we propose a domain-specific modeling language, called *Combination Modeling Language*. To make it concrete we use the example of Capella, a widely used design platform, which provides (among others) comprehensive support for functional analysis from the requirements down to the deployment of components. Yet, Capella does not provide direct support for non-functional features such as security analysis. On the other hand modeling languages dedicate to security and safety analyses do not provide direct support for functional analysis. In our example we consider SysML-Sec. Rather than trying to extend either Capella or SysML-Sec into more expressive languages to add the missing features, we use the Combination Modeling Language to extract relevant subsets of both languages and build a view adequate for conducting a security and

safety analysis of Capella functional models. Our language is generic enough to extract proper subsets of languages and combine them to build views for different experts. Moreover, it maintains a global consistency between the different views.

The chapter is structured as follows. At the beginning of this chapter, we present the Combination Modeling Language. Our language relies on patterns and corresponding transformation rules. Then, we present the abstract syntax of this language. Next, we give the definition of meta symbols and notations. We use EBNF to define rule expression. Next, we present the operators and their semantics, before concluding.

4.2 The Combination Modelling Language

The proposed modeling language is a dedicated (meta-) language to extend and enrich one DSML capability by combining other DSMLs. With this language, an integration engineer can explicitly capture combination scenarios at the language level. Combination patterns are used to specify different combination relationships. Specific operators are provided to build up Transformation Rule Expression (TRE), a set of TREs define a Transformation Rule Library (TRL) which specifies how to combine different (meta) model elements. Once the TRL is completed, it can be parsed by an automatic tool that we presented in the previous chapter. Afterwards, the tool can perform the transformation automatically. The concept of combination language is illustrated in Figure 4.1.

4.2.1 Specification

A specification consists of combination patterns and corresponding TRL. It defines what and how elements from different models are combined. Once it is specified, integration experts can share this specification thus allowing the reuse and tuning of TRL. As a specification can explicitly describe combination relationships, it can also be used to decompose models by bi-directional techniques for some decomposition needs.

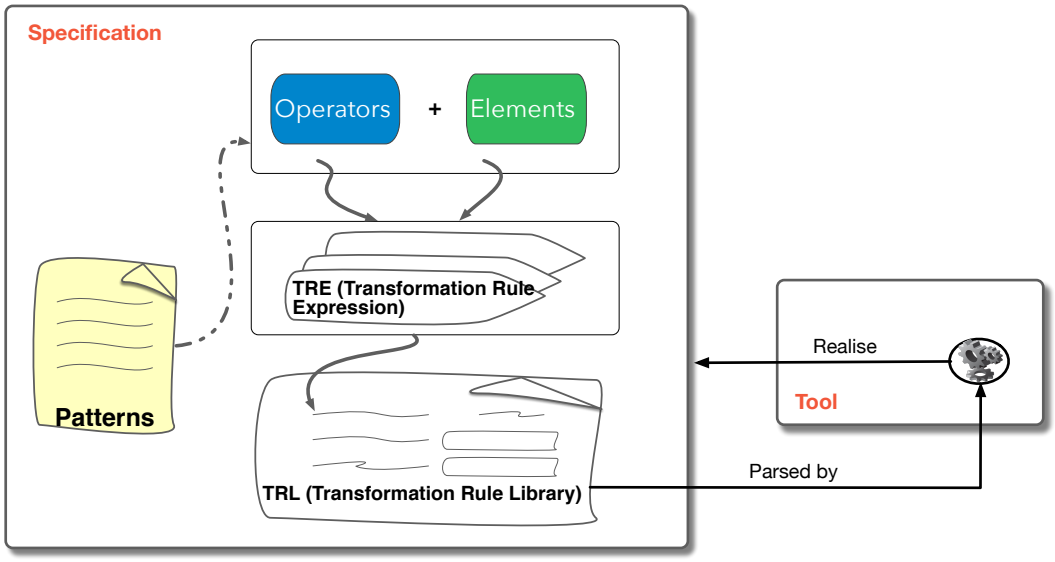


Fig. 4.1 Concept of Combination Modeling Language (CML)

4.2.2 Combination Patterns

Currently, we predefine a number of essential combination patterns, which provide all the declarations used in all the following examples. However, thanks to our language, designers can build other combination patterns depending on their problems and requirements. In that case, they have to define some new combination patterns in the form of TRL.

1. ***Association***: The *Association* pattern is the most common phenomenon and easier to understand. It is used to indicate one element which associates to another element and their related sub-elements (for example, its embedded element or associated attributes).

2. ***Removal***: The *Removal* pattern indicates the situation, where some elements are not considered for new models according to requirements.

3. ***Correspondence***: The *Correspondence* pattern indicates building an equivalence relationship among a set of elements.

4. ***Annotation***: The *Annotation* pattern aims to add information that do not exist in the model, for example, the dependency relationship among the model elements, and the nature of the elements.

4.2.3 Abstract syntax of CML

We give the abstract syntax of the Combination Modeling Language by using a metamodel expression in a class diagram (shown in Figure 4.2). The major element is a Specification that contains Patterns, Operators and TRL. A Specification requires importing at least two (meta) models. The imported (meta) models serve as a source of a set of candidate elements for the following operations. An operator selects the elements and their attributes from the imported (meta) models, and it also specifies how to combine selected elements.

Each operator contains a transformation expression that relies on a strict definition by Extended Backus Naur Form (EBNF). EBNF are extensions of the basic Backus Naur form (BNF) meta syntax notation. Symbols are used to construct the TRE. For instance, for adding security properties to a logical component of Capella, it has to specify the corresponding element and their related attributes in TTool by using TRE.

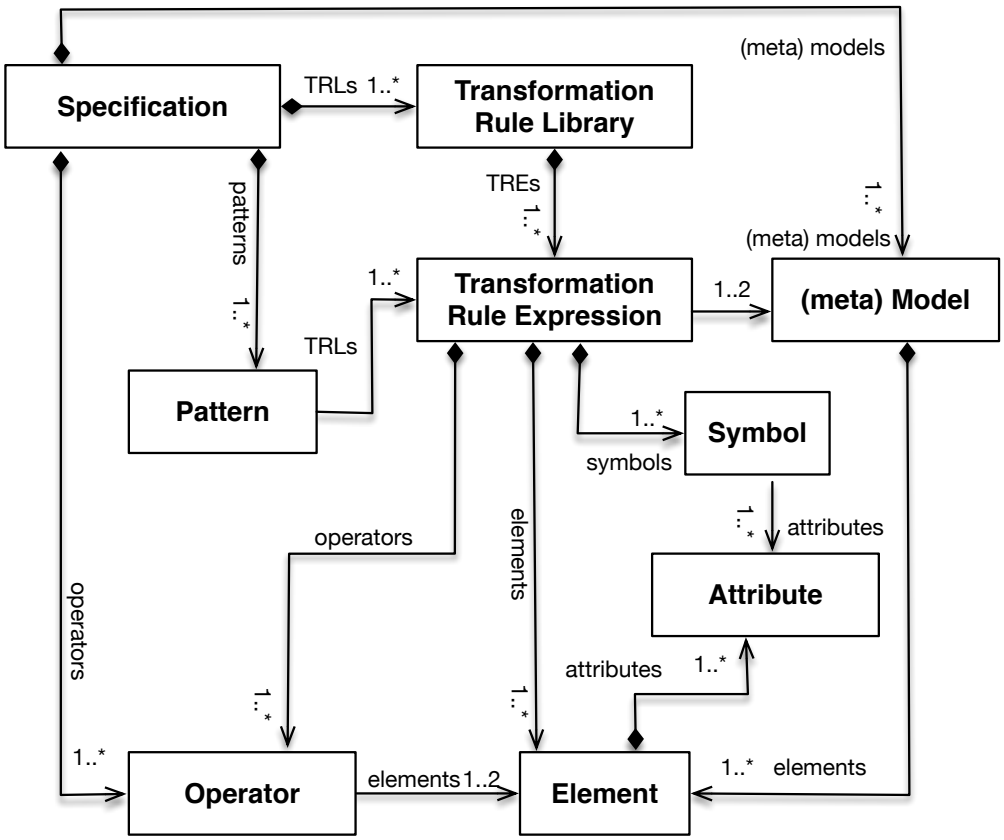


Fig. 4.2 A simplified view of abstract syntax of combination language

4.2.4 Meta symbols and notations rule expression

In this subsection, we firstly introduce some notations and meta symbols which are fundamental elements for constructing the well-defined Transformation Rule Expressions (see table 4.1). We use EBNF to strictly define non-ambiguous Transformation Rule Expression. EBNF is a notation technique for context-free grammars which is often used to describe the syntax of languages [182].

Symbol	**Meaning**
Γ	Beginning of transformation Rule
;	End of transformation rule
:	Separate elements
$\rightarrow$	Transforming
<>	Parent node
{ }	Attribute
[ ]	Optional value
|	Alternative
+	Object to be created
¬	Ignoring
@	Tagging

Tab. 4.1 Symbols of transformation rule expression

The detailed literal meaning of symbols is given below:

1. A Transformation Rule Expression begins with "Γ" and ends with ";". The symbol "Γ" [1] also can be used as a Boolean function. If $\Gamma(source, target)$ is true, it means there is a relationship between *source* and *target*.

2. The symbol "$\rightarrow$" indicates a transforming action.

3. A transforming action contains the source elements which in the left side of "$\rightarrow$" and the target elements in the right side. A simple example is given below:

$$\Gamma < parent > source \rightarrow target;$$

[1] *In the real TRL file, we use "#" instead of symbol "Γ" to facilitate input.*

i.e. We intend to transform a "*source*" object into a "*target*" object, "*parent*" points at the parent of the source object (if it has one).

4. Symbol ":" separates each part of TRE.
   e.g, $P_{ort}\{Direction\} : \{Type\} : \{Secure\}$
   It means there is an element P_{ort}. This element has three attributes, "Direction", "Type" and "Secure". We use ":" to separate these attributes.

5. An angle bracket "<>" encloses the parent node if the element has one or more parent nodes.

6. A curly bracket "{ }" encloses some attributes.

7. A square braces "[ ]" delimits optional elements.

8. The alternative value is separated by a pipe "|". For example, The *port* has a directional attribute called *Direction* which could be *in* or *out* shown as:

$$P_{ort}\{Direction[in|out]\}$$

9. Symbol "@" indicates tags used to add some extra informations such as dependency and nature. The extra information is handled in a similar way as operational values: enclosed in [ ]; separated by ",". For example, $P_{ort}@[ModelA, Security]$ means element Port belongs to ModelA and is used for Security purpose (view). In such a situation, it makes tools automatically display or hide the element P_{ort} which is in modelA and for security view in the following process.

With those symbols, we can build up several TREs. Some more detailed examples of Transformation Rule Expressions are shown in the listing 4.1.

Remarks:

- The symbol Γ in function expressions does not have the same meaning as when used at the beginning of transformation rules. To distinguish the function and the transformation rules, the formula is underlined.

- The relationships are defined in subsection 4.2.6.1.

4.2.5 Abstract syntax of rule expression in EBNF

As we mentioned in the previous subsection, the TRE consists of one or more sequences of symbols. We define here the concrete syntax in EBNF.

$\langle expression \rangle ::= \Gamma \langle term \rangle \rightarrow \langle term \rangle ; |$
$\langle expression \rangle : \langle term \rangle ;$

$\langle term \rangle ::= \langle element \rangle |$
$\langle operator \rangle \langle element \rangle |$
$\langle element \rangle \langle operator \rangle \langle element \rangle$

$\langle operator \rangle ::= \text{'@'} \mid \text{'+'} \mid \text{'}\neg\text{'} \mid \text{'}\rightarrow\text{'}$

$\langle element \rangle ::= \langle element \rangle (\langle attribute \rangle \mid \langle optional\ value \rangle)$

4.2.6 Operators and Semantics

The context-sensitive syntax and the operational rules could also be considered as semantics instead of syntax. For example, the context-sensitive syntax is called static semantics in the UML specification documents from OMG [70]. In our case, it specifies how an instance of a construct can be meaningfully connected to other instances.

In order to make the TRE more clear and precise, we firstly present a set of relationships definitions. This should help the reader understand the semantics of operators. they may also help users understand the TRE examples shown later.

4.2.6.1 Definition of relationships

Here we define a set of essential relationships, which are used to describe the logical links between two elements of model. We use set theory. Capital symbols (e.g., A, B) usually represent sets of elements, while lower case symbols are elements of those sets (e.g., $a, b, c \in A$).

- **Relationship**: When we identify a relation $\mathcal{R}$ between a and x, we denote $(a, x) \in \mathcal{R}$ or $\mathcal{R}(a, x)$ or $a\mathcal{R}x$ depending on the context. For each relation we assume the existence of a Boolean function such that $\mathcal{R}(a, x)$ if there exist a relation between a and x. When such a relation is identified, then the transformation becomes possible, from a to x:

$$\mathcal{R}(a, x) \implies \underline{\Gamma(a, x)}$$

  Note: We use relations here to transform a source into a target (not symmetric) but it also allows the reverse transformation if we want to trace back to the initial metamodel.

- **Equivalence**: $\mathcal{E}(a, x)$ is a Boolean function that is true if and only if a is semantically equivalent to x. Similarly, an equivalence is stronger than a mere Relationship, it may also lead to a transformation and therefore

$$\mathcal{E}(a, x) \implies \mathcal{R}(a, x) \implies \underline{\Gamma(a, x)}$$

- **NotIn**: if $X = \{x, y, z\}$ is a set, we lift the Relationship and Equivalence to sets to identify sets of elements that are neither in relationship nor equivalence.

$$\neg\mathcal{R}(a, X) \implies \forall x \in X, \neg\mathcal{R}(a, x)$$

  similarly,

$$\neg\mathcal{E}(a, X) \implies \forall x \in X, \neg\mathcal{E}(a, x)$$

  Obviously, no transformation is possible in such cases.

4.2.6.2 Operators

(a) ***Transforming operator:*** We use $\rightarrow$ indicates transforming operator, for example, $a \rightarrow x$ means that we transform a into x.

(b) ***Creating operator:*** In the case of creating a new attribute, the attribute name is in the curlybrackets with plus "{ }+". For example, $\Gamma a \rightarrow x\{t\}+$ means that we transform a to x while adding attribute t. As an example, let us consider

$$\Gamma Port_{fun} \rightarrow Port_{comm}\{type[data|event|dataevent]\}+;$$

A function port $Port_{fun}$ must be transformed into a communication port $Port_{comm}$, while creating a new attribute *type*, a enumerate with three possible literal values (data, event, dataevent).

(c) ***Ignoring operator:*** This operator is used to ignored attributes or objects. It is denoted with symbol "¬" in front of the object. For example, $\neg a$ means a is **NotIn** object for a set B, in other words, we can neither find out ***Relationship*** nor ***Equivalence*** between a and B. For example,

$$\Gamma \neg P_{ort}\{ordering\} \rightarrow P_{ort};$$

or simplify,

$$\Gamma \neg P_{ort}\{ordering\};$$

the attribute "ordering" of P_{ort} of original model is not existing in target P_{ort} of target model. Therefore, *Ignoring operator* shows this transformation rule. The parser will get this information and ignore transforming this attribute afterward.

(d) ***Tagging Operator:*** This operator is used for tagging the nature of an element attribute. As an example: $P_{ort}@[ModelA, Security]$ presents two attributes of element P_{ort} with two tags. One is *ModelA*, indicating that the element P_{ort}

belongs to *ModelA*. In other words, It represents a dependency relationship between this element P_{ort} and element *ModelA*. Another is *Security*, representing an element P_{ort} for Security purpose. It would be used to catalog the elements for displaying or fast selecting purpose.

4.2.7 Operational Transformation Rules

TRE is used to represent the transforming relationships. It would be used to guide the integration engineer and to allow automatic parsing by the transformation engine. We explain how it does work by using some more detailed examples of TRE. Please refer to the TRE table which is in the listing 4.1.

On line 1 of this example, we firstly transform an element *port* (it has direction attribute) of source model to an target object element *port*, adding a new attribute *Type* with three possible values (date, event or dataevent). These "type value" can be recognised by target model's DSML and the their support tool. On line 2, it is similar to the previous one, but the object element *function* has a parent node called *CompositeComponent* which is enclosed in a pair of angle brackets.

Line 3 shows an ignored element, in which the source element cannot find a corresponding one in the object model, or the source element is not needed by the object model. Finally, lines 4 and 5 show *Equivalence* relationships between the source element and the object element, in other words, a set of one by one transformations which transform "Ex_{fun}" to "$connection$", "Source" to "source" and "Target" to "target", respectively.

```
1 ΓPort{Direction[in|out]} –> PrimitivePort {Direction [in|out]}:{Type[data|event|data event]}+;
2 ΓFunction –> <CompositeComponent>PrimitiveComponent{AccessIdentifier}+:{InitialValue
    }+:{Type[Natural|Boolean]}+;
3 Γ¬Port{ordering};
4 ΓEx_fun{Source} –> <connections>:connection:{source};
5 ΓEx_fun:{Target} –> <connections>:connection:{target};
```

Listing 4.1 The example of Transformation Rule Expressions

4.3 Conclusion

In this chapter, we present our language, Combination Modeling Language, is used to combine different modeling views.

Our language is rule based, the transformation specification consists of a set of transformation rule expressions. The transformation engine/tool produces the target model from the source model according to MT expressions. The input models must be valid, as we do not check it.

Although we have applied this language to several examples, there are still some remaining drawbacks to overcome. The major one is that the traceback function is not yet implemented automatically.

In our future work, we plan to have a graphical syntax to simplify the process for users. We also have to implement a mechanism to go back from the target model back to the initial modeling elements. This is sketched in our workflow with a dotted arrow. Furthermore, we have not studied the completeness of our language. We implemented operators that we met in our case studies, that comes directly from our industrial partners.

5 Support tool

> *The Pareto principle (also known as the 80/20 rule, the law of the vital few, or the principle of factor sparsity)states that, for many events, roughly 80% of the effects come from 20% of the causes.*
>
> — **Vilfredo Pareto**
> Italian economist

As we introduced in previous chapters, we have proposed a DSML which is dedicated to the composition of heterogeneous (view) models through their meta-models. We have called this new modeling language *Combination Modeling Language*.

In this chapter, we present the models combination workbench, which reduces the effort of models combination by syntax parser and adaptors. Instead of doing combination manually, a support tool (CMT) is designed to accomplish the process automatically. It can ensure the correctness of generating a new combined (meta) model and export the new (meta) model in an easy way.

5.1 Introduction

We have defined the syntax and grammar of combination modeling language. However, a language needs an operational environment to support its execution. Just like for a program, the (source) code needs a compiler to obtain an execution file so as to be executed. Instead of combining models manually, a *support tool* is required to accomplish the process automatically. Our tool is called Combination Modeling Tool (CMT). Once the integration engineers have prepared the TRL; they must import the source models and TRL, then the CMT combines models automatically according to TRL.

Furthermore, we know that manually combining models is error-prone and wastes a lot of time. The integration engineers have to pay much more attention to building a new model according to rules. Any mistake can lead to unpredictable results, and it is difficult to detect those mistakes. The CMT should be able to detect grammar errors of TRE.

In addition, to simply the deployment of our tool, we have developed a web front-end. That allows integration engineers to work with a friendly interface and a thin, platform-independent, client.

We summarise here the requirements of CMT:

- Importing source models and exporting target (combined) model
- Detecting error of TRE
- Executing process are automatic and effective
- Easy deployment and multi-user support

5.2 Architecture

Web-based architecture has been selected for fulfilling easy deployment and multi-user. By leveraging web-based architecture strength, we only focus on server-side.

Moreover, we need to maintain the server-side program when fixing bugs and updating.

Node (also called Node.js) is one of the better-known frameworks and environments that support server-side JavaScript development. A node server process usually invoked from the command line, runs single-threaded, yet can serve many clients concurrently.

Vue.js is one of the most used JavaScript projects in recent years due to its flexibility and adoption by a large community. Vue.js can be used as a decorator to build user interfaces.

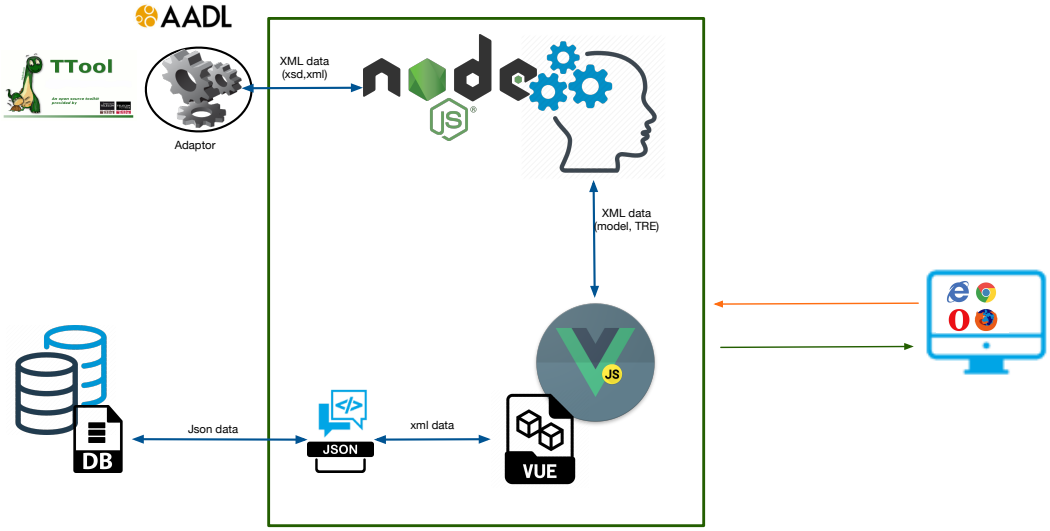


Fig. 5.1 Architecture of combination tool

We illustrate the architecture of the combination tool in Fig 5.1. The integration engineers (end-user) open a web browser with an address of service. Loading source models and preparing TREs, then end-user sends the request to web server applications. The engine of combination (javascript program) runs according to input data (XML format). When the engine gets the task done, it returns the results (models) to the web client (end-user) and saves models data to the server's database as a JSON file. Depending on the request, if the end-user requires a concrete MT, the engine invokes an adaptor for adapting object models such as AADL (osate) or TTool. By doing so, the end-user gets an XML file of the model.

5.3 Instrumentation

5.3.1 Meta-model level

Starting tool. The Graphical User Interface (GUI) is shown in Fig 5.2. The exact look may change depending on the actual browser used.

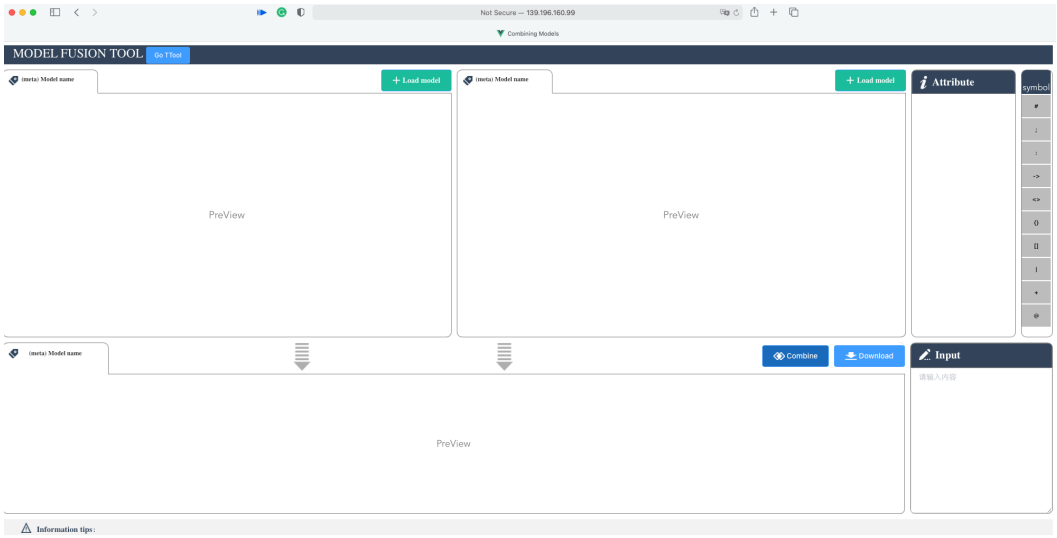


Fig. 5.2 Web-based GUI

Loading (meta-) models from xml files. Users can load (meta-)models by clicking "+load model" button (green), and then a pop-up window appears, prompting you to select a xml file for the metadata import(see Fig 5.3). The (meta-) models is composed of structural data, attributes and values. All of data are loaded with model files.

Models appear in the model's zone (red rectangle), the topology of components and their captions are illustrated as well (see Fig 5.4).

Attributes and values: When the focus moves on one element (component of the model), this element attributes and their values are loaded in the "Attribute" zone (yellow rectangle). Users can change the value by combo, choice box, or text fields.

TRE Box: "Input" zone (green rectangle) is a container of TRE. People can write rules in this zone and input symbols by clicking symbol buttons (above input zone).

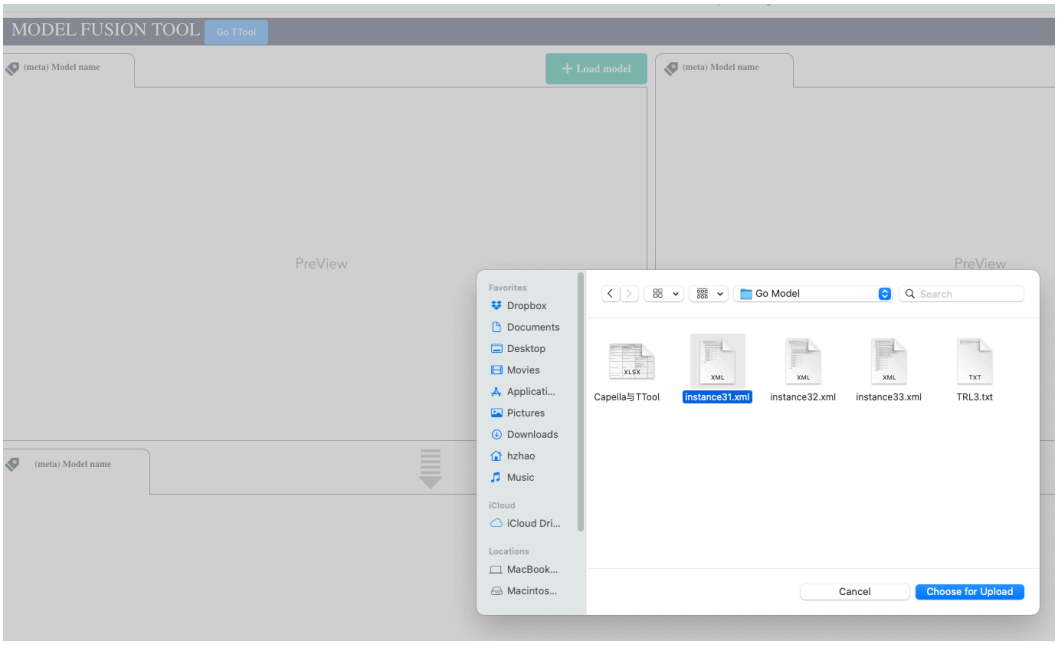


Fig. 5.3 Loading (meta-) models files

Result: If source models have been loaded and TREs is ready, the user can click "combine", then obtaining the result model, which appears in the result zone (violet zone). There are two choices. One is saving the result model to a local folder by clicking "download" button. Another way is to go to "Specific Model Combinations" by clicking the "GO TTool" button (on the top bar).

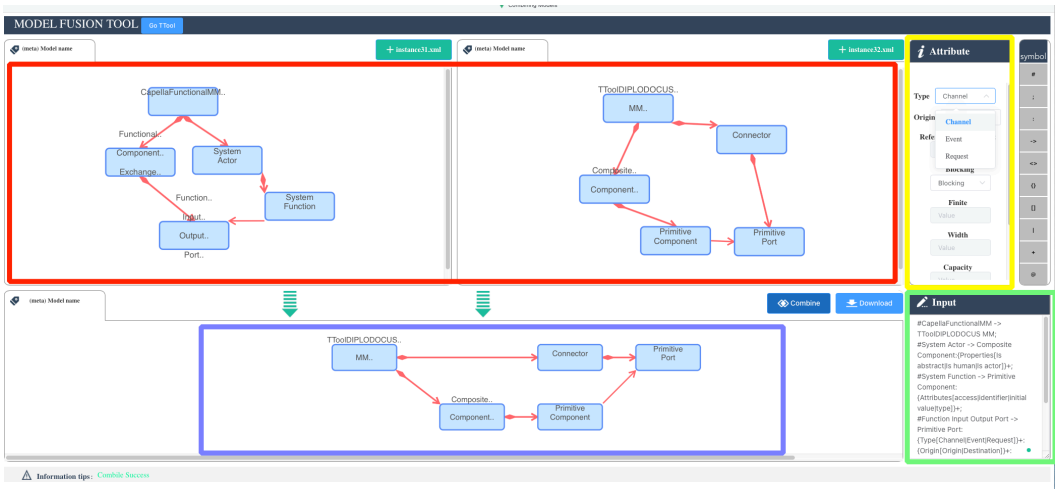


Fig. 5.4 Functions and zones

5.3.2 Specific model level

We mentioned that the combination process contains two parts: at the different model levels, meta-model and concrete model. The specific model combination is at the concrete model level. The interface is shown in Fig 5.5

"Load model" button (①) guides people to load a functional model (e.g., Capella model) conform to its meta-model.

Chose a combined meta-model, users have to load a combined meta-model to guide the concrete model combination. "Load model" button (②) loads a combined model from a local file. "Import" button (③) imports a combined model from the previous step (saved in the server).

Combine, this button (④) is to combine models and return a specific model.

Download, people download the result model by clicking "Download" button (⑤) and save in a local file.

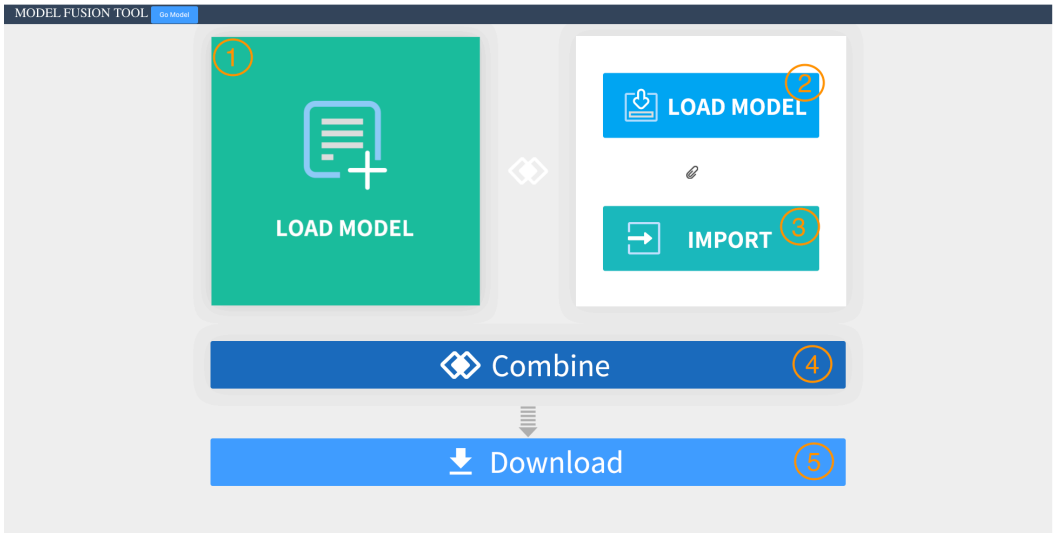


Fig. 5.5 Specific model

5.4 Tool comparison

We evaluate how well the various tools may perform to handle large and complex transformations. The result and information are from our practice, published papers

and online documents of user's feedback. Thus, this is not a formal evaluation. It just provides an overall picture of the potential of each tool to work with large and complex models.

To measure the complexity of a transformation, metrics such as the number of expressions, transformation rules (e.g., 60) and elements (e.g., 20 of one model). Our comparison was done on the same PC with an Intel i5 with 8Gb of memory, both Fedora Linux 64bit and MacOS 10.5. We take the installation of the Oracle Java 8 virtual machine into account as a part of the deployment. In Table 5.1, we evaluate our tool according to these metrics:

- Memory Usage, Disk Usage: excluding JVM and saved data.
- Time of execution
- Independent: tool is not a plugin and depends on some basic environment such as Eclipse.
- Pr Lang: programming language of the tool.
- Installation: Hard means that it is hard to install the tool. Medium means that it is not hard, including tool and support environment. Easy means that it is easy to install the tool but needs basic knowledge.
- Time of deployment: How long to deploy the tool.
- Deployment: Local means that the tool is installed and runs in a local machine. Web-based means opening with browser.
- Multi-user: If the tool supports multiple users.

5.5 Conclusion

In this chapter, we have presented our support tool CMT which makes the combination process automatic. After collecting the requirements for the support tool, we have proposed a web-based architecture and related technologies to fulfil those

Tools	Memory Usage	Disk Usage	Time of execution	Independent	Pr Lang	Installation	Time of deployment	Deployment	Multi-user
UML-RSDS	∼	60 MB	1 s	Y	Java	Medium	20 m	Local	N
JTL	512 MB	70 MB	>2 s	N	Java	Medium	>40 m	Local	N
Tefkat	512 MB	∼	>2 s	N	Java	Medium	∼	Local	N
PTL	512 MB	∼	>3 s	N	Java	Medium	>40 m	Local	N
mediniQVT	512 MB	∼	∼	N	Java	Medium	>40 m	Local	N
QVTR-XSLT	512 MB	∼	∼	N	Java	Medium	>40 m	Local	N
CMT	100 MB	10 MB	1 s	Y	Node.js	Easy	15 m	Web-based	Y
TXL	1 G	50 MB	>3 s	N	Java	Medium	∼	Local	N
Xtend	512 MB	100 MB	1 s	N	Java	Medium	>30 m	Local	N
QVTo-Eclipse	512 MB	∼	>2 s	N	Java	Hard	>40 m	Local	N
MetaEdit+	512 MB	90 MB	1 s	Y	MERL	Easy	10 m	Local	N
Kermeta2	512 MB	∼	>2 s	N	Java	Medium	>30 m	Local	N
Melange	512 MB	100 MB	2 s	N	Java	Medium	>30 m	Local	N
JQVT	512 MB	50 MB	2 s	N	Java	Medium	>30 m	Local	N
Together	1 G	50 MB	1 s	N	Java	Easy	>30 m	Local	N

Tab. 5.1 Evaluation of MT tools on Consumption, Deployment, Complex and Performance

requirements. Moreover, we have provided an instrumentation of CMT that can guide integration engineers to use this tool. We have evaluated the MT tools from resource consumption, deployment, complex and performance by our practice and the published papers to outline the strong points of our tool. In the following chapters, CMT is used to run different case studies inspired by models of our industrial partners.

6 Bridging Capella with AADL for schedulability analysis

> *Education is what remains after one has forgotten everything he learned in school.*
>
> — **Albert Einstein**
> Physicist

In this chapter, we explore a model-based approach for systems engineering that advocates the composition of several heterogeneous artifacts (called views) into a sound and consistent system model. Relying on the proposed modeling language CML. Thus, rather than trying to build a universal language able to capture all possible aspects of systems, the proposed language proposes to relate small subsets of languages to offer specific analysis capabilities while keeping a global consistency between all joined models. We demonstrate the interest of our approach through an industrial process based on Capella, which provides (among others) a large support for functional analysis from requirements to components deployment. Even though Capella is already quite expressive, it lacks support for schedulability analysis. AADL is also a language dedicated to system analysis. If it is connected to backend tools for schedulability analysis, it lacks an extensive support for functional analysis. Thus, instead of proposing ways to add missing aspects in either Capella or AADL, we would rather extract a relevant subset of both languages to build a view adequate for conducting schedulability analysis of Capella functional models. Finally, our combination language is generic enough to extract pertinent subsets of languages and combine them to build views for different experts. It also helps maintaining a global consistency between different modeling views.

6.1 Introduction

In our approach, we combine meta-models, while keeping each language (or tool) isolated.

Our language combines two modeling languages by defining rules.

To validate the contribution of the proposed approach, SysML and AADL are selected as two target languages, and their support environments (tools) Capella/Arcadia and OSATE2[1] are used to show the design of example system.

This chapter is organized as follows. In section 6.2, we first identify the workflow of the proposed approach. Then, we present the reinforced language and the operators in section 4.2. In section 6.3, we apply these operators on functional and physical views. To evaluate the proposed formal approach, a train traction control system is used as an illustration in section 6.4.

6.2 Overview of our approach

In this section, we describe the proposed workflow using an example based on Arcadia and AADL, as shown in Figure 6.1 [183]. Arcadia is well adapted to describe how to allocate functions, while AADL focuses on the concrete execution behaviors of components. In this chapter, we use transformation to enhance Arcadia with the schedulability analysis features of AADL. The transformation is performed by proposing a set of rules and operators to specify the relationships at the M2 level. Those relations are used for MT purpose and a set of all relationships is called Transformation Rule Library (TRL). More specifically, these rules are used to establish a relationship between Arcadia and AADL metamodels in a TRL. We assume that Arcadia and AADL define concepts that can be put in relation thanks to the proposed rules.

In Figure 6.1, the green part represents the concepts borrowed from Arcadia while the red part represents the extensions borrowed from AADL (e.g., period and exe-

[1] http://osate.org/index.html

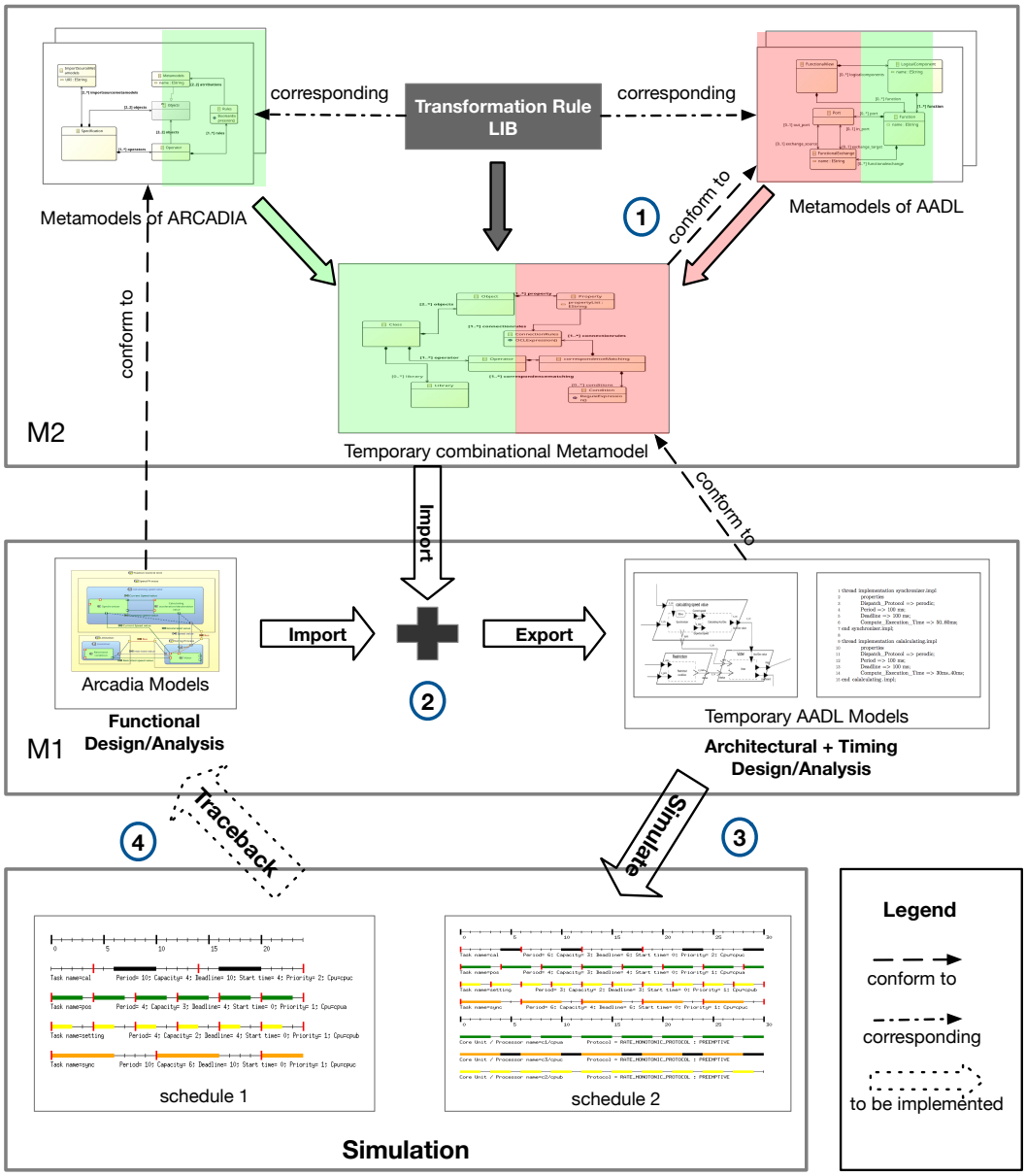


Fig. 6.1 Overview of Workflow

cution time). Then, the elements of metamodels are chosen manually depending on the expectations. The workflow has four steps. In step one, we can get a temporary combinational metamodel (TCM) at run time by using TRL once the equivalence relations between the two metamodels have been settled. In step two, the TCM can be used to combine an AADL model with elements of an Arcadia model, then the new AADL model can be exported into OSATE for further editing. In step three, the Cheddar analysis tool [184] is used to conduct schedulability analysis. This tool can be used to detect design flaws, time and resources conflicts. In step four, the

results are mapped back onto the initial Arcadia model in order to help the designer enhance his/her model.

6.3 Transformation Rule Library (TRL)

In previous content, we mentioned TRE, which plays an important role in the transformation process. Hence, in this section, we will show how the TRL is constructed by a set of TREs. We also respectively present functional view and physical view in Arcadia (SysML) and AADL. Each view contains one or more metamodels.

6.3.1 Functional view

6.3.1.1 Logical components in Arcadia

The logical components in Arcadia contain a set of member elements, such as logical component containers, functions, ports, and functional exchanges. In the Arcadia, functional diagrams consist of a set of SysML blocks and its interactions, named *Logical components*; The notion of logical components enables better expression of system engineering semantics compared to SysML, and particularly, reduces the bias towards software. SysML block definition diagrams (BDDs) and internal block diagrams (IBDs) are assigned to different abstract and refined layers, respectively. The definition of a block in SysML can be further detailed by specifying its parts; ports, specifying its interaction points; and connectors, specifying the connections among its parts and ports. This information can also be visualized using logical components in Arcadia. In the definition of logical component, we present a metamodel of an instance of logical components.

Definition: Logical Component (LC)
A logical component (**LC**) is a 5-tuple,

$$\mathcal{LC} =< C_{omp}, F_{un}, P_{ort}, Ex_{fun}, M_{cf} >$$

, where

$$C_{omp} = \bigcup_{i} F_{un}^{i}$$

is a logical component container which contains a set of functional elements.

F_{un} is a finite set of functional blocks including their name and id attributes. P_{ort} is a finite set of functional ports including directions and allocation attributes. $Ex_{fun} \subseteq P_{ort} \times P_{ort}$ denotes a finite set of functional exchange (connection) between two functional ports, in which it must be a pair of one source and one target. $M_{cf} : F_{un} \rightarrow C_{omp}$ allocate functions to a logical component container.

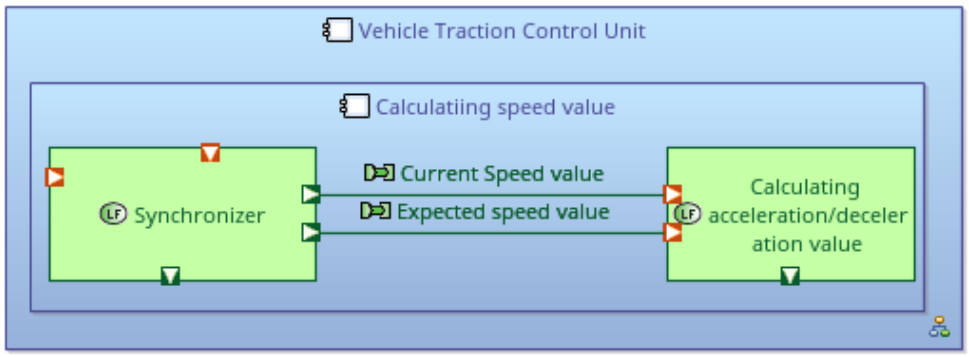


Fig. 6.2 An example of functional view of vehicle traction control unit in ARCADIA

As shown in Figure 6.2, there is a functional instance model of a part of a vehicle traction control unit in ARCADIA. The blue rectangle is named logical component in Arcadia, a *logical component container*, C_{omp}. The green rectangles are functions F_{un} which are contained by C_{omp}. The element M_{cf} represents this allocation relationship between logical component containers and functions $M_{cf} : F_{un} \rightarrow C_{omp}$. The deep green square with the white triangle is the outgoing port (P_{ort}), which connects an incoming port (P_{ort}) that is drawn as a red square with white triangle, and the green line is the functional exchange between two functional ports (Ex_{fun}).

6.3.1.2 The metamodels of software in AADL

AADL is able to model a real-time system as a hierarchy of software components, predefined software component types in the category of the components such as thread, thread group, process, data, in which subprogram are used to model the software architecture of the system.

Definition: Software Composition (SC)

A **SC** is a 4-tuple:

$$\mathcal{SC} =< Type, Port, Connection, Annex >$$

where $Type$ specifies the type of components (e.g, system, process, thread). $Port$ is a set of communication point of component. Port could be different types such as **data** port, **event** port and **data event** port. And, port can specify the direction such as **in** port, **out** port, **in out** port. $Connection$ is used to connect ports in the direction of data/control flow in uni- or bi-directional. $Annex$ is defined for the refinement of component. In this chapter, we used hybrid annex to explicitly describe both discrete and continuous behavior of the train traction control system.

6.3.1.3 Hybrid annex

We use the HA to declare both discrete and continuous variables in the *Variables* section, and the initial values of constants are given in *constant* section. *Assert* is used to declare predicates which may be used with invariants to define a condition of operation. The *behavior* section is used to specify the continuous behavior of the annotated AADL component as concurrently executing processes, and has a continuous evolution — a differential equation specifies the behavior of a physical controlled variable of a hybrid system. The communication between computing units and physical components are an essential part of a hybrid system, communication between physical process uses the channels declared in the *channel* section, and communicates with an AADL component that relies on the declared ports in the component type. Continuous process evolution may be terminated after a specific

time or on a communication event. They are invoked through timed and communication interrupt, respectively. A timed interrupt preempts the continuous evolution after a given amount of time. A communication interrupt preempts the continuous evolution whenever a communication takes places on any of the named ports or channels. The Hybrid Annex of AADL has no direct equivalent in SysML.

Definition: Hybrid Annex (HA)

A *Hybrid Annex* is a 8-tuple:

$$\mathcal{HA} =< Ass, Ivar, Var_{hd}, Cons_{hd}, P_{roc}, ChP, Itr, B_{itr} >$$

where Ass is a finite set of assert for declaring predicates applicable to the intended continuous behavior of the annotated AADL component. $Ivar$ is associated with assert to define a condition of operation that must be true during the lifetime. Var_{hd} is a finite set of discrete and continuous variables. $Cons_{hd}$ is a finite set of constants which must be initiated at declaration. P_{roc} is a finite set of process that are used to specify continuous behaviors of AADL components. ChP is a finite set of channels and ports for synchronizing process. Itr is a finite set of time or communication interrupts. $B_{itr} : Itr \rightarrow P_{roc}$ binds interrupts to related process.

6.3.1.4 Functional elements transformation rules

The table 6.1 shows the correspondence between AADL and Arcadia elements. The additional attributes column are the attributes to be created during the transformation. According to this table, we can write the transformation rules to transforming Arcadia to AADL on the functional parts.An example as below (listing 6.1 [183]):

```
1 ΓC_omp -> Type[ system|process]:{Runtime_Protection[true|false]}+;
2 ΓF_un -> Type[abstract|thread]:{Dispatch_Protocol[Periodic|Aperiodic|Sporadic|Background|
      Timed|Hybrid]}+;
3 ...
```

Listing 6.1 Functional elements transformation rules example

Arcadia	**AADL**	**Additional attributes**	**Notation**
Logical component container (C_{omp})	System,Process	{Runtime_Protection[true|false]}+	@[function|AADL|scheduling]
Function (F_{un})	Abstract, Thread	{Dispatch_Protocol[Periodic|Aperiodic|Sporadic|Background| Timed|Hybrid]}+	 @[function|AADL|scheduling]
Port (P_{ort})	Port	{Type[data|event|data event]}+	@[function|AADL|scheduling]
Functional Exchange (Ex_{fun})	Connection	$\varnothing$	
$\varnothing$	Annex	{Type[abstract| thread]}:{annex}+	@[function|AADL|scheduling]
Physical Node (N_{ode})	Device,Memory,Processor,Bus	{Dispatch_Protocol}+:{Period}:{Deadline}+:{priority}+	@[physic|AADL|scheduling]
Physical Port (PP)	$\varnothing$	$\neg PP$	@[physic|AADL|scheduling]
Physical Link (PL)	Bus/BusAccess	{Allowed_Connnection_Type}+:{Allowed_Message_Size}+: {Allowed_Physical_Access}+:{Transmission_Time}+	 @[physic|AADL|scheduling]

Tab. 6.1 Functional and Physical elements correspondence table

6.3.2 Physical view

6.3.2.1 Execution platform in AADL

Processor, memory, device, and bus components are the execution platform components for modeling the hardware part of the system. Ports and port connections are provided to model the exchange of data and event among components. Functional and non-functional properties, like scheduling protocol and execution time of the thread, can be specified in components and their interactions.

Definition: Execution Platform (EP)

A **EP** component is defined as a 3-tuple:

$$\mathcal{EP} =< EC, BA, C_{onn} >$$

, where
EC defines the execution component such as processor, memory, bus and device. BA defines the BusAccess which is interactive approach between **bus** component and other execution platform components. $C_{onn} \subseteq EC \times EC$ denotes a finite set of connections between two components connected via a bus device.

6.3.2.2 Physical components in Arcadia

The physical component in Arcadia consists of physical Node, Port and Link. The Physical Port and Link corresponds to port and bus connections in AADL. There are some choices when a physical Node is translated to AADL such as device, memory, and processor, hence the designer has to point out what type of target component during transformation by using transformation rule express.

Definition: Physical Components (PC)

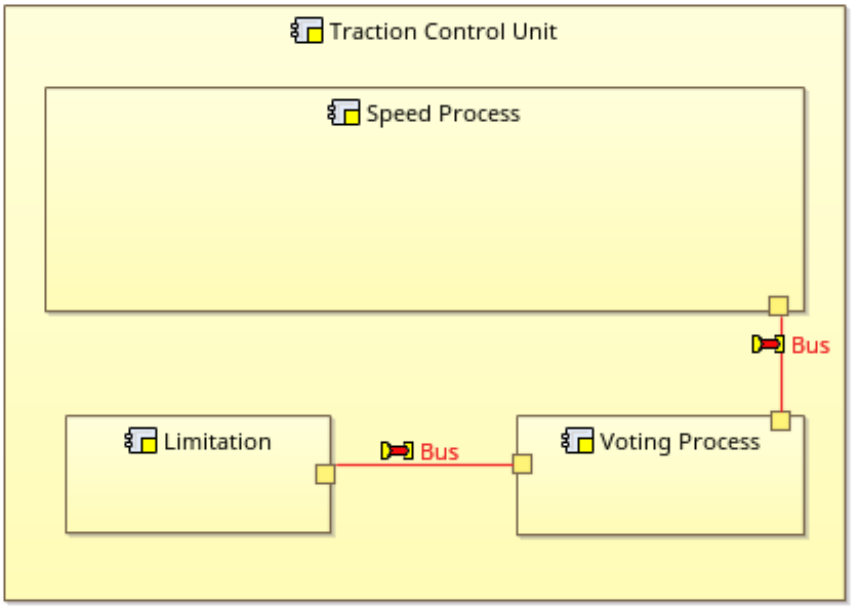


Fig. 6.3 An example of physical view of vehicle traction control unit in ARCADIA

A physical components is a 3-tuple,

$$\mathcal{PC} =< N_{ode}, PP, PL >$$

where, N_{ode} is a execution platform, named *node* in Arcadia. It could be different type of physical components (e.g, processor, board). PP is the physical component port. PL is physical link, which could be assigned a concrete type such as *bus*.

Figure 6.3 shows a part of physical instance model of vehicle traction control unit in Arcadia. The yellow parts are physical nodes (N_{ode}) and the red line is the physical link (PL) named bus in this case. The bus connects to two physical ports (PP), which are the small squares in dark yellow.

6.3.2.3 Physical elements transformation rules

According to the table 6.1, we can write transformation rules for physical elements. Listing 6.2 [183] is shown as a part of the code to transform the physical component from Arcadia to AADL.

```
1 Γ$N_{ode}$ –> [$Device|Process|Memory|Bus$]:{Dispatch_Protocol}+:{Period}:{Deadline}+:{
      priority}+;
2 Γ$PP- >\neg$ PP;
3 Γ$PL- >$ Bus/BusAccess:[{Allowed_Connnection_Type}+:{Allowed_Message_Size}+|{
      Allowed_Physical_Access}+:{Transmission_Time}+];
```

Listing 6.2 Physical elements transformation rules example

What we have to explain is the physical link part (see line 3). The Bus device could be a logical resource or hardware component. Hence, the bus device has different properties depending on the role. When the bus is considered as a logical resource, it contains the properties *Allowed_connection_type and Allowed_Message_Size*. When the bus is hardware, it contains
Allowed_Physical_Access and Transmission_Time. Therefore, we write the rules that either

$$\{Allowed\_Connnection\_Type\}+ : \{Allowed\_Message\_Size\}+$$

or

$$\{Allowed\_Physical\_Access\}+ : \{Transmission\_Time\}+$$

6.3.2.4 Binding

In AADL, the pre-defined property set includes *deployment_properties*, which is used to describe the deployment relationship from the software component to execution platform component. Here, we define *bind* as an operator between application software components and execution platform components. (Binding)

In the system with multiple processors, *bind* is a tuple:

$$\mathcal{B} =< SFC, EP, B >$$

, where

1. SFC is a set of application software components;

2. EP is a set of hardware components;

3. B is a binding relation between software components and hardware components.

Arcadia presents a methodology to define, design, analyze and validate systems with software and hardware architecture. It provides a hierarchical structure with logical/functional components, physical components. Logical components deploy into physical components. Here, we define $allocate$ as an operator to describe the relationship of functional components with physical components. An $allocate$ operator is a tuple:

$$< C_{logi}, C_{Phy} >$$

6.3.2.5 Port and connection

Ports are the logical connection points between different components. AADL defines three types of component ports, for the data transmission by data port, control information by event port and both of them by data event port. There are two directions of port, input and output. The output port is connected to the input port to constitute the port connection. Arcadia defines only directions (in and out), in which the type of port is omitted. Hence, we ought to add the type attribute to complete the form in AADL when doing a transformer. The translating rule writes as an example in list 6.2 at line 1. It means the transformation between one functional port P_{ort} of Arcadia and a port of AADL (within the parent node <*feature*>). The direction attribute and its values *in or out* can transfer to counterpart directly, and the data type is additional option, it will be added with its values *data, event, data event*, denoted *{Type[data|event|data event]}+*. For some attribute which does not exist in AADL such as *ordering* (see list at line 3), we can write one line with the symbol ¬, it means the ordering attribute will be ignored for transformation.

A connection is an interaction between two objects via ports. One connection must have only one *source* and one *target*. It is the same in both Arcadia and AADL. An example of transformation expression is shown in line 4.

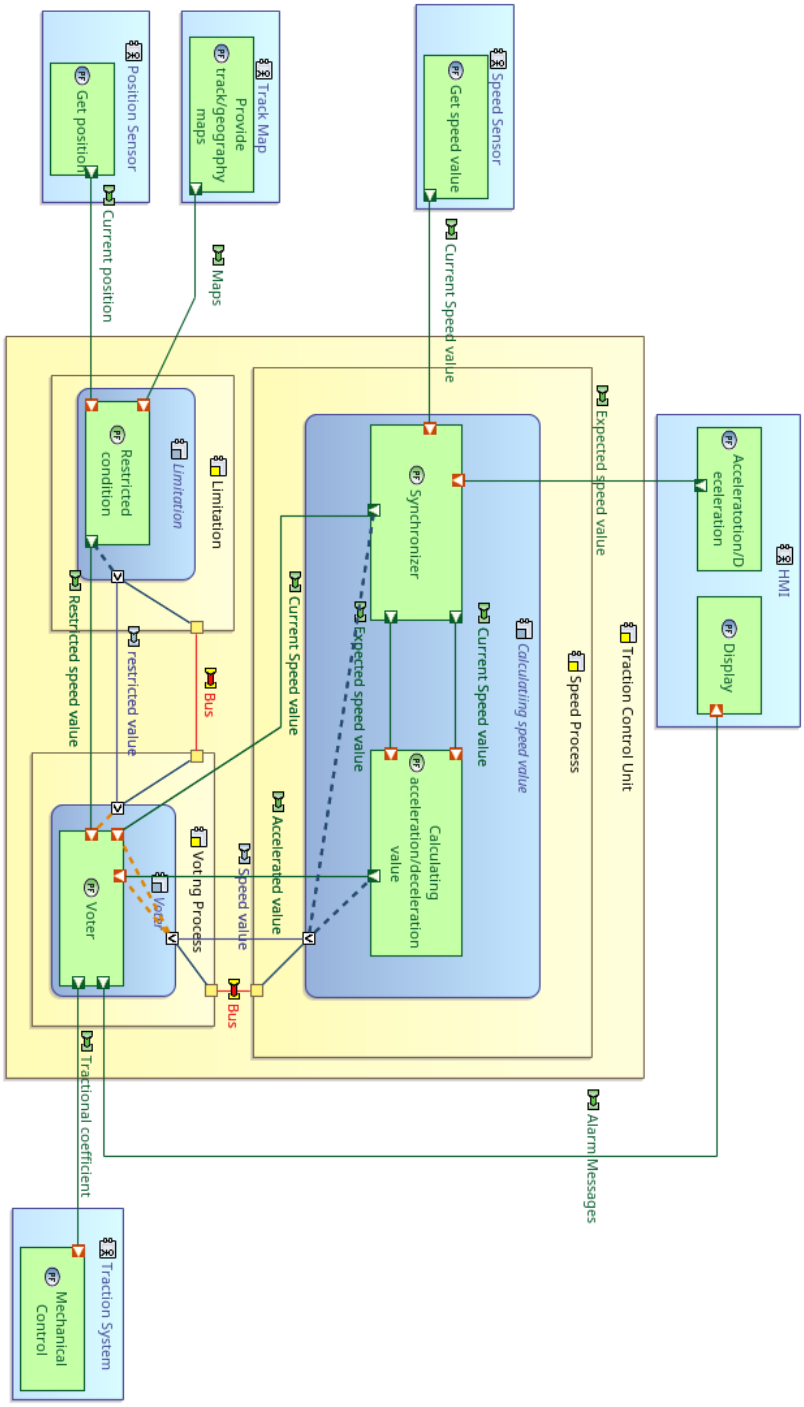


Fig. 6.4 Arcadia model of TCU system

6.4 Case study

To illustrate our approach in transforming and using produced AADL models to analyze the properties, this section presents the experimental results of analyzing the traction controlling unit of railway signaling system. By using our proposed

approach, we transfer and extend Arcadia metamodel, with AADL constructs and we used OSATE2 with the generated metamodel. Once the concrete models are created, the scheduling property is chosen to show analysis ability through Cheddar tool.

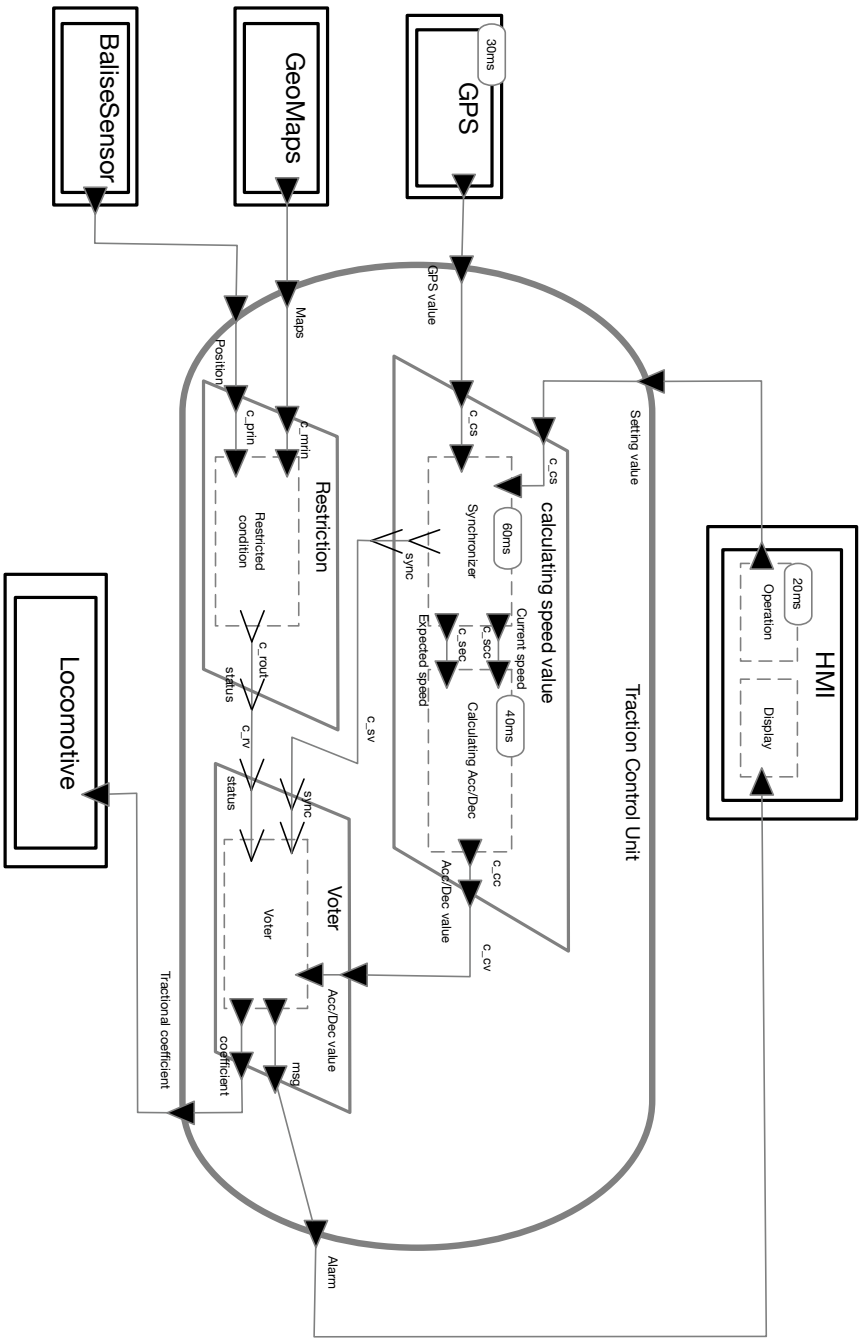


Fig. 6.5 AADL model of TCU system

6.4.1 Train traction control system

Train movement is the calculation of the speed and distance profiles when a train is traveling from one point to another according to the limitations imposed by the signaling system and traction equipment characteristics. As the train has to follow the track, the movement is also under the constraints of track geometry, speed restrictions, then the calculation becomes position-dependent. The subsystem of calculating the traction effectively under speed restrictions is therefore critical to achieving train safe running. Nowadays, Communication Based Train Control (CBTC) system is the main method of rail transit (both urban and high-speed train) which adopts wireless local area networks as the bidirectional train-ground communication [185]. To increase the capacity of rail transit lines, many information-based and digital components have been applied for networking, automation and system inter-connection, including general communication technologies, sensor networks, and safety-critical embedded control system. A large number of subsystems consist of modern signaling systems of railways, therefore, system integration is one of the key technologies of signaling systems. It plays a significant role in maintaining the safety of the signaling system [186].

This section uses a subsystem called Traction Control Unit system (TCU) for signaling system of high-speed trains. We use this TCU system to illustrate the MT from engineering level to detailed architectural level and verified the instance models. The functional modules such as calculation and synchronization will be transformed using our approach, and then non-functional properties such as timing correctness and resource correctness will be verified by Cheddar [184].

First, we start with component functional views and physical view analysis by designing system models in Arcadia (shown in Figure of TCU 6.4 [183]). The function of using the traction control system is to collect the external data by sensors, such as a speed sensor. The data from Balise sensors is used to determinate the track block, and then it is going to seek the speed restriction conditions by matching accurate positioning (if the track blocks are divided fine enough) and digital geometric maps data. Meanwhile, calculating speed unit receives the speed data from GPS and speed control commands from HMI (Human-Machine Interface) periodically.

GPS data provides speed value periodically (we set a period of 30 seconds in this case), and HMI data sustainedly sends the operation command with the period of 20 seconds till the value changed (e.g., expected speed value), then the calculating unit has to output an acceleration value and export to the locomotive mechanical system. Although they are periodic, the external data does not always arrive on time due to transmission delay or jitter. Therefore, we should use a synchronizer to make sure they are synchronized. Otherwise, the result would be wrong with asynchronous data. Similarly, to ensure the correctness of the command of acceleration (or deceleration), we apply a voting mechanism which can ensure the result is correct as much as possible. The voter must have the synchronized signal and restriction condition to dedicate to output the acceleration coefficient request to the locomotive system. The AADL diagram is shown in Figure 6.5 [183].

6.4.2 Model transformation

Using the combination tool, the metamodel of the TCU system in Capella is translated into the corresponding AADL metamodel with the rules and approach which describes in section 6.3. For instance, on one hand, the function class is translated into the thread in AADL. To analyze the timing properties, several attributes also have been added such as protocol type, deadline, execution time, period. On the other hand, the physical part element Node translates to the processor in this case. Different from simple physical Node in Arcadia, the processor element attaches rich properties such as scheduling protocol (scheduler type), process execution time. The allocation relationships on both physical and functional parts are translated into AADL as well.

6.4.3 Schedule verification

It is an essential safety requirement of the system to ensure external data and internal process work sequentially, and each task should be scheduled properly. However, in real-world, the risk of communication quality and rationality of scheduling must be taken into account. Therefore, the schedule verification is a way to evaluate system timing property. Cheddar provides a support to check if a real-time application

meets its temporal constraints. The framework is based on the real-time scheduling theory and is mostly written for educational purposes [187].

```
thread implementation synchronizer.impl
   properties
   Dispatch_Protocol => perodic;
   Period => 100 ms;
   Deadline => 100 ms;
   Compute_Execution_Time => 50..60ms;
end synchronizer.impl;

thread implementation calalculating.impl
   properties
   Dispatch_Protocol => perodic;
   Period => 100 ms;
   Deadline => 100 ms;
   Compute_Execution_Time => 30ms..40ms;
end calalculating.impl;

thread implementation gps.position
   properties
   Dispatch_Protocol => perodic;
   Period => 40 ms;
   Deadline => 40 ms;
   Compute_Execution_Time => 30ms..40ms;
end gps.position;

thread implementation HMI.setting
   properties
   Dispatch_Protocol => perodic;
   Period => 30 ms;
   Deadline => 30 ms;
   Compute_Execution_Time => 20ms..30ms;
end HMI.setting;
```

Listing 6.3 Setting of scheduling properties

Listing 6.3 shows a set of 4 periodic tasks (cal, pos, sync and setting) of TCU respectively, defined by the periods 100, 100, 40 and 30, the capacities 60, 40, 30 and 20, and the deadlines 100, 100, 40 and 30. These tasks are scheduled with a

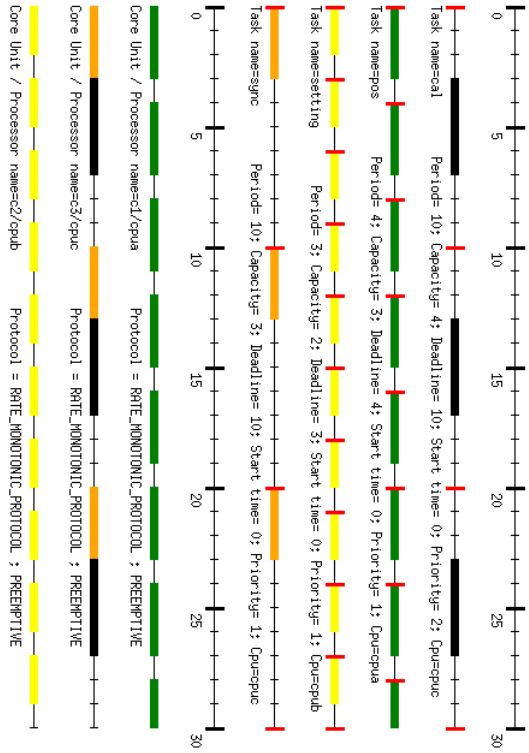
preemptive Rate Monotonic scheduler (the task with the lowest period is the task with the highest priority).

For a given task set, if a scheduling simulation displayed XML results in the Cheddar, one can find the concurrency cases or idle periods (see left of figure 6.6, and comprise the software part and physical device part). People change the parameters directly and reload simulation; a feasible solution can be applied instead. After tuning, finally, the appropriate setting is displayed in the right part of figure 6.6. According to this simulation result, people can correct the properties value in AADL, thereby ensuring the correctness of system behavior timing properties.

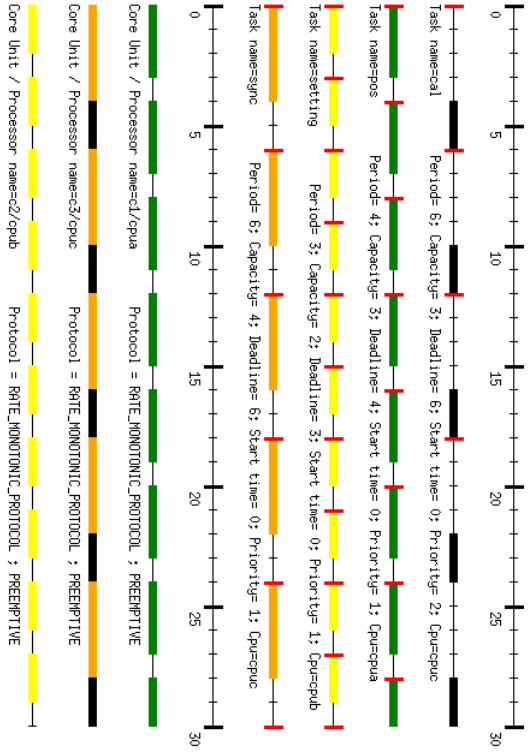
6.5 Summary

In this chapter, we have used our language for combining different modeling design artifacts (called views). We select system engineering methodology Arcadia (based on SysML) and architectural design language AADL as a vehicle for demonstrating our approach and of model combination language for schedulability analysis. We did so for two reasons. Firstly, the integration of heterogeneous components and elaborate model integrity concept in system design are challenging problems while using numerous model elements to describe different views of one system (or subsystem). Our proposed language attempts to be generic so that other cases can also be addressed. Secondly, enriching the functional design with scheduling ability can uncover conflicts that were not detected on the pure functional model. Hence, our language is good enough for the composition of several heterogeneous artifacts (views).

For helping the reader, we have briefly introduced the key elements of Arcadia and AADL that we have used. We also have given some examples of transformation rules which guide the transformation from Arcadia to AADL. Finally, a case study of train traction controlling system is used to demonstrate the transformation from engineering concerned design into an architectural refinement design which can be further analyzed by Cheddar.



(a) Schedule 1 with idle time



(b) Schedule 2 with compact time

Fig. 6.6 Simulation results of tasks schedule

7 Promoting functional design with safety and security properties

> *Eadem mutata resurgo.* $[\theta = \frac{1}{b} \ln (\frac{r}{\alpha})]$
> *Although changed, I arise the same.*
>
> — **Jacob Bernoulli**
> Mathematician

The design flaws and attacks on CPSs can lead to severe consequences. Thus, security and safety issues should be taken into account with functional design as early as possible during the development process.

In this chapter, we explore the model combination with security and safety requirements. We rely on the proposed modeling language CML to accomplish this goal. We take Capella, a widely used design platform, which provides (among others) comprehensive support for functional analysis from the requirements to the deployment of components. However, Capella does not provide direct support for security analysis. SysML-Sec is an extension of SysML dedicated to security and safety analysis, but it does not directly support functional analysis. Rather than trying to extend either Capella or SysML-Sec into more expressive languages to add the missing features, we extract proper subsets of both languages to build a view adequate for conducting a security and safety analysis of Capella functional models. The proposed CML is generic enough to extract proper subsets of languages and combine them to build views for different experts. Moreover, a case study is used to show that CML maintains a global consistency between functional and safety and security views.

7.1 Introduction

With an exponential growth in the development and deployment of various CPSs, new security and safety challenges have emerged [42, 47]. Various vulnerabilities, threats, attacks, and controls have been introduced for the new generation of CPSs and increase rapidly. The hacker also targets industrial systems whose sensors are increasingly commonly connected with vulnerable information systems. Attacks threaten the dependability of such systems with various objectives ranging from extortion to terrorist acts. For instance, recently, an American oil pipeline company "Colonial Pipeline" has been attacked by a hacker team and paid 4.4M dollar ransom[1]. There is also an example of impact on people's daily life. Two researchers have shown that they added a privilege escalation exploit such as CVE-2021-3347 to hack a car[2].

In order to accurately understand the growing trend, we developed a "crawler"[3] which can automatically collect the NVD[4] data files, a U.S. government repository of standards based vulnerability management data. These data can help us to identify the main kind of vulnerability and reasonably choose the properties to verify.

We collect the datum covers from 2002 (including the years before 2002) to 2021 (the first half of the year). Figure 7.1shows the growth trend of number of vulnerability per year according to collected NVD data. The curve shows that the number of vulnerability incresed at a rapid pace in the past few years. The recent, rapid growth phase coincides with increased commercial and popular interest in CPSs. The greatest success in the development of CPSs has been achieved in the U.S. and E.U. Recently, China has joined this race, which is investing huge amounts of money in this area [188]. Therefore, security and safety issues should be taken into account and identify flaws and vulnerabilities as early as possible in the system developing process [189]. In this way, their security vulnerabilities and safety flaws should be detected and mitigated.

[1] https://www.bloomberg.com/news/articles/2021-05-09/colonial-hackers-stole-data-thursday-ahead-of-pipeline-shutdown
[2] https://www.securityweek.com/tesla-car-hacked-remotely-drone-zero-click-exploit
[3] https://github.com/conanbos/crawl_cve_data
[4] https://nvd.nist.gov

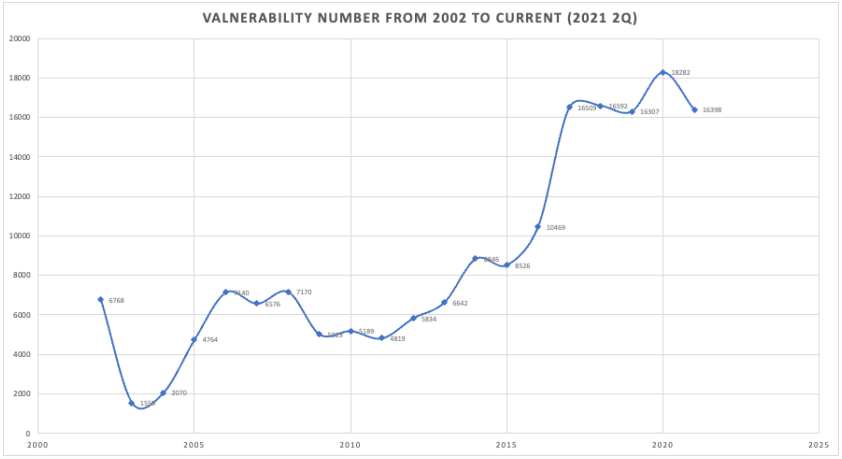


Fig. 7.1 Vulnerability trend from 2002 to 2021

In contrast, security is traditionally considered as data or communications security problem to be handled by computer scientists and/or engineers [190]. However, CPSs have open up a vast new range of potential problems that do not always show up on the traditional view. CPSs have additional properties that provide opportunities to attackers; for example, their real-time behavior means that attackers can cause havoc without stealing or corrupting data—simply changing the timing of key computations is sufficient to put the system into an unsafe state. Therefore, CPSs require us to take unified view among security, safety, functionality, architecture and their relationship (allocation, connection). When the design of a system requires different expertise, it is a usual practice to split its design among different teams that rely on specific views related to their domain of expertise. In this sense, MDE is suitable for CPSs design as it helps handling its complexity at design time. Our contribution is providing a manner to combine different views (models) in a reasonable way.

This chapter shows how the combination of a function and safety and a security model leads to obtain an enriched model. Finally, the enriched concrete model can be used for further analysis on security and safety, i.e., the functional models are enriched with safety and security properties which are verified by dedicate toolchain such as TTool. A case study demonstrates how to combine SysML-Sec-based models [16] with UML-like models, then the security and safety properties are added to UML-like models. The new generated model is able to perform security verifications and/or simulations by support environment TTool [60, 62].

The chapter is structured as follows. The next section explains how SysML-Sec differs from SysML and presents the motivation of our work. In section 7.3, we identify the security and safety issues and related properties. We also present the workflow and the process of transforming among different meta-models. Next, we illustrate a case study about ADAS which demonstrates our approach and language are effective. At the end of this chapter, we give a conclusion of this chapter.

7.2 Motivation

At present, the topic of autonomous-vehicles [191] are still one of the most promising research areas as well as the hottest topic in the automotive industry. Autonomous driving consists of many technologies, including sensing, perception, planning and operation. These new technologies enhance the safety of drivers and other road participant, mitigate the emission and promote the efficiency of travel [192]. After the press release from Nissan at Aug. 2013, several major OEMs (car makers) and Tier 1 suppliers (ECU providers) planned to introduce autonomous driving products into the market by 2020. However, a set of compliance of standards (ISO 26262 [193], ISO draft 21448, ISO/SAE 21434) are required by OEMs and Tier 1 suppliers for their subcontractors. These standards can ensure that the security and safety requirements are fulfilled. The subcontractors (including Tier 1 suppliers) and OEMs themselves have to reach a corresponding SIL (system integrity level) for safety part and SL (security level) for security part. Both SIL and SL are required on system and component level.

To reach the security and safety goal of the system, the requirements are essential. The security and safety requirements are defined by engineers, or the requirements are input from Stakeholder. Once the security and safety requirements are determined (including derived requirements, derived requirement are requirements that are not explicitly stated in the set of Stakeholder requirements, and yet is required to satisfy one or more of them). The security and safety engineers analyse these requirements and cooperate with system engineers (system architect) to design system architecture. The analysis and design are handled by MDE. MDE helps security and

safety engineer to select the appropriate countermeasure (algorithms, architectures) in an easy way.

According to my experience[5], we assessed and audited lots of project of Tier 1 suppliers. How to fulfill the requirements of safety and security of the standard are the widespread problems. In fact, engineers generally focus on functionality design, and in the end of project, they spent more time to pass the compliance testing, e.g., "achilles test"[6]. In most cases, due to lacking of security and safety consideration, engineers have to turn parameters or change architectures to pass the testing.

As we understood that the security and safety have to be considered at the early stage of development. The OEM and Tier 1 engineers get used to function-oriented design by using MDE. Few security and safety engineers use MDE to design model from security and safety view. There is a gap between functional and security and safety design, and a unified methodology to solve this problem is still lacking. We also notice that the traditional function-oriented MDE environment (such as Capella) could not support security and safety design. Despite the large support offered by Capella, there is no direct native support for dealing with security and safety issues, while there are now several tools specifically tailored for security and safety, such as TTool. Since TTool is based on SysML-Sec, and Capella is also basically based on a UML profile, they both rely on the same core technology but with different specific features (see Figure 7.2). The similarities between Capella and TTool (SysML-Sec) opens a way to leverage TTool somehow to enrich Capella's security and safety analysis capabilities. The question that we address here is whether we can benefit from both Capella and security and safety tools without extending Capella. Extending Capella (integrating security and safety analysis capabilities into Capella) can make it bigger and more complicated, while it brings new problems such as maintenance and consistency. Rather than trying to extend Capella to adapt to all aspects, we propose to bring together small subsets of each tool to focus on specific analysis capabilities while keeping the independence and global consistency of all the small pieces.

[5] I worked as an auditor at TÜV SÜD Industries Service Dept

[6] https://www.ge.com/digital/sites/default/files/download_assets/achilles-test-platform-from-ge-digital-datasheet.pdf

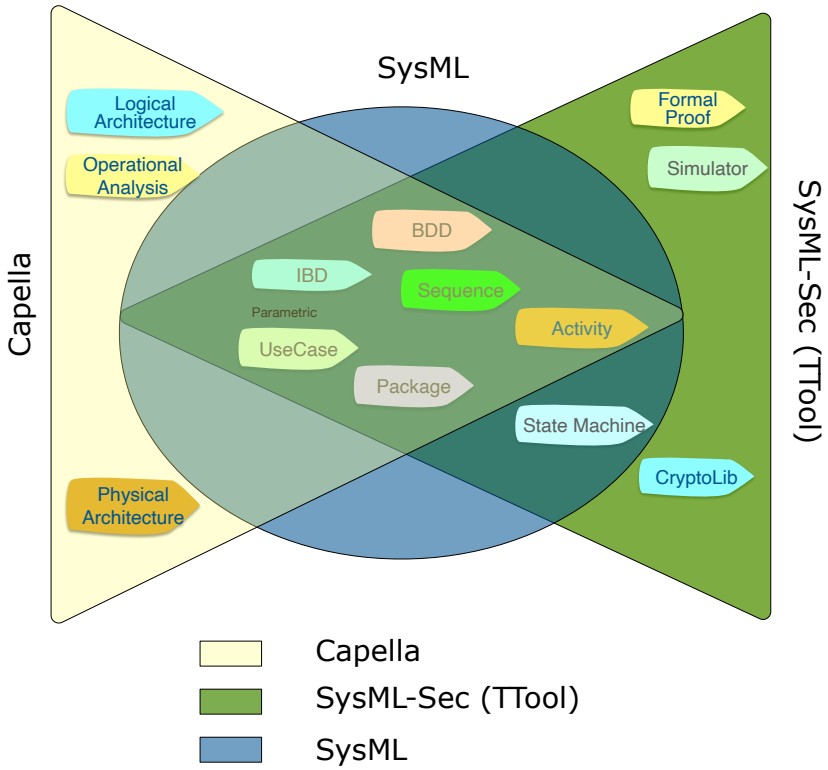


Fig. 7.2 Excerpt of relationships between SysML and SysML-Sec

We choose Capella as an engineering modeling platform and TTool as a security and safety analysis and proof tool. Our approach consists of security and safety features which are extracted from the metamodel level and a set of operational methods. The former is an abstract representation of security that allows us to identify and verify security and safety properties formally, and the latter defines the operational process that is used to conducting transformation. ADAS serves as a use case that is used to demonstrate how engineering modeling design combines security and safety analysis with our proposed approach.

7.3 Multi-view modeling approach for security and safety design

In this section, we introduce a security/safety-oriented multi-view modeling approach with the objective to analyze the cyber security/safety of Capella artifacts, as well as the possible countermeasures and their impact on the performance of the system, we use TTool as the underlying proof framework.

7.3.1 Workflow

Our workflow is depicted in Figure 7.3. Firstly, we give two metamodels as the original objects which are to be combined. Secondly, we construct a TRL. Once the TRL is built in the correct way, it is then imported into Step (1), which represents a generating process of a new metamodel $\boxed{A'}$. The step (2) and (3) are the steps for importing the security metamodel and functional metamodel. With step (1), it can produce a resulting metamodel $\boxed{A'}$ at step (4) that includes functional and security entities. In the Next step (5), we import instance models $\boxed{a1{:}A}$ into the new metamodel $\boxed{A'}$. Finally we get a new instance model $\boxed{a1'{:}A'}$, called *Resulting model* that strictly conforms to metamodel $\boxed{A'}$. The instance model can be imported to the security framework TTool to perform security analysis, even more post-processings.

7.3.2 Security and safety scopes

Security requirements can be captured as constraints that depend on security concerns. Security requirements are the needs of stakeholders' security objectives that consider the identified threat and assumed system architecture. These requirements do not say how to satisfy the constraints, but only define the constraints. The security requirements are based on the use cases and technical analysis such as "attack tree", and derived in a systematic manner. The requirements of security can be classified according to security properties, such as: *Confidentiality, Authenticity, Integrity, Privacy/anonymity, Freshness, Availability, Controlled access, and Non-repudiation*. As to identification of security requirements, the EVITA project [192] has shown that they identify the security requirements from two viewpoints. One is based on functional representation of use cases, providing security requirement by property (confidentiality, authenticity), another is based on mapping functional representation to an architecture, providing both functional and architectural requirements.

Safety is also called functional safety (ISO 61508 [194], 5012x [195], 26262 [193]). Based on the safety goals, a functional safety concept is developed considering the preliminary architectural assumptions. The functional safety concept is developed

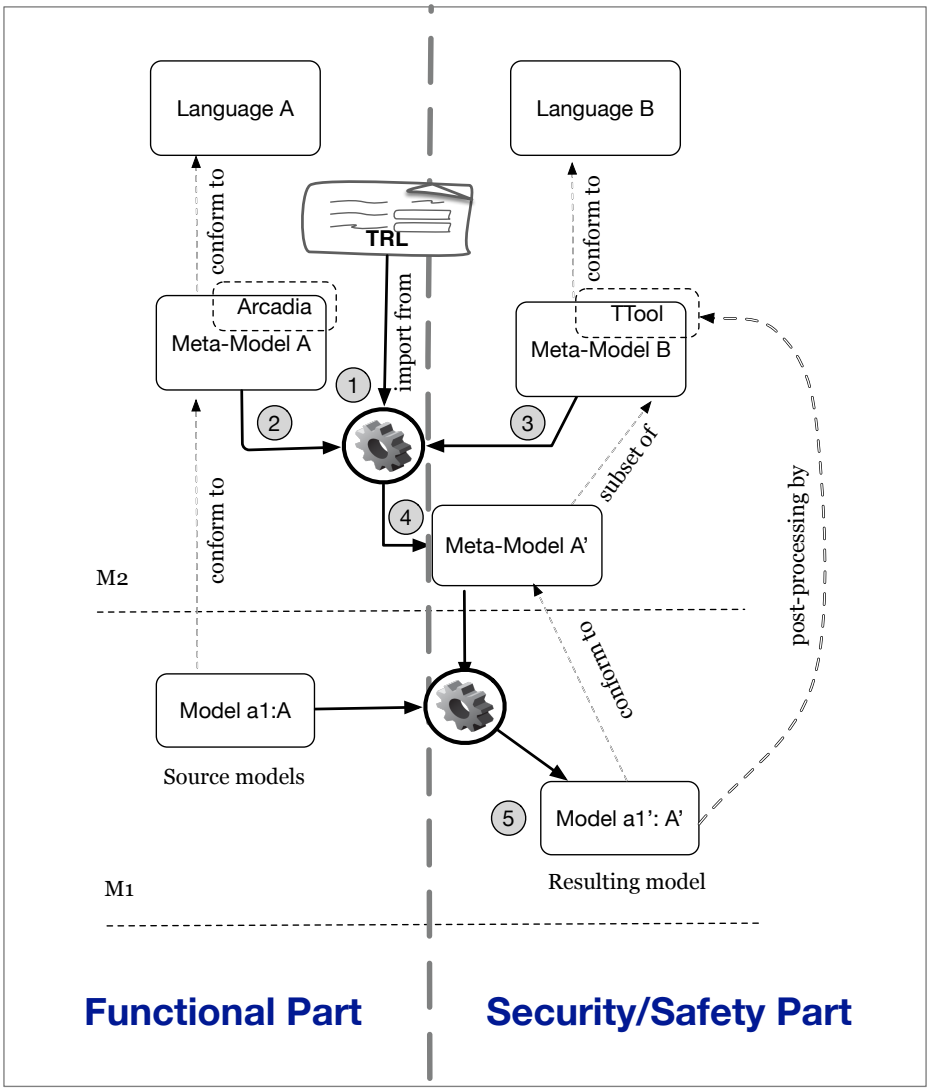


Fig. 7.3 The workflow for combining safety and security models

with deriving functional safety requirements from the safety goals and allocating these functional safety requirements to the elements of the item. The functional safety concept may also include other technologies or rely on external measures. In those cases, the corresponding assumptions or expected behaviours are validated.

Safety is a key factor to evaluate the system. The requirements of safety are based on use cases and connecting to functional representation. The safety requirements can be classified according to safety properties, such as liveness, reachability and deadlock.

Identification of safety requirements should consider multiple factors (e.g., failure mode, MTBF, BFR) and involve technical analysis (e.g., hazard analysis, risk as-

sessment, impact analysis, failure mode and effects analysis). Conventional safety suggests that a system should not contain software and hardware flaws which can prevent a correct functioning. "Safety of the Intended Function" involves avoiding the situations which the system or its components cannot handle, such as adverse extreme environmental conditions. *Timing* can be critical for certain real-time systems, as the system will need to respond to certain events as quickly as possible, such as obstacle avoidance, and reducing speed, within a set period to avoid dangerous situations. Any delay could result in a quite severe consequence.

7.3.3 Properties to verify

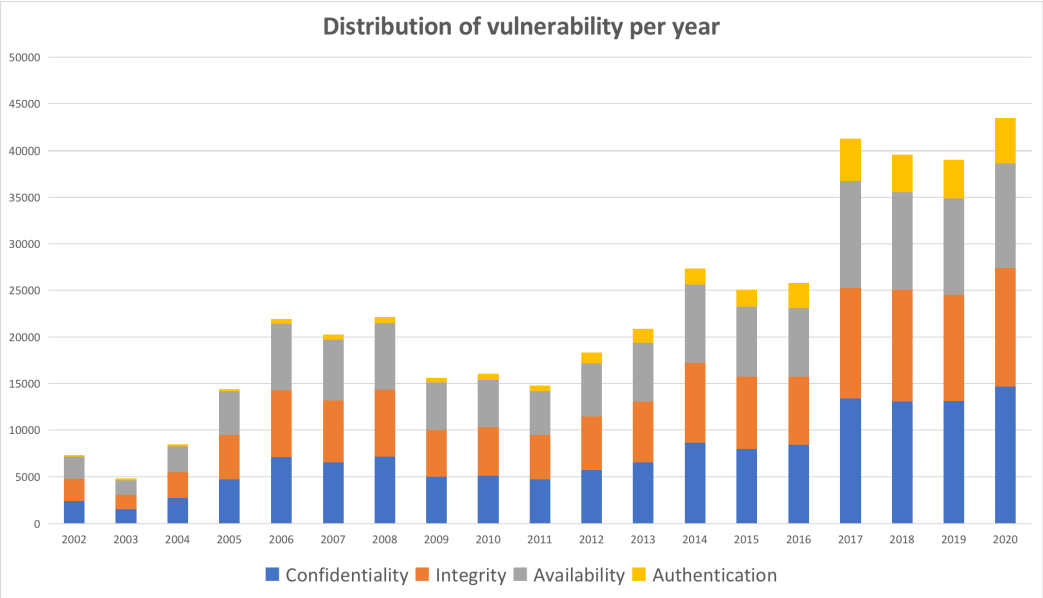
The first step of V-cycle is requirement determination, including functional, performance, security and safety. Once the requirement has been determinated, the properties to verify should be defined. To ensure that the system works as designed, safety and security verifications are useful means. What properties to be verified is a question for engineers.

As shown in the Figure 7.4, the results of statistics for the terms *confidentiality, integrity, availability and authenticity*[7]. Note that the results were restricted to those vulnerabilities with the relevant terms in the assessment to capture those with a significant focus on the subject. The results show a steady growth of these four kinds of vulnerabilities from less than 10,000 in 2006 to more than 100,000 in 2020. Therefore, OEMs and Tier 1 suppliers consider "confidentiality, integrity, authentication, liveness and availability" as main properties of security and safety in most cases in their projects.

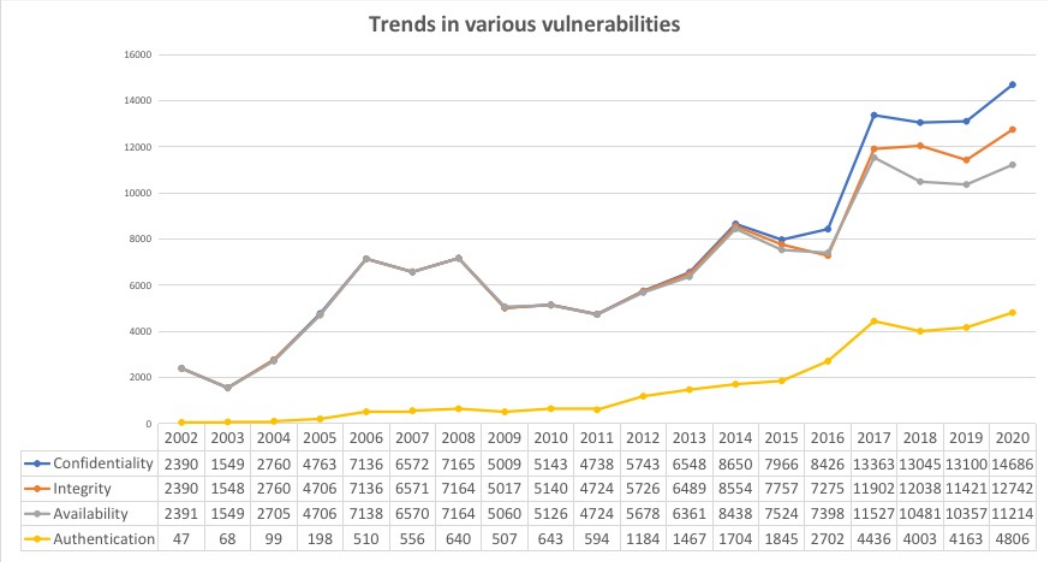
These properties can be formalized and checked by the model-checker such as UPPAAL or with reachability graphs [75]. TTool relies on its internal model-checker and get results to notify users [196]. As for security properties, TTool is also able to verify these security properties such as authenticity, confidentiality.

To clearly and completely understand the security and safety properties, we give the key definitions of safety and security properties as below:

[7]Original data are from NVD dataset



(a) Distribution of vulnerability per year



	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020
Confidentiality	2390	1549	2760	4763	7136	6572	7165	5009	5143	4738	5743	6548	8650	7966	8426	13363	13045	13100	14686
Integrity	2390	1548	2760	4706	7136	6571	7164	5017	5140	4724	5726	6489	8554	7757	7275	11902	12038	11421	12742
Availability	2391	1549	2705	4706	7138	6570	7164	5060	5126	4724	5678	6361	8438	7524	7398	11527	10481	10357	11214
Authentication	47	68	99	198	510	556	640	507	643	594	1184	1467	1704	1845	2702	4436	4003	4163	4806

(b) Trends in various vulnerabilities

Fig. 7.4 Distribution and trends in various vulnerabilities

7.3.3.1 Safety properties

- **Reachability** is a property that determine if a function or condition is present in at least one execution path of the system. It also can indicate if the model

is correct and all the functions and conditions can be executed as good as designed.

- **Liveness** is a property that cannot be violated in a finite execution of an embedded system. In other words, liveness expresses that eventually"something good must happen" during an execution [197].

- **Deadlocks** is a situation in which no further action can be taken, e.g., two functions mutually wait for the other one to make a step before proceeding.

7.3.3.2 Security properties

- **Confidentiality** property applies to a quantum of information and a set of authorized entities. If there are only the authorized entities that can know the quantum of information, the property is satisfied. Privacy relies on confidentiality and can be considered as a special case of confidentiality [192].

- **Authenticity** is a property that applies to a quantum of information. The property is satisfied when the data come from a claimed author without any modification. Note that in most security-oriented frameworks data origin authenticity implies integrity [192].

- **Integrity** is also called weak authenticity, which is a property applies to a quantum of information between two observations. The property is satisfied when the quantum of information has not been modified by an attacker or unauthorized individual. It guarantees for instance that the quantum of information has not been modified between two given read operations, or that a message sent on a communication channel has not been altered during its journey. Compare to integrity, strong authenticity is a property related to communications. Weak authenticity only determines if a message has been modified by an attacker, while strong authenticity ensures that messages being received in a certain communication exchange must have been sent in that exchange. For example, if an attacker captures and replays a message, then that communication satisfies the property of "Integrity" but not "Strong authenticity".

- **Availability** is a metric that measures the system usability. In other words, an availability property or requirement applies to a service or a physical device providing a service, under given conditions over its defined lifetime. The property is satisfied when service is operational. Denial of service attacks aim at compromising the availability of their target. For example, if the system can provide services immediately when requested by authorized users.

- **Access control** is a security technique that allows only authorized entities to use resources or perform specific actions in the computing system. It can be related to both Confidentiality and Authenticity [75]. As an unauthorized entity is not able to access confidential data, it should not be able to modify any code of a system and invoke any internal components of the system. Access control techniques should prevent insecurity actions and deny unauthorized service requests. It is a fundamental concept in security that minimizes risk to the system.

7.3.4 Transformation rule library for security and safety

By using the proposed combination language, we can construct a set of relationships between functional meta-models and security/safety-oriented meta-models. The set of relationships is called TRL, which we mentioned in the above sections. Once the TRL is established, the following process of generating could be automatic by the tool. As the combined models include both the functional and security parts, we can import those models to TTool for security/safety analysis (simulation, verification). The results can be traced back to the functional design part.

The table 7.1 shows the correspondence between Capella and TTool elements. The additional attributes column are the attributes to be created during the transformation. According to this table, we can easily write the transformation rules to transforming Capella to TTool on functional parts.

A simplified schema of relationships between Arcadia meta-model and TTool meta-model is shown in Figure 7.5. The green flash represents component equivalence that means the two components are linked by am equivalence relationship (refer

Capella	TTool	Additional attributes
Function	PrimitiveComp	
Interaction	Connection	
Port	Port	Type
Port	Type	[request, event, channel]
Port	<channel>Confidentiality	[true, false]
Port	<channel>Authenticity	[true, false]

Tab. 7.1 Functional and security and safety elements correspondence

to 4). In the box on the left bottom of the figure, these are the first parameters of the system model. These parameters associate with the "channel" component in TTool. The "channel" is a kind of port component in TTool which is equivalent to "Functional Exchange" and " Functional ports (Input, Output)" in Capella (SysML). The red dotted line flash shows the extended capabilities by model transformation, in other words, the components in one meta-model can be transformed to another meta-model by CML, then their capability is enhanced. For example, if the "confidentiality" (or "Authenticity") is checked, the corresponding algorithm will be applied to encrypt the data which is sent by this channel. In other words, the functional components in Capella can be seen as having additional security and safety properties as long as they are linked to a TTool (SysML-Sec) component using our proposed language.

Cryptographic configuration are first made to specify security algorithms of the system model (e.g., AES). Within activity diagrams, they appear as an upside-down pentagon marked with their type, as shown in Figure 7.5, where 'AE' represents Asymmetric Encryption and 'D' represents Decryption. Cryptographic Configurations can be typed as follows: Symmetric Encryption and Asymmetric Encryption patterns encrypt data along with a key/keys specific to the pattern. A MAC can be added to messages to authenticate it and determine if it has been modified. Hash calculates a hash of the data. "Nonces" can be concatenated to messages before verify authenticity of encryption. Advanced algorithm allows the user to indicate their own encryption scheme, such as combinations of cryptographic operations. Figure 7.5 (bottom left) shows the specification of a *Cryptographic Configuration*. The designer can choose the algorithm and the corresponding performance properties. The balance between security algorithms and performance requirements have

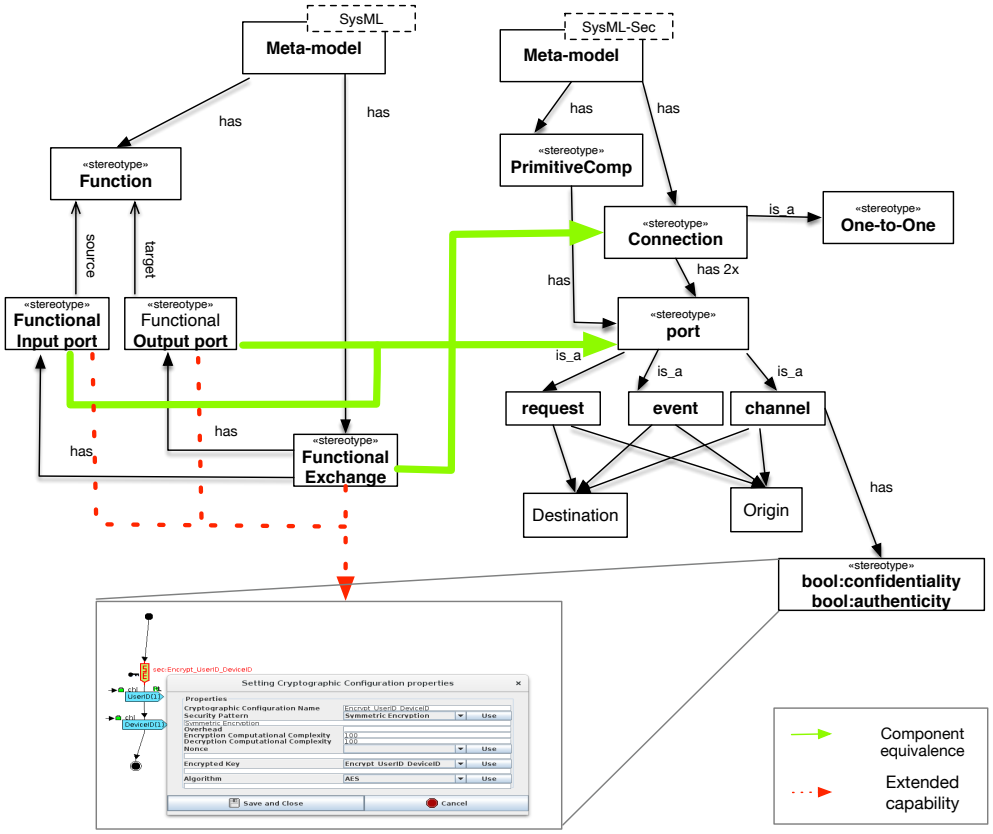


Fig. 7.5 A simple schema of relationships between security meta-model of TTool and functional meta-model of Arcadia

to be established [198]. An extensive "Secure Software Development Life Cycle" catalog is used together with TTool for performance trade-off analysis.

7.3.4.1 An instance of TRL

TRL serves transforming meta-model of Capella into a new meta-model, which is added as part of TTool's meta-model. We write the most of TRL details for transforming Capella's element to TTool in Listing 7.1.

7.4 Case study

Advanced Driver-Assistance Systems (ADAS) are the typical CPSs. ADAS take an important role in an autonomous vehicle. Conventional ADAS technology can detect some obstacles, alert the driver of hazardous road conditions, in some cases, slow or stop the vehicle. This level of ADAS is great for applications like blind-

spot monitoring, lane-centering assistance, obstacle avoiding, and forward collision warning. It means that the "driver is disengaged from physically operating the vehicle by having driver's hands off the steering wheel and foot off the pedal at the same time". However, the freedom given to the driver also brings great risks, e.g, the underlying flaws are used by attackers to hijack the vehicle such as getting a remote control, or delaying system response time.

In this case study, we demonstrate how to add safety and security verification abilities for Capella's functional design by using the proposed approach. The SysML-Sec further adds the safety and security properties for functional design. Then, we can perform verification to check if security and safety properties are satisfied. All the results get back to Capella to correct or adjust the functional design. We illustrate the whole workflow that is from the meta-model phase to the final verification phase, refer to Figure 7.6.

We start with meta-models combination at the meta-model phase (as shown in Figure 7.3). We use the proposed language to build up TRL, which is presented in 7.3.4. Once the TRL is done, we enter model phase for functional design on Capella (shown in the middle of the Figure 7.3.4). All of the sensors (radar, camera…) and ADAS control system tasks (Perception and Navigation) are designed as functions on the Capella model, while modeling all the function exchanges.

```
1 ΓSystem Component -> Composite Component:{Properties[Is abstract|Is human|Is actor ]}+;
2 ΓSystem Function -> Primitive Component:{Attributes[access| identifier | initial value|type
    ]}+;
3 ΓFunction Input Port -> Primitive Port:{Type[Channel|Event|Request]}+;
4 ΓFunction Input Port -> Primitive Port:{Origin[Origin| Destination ]}+:{Reference
    Requirement}+:{Blocking[Blocking|Non blocking FIFO]}+:{Finite}+:{Dataflow type}+;
5 ΓFunction Output Port -> Primitive Port:{Type[Channel|Event|Request]}+;
6 ΓFunction Output Port -> Primitive Port:{Origin[Origin| Destination ]}+:{Reference
    Requirement}+:{Blocking[Blocking|Non blocking FIFO]}+:{Finite}+:{Dataflow type}+;
7 ΓFunctional Exchange:{source} -> Connector:{p1};
8 ΓFunctional Exchange:{target} -> Connector:{p2};
```

Listing 7.1 An example of TRL for transforming to TTool

Next, leveraging TRL (listing 7.1), we transform Capella models into the SysML-Sec models for further safety and security design and analysis. All the required attributes and properties would be filled in TTool/SysML-Sec such as port properties (direction, type). For example, according to the TRL (see listing 7.1), firstly, we write a TRE is "Γ System Function -> Primitive Component", while all the *System Function* in Capella model are transformed to *PrimitiveComp* in TTool model with their name. Secondly, the next TRE is "Γ Function Input Port -> Primitive Port", there are two types of ports in Capella, "Function Input Port" and "Function Output Port". The *Function Input Port* will be transformed to *Primitive Port* in TTool with attribute Origin being "Origin", and *Function Output Port* is transformed to *Primitive Port* with attribute Origin being "Destination". In this case, the *Primitive Port*'s type is "Channel" by default, because there are no "Event" and "Request" type in Capella. Other attributes, such as "Reference Requirement", "Blocking", we let it be empty at this moment, we further assign their values in TTool.

Once the TRL has been established, we then use "Combination Modeling Tool" (refer to chapter5) to generate a new meta-model. CML is able to transform the meta-model according to TRL. The new concrete TTool model is generated according to this new meta-model and Capella functional model.

Figure 7.6 shows the whole process. In the left side of the figure, the schema of transformation is defined. In the model phase (in the middle of Figure 7.6), a

Capella functional model is transformed into TTool format automatically. In the right side of the figure, we illustrate further design for security and safety purpose. For example, we set up essential security properties in TTool such as cryptographic configuration. Similarly, we also select the safety properties which are to be verified such as reachability, liveness and absence of deadlock. TTool then performs verification automatically if the grammar check passes without errors. TTool gives feedback when the verification process is finished. Engineers can revise the design according to the results.

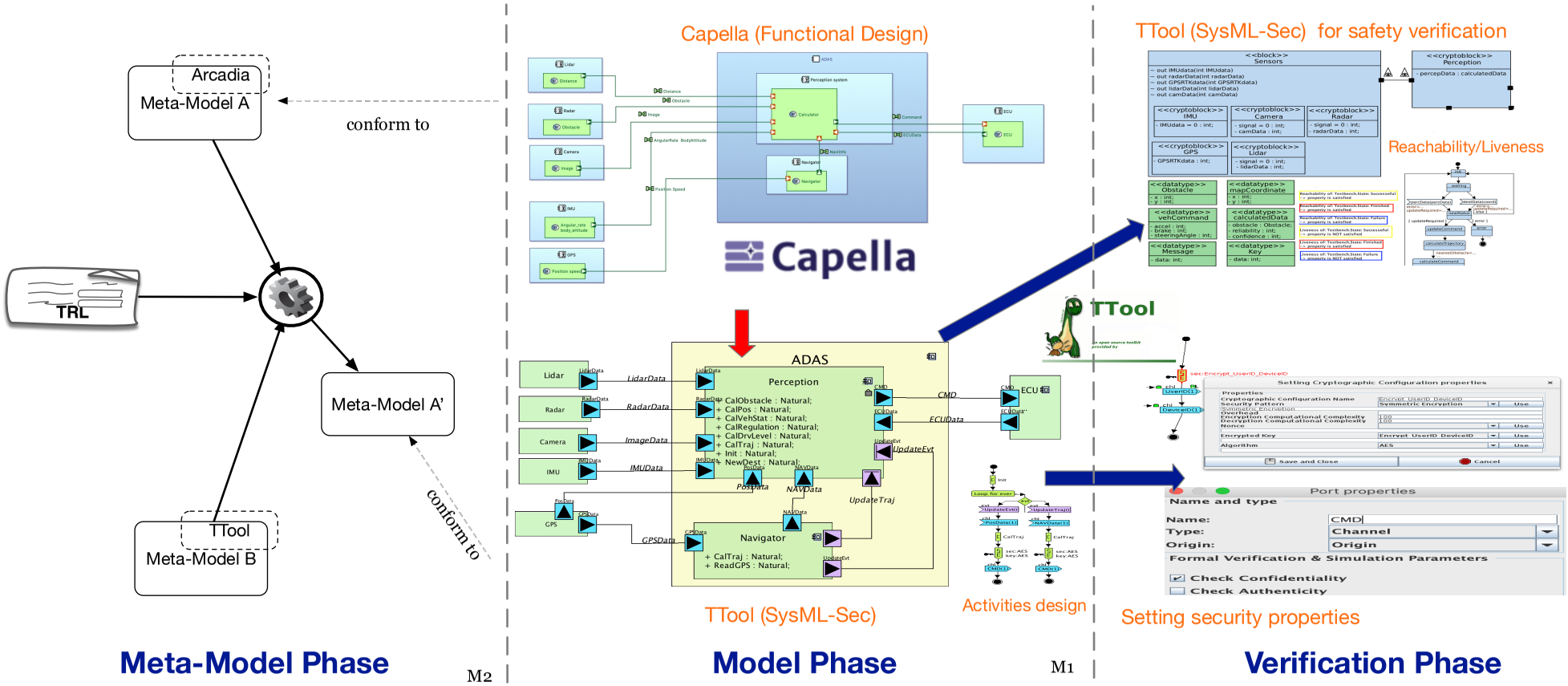


Fig. 7.6 Workflow of ADAS design from modeling phase to verification phase for security and safety purposes

7.5 Conclusion

In this chapter, we have presented the growth trend of security and safety issues and impacts of CPSs. Thus security and safety aspects have to be considered at an early stage of CPSs development along with functionality considerations. MDE is therefore proposed to handle CPSs design. Yet, general functional modeling languages such as SysML have limitations to describe security and safety properties, which ad-hoc languages can do it pretty well. Therefore, SysML's extension, SysML-Sec is used to fill this gap. We also identify the security and safety properties and explain how we chose the properties to verify.

In a similar way as our previous work on combining Capella and AADL models so as to perform scheduling verification, we proceed here to address safety and security. Reusing the same proposed language-based design approach for combining safety and security artifacts with functional models, we explicitly introduce the workflow of the proposed approach that identifies security and safety issues and related properties.

A safety and security-aware design case of an autonomous vehicle system was used to illustrate how the functional models is equiped with safety and security capabilities by using the proposed language. The analysis and verification are then performed by the TTool toolchain.

8 Conclusion and Perspective

> 为天地立心，为生民立命。
> 为往圣继绝学，为万世开太平。
>
> — 张载（字 子厚）
> 北宋, 横渠先生

In this chapter, we conclude the content of my thesis, including main contributions, I.) a dedicated modeling language CML to specify combination patterns among heterogeneous modeling languages; II.) two practices of combining different views which can help the designer understand the application of this modeling language. A support tool makes the process easier and automated. The proposed CML seems to be generic enough with two practices of verifications of scheduling and security & safety properties. We now discuss the limitations of CML and future works.

8.1 Conclusion

8.1.1 Overview

CPS consists of various components and their interconnections. Thus, the design of the CPS spans numerous domains and expertises. Handling requirements from different domains with different characteristics pushes model-based approaches to their limits. Hence, we intend to find an appropriate way to mitigate the complexity of CPS design and to use different modeling tools or languages for unified system design. The CPS have been held to a higher reliability and predictability standard than general-purpose computing [2]. For example, in a general-purpose embedded system, the execution time of computation is a factor in evaluating the system performance. Taking a longer time to perform tasks is not a critical issue. It is merely less convenient and less valuable. But in the CPSs, overtime can put the system into an unsafe situation, moreover, it can also lead to an accident when being used in a safety-critical system such as a railway signaling system.

MDE is considered as a well-established development approach that uses abstraction to bridge the gap between the problem space and the system implementation [65, 199]. MDE uses models to describe complex systems at multiple levels of abstraction. Models are instances of modeling languages that define their abstract syntax, concrete syntax, and semantics [200]. As an important issue of MDE, multi-view modeling integrates different models using various DSMLs and abstracts various aspects of systems and subsystems, such as scheduling, behaviors and functionalities. Therefore, it is critical to understand the relationship among (meta) models.

Since CPS development is extremely complex, the design of CPS requires many experts with different domains. We have identified the characteristics of CPS: they are *heterogeneous* systems, they capture the different aspects and views and the design relies on a variety of models. CPS are also *platform-aware*, they can execute on many platforms and should adapt to the platform with some non-functional properties. The execution time and safety & security issues are significant issues to CPS, because they may lead to an unacceptable result. We also considered systems that are *time-sensitive and safety&security-critical*. Compared to more traditional

embedded systems, CPSs usually contain heterogeneous interconnected embedded subsystems which are *widely distributed*. Then, we have identified the challenges (in section 2.1.3) for CPS design, which is to be addressed in industrial applications such as safety & security, real-time, verification & validation, and training cost. We summarise the main challenges as *Complexity of system and heterogeneous subsystems* and *Systems consistency*.

To tackle these challenges, we propose a Combination Modeling Language which enables system engineers to combine and reuse the artifacts (models) of domain experts. The major element of CML is a specification that contains Patterns, Operators and TRL. We gave for each element a number of examples to illustrate how they work. CML is devised to enable system engineer to reuse models designed by other engineers.

8.1.2 Contributions

In this thesis, I devote my efforts to deal with the significant issues of designing heterogeneous systems. These are my contributions:

1. Propose a combination modeling language CML to combine heterogeneous (meta) models.

2. Develop a support tool, which makes the combining process automatic with a friendly GUI.

3. Show that the proposed language is useful and generic enough with two different use cases from two domains:

    - Verifying the schedulability by combining AADL design

    - Identifying safety&security properties and conducting verification by TTool.

This thesis discusses the characteristics and challenges of CPS from a designer's view. To handle these issues, a new approach is required to efficiently take strengths

of existing languages and combine them together. The existing approaches can be classified into two types. The first type is to continuously integrate the necessary languages into an existing development platform and then progressively build a comprehensive development platform. However, this type of approach could encounter a never-ending process and result in a gigantic framework, thus it is difficult to use and maintain. The second type is to keep each language (or tool) isolated, and relate some of the elements from each language with (sub) meta-model so as to allow different kinds of analyses offered by each method (e.g., scheduling analysis, safety analysis). Furthermore, each domain expert can work independently using the second type of approach. However, since each language has its own characteristics, the gaps must be eliminated to handle consistency issues. Therefore, we have proposed a modeling language to establish a set of relationships among (meta) models.

The proposed *CML* is a dedicated (meta) language to extend and enrich one DSML's capabilities by combining with other DSMLs. By using CML, system experts can capture a set of scenarios and co-work with different domain experts at the language level. We used EBN form to define context-free grammar for the syntax part. Moreover, the combination pattern is used to specify different combination relationships for the semantics part.

CML enables several modeling views that can be considered and designed at the same abstract level, and allows different modeling framework to reuse each other artifacts. It essentially augments the system design efficiency, reduces the complexity, and should hopefully ensure the system consistency.

Since combining models manually is error-prone and time consuming, the integration engineers have to pay much more attention to building a new model according to rules. Any mistake can lead to unpredictable results, moreover it is difficult to detect those mistakes. Instead of combining models manually, a *support tool* is designed to accomplish the process automatically. It can ensure the correctness of generating a new combined (meta) model and export the new (meta) model in an easy way. This tool is web-based, it allows integration engineers to work with a friendly interface and a thin client (using a browser and being platform-independent). With

web-based features, this tool is also able to serve some integration engineers simultaneously.

To validate our approach, we first address the field of schedulability analysis by combining Capella and AAD. We used the example of a train control system. AADL tools carry out the analyses to verify that time expectations (e.g., preset values) are correct.

Then, we have turned to safety and security properties (such as reachability, liveness, and authenticity) using TTool, a SysML-Sec support environment. We have demonstrated how to verify each of the capabilities of our approach to improve the safety and security of the ADAS of an autonomous vehicle as our running example. Since the verified models are transformed from a functional design environment (using Capella in this thesis), the models to simulate are conformed to functional design.

8.1.3 Limitations

There are still some remaining drawbacks that we try to analyze objectively:

- The integration engineers have to spend time learning the syntax of rule
- The writing of rules is error-prone
- We do not yet implement reverse direction transformation automatically.

In MDE, there is always a learning curve that is sometimes important. Our approach is not different in this respect.

As we define relations between two meta-models they should be exploited both ways both for forward and reverse transformations. However, our tool does not provide, at the moment, any facility to exploit the reverse transformations. This is a major limitation.

8.2 Perspectives

As we mentioned in the limitations, currently, CML does not support traceback to the original models. In some scenarios with the loss of the original models (functional or others), the engineer requests to find the original models back from combined models with TRL, i.e., the transformation should support bi-directional operations. We will improve CML's functionality and add more information and operators for bi-directional operations in our future work.

It is difficult to find the right limit of expressiveness for a language. CML is sufficient for our examples but we anticipate that having a repetitive capability (like for-loop or recursive enumeration) could be useful in the future.

The rise of AI computing and Machine Learning technologies have led to new demands for Machine Learning systems to learn complex models with parameters that promise adequate capacity to offer powerful and real-time predictive analytics. System models are tailored to the unique properties of ML algorithms, and algorithms are re-designed to better fit into the system models (so-called system and ML algorithm co-design) [201]. We could imagine using ML technologies to improve the model combination process. In the meantime, we intend to provide tips for TRE writing and check the logic errors of rules by analyzing the elements of models.

List of Abbreviations

AADL Architecture Analysis & Design Language. xii, 31

ADAS Advanced Driver-Assistance Systems. 137

CML Combination Modeling Language. 35

CPS Cyber-Physical System. xi, 27

DE Discrete Event. 49

DE Differential Equation. 49

DSL Domain-Specific Language. 35, 46, 51

DSML Domain-Specific Modeling Language. xii, 51, 70

DT Digital Twin. 40

EBNF Extended Backus Naur Form. 33

EMF Eclipse Modeling Framework. 31

GUI Graphical User Interface. 33, 45, 97

HMI Human-Machine Interface. 40, 41

IoT Internet-of-Things. 39

M2M Model to Model. 46, 64

M2T Model to Text. 46, 64

MARTE Modeling and Analysis of Real-time and Embedded system. 58

MBSE Model-Based System Engineering. xi, 56

MDE Model-Driven Engineering. 31, 45

MPM Multi-Paradigm Modeling. 45

MT Model Transformation. 31

MV Multi-View. 35

MVM Multi-View Modeling. 47

OCL Object Constraint Language. 50

OMG Object Management Group. 50

PDE Partial Differential Equation. 29, 49

SysML System Modeling Language. 50

TRE Transformation Rule Expression. 33, 82

TRL Transformation Rule Library. 33, 82

UML Unified Modeling Language. 46, 50

Bibliography

[1] Edward Lee. “The Past, Present and Future of Cyber-Physical Systems: A Focus on Models”. In: *Sensors* 15.3 (Feb. 2015), pp. 4837–4869 (cit. on pp. 27, 39, 42).

[2] Edward A Lee. “Cyber Physical Systems: Design Challenges”. In: *International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*. IEEE, May 2008, pp. 363–369 (cit. on pp. 27, 28, 42, 144).

[3] Siddhartha Kumar Khaitan and James D McCalley. “Design techniques and applications of cyberphysical systems: A survey”. In: *IEEE Systems Journal* 9.2 (July 2014), pp. 350–365 (cit. on p. 27).

[4] Frédéric Mallet, Eugenio Villar, and Fernando Herrera. “MARTE for CPS and CP-SoS”. In: *Cyber-Physical System Design from an Architecture Analysis Viewpoint*. Springer, 2017, pp. 81–108 (cit. on pp. 27, 28, 58).

[5] Patricia Derler, Edward A. Lee, and Alberto Sangiovanni Vincentelli. “Modeling Cyber-Physical Systems”. In: *Proceedings of the IEEE* 100.1 (Jan. 2011), pp. 13–28 (cit. on pp. 27, 42, 49, 58).

[6] Marc Raibert, Kevin Blankespoor, Gabriel Nelson, and Rob Playter. “BigDog, the Rough-Terrain Quadruped Robot”. In: *IFAC Proceedings Volumes* 41.2 (2008), pp. 10822–10825 (cit. on p. 27).

[7] David Wooden, Matthew Malchano, Kevin Blankespoor, et al. “Autonomous Navigation for BigDog”. In: *International Conference on Robotics and Automation* (2010), pp. 4736–4741 (cit. on p. 27).

[8] Ryosuke Okuda, Yuki Kajiwara, and Kazuaki Terashima. “A survey of technical trend of ADAS and autonomous driving”. In: *International Symposium on VLSI Design, Automation and Test* (2014), pp. 1–4 (cit. on pp. 27, 42).

[9] David Broman, Edward A Lee, Stavros Tripakis, and Martin Törngren. “Viewpoints, formalisms, languages, and tools for cyber-physical systems”. In: *6th International Workshop on Multi-Paradigm Modeling*. ACM. 2012, pp. 49–54 (cit. on p. 29).

[10]Jean-Raymond Abrial and Stefan Hallerstede. “Refinement, decomposition, and instantiation of discrete models: Application to Event-B”. In: *Fundamenta Informaticae* 77.1-2 (2007), pp. 1–28 (cit. on p. 30).

[11]Jean-Raymond Abrial. *Modeling in Event-B: system and software engineering*. Cambridge University Press, 2010 (cit. on p. 30).

[12]Jean-Raymond Abrial and Jean-Raymond Abrial. *The B-book: assigning programs to meanings*. Cambridge university press, 2005 (cit. on p. 30).

[13]Régine Laleau and Amel Mammar. “An overview of a method and its support tool for generating B specifications from UML notations”. In: *International Conference on Automated Software Engineering*. IEEE. 2000, pp. 269–272 (cit. on pp. 30, 77).

[14]Régine Laleau, Farida Semmak, Abderrahman Matoussi, et al. “A first attempt to combine SysML requirements diagrams and B”. In: *Innovations in Systems and Software Engineering* 6.1-2 (2010), pp. 47–54 (cit. on pp. 30, 76, 77).

[15]Steve Jeffrey Tueno Fotso, Régine Laleau, Hector Ruiz Barradas, Marc Frappier, and Amel Mammar. “A Formal Requirements Modeling Approach: Application to Rail Communication.” In: *Proceedings of the 14th International Conference on Software Technologies (ICSOFT)*. SciTePress, 2019, pp. 170–177 (cit. on p. 30).

[16]Ludovic Apvrille and Yves impRoudier. “SysML-sec: A SysML environment for the design and development of secure embedded systems”. In: *Asia-Pacific Council on Systems Engineering (APCOSEC)* (Sept. 2013), pp. 8–11 (cit. on pp. 31, 54, 76, 126).

[17]Clarity. *“Clarity project” http://www.clarity-se.org*. 2015 (cit. on p. 32).

[18]Christophe Boudjennah, Benoit Combemale, Daniel Exertier, Stéphane Lacrampe, and Marie-Agnès Peraldi-Frati. “CLARITY: Open-Sourcing the Model-Based Systems Engineering Solution Capella”. In: *Second Workshop on Open Source Software for Model Driven Engineering*. CEUR. 2015 (cit. on p. 32).

[19]Capella. *“Introduction to Arcadia” http://www.polarsys.org/capella/arcadia.html*. 2014 (cit. on p. 32).

[20]Pascal Roques. “MBSE with the ARCADIA Method and the Capella Tool”. In: *8th European Congress on Embedded Real Time Software and Systems*. 2016 (cit. on pp. 32, 46).

[21]Hui Zhao, Ludovic Apvrille, and Frédéric Mallet. “Multi-View Design for Cyber-Physical Systems”. In: *PhD Symposium at 13th International Conference on ICT in Education, Research, and Industrial Applications*. 2017, pp. 22–28 (cit. on p. 32).

[22] Jay Lee, Behrad Bagheri, and Hung-An Kao. “A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems”. In: *Manufacturing Letters* 3 (Jan. 2015), pp. 18–23 (cit. on pp. 39, 41).

[23] Andreas Wortmann, Benoît Combemale, and Olivier Barais. “A Systematic Mapping Study on Modeling for Industry 4.0.” In: *2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)* (2017), p. 25 (cit. on pp. 39, 55, 63).

[24] Kleanthis Thramboulidis and Foivos Christoulakis. “UML4IoT—A UML-based approach to exploit IoT in cyber-physical manufacturing systems”. In: *Computers in Industry* 82.C (Oct. 2016), pp. 259–272 (cit. on pp. 39, 42, 75).

[25] Ragunathan Rajkumar, Lee Insup, Sha Lui, and John Stankovic. “Cyber-physical systems: the next computing revolution”. In: *Design Automation Conference*. June 2010, pp. 731–736 (cit. on p. 39).

[26] Michael Hübner. “Introduction to the special section on multiprocessor system-on-chip for cyber-physical systems”. In: *ACM Trans. Embed. Comput. Syst.* 12.1s (Mar. 2013), pp. 1–1 (cit. on p. 40).

[27] Eloy Garcia, Panos J Antsaklis, Luis A Montestruque, et al. *Model-based control of networked systems*. Springer International Publishing, 2014 (cit. on p. 40).

[28] Peter Hehenberger, Birgit Vogel-Heuser, David Bradley, et al. “Design, modelling, simulation and integration of cyber physical systems: Methods and applications”. In: *Computers in Industry* 82 (2016), pp. 273–289 (cit. on pp. 40, 45, 49).

[29] Shiyong Wang, Jiafu Wan, Di Li, and Chunhua Zhang. “Implementing smart factory of industrie 4.0: an outlook”. In: *International Journal of Distributed Sensor Networks* 12.1 (2016), p. 3159805 (cit. on p. 40).

[30] Sakshi Popli, Rakesh Kumar Jha, and Sanjeev Jain. “A survey on energy efficient narrowband internet of things (NBIoT): architecture, application and challenges”. In: *IEEE Access* 7 (2018), pp. 16739–16776 (cit. on p. 40).

[31] Fei Tao, He Zhang, Ang Liu, and A. Y. C. Nee. “Digital Twin in Industry: State-of-the-Art”. In: *IEEE Transactions on Industrial Informatics* 15.4 (2018), pp. 2405–2415 (cit. on pp. 41, 42).

[32] Baheti Radhakisan and Helen Gill. “Cyber-physical systems”. In: *The Impact of Control Technology* 12.1 (Mar. 2011), pp. 161–166 (cit. on p. 42).

[33] Krishna Sampigethaya and Radha Poovendran. “Aviation Cyber-Physical Systems: Foundations for Future Aircraft and Air Transport”. In: *Proceedings of the IEEE* 101.8 (2013), pp. 1834–1855 (cit. on p. 42).

[34] Junhua Zhao, Fushuan Wen, Yusheng Xue, Xue Li, and Zhaoyang Dong. "Cyber physical power systems: architecture, implementation techniques and challenges". In: *Automation of Electric Power Systems* 34.16 (2010), pp. 1–7 (cit. on p. 42).

[35] Lichen Zhang. "Multi-view approach to model aerospace cyber-physical systems." In: *19th International Conference on Automation and Computing (ICAC)* (2013), pp. 1–6 (cit. on pp. 42, 47).

[36] Samarjit Chakraborty, Mohammad Abdullah Al Faruque, Wanli Chang, et al. "Automotive Cyber‐Physical Systems: A Tutorial Introduction". In: *IEEE Design & Test* 33.4 (2016), pp. 92–108 (cit. on p. 42).

[37] Jiateng Yin, Tao Tang, Lixing Yang, et al. "Research and development of automatic train operation for railway transportation systems: A survey". In: *Transportation Research Part C: Emerging Technologies* 85 (Dec. 2017), pp. 548–572 (cit. on p. 42).

[38] Armin Zimmermann and Günter Hommel. "A train control system case study in model-based real time system design". In: *International Parallel and Distributed Processing Symposium*. IEEE. 2003, pp. 8–14 (cit. on p. 42).

[39] SungHyun Kim and Sungbum Park. "CPS (cyber physical system) based manufacturing system optimization". In: *Procedia computer science* 122 (2017), pp. 518–524 (cit. on p. 42).

[40] Ahmadzai Ahmadi, Chantal Cherifi, Vincent Cheutet, and Yacine Ouzrout. "A review of CPS 5 components architecture for manufacturing based on standards". In: *11th International Conference on Software, Knowledge, Information Management and Applications (SKIMA)*. IEEE. 2017, pp. 1–6 (cit. on p. 42).

[41] Marco Garetti, Luca Fumagalli, and Elisa Negri. "Role of ontologies for CPS implementation in manufacturing". In: *Management and Production Engineering Review* (2015) (cit. on p. 42).

[42] Abdulmalik Humayed, Jingqiang Lin, Fengjun Li, and Bo Luo. "Cyber-physical systems security—A survey". In: *IEEE Internet of Things Journal* 4.6 (2017), pp. 1802–1831 (cit. on pp. 42, 125).

[43] Insup Lee and Oleg Sokolsky. "Medical cyber physical systems". In: *Design automation conference*. IEEE. 2010, pp. 743–748 (cit. on p. 42).

[44] Yin Zhang, Meikang Qiu, Chun-Wei Tsai, Mohammad Mehedi Hassan, and Atif Alamri. "Health-CPS: Healthcare cyber-physical system assisted by cloud and big data". In: *IEEE Systems Journal* 11.1 (2015), pp. 88–95 (cit. on p. 42).

[45] Arjen A van der Meer, Peter Palensky, Kai Heussen, et al. “Cyber-physical energy systems modeling, test specification, and co-simulation based testing”. In: *Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*. IEEE. 2017, pp. 1–9 (cit. on p. 42).

[46] Azfar Khalid, Pierre Kirisci, Zeashan Hameed Khan, et al. “Security framework for industrial collaborative robotic cyber-physical systems”. In: *Computers in Industry* 97 (2018), pp. 132–145 (cit. on p. 42).

[47] Jiang Wan, Arquimedes Canedo, and Mohammad Abdullah Al Faruque. “Security-aware functional modeling of cyber-physical systems”. In: *20th Conference on Emerging Technologies & Factory Automation (ETFA)*. IEEE. 2015, pp. 1–4 (cit. on pp. 42, 58, 125).

[48] Mike Burmester, Emmanouil Magkos, and Vassilis Chrissikopoulos. “Modeling security in cyber–physical systems”. In: *International Journal of Critical Infrastructure Protection* 5.3-4 (2012), pp. 118–126 (cit. on pp. 42, 43, 45).

[49] Geng Yang, Zhibo Pang, M Jamal Deen, et al. “Homecare robotic systems for healthcare 4.0: visions and enabling technologies”. In: *IEEE journal of biomedical and health informatics* 24.9 (2020), pp. 2535–2549 (cit. on p. 42).

[50] Hongpeng Wang, Jingtai Liu, and Jianda Han. “RS-CPS: A distributed architecture of robotic surveillance cyber-physical system in the nature environment”. In: *International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)*. IEEE. 2015, pp. 1287–1292 (cit. on p. 42).

[51] Gang Yang, Xingshe Zhou, and Yuanyuan Lian. “Constraint-Based Consistency Checking for Multi-View Models of Cyber-Physical System”. In: *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. July 2017, pp. 370–376 (cit. on pp. 42, 75).

[52] Christos Tsigkanos, Timo Kehrer, and Carlo Ghezzi. “Modeling and Verification of Evolving Cyber-physical Spaces”. In: *11th Joint Meeting on Foundations of Software Engineering*. New York, NY, USA: ACM, 2017, pp. 38–48 (cit. on pp. 42, 44).

[53] Yongxiang Bao, Mingsong Chen, Qi Zhu, et al. “Quantitative Performance Evaluation of Uncertainty-Aware Hybrid AADL Designs Using Statistical Model Checking”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2017), pp. 18–24 (cit. on p. 43).

[54] Muhammad Sabir Idrees, Yves Roudier, and Ludovic Apvrille. “Model the System from Adversary Viewpoint - Threats Identification and Modeling.” In: *Electronic Proceedings in Theoretical Computer Science* 165 (Oct. 2014), pp. 45–58 (cit. on p. 43).

[55] Moussa Amrani, Dominique Blouin, Robert Heinrich, et al. “Towards a formal specification of multi-paradigm modelling”. In: *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE. 2019, pp. 419–424 (cit. on p. 45).

[56] Moussa Amrani, Dominique Blouin, Robert Heinrich, et al. “Multi-paradigm modelling for cyber–physical systems: a descriptive framework”. In: *Software and Systems Modeling* (2021), pp. 1–29 (cit. on p. 45).

[57] Ankica Barišić, Ivan Ruchkin, Dušan Savić, et al. “Multi-paradigm modeling for cyber‑physical systems: A systematic mapping review”. In: *Journal of Systems and Software* 183 (2022), p. 111081 (cit. on pp. 45, 63).

[58] Jean-Marc Jezequel. “Model driven design and aspect weaving”. In: *Software and Systems Modeling* 7.2 (Feb. 2008), pp. 209–218 (cit. on pp. 45, 74, 75).

[59] Jean Bézivin. “Model driven engineering: An emerging technical space”. In: *Generative and transformational techniques in software engineering*. Springer, 2006, pp. 36–64 (cit. on p. 45).

[60] Ludovic Apvrille, Letitia Li, and Yves Roudier. “Model-Driven Engineering for Designing Safe and Secure Embedded Systems”. In: *2016 Architecture-Centric Virtual Integration (ACVI)*. IEEE, Apr. 2016, pp. 4–7 (cit. on pp. 45, 55, 58, 76, 126).

[61] Aditya A Shah, Aleksandr A Kerzhner, Dirk Schaefer, and Christiaan JJ Paredis. “Multi-view modeling to support embedded systems engineering in SysML”. In: *Graph transformations and model-driven engineering*. Berlin, Heidelberg: Springer, 2010, pp. 580–601 (cit. on pp. 45, 63).

[62] Yves Roudier and Ludovic Apvrille. “SysML-Sec: A model driven approach for designing safe and secure systems”. In: *International Conference on ModelDriven Engineering and Software Development (MODELSWARD)*. Feb. 2015, pp. 655–664 (cit. on pp. 46, 54, 58, 76, 126).

[63] Phu H Nguyen, Max Kramer, Jacques Klein, and Yves Le Traon. “An extensive systematic review on the Model-Driven Development of secure systems”. In: *Information and Software Technology* 68 (2015), pp. 62–81 (cit. on p. 46).

[64] Florian Lugou, Letitia W Li, Ludovic Apvrille, and Rabéa Ameur-Boulifa. “Sysml models and model transformation for security”. In: *International Conference on ModelDriven Engineering and Software Development (MODELSWARD)*. IEEE. 2016, pp. 331–338 (cit. on p. 46).

[65] Hüseyin Ergin, Eugene Syriani, and Jeff Gray. “Design pattern oriented development of model transformations”. In: *Computer Languages, Systems & Structures* 46 (Nov. 2016), pp. 106–139 (cit. on pp. 46, 63, 144).

[66] ISO IEC IEEE. “Systems and Software Engineering–Architecture Description”. In: *ISO/IEC/IEEE 42010: 2011 (E)(Revision of ISO/IEC 42010: 2007 and IEEE Std 1471-2000)* (2011) (cit. on p. 47).

[67] Matias Ezequiel Vara Larsen, Julien DeAntoni, Benoît Combemale, and Frédéric Mallet. “A Behavioral Coordination Operator Language (BCOoL)”. In: *18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 2015, pp. 186–195 (cit. on pp. 49, 58, 67).

[68] Michael Heymann, Feng Lin, George Meyer, and Stefan Resmerita. “Analysis of zeno behaviors in hybrid systems”. In: *41st IEEE Conference on Decision and Control, 2002.* Vol. 3. IEEE. 2002, pp. 2379–2384 (cit. on p. 49).

[69] Bran Selic. “UML profile for MARTE: modeling and analysis of real-time embedded systems”. In: *Technical Report, report number: formal/2011-06-02* (June 2011), pp. 1–754 (cit. on pp. 50, 58).

[70] OMG. *Unified Modeling Language*. [Online]. Available: https://www.omg.org/spec/UML/About-UML/. Apr. 2019 (cit. on pp. 50, 58, 87).

[71] Julien DeAntoni and Frédéric Mallet. “ECL: the Event Constraint Language, an Extension of OCL with Events”. In: *Research report* (July 2012), p. 24 (cit. on p. 50).

[72] Aamir M Khan, Frédéric Mallet, and Muhammad Rashid. “Combining SysML and Marte/CCSL to Model Complex Electronic Systems”. In: *2016 International Conference on Information Systems Engineering (ICISE)*. IEEE, 2016, pp. 12–17 (cit. on p. 50).

[73] Yves Roudier, Muhammad Sabir Idrees, and Ludovic Apvrille. “Towards the model-driven engineering of security requirements for embedded systems.” In: *2013 3rd International Workshop on Model-Driven Requirements Engineering (MoDRE)* (2013), pp. 55–64 (cit. on p. 51).

[74] Letitia W Li, Florian Lugou, and Ludovic Apvrille. “Security Modeling for Embedded System Design”. In: *International Workshop on Graphical Models for Security*. Springer. 2017, pp. 99–106 (cit. on p. 51).

[75] Letitia Li. “Safe and secure model-driven design for embedded systems”. PhD thesis. Université Paris-Saclay, Sept. 2018 (cit. on pp. 51, 132, 135).

[76] V Normand and D Exertier. “Model-driven systems engineering: SysML & the MDSysE approach at Thales”. In: *Ecole d’été CEA-ENSIETA, Brest, France* (2004) (cit. on p. 52).

[77] Capella. *Website of Capella/Arcadia*. [Online]. Available: https://www.polarsys.org/capella/arcadia.html. 2019 (cit. on p. 52).

[78] Letitia W. Li, Florian Lugou, and Ludovic Apvrille. "Security-Aware Modeling and Analysis for HW/SW Partitioning". In: *International Conference on ModelDriven Engineering and Software Development (MODELSWARD)*. Porto, Portugal, Feb. 2017, pp. 302–311 (cit. on pp. 55, 58, 76).

[79] L Li, L Apvrille, and D Genius. "Virtual Prototyping of Automotive Systems: Towards Multi-level Design Space Exploration". In: *Conference on Design & Architectures for Signal & Image Processing (DASIP'2016)* (2016) (cit. on pp. 55, 76).

[80] Daniela Genius, Letitia Li, and Ludovic Apvrille. "Model-Driven Performance Evaluation and Formal Verification for Multi-level Embedded System Design". In: *5th International Conference on Model-Driven Engineering and Software Development*. SCITEPRESS, 2017, pp. 78–89 (cit. on pp. 55, 76).

[81] David Alexandre, Kim G Larsen, Axel Legay, Marius Mikučionis, and Danny Bøgsted. "Uppaal SMC tutorial". In: *International Journal on Software Tools for Technology Transfer* 17.4 (2015), pp. 397–415 (cit. on pp. 55, 76).

[82] Bruno Blanchet, Ben Smyth, Vincent Cheval, and Marc Sylvestre. *ProVerif 2.00: Automatic Cryptographic Protocol Verifier,User Manual and Tutorial*. May 2018 (cit. on pp. 55, 76).

[83] Jose Fran Ruiz, Carsten Rudolph, Antonio Mana, and Marcos Arjona. "A security engineering process for systems of systems using security patterns". In: *8th Annual IEEE Systems Conference (SysCon)*. IEEE, 2014, pp. 8–11 (cit. on p. 55).

[84] L Zhang. "Modeling large scale complex cyber physical control systems based on system of systems engineering approach". In: *20th International Conference on Automation and Computing* (2014), pp. 55–60 (cit. on p. 55).

[85] Muhammad Rashid, Muhammad Waseem Anwar, and Aamir M Khan. "Toward the tools selection in model based system engineering for embedded systems—A systematic literature review". In: *Journal of Systems and Software* 106 (Aug. 2015), pp. 150–163 (cit. on p. 55).

[86] Ilge Akkaya, Patricia Derler, Shuhei Emoto, and Edward A Lee. "Systems engineering for industrial cyber–physical systems using aspects". In: *Proceedings of the IEEE* 104.5 (2016), pp. 997–1012 (cit. on p. 55).

[87] T Watteyne, P Tuset-Peiro, and X Vilajosana. "Teaching Communication Technologies and Standards for the Industrial IoT? Use 6TiSCH!" In: *IEEE Communication Magazine* (2017), pp. 132–139 (cit. on p. 56).

[88] OMG. *Systems Modeling Language*. May 2017 (cit. on p. 58).

[89] Phu H Nguyen, Shaukat Ali, and Tao Yue. “Model-based security engineering for cyber-physical systems: A systematic mapping study”. In: *Information and Software Technology* 83 (Mar. 2017), pp. 116–135 (cit. on p. 58).

[90] Charles André and Frédéric Mallet. “Specification and verification of time requirements with CCSL and Esterel”. In: *2009 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems*. 2009, pp. 167–176 (cit. on p. 58).

[91] N Kahani and JR Cordy. “Comparison and evaluation of model transformation tools”. In: *Queen’s University, Kingston, Tech. Rep* (2015) (cit. on pp. 58, 64, 66, 68).

[92] Michael Brunner, Michael Huber, Clemens Sauerwein, and Ruth Breu. “Towards an Integrated Model for Safety and Security Requirements of Cyber-Physical Systems”. In: *International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, pp. 334–340 (cit. on p. 58).

[93] Ayan Banerjee, Krishna K Venkatasubramanian, Tridib Mukherjee, and Sandeep Kumar S Gupta. “Ensuring Safety, Security, and Sustainability of Mission-Critical Cyber−Physical Systems”. In: *Proceedings of the IEEE* 100.1 (2011), pp. 283–299 (cit. on p. 58).

[94] Florian Lugou, Ludovic Apvrille, and Aurélien Francillon. “SMASHUP - a toolchain for unified verification of hardware/software co-designs.” In: *J. Cryptographic Engineering* 7.1 (2017), pp. 63–74 (cit. on p. 58).

[95] Verislav Djukić, Aleksandar Popović, and Juha-Pekka Tolvanen. “Domain-specific modeling for robotics: from language construction to ready-made controllers and end-user applications”. In: *3rd Workshop on Model-Driven Robot Software Engineering*. NY, USA, July 2016, pp. 47–54 (cit. on p. 58).

[96] Rohit Sinha, Sriram K Rajamani, Sanjit A Seshia, and Kapil Vaswani. “Moat - Verifying Confidentiality of Enclave Programs.” In: *ACM Conference on Computer and Communications Security* (2015), pp. 1169–1184 (cit. on p. 58).

[97] T Correa, L B Becker, J M Farines, et al. “Supporting the Design of Safety Critical Systems Using AADL”. In: *15th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*. IEEE, 2010, pp. 331–336 (cit. on p. 58).

[98] D. Ameller, X. Franch, C. Gómez, et al. “Dealing with Non-Functional Requirements in Model-Driven Development: A Survey”. In: *IEEE Transactions on Software Engineering* (2019), pp. 1–1 (cit. on p. 63).

[99] Fang Li, Jiafu Wan, Ping Zhang, and Di Li. “A multi-view integration modeling approach for cyber-physical robot system”. In: *International Conference on Machine Learning and Cybernetics*. Vol. 1. IEEE, 2013, pp. 387–392 (cit. on p. 63).

[100] Bernhard Hoisl, Zhenjiang Hu, and Soichiro Hidaka. “Towards Co-evolution in Model-Driven Development Via Bidirectional Higher-Order Transformation.” In: *International Conference on ModelDriven Engineering and Software Development (MODELSWARD)* (July 2014) (cit. on p. 63).

[101] Manuel Wimmer, Angelika Kusel, Werner Retschitzegger, et al. “Reusing Model Transformations across Heterogeneous Metamodels”. In: *Electronic Communications of the EASST* 50 (Jan. 2012) (cit. on pp. 63, 71).

[102] Angelika Kusel, Johannes Schönböck, M. Wimmer, Werner Retschitzegger, and W. Schwinger. “Reuse in model-to-model transformation languages: are we there yet?” In: *Software & Systems Modeling* 14 (May 2013), pp. 1–36 (cit. on pp. 63, 64, 71).

[103] Jean-Michel Bruel, Benoît Combemale, Esther Guerra, et al. “Comparing and classifying model transformation reuse approaches across metamodels”. In: *Software and Systems Modeling* 19.2 (2020), pp. 441–465 (cit. on pp. 63, 66).

[104] Jesús Sánchez Cuadrado, Esther Guerra, and Juan De Lara. “Generic model transformations: write once, reuse everywhere”. In: *International Conference on Theory and Practice of Model Transformations*. Springer. 2011, pp. 62–77 (cit. on pp. 63, 71).

[105] Degueule Thomas, Benoît Combemale, Arnaud Blouin, Olivier Barais, and Jean-Marc Jezequel. “Melange: A meta-language for modular and reusable development of dsls”. In: *Conference on Software Language Engineering*. 2015, pp. 25–36 (cit. on pp. 64, 74, 75).

[106] Dániel Varró and András Pataricza. “Generic and meta-transformations for model transformation engineering”. In: *International Conference on the Unified Modeling Language*. Springer. 2004, pp. 290–304 (cit. on pp. 64, 65).

[107] Arnaud Cuccuru, Chokri Mraidha, François Terrier, and Sébastien Gérard. “Templatable metamodels for semantic variation points”. In: *European Conference on Model Driven Architecture-Foundations and Applications*. Springer. 2007, pp. 68–82 (cit. on p. 65).

[108] Cláudio Gomes, Bruno Barroca, and Vasco Amaral. “Classification of model transformation tools: pattern matching techniques”. In: *International Conference on Model Driven Engineering Languages and Systems*. Springer. 2014, pp. 619–635 (cit. on p. 66).

[109] Soichiro Hidaka, Massimo Tisi, Jordi Cabot, and Zhenjiang Hu. “Feature-based classification of bidirectional transformation approaches”. In: *Software & Systems Modeling* 15.3 (2016), pp. 907–928 (cit. on p. 66).

[110] Gabriele Taentzer, Karsten Ehrig, Esther Guerra, et al. “Model transformation by graph transformation: A comparative study”. In: *Workshop Model Transformation in Practice* (2005) (cit. on p. 66).

[111] Edgar Jakumeit, Sebastian Buchwald, Dennis Wagelaar, et al. “A survey and comparison of transformation tools based on the transformation tool contest”. In: *Science of computer programming* 85 (2014), pp. 41–99 (cit. on p. 66).

[112] Tom Mens and Pieter Van Gorp. “A taxonomy of model transformation”. In: *Electronic notes in theoretical computer science* 152 (2006), pp. 125–142 (cit. on p. 66).

[113] Krzysztof Czarnecki and Simon Helsen. “Feature-based survey of model transformation approaches”. In: *IBM systems journal* 45.3 (2006), pp. 621–645 (cit. on p. 66).

[114] Jean-Michel Bruel, Benoit Combemale, Esther Guerra, et al. “Model Transformation Reuse Across Metamodels”. In: *International Conference on Theory and Practice of Model Transformations*. Springer. 2018, pp. 92–109 (cit. on p. 66).

[115] Hugo Bruneliere, Jordi Cabot, Frédéric Jouault, and Frédéric Madiot. “MoDisco: A Generic and Extensible Framework for Model Driven Reverse Engineering”. In: *International Conference on Automated Software Engineering*. NY, USA: ACM, 2010 (cit. on p. 66).

[116] Michael Lawley and Jim Steel. “Practical declarative model transformation with Tefkat”. In: *International Conference on Model Driven Engineering Languages and Systems*. Springer. 2005, pp. 139–150 (cit. on p. 67).

[117] Jesús M Almendros-Jiménez, Luis Iribarne, Jesús López-Fernández, and Ángel Mora-Segura. “PTL: A model transformation language based on logic programming”. In: *Journal of Logical and Algebraic Methods in Programming* 85.2 (2016), pp. 332–366 (cit. on p. 67).

[118] Kevin Lano and Shekoufeh Kolahdouz-Rahimi. “Specification and verification of model transformations using UML-RSDS”. In: *International Conference on Integrated Formal Methods*. Springer. 2010, pp. 199–214 (cit. on p. 67).

[119] Antonio Cicchetti, Davide Di Ruscio, Romina Eramo, and Alfonso Pierantonio. “JTL: a bidirectional and change propagating transformation language”. In: *International Conference on Software Language Engineering*. Springer. 2010, pp. 183–202 (cit. on p. 67).

[120] Dan Li, Xiaoshan Li, and Volker Stolz. “QVT-based model transformation using XSLT”. In: *ACM SIGSOFT Software Engineering Notes* 36.1 (2011), pp. 1–8 (cit. on p. 67).

[121] Nuno Macedo and Alcino Cunha. “Implementing QVT-R bidirectional model transformations using Alloy”. In: *International Conference on Fundamental Approaches to Software Engineering*. Springer. 2013, pp. 297–311 (cit. on p. 67).

[122] Christopher Gerking and Christian Heinzemann. “Solving the Movie Database Case with QVTo.” In: *TTC@STAF*. 2014, pp. 98–102 (cit. on pp. 67, 68).

[123] Steven Kelly, Kalle Lyytinen, and Matti Rossi. "Metaedit+ a fully configurable multi-user and multi-tool case and came environment". In: *International Conference on Advanced Information Systems Engineering*. Springer. 1996, pp. 1–21 (cit. on p. 67).

[124] Zoé Drey, Cyril Faucher, Franck Fleurey, Vincent Mahé, and Didier Vojtisek. "Kermeta language reference manual". In: *Manuscript available online http://www. kermeta. org* (2009) (cit. on p. 67).

[125] Eugene Syriani, Hans Vangheluwe, Raphael Mannadiar, et al. "AToMPM: A web-based modeling environment". In: *16th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 2013, pp. 21–25 (cit. on p. 69).

[126] Martin Fleck, Javier Troya, and Manuel Wimmer. "Marrying search-based optimization and model transformation technology". In: *Proc. of NasBASE* (2015), pp. 1–16 (cit. on p. 69).

[127] Arend Rensink. "The GROOVE simulator: A tool for state space generation". In: *International Workshop on Applications of Graph Transformations with Industrial Relevance*. Springer. 2003, pp. 479–485 (cit. on p. 69).

[128] Claudia Ermel, Michael Rudolf, and Gabriele Taentzer. "The AGG approach: Language and environment". In: *Handbook Of Graph Grammars And Computing By Graph Transformation: Volume 2: Applications, Languages and Tools*. World Scientific, 1999, pp. 551–603 (cit. on p. 69).

[129] Peter Braun and Frank Marschall. "Transforming object oriented models with BOTL". In: *Electronic Notes in Theoretical Computer Science* 72.3 (2003), pp. 103–117 (cit. on p. 69).

[130] Soichiro Hidaka, Zhenjiang Hu, Kazuhiro Inaba, Hiroyuki Kato, and Keisuke Nakano. "GRoundTram: An integrated framework for developing well-behaved bidirectional model transformations". In: *26th IEEE/ACM International Conference on Automated Software Engineering*. IEEE. 2011, pp. 480–483 (cit. on p. 69).

[131] Andy Schürr. "Specification of graph translators with triple graph grammars". In: *International Workshop on Graph-Theoretic Concepts in Computer Science*. Springer. 1994, pp. 151–163 (cit. on p. 69).

[132] Thorsten Arendt, Enrico Biermann, Stefan Jurack, Christian Krause, and Gabriele Taentzer. "Henshin: advanced concepts and tools for in-place EMF model transformations". In: *International Conference on Model Driven Engineering Languages and Systems*. Springer. 2010, pp. 121–135 (cit. on p. 69).

[133] Lilija Klassen and Robert Wagner. "EMorF-A tool for model transformations". In: *Electronic Communications of the EASST* 42.C (2012) (cit. on p. 69).

[134] OMG. *Meta Object Facility (MOF) Core Specification*. Oct. 2016 (cit. on p. 70).

[135] Gordon D Plotkin. “A structural approach to operational semantics”. In: *Computer Science Department, Aarhus University* (1981) (cit. on p. 70).

[136] Charles Antony Richard Hoare. “An axiomatic basis for computer programming”. In: *Communications of the ACM* 12.10 (1969), pp. 576–580 (cit. on p. 70).

[137] Lars-åke Fredlund, Bengt Jonsson, and Joachim Parrow. “An implementation of a translational semantics for an imperative language”. In: *International Conference on Concurrency Theory*. Springer Berlin Heidelberg, 1990, pp. 246–262 (cit. on p. 70).

[138] Muhammad Waqar Aziz and Muhammad Rashid. “Domain Specific Modeling Language for Cyber Physical Systems”. In: *2016 International Conference on Information Systems Engineering (ICISE)*. IEEE, 2016, pp. 29–33 (cit. on p. 70).

[139] John Edward Hutchinson, Jon Whittle, Mark Rouncefield, and Steinar Kristoffersen. “Empirical assessment of MDE in industry.” In: *Proceedings of the 33rd international conference on software engineering (ICSE)* (2011), p. 471 (cit. on p. 70).

[140] Paul Hudak. “Modular Domain Specific Languages and Tools”. In: *5th International Conference on Software Reuse* (June 1999) (cit. on p. 70).

[141] Markus Völter. “Language and IDE Modularization, Extension and Composition with MPS”. In: *Pre-proceedings of Summer School on Generative and Transformational Techniques in Software Engineering (GTTSE)* 7680 (July 2011), pp. 359–431 (cit. on pp. 70, 71).

[142] Markus Völter and Konstantin Solomatov. “Language Modularization and Composition with Projectional Language Workbenches illustrated with MPS”. In: *Software Language Engineering* 16.3 (2010) (cit. on p. 71).

[143] Martin Erwig, Richard F Paige, and Eric Van Wyk. “The state of the art in language workbenches-conclusions from the language workbench challenge”. In: *Software Language Engineering-6th International Conference (SLE)*. Vol. 8225. Springer. 2013, pp. 197–217 (cit. on p. 71).

[144] Jean-Marc Jézéquel, Benoit Combemale, Olivier Barais, Martin Monperrus, and Francois Fouquet. “Mashup of Meta-Languages and its Implementation in the Kermeta Language Workbench”. In: *Software & Systems Modeling* 14 (June 2013) (cit. on p. 71).

[145] Joao Saraiva. “Component-based programming for higher-order attribute grammars”. In: *International Conference on Generative Programming and Component Engineering*. Springer. 2002, pp. 268–282 (cit. on p. 71).

[146] Uwe Kastens and William Waite. “Modularity and Reusability in Attribute Grammars.” In: *Acta Informatica* 31 (July 1994), pp. 601–627 (cit. on p. 71).

[147] Holger Krahn, Bernhard Rumpe, and Steven Völkel. "MontiCore: A framework for compositional development of domain specific languages". In: *nternational journal on software tools for technology transfer* 12.5 (Sept. 2010), pp. 353–372 (cit. on p. 71).

[148] Srdjan Zivkovic and Dimitris Karagiannis. "Towards Metamodelling-In-The-Large: Interface-Based Composition for Modular Metamodel Development". In: vol. 214. June 2015, pp. 414–428 (cit. on p. 71).

[149] Edoardo Vacchi and Walter Cazzola. "Neverlang: A framework for feature-oriented language development". In: *Computer Languages, Systems & Structures* 43 (Feb. 2015) (cit. on p. 71).

[150] Akos Ledeczi, Árpád Bakay, Miklós Maróti, et al. "Composing Domain-specific Design Environments". In: *Computer* 34.11 (Dec. 2001), pp. 44–51 (cit. on p. 71).

[151] Jacky Estublier, German Vega, and Anca Daniela Ionita. "Composing domain-specific languages for wide-scope software engineering applications". In: *International Conference on Model Driven Engineering Languages and Systems*. Springer. 2005, pp. 69–83 (cit. on p. 71).

[152] Sebastian Erdweg, Paolo G Giarrusso, and Tillmann Rendel. "Language composition untangled." In: *Proceedings of the Twelfth Workshop on Language Descriptions, Tools, and Applications* (2012), pp. 1–8 (cit. on p. 71).

[153] Moritz Eysholdt and Heiko Behrens. "Xtext: implement your language faster than the quick and dirty way". In: *International conference companion on Object oriented programming systems languages and applications companion*. ACM, 2010, pp. 307–309 (cit. on p. 71).

[154] Dimitrios S Kolovos, Louis M Rose, Nicholas Matragkas, et al. "A research roadmap towards achieving scalability in model driven engineering". In: *The Workshop on Scalability in Model Driven Engineering*. 2013, pp. 1–10 (cit. on p. 71).

[155] Jim Steel and Jean-Marc Jézéquel. "On model typing". In: *Software & Systems Modeling* 6 (Dec. 2007), pp. 401–413 (cit. on p. 71).

[156] Juan de Lara, Esther Guerra, and Jesús Sánchez-Cuadrado. "Abstracting modelling languages: A reutilization approach". In: *International Conference on Advanced Information Systems Engineering*. Springer. 2012, pp. 127–143 (cit. on p. 71).

[157] Juan De Lara and Esther Guerra. "Generic meta-modelling with concepts, templates and mixin layers". In: *International Conference on Model Driven Engineering Languages and Systems*. Springer. 2010, pp. 16–30 (cit. on p. 71).

[158] Sebastian Erdweg, Paolo G Giarrusso, and Tillmann Rendel. “Language composition untangled”. In: *12th Workshop on Language Descriptions, Tools, and Applications*. 2012, pp. 1–8 (cit. on p. 71).

[159] Maged Elaasar, Florian Noyrit, Omar Badreddin, and Sébastien Gérard. “Reducing UML Modeling Tool Complexity with Architectural Contexts and Viewpoints.” In: *International Conference on ModelDriven Engineering and Software Development (MODELSWARD)*. 2018, pp. 129–138 (cit. on p. 72).

[160] A. Haber, M. Look, A. N. Perez, et al. “Integration of heterogeneous modeling languages via extensible and composable language components”. In: *International Conference on ModelDriven Engineering and Software Development (MODELSWARD)*. Feb. 2015, pp. 19–31 (cit. on p. 72).

[161] Pierre De Saqui-Sannes and Jérôme Hugues. “Combining SysML and AADL for the design, validation and implementation of critical systems”. In: *ERTS*. 2012, p. 117 (cit. on pp. 72, 73).

[162] Razieh Behjati, Tao Yue, Shiva Nejati, Lionel Briand, and Bran Selic. “Extending SysML with AADL concepts for comprehensive system architecture modeling”. In: *European Conference on Modelling Foundations and Applications*. Springer. 2011, pp. 236–252 (cit. on pp. 72, 73).

[163] Matthias Brun, Thomas Vergnaud, Madeleine Faugere, and Jérôme Delatour. “From UML to AADL: an Explicit Execution Semantics Modelling with MARTE”. In: *ERTS*. 2008 (cit. on p. 73).

[164] Skander Turki, Eric Senn, and Dominique Blouin. “Mapping the MARTE UML profile to AADL”. In: *3rd International Workshop on Model Based Architecting and Construction of Embedded Systems*. Citeseer. 2010, pp. 11–20 (cit. on p. 73).

[165] Bassem Ouni, Pierre Gaufillet, Eric Jenn, and Jérôme Hugues. “Model Driven Engineering with Capella and AADL”. In: *ERTSS* (Jan. 2016) (cit. on p. 73).

[166] Kunal Suri, Arnaud Cuccuru, Juan Cadavid, et al. “Model-based Development of Modular Complex Systems for Accomplishing System Integration for Industry 4.0.” In: *International Conference on ModelDriven Engineering and Software Development (MODELSWARD)*. 2017, pp. 487–495 (cit. on p. 73).

[167] S. Apel, M. Mauch, and V. Schau. “Model-driven engineering tool comparison for architectures within heterogenic systems for Electric vehicle”. In: *International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*. Feb. 2016, pp. 671–676 (cit. on p. 74).

[168] Ivan Kurtev, Mathijs Schuts, Jozef Hooman, and Dirk-Jan Swagerman. “Integrating Interface Modeling and Analysis in an Industrial Setting.” In: *International Conference on ModelDriven Engineering and Software Development (MODELSWARD)*. 2017, pp. 345–352 (cit. on p. 74).

[169] F. Scippacercola, R. Pietrantuono, S. Russo, and A. Zentai. “Model-driven engineering of a railway interlocking system”. In: *International Conference on ModelDriven Engineering and Software Development (MODELSWARD)*. Feb. 2015, pp. 509–519 (cit. on p. 74).

[170] Rodrigo Ramos, Olivier Barais, and Jean-Marc Jezequel. “Matching model-snippets”. In: *International Conference on Model Driven Engineering Languages and Systems*. Springer. 2007, pp. 121–135 (cit. on p. 74).

[171] Antonio Cicchetti, Federico Ciccozzi, and Thomas Leveque. “Supporting incremental synchronization in hybrid multi-view modelling”. In: *International Conference on Model Driven Engineering Languages and Systems*. Springer. 2011, pp. 89–103 (cit. on p. 74).

[172] Carlos Gomez, Julien DeAntoni, and Frédéric Mallet. “Multi-view Power Modeling Based on UML, MARTE and SysML.” In: *38th Euromicro Conference on Software Engineering and Advanced Applications* (Sept. 2012), pp. 17–20 (cit. on p. 74).

[173] Frédéric Boulanger, Christophe Jacquet, Cécile Hardebolle, and Elyes Rouis. “Modeling heterogeneous points of view with ModHel’X”. In: *International Conference on Model Driven Engineering Languages and Systems*. Springer. 2009, pp. 310–324 (cit. on p. 74).

[174] Mahmoud Nassar. “VUML: a Viewpoint oriented UML Extension”. In: *18th International Conference on Automated Software Engineering, 2003. Proceedings.* IEEE. 2003, pp. 373–376 (cit. on p. 74).

[175] Pierre-Alain Muller, Franck Fleurey, and Jean-Marc Jézéquel. “Weaving executability into object-oriented meta-languages”. In: *International Conference on Model Driven Engineering Languages and Systems*. Springer. Oct. 2005, pp. 264–278 (cit. on p. 75).

[176] Ajinkya Bhave, Bruce H Krogh, David Garlan, and Bradley Schmerl. “View Consistency in Architectures for Cyber-Physical Systems”. In: *2011 IEEE/ACM Conference on Cyber-Physical Systems*. Apr. 2011, pp. 151–160 (cit. on p. 75).

[177] Jörg Kienzle, Wisam Al Abed, and Jacques Klein. “Aspect-oriented Multi-view Modeling”. In: *8th ACM Conference on Aspect-oriented Software Development*. 2009, pp. 87–98 (cit. on p. 75).

[178] Arnaud Albinet, Jean-Louis Boulanger, Hubert Dubois, et al. “Model-based methodology for requirements traceability in embedded systems”. In: *Proceedings of 3rd European Conference on Model Driven Architecture Foundations and Applications*. 2007 (cit. on pp. 76, 77).

[179] Lucio F Vismari, João Batista Camargo, Jorge Rady de Almeida, et al. “A practical analytical approach to increase confidence in software safety arguments”. In: *IEEE Systems Journal* 11.4 (2015), pp. 2072–2083 (cit. on p. 77).

[180] Jean-François Pétin, Dominique Evrot, Gérard Morel, and Pascal Lamy. “Combining SysML and formal methods for safety requirements verification”. In: *22nd International Conference on Software & Systems Engineering and their Applications*. 2010 (cit. on p. 77).

[181] Romaric Guillerm, Hamid Demmou, and Nabil Sadou. “Safety evaluation and management of complex systems: A system engineering approach”. In: *Concurrent Engineering* 20.2 (2012), pp. 149–159 (cit. on p. 77).

[182] Daniel D McCracken and Edwin D Reilly. “Backus-naur form (BNF)”. In: *Encyclopedia of Computer Science* (2003), pp. 129–131 (cit. on p. 85).

[183] Hui Zhao, Ludovic Apvrille, and Frédéric Mallet. “Meta-models Combination for Reusing Verification Techniques”. In: *7th International Conference on Model-Driven Engineering and Software Development*. SCITEPRESS-Science and Technology Publications. 2019, pp. 39–50 (cit. on pp. 105, 110, 113, 118, 119).

[184] Frank Singhoff, Jérôme Legrand, Laurent Nana, and Lionel Marcé. “Cheddar - a flexible real time scheduling framework.” In: *Proceedings of the 2004 annual ACM SIGAda international conference on Ada: The engineering of correct and reliable software for real-time distributed systems using Ada and related technologies* (2004), pp. 1–8 (cit. on pp. 106, 118).

[185] Li Zhu, Yan Zhang, Bin Ning, and Hailin Jiang. “Train-ground communication in CBTC based on 802.11 b: Design and performance research”. In: *International Conference on Communications and Mobile Computing*. IEEE. 2009, pp. 368–372 (cit. on p. 118).

[186] Junfeng Wang and Jungang Wang. “A new early warning method of train tracking interval based on CTC”. In: *IEEE Transactions on Intelligent Transportation Systems* (2017), pp. 1–7 (cit. on p. 118).

[187] L Marcé, F Singhoff, J Legrand, and L Nana. “Scheduling and Memory Requirements Analysis with AADL”. In: *International Conference on Ada*. NY, USA: ACM, 2005, pp. 1–10 (cit. on p. 120).

[188] Alexey V Bataev and Ariadna Aleksandrova. “Digitalization of the World Economy: Performance Evaluation of Introducing Cyber-Physical Systems”. In: *International Conference on Industrial Technology and Management (ICITM)*. IEEE. 2020, pp. 265–269 (cit. on p. 125).

[189] George Fortney. “Model based systems engineering using validated executable specifications as an enabler for cost and risk reduction”. In: *Ground Vehicle Systems Engineering and Technology Symposium (GVSETS)*. 2014 (cit. on p. 125).

[190] Marilyn Wolf and Dimitrios Serpanos. “Safety and security in cyber-physical systems and internet-of-things systems”. In: *Proceedings of the IEEE* 106.1 (2017), pp. 9–20 (cit. on p. 126).

[191] Shinpei Kato, Eijiro Takeuchi, Yoshio Ishiguro, et al. “An open approach to autonomous vehicles”. In: *IEEE Micro* 35.6 (2015), pp. 60–68 (cit. on p. 127).

[192] EVITA. *EVITA Project*. [Online]. Available: https://www.evita-project.org. 2009 (cit. on pp. 127, 130, 134).

[193] ISO 26262. *Road vehicles – Functional safety*. Norm. 2011 (cit. on pp. 127, 130).

[194] IEC61508 IEC. “61508 functional safety of electrical/electronic/programmable electronic safety-related systems”. In: *International electrotechnical commission* (1998) (cit. on p. 130).

[195] CENELEC. “EN50126: Railway applications-The specification and demonstration of Reliability”. In: *Availability, Maintainability and Safety (RAMS)* (1999) (cit. on p. 130).

[196] Alessandro Tempia Calvino and Ludovic Apvrille. “Direct Model-checking of SysML Models.” In: *MODELSWARD*. 2021, pp. 216–223 (cit. on p. 132).

[197] Ekkart Kindler. “Safety and liveness properties: A survey”. In: *Bulletin of the European Association for Theoretical Computer Science* 53.268-272 (1994), p. 30 (cit. on p. 134).

[198] Radek Fujdiak, Petr Blažek, Ludovic Apvrille, et al. “Modeling the trade-off between security and performance to support the product life cycle”. In: Budva, Montenegro, June 2019 (cit. on p. 137).

[199] Thomas Stahl, Markus Voelter, and Krzysztof Czarnecki. *Model-driven software development: technology, engineering, management*. John Wiley & Sons, Inc., 2006 (cit. on p. 144).

[200] David Harel and Bernhard Rumpe. *Modeling languages: Syntax, semantics and all that stuff*. Tech. rep. Technical Report MCS00-16, The Weizmann Institute of Science, Rehovot, Israel, Aug. 2000 (cit. on p. 144).

[201] Eric Xing. “SysML: On System and Algorithm Co-design for Practical Machine Learning”. In: *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2018, pp. 2880–2888 (cit. on p. 148).