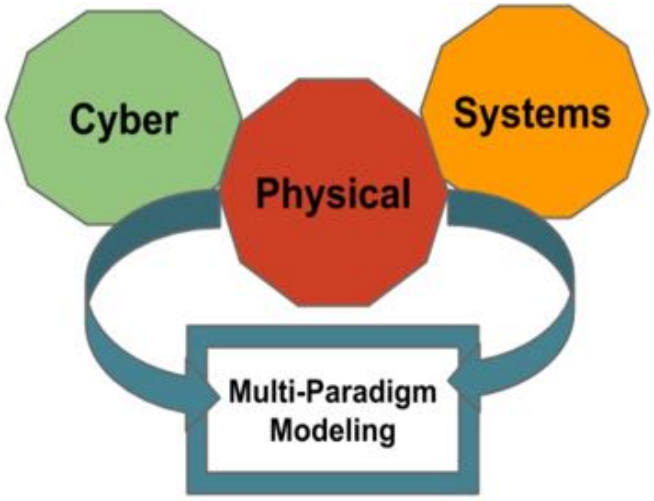

ICT COST Action IC1404

Framework to Relate / Combine Modeling Languages and Techniques

Rima Al-Ali, Moussa Amrani, Soumyadip Bandyopadhyay, Ankica Barišić, Fernando Barros, Dominique Blouin, Ferhat Erata, Holger Giese, Mauro Iacono, Stefan Klikovits, Eva Navarro, Patrizio Pelliccione, Kuldar Taveter, Bedir Tekinerdogan, Ken Vanherpen

Deliverable: WG1.2

Core Team
University of Antwerp, Belgium
New University of Lisbon, Portugal
Telecom ParisTech, Paris, France
Hasso-Plattner Inst., Potsdam, Germany
University of Geneva, Switzerland
Charles University, Prague, Czech Republic
University of Manchester, UK
University of Namur, Belgium
Wageningen University, Netherlands
University of Coimbra, Portugal
Ege University, Izmir, Turkey
University of Campania Luigi Vanvitelli, Caserta, Italy
Tallinn University of Technology, Estonia
Chalmers University of Technology, Goteborg, Sweden

Document Info

Deliverable	WG1.2
Dissemination	Restricted
Status	Final
Doc's Lead Partner	Hasso-Plattner Inst.
Date	January 7, 2019
Version	3.0
Pages	97

Contents

List of Figures

1 Introduction

This document reports on the Framework to Relate / Combine Modeling Languages and Techniques of Working Group1 on Foundations of the ICT COST Action IC1404 Multi-Paradigm Modelling for Cyber-Physical Systems (MPM4CPS). It first presents an ontology of Cyber Physical Systems in chapter 3 and then an ontology of Multi-Paradigm Modeling in chapter 4. Then, these ontologies are combined to define an ontology of Multi-Paradigm Modeling for Cyber-Physical Systems presented in chapter 5. Finally, a number of megamodel examples are presented in chapter 6 that instantiate the core ontologies and make use of the catalog of languages and tools individuals.

The work of working group 1 on foundations revealed that the dependencies between the framework targeted in this report and the state-of-the-art report in form of deliverable D1.1 (78) was much more tight than initially expected. To avoid capturing some content of the state-of-the-art report also in a redundant form in the ontologies of the framework of this report, it was decided instead to include the relevant information in the ontologies and extract it from there automatically for generating the state-of-the-art report.

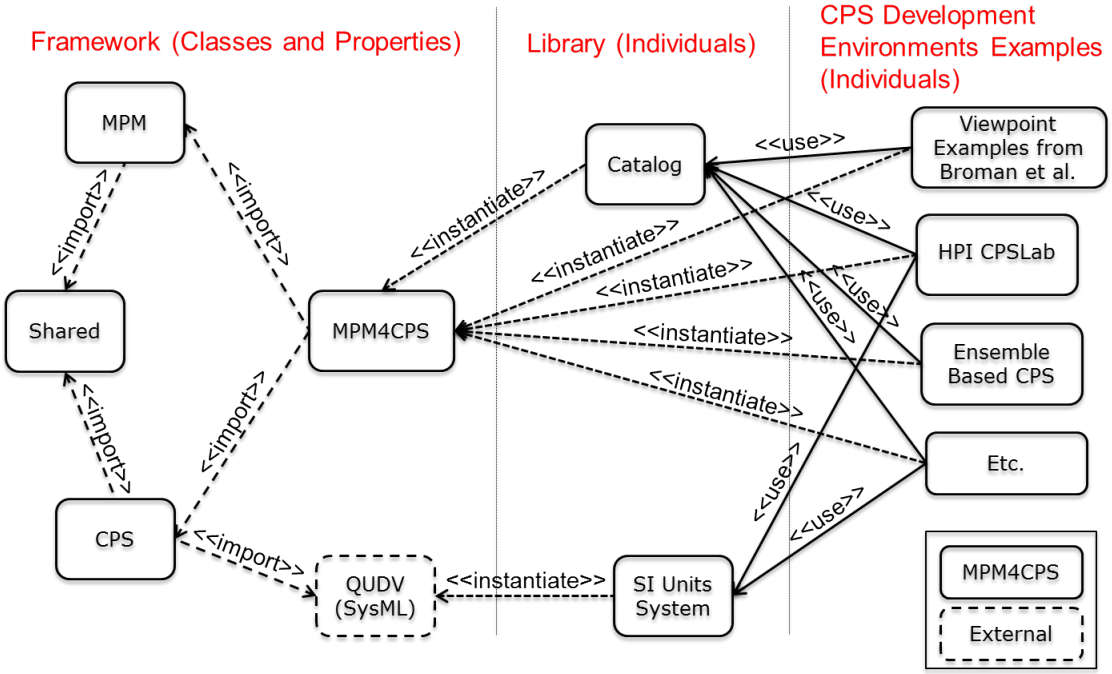

Figure 1.1: Overview of the structure of the MPM4CPS ontology

In figure 1.1, the structure of the framework and its elements in form of the different ontologies and its instances is presented.

The first column depicts the framework and its ontologies as presented in this report. This includes the ontology of Cyber-Physical Systems presented in chapter 3, the ontology of Multi-Paradigm Modeling presented in chapter 4 and the combined ontology of Multi-Paradigm Modeling for Cyber-Physical Systems presented in chapter 5.

The Glossary of Terms for Cyber-Physical Systems presented in the report on the State-of-the-art on Current Formalisms used in Cyber-Physical Systems Development covered by deliverable D1.1 (78) is extracted automatically from these ontologies and the contained concepts defining the framework.

In the second column, the catalog of modeling languages and tools that is an instance of the MPM4CPS ontology presented in the report on the State-of-the-art on Current Formalisms used in Cyber-Physical Systems Development covered by deliverable D1.1 (78) is depicted. The catalog of that deliverable will be automatically derived from this instance such that ontology and instances can be kept consistent with only minimal coordination efforts.https://www.sharelatex.com/project/5ae2fed74797f945fcb70b13

In the third column, some examples for CPS employing MPM in form of mega models are depicted that are presented in detail in the Catalog of Megamodel Examples in chapter 6. As shown in the figure, these examples employ the languages and tools listed in the report on the State-of-the-art on Current Formalisms used in Cyber-Physical Systems Development and covered by deliverable D1.1 (78), and also instantiate the MPM4CPS ontology.

1.1 Ontology Development Approach

To define the ontologies of WG1, we have carried out a domain analysis process (97). The domain analysis process can be defined as the process of identifying, capturing and organizing domain knowledge about the problem domain with the purpose of making it reusable when creating new systems. A domain is usually defined as an area of knowledge or activity characterized by a set of concepts and terminology understood by practitioners in that area. In our context, the domains of consideration are the domains of CPS and MPM, and we aim to derive and model the concepts of these domains. Figure 1.2 represents the common structure of domain analysis methods as it has been derived from survey studies on domain analysis methods.

Conventional domain analysis methods consist generally of the activities Domain Scoping and Domain Modeling: Domain Scoping identifies the domains of interest, the stakeholders, and their goals, and defines the scope of the domain. Domain Modeling is the activity for representing the domain, or the domain model. In our study the outputs of the domain modeling process will be the set of ontologies for CPS and MPM as identified by figure 1.1.

The domain model can be represented in different forms such as ontological languages, object-oriented language, algebraic specifications, rules, conceptual models etc. Typically, a domain model is formed through a commonality and variability analysis to concepts in the domain. A domain model is used as a basis for engineering components intended for use in multiple applications within the domain.

One of the popular approaches for domain modeling is feature modeling. A feature is a system property that is relevant to some stakeholder and is used to capture commonalities or discriminate between. A feature model is a model that defines features and their dependencies. Feature models are usually represented in feature diagrams (or tables). A feature diagram is a tree with the root representing a concept (e.g., a software system), and its descendent nodes are features. Relationships between a parent feature and its child features (or sub-features) are categorized as:

- Mandatory - child feature is required.
- Optional - child feature is optional.
- Or -at least one of the sub-features must be selected.
- Alternative (xor) - one of the sub-features must be selected

A feature configuration is a set of features which describes a member of an SPL. A feature constraint further restricts the possible selections of features to define configurations. The most common feature constraints are:

- A requires B - The selection of A in a product implies the selection of B.

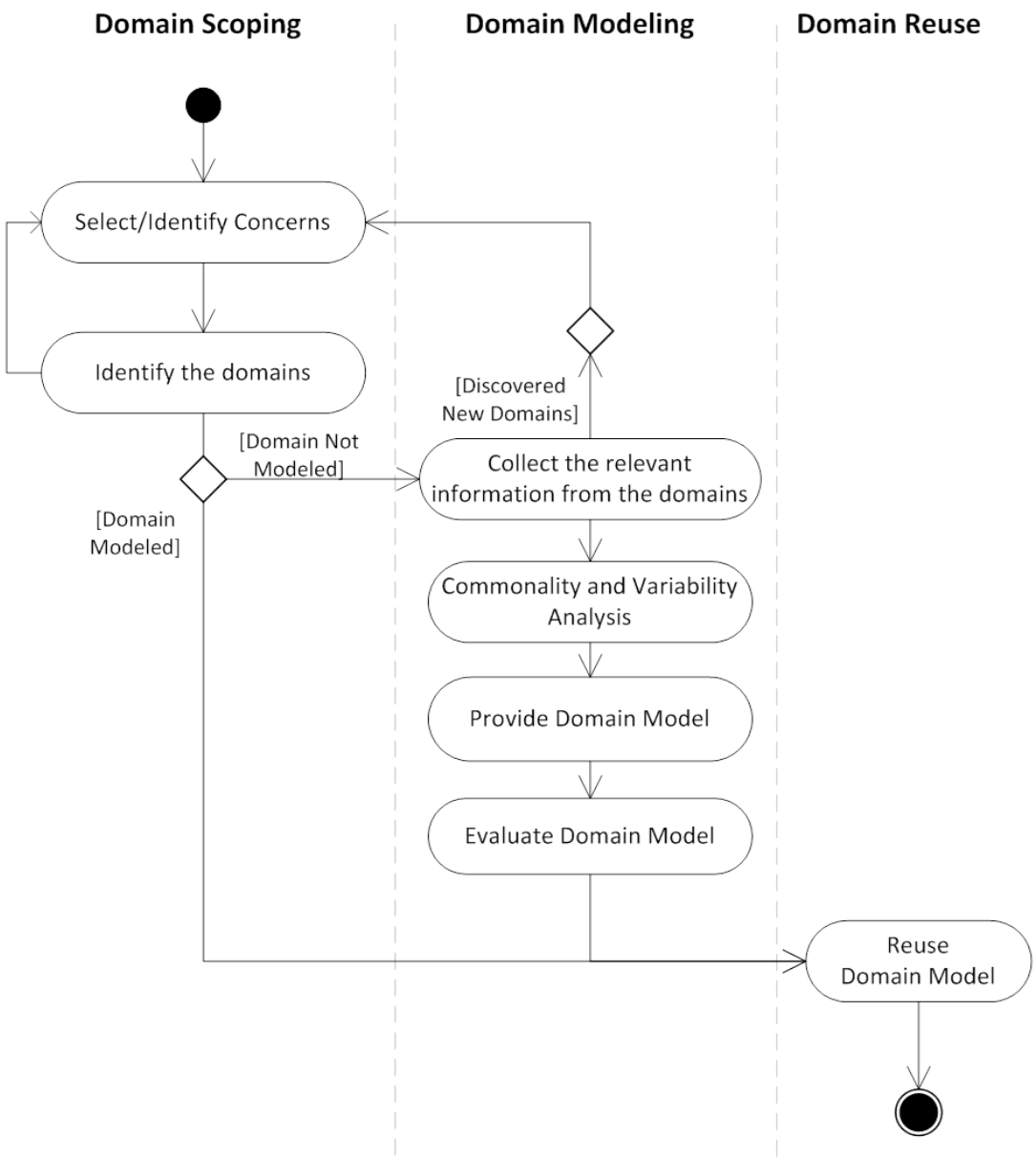

Figure 1.2: Common structure of domain analysis methods (adopted from: (131))

- A excludes B - A and B cannot be part of the same product.

Feature modeling is a domain modeling technique, which is widely used to model the commonality and variability of a particular domain or product family. Another domain modeling technique that is used in software engineering is ontology modeling. A commonly accepted definition of an ontology is "an explicit specification of conceptualization" (84). An ontology represents the semantics of concepts and their relationships using some description language. Basic feature modeling is also a concept description technique that focuses on modeling both the commonality and variability. It has been indicated that feature models can be seen as views on ontologies (54).

To develop the WG1 ontologies presented in details in the following chapters, the aforementioned techniques have been used. For the CPS ontology, feature modeling has been used while for the MPM ontology, modeling with the W3C OWL language and its tool Protege has been used.

2 Ontology of Shared Concepts

As outlined in the introduction of Chapter 1 in Figure 1.1, the structure of the framework and its elements are organized in the form of different ontologies providing classes for the covered domains and individuals (instances) for these classes. The ontology presented in this chapter serves in this context as foundation to define an ontology of Cyber-Physical Systems later in Chapter 3 and an ontology of Multi-Paradigm Modeling in Chapter 4. Then, these ontologies together with the one of this chapter are combined to define an ontology of Multi-Paradigm Modeling for Cyber-Physical Systems presented in Chapter 5.

2.1 Ontology Overview

The shared ontology defines concepts that do not pertain to the CPS and neither to the MPM domain, but that are still required by one of these domains or both. It is a place to define concepts reusable by all the ontologies developed in this effort. The class provided in this ontology may be refined to provide extend the definitions to more specific domains.

Figure 2.1 shows an overview of the shared ontology. The details of each concept are provided in the following subsections.

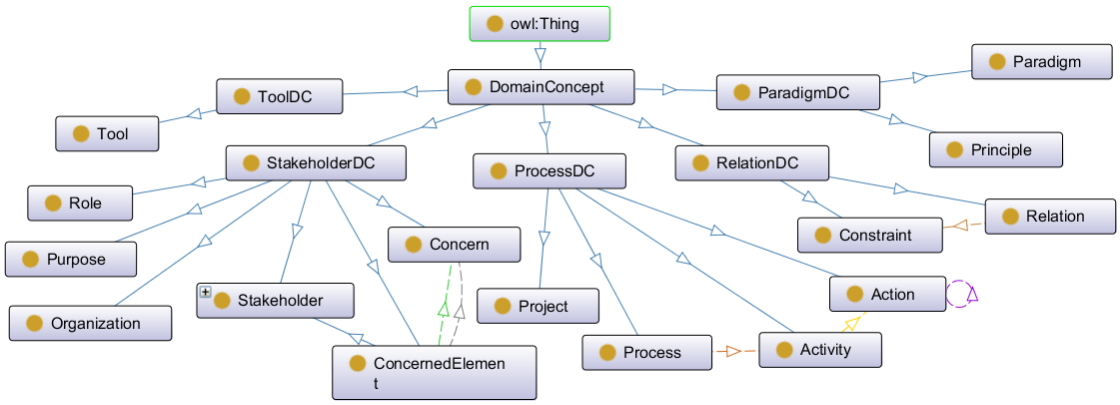

Figure 2.1: Overview of the shared ontology

2.2 DomainConcept

This class groups all concepts of the MPM4CPS ontology. It is further divided into into sub-classed whose names end with DC in order to organize the ontology into sub-domains thus facilitating the navigation across the many concepts of the MPM4CPS ontology. Note that the sub-domain classes are not necessarily disjoints so that a class may belong to several domain concepts.

2.2.1 ParadigmDC (Paradigm Domain Concepts)

This class groups concepts that are related to the paradigm domain.

2.2.1.1 Paradigm

In the usual sense, a paradigm is a set of concepts, patterns, theories, research methods, postulates, and standards that together constitutes a contribution to a domain.

Subclass of

- **ParadigmDC** (see section 2.2.1)

References

- https://en.wikipedia.org/wiki/Paradigm

2.2.1.2 Principle

A principle is a law or rule that has to be, or usually is to be followed, or can be desirably followed, or is an inevitable consequence of something, such as the laws observed in nature or the way that a system is constructed. The principles of such a system are understood by its users as the essential characteristics of the system, or reflecting system's designed purpose, and the effective operation or use of which would be impossible if any one of the principles was to be ignored.

Subclass of

- **ParadigmDC** (see section 2.2.1)

2.2.2 ProcessDC (Process Domain Concepts)

This class groups concepts that are related to the development process domain.

2.2.2.1 Action

This class represents an action performed during an activity.

Subclass of

- **ProcessDC** (see section 2.2.2)

2.2.2.2 Activity

This class represents an activity performed during a process.

Subclass of

- **ProcessDC** (see section 2.2.2)

2.2.2.3 Process

A process is a sequence of activities executed in order to achieve a result or a goal.

Subclass of

- **ProcessDC** (see section 2.2.2)

References

- https://en.wikipedia.org/wiki/Process

2.2.2.4 Project

Subclass of

- **ProcessDC** (see section 2.2.2)

2.2.3 RelationDC (Relation Domain Concepts)

This class groups concepts that are related to the relation domain.

2.2.3.1 Constraint

A limitation or a restriction over a relation.

Subclass of

- **RelationDC** (see section 2.2.3)

2.2.3.2 Relation

Relation or relations may refer to anything that involves communicating with another person, group, society or country.

Subclass of

- **RelationDC** (see section 2.2.3)

2.2.4 StakeholderDC (Stakeholder Domain Concepts)

This class groups concepts that are related to the stakeholder domain.

2.2.4.1 Concern

In computer science, a concern is a particular set of information that has an effect on the code of a computer program. A concern can be as general as the details of database interaction or as specific as performing a primitive calculation, depending on the level of conversation between developers and the program being discussed. IBM uses the term concern space to describe the sectioning of conceptual information.

Subclass of

- **StakeholderDC** (see section 2.2.4)

2.2.4.2 ConcernedElement

Subclass of

- **StakeholderDC** (see section 2.2.4)
- **ConcernedElement** (see section 2.2.4.2)

2.2.4.3 Organization

Subclass of

- **StakeholderDC** (see section 2.2.4)

2.2.4.4 Purpose

The object for which something exists

Subclass of

- **StakeholderDC** (see section 2.2.4)

2.2.4.5 Role

A role (also role or social role) is a set of connected behaviours, rights, obligations, beliefs, and norms as conceptualized by people in a social situation. It is an expected or free or continuously changing behaviour and may have a given individual social status or social position. It is vital to both functionalist and interactionist understandings of society.

Subclass of

- **StakeholderDC** (see section 2.2.4)

2.2.4.6 Stakeholder

A stakeholder or stakeholders, as defined in its first usage in a 1963 internal memorandum at the Stanford Research Institute, are "those groups without whose support the organization would cease to exist." The theory was later developed and championed by R. Edward Freeman in the 1980s. Since then it has gained wide acceptance in business practice and in theorizing relating to strategic management, corporate governance, business purpose and corporate so-

cial responsibility (CSR). A corporate stakeholder can affect or be affected by the actions of a business as a whole.

Subclass of

- **ConcernedElement** (see section 2.2.4.2)
- **StakeholderDC** (see section 2.2.4)

2.2.4.7 ToolProvider

Who provides the product and gives the licenses (e.g. company, university, group, ..)

Subclass of

- **Stakeholder** (see section 2.2.4.6)

2.2.5 ToolDC (Tool Domain Concepts)

This class groups concepts that are related to the tool domain.

2.2.5.1 Tool

Set of different tools that are used during system development.

Subclass of

- **ToolDC** (see section 2.2.5)

2.3 Properties

3 Ontology of Cyber-Physical Systems

3.1 State-of-the-art

Cyber-Physical Systems (CPS) are systems that tightly integrate computation with networking and physical processes. Such systems form large networks that communicate with each other and rely on actuators and sensors to monitor and control complex with physical processes, creating complex feedback loops between the physical and the cyber worlds. CPS bring innovation in terms of economic and societal impacts for various kinds of industries, creating entirely new markets and platforms for growth. CPS have growing applications in various domains, including healthcare, transportation, precision agriculture, energy conservation, environmental control, avionics, critical infrastructure control (electric and nuclear power plants, water resources, and communications systems), high confidence medical devices and systems, traffic control and safety, advanced automotive systems, process control, distributed robotics (telepresence, telemedicine), manufacturing, and smart city engineering. The positive economic impact of any one of these applications areas is enormous.

Technically, CPS systems are inherently heterogeneous, typically comprising mechanical, hydraulic, material, electrical, electronic, and computational components. The engineering process of CPS requires distinct disciplines to be employed, resulting in a collection of models that are expressed using correspondingly distinct modelling formalisms.

An important realization is that distinct models need to be weaved together consistently to form a complete representation of a system that enables, among other global aspects, performance analysis, exhaustive simulation and verification, hardware in the loop simulation, determining best overall parameters of the system, prototyping, or implementation. A new framework is required that is able to represent these connections between models and, moreover, enable reasoning about them. No single formalism is able to model all aspects of a system; modelling of a CPS system is inherently multi-paradigm, which calls for a trans-disciplinary approach to be able to conjoin abstractions and models from different worlds. Physically, CPS systems are inherently heterogeneous, typically comprising mechanical, hydraulic, material, electrical, electronic, among others. Those areas correspond to engineering disciplines with their own models and abstractions designed to best capture the dynamics of physical processes (e.g., differential equations, stochastic processes, etc.). Computationally, CPS systems leverage the half-century old knowledge in computer science and software engineering to essentially capture how data is transformed into other useful data, abstracting away from core physical properties occurring in the real world, and particularly the passage of time in physical processes.

The key challenge, as identified a decade ago, is then to provide mathematical and technical foundations to conjoin physical abstractions that describe the dynamics of nature in various engineering domains, as described earlier, with models focusing solely on data transformation. This is necessary to adequately capture and bridge both aspects of a complex, realistic cyber-physical system, and become able to reason and explore system designs collaboratively, allocating responsibilities to software and physical elements, and analyzing trade-offs between them.

Currently, there is a few common design and modelling approaches that allow engineers to handle both aspects of CPS, allowing to bridge the involved disciplines into a shared, common one. Among the existing ones, *co-simulation* showed that it is possible for computation and physics engineers to cooperate efficiently without enforcing new tools or design methods.

3.2 Ontology Overview

After a domain analysis to CPS we have derived the feature model as shown in figure 3.1. The different components of a CPS system can be designed together or separately: in the latter case, the various different components need to be integrated. A CPS has different component types which can be computational or physical. Furthermore, a CPS has a network which can have different configurations and protocols.

Interoperability relates to how well the different components can operate together. Here we can have syntactic or semantic interoperability. Since both are required in CPS, these two features are considered mandatory, rather than alternative. Different components in a CPS can be of the same type or of different types, thereby distinguishing between homogeneous or heterogeneous CPS. The final feature in the feature model presents the various application domains in which CPS can be applied including manufacturing, healthcare, transportation etc.

3.2.1 Ontology Diagram

In this section we describe the metamodel for CPS which represents the concepts and their relations. The metamodel is shown in figure 3.2. A CPS system consists of CPS Components that interact by using one or more Communication Protocols running on a Communication Network. CPS Components interact with other components. A CPS Component is a Computational Component or Physical Component. A computational component is a Software Component or Hardware Component that can include zero or more Sensors and Actuators. Sensors monitor the Physical Component while Actuators can drive them. Essentially, sensors take a mechanical, optical, magnetic or thermal signal and convert it into voltage and current. This provided data can then be processed and used to define the required action. Both computational components and physical components could have a virtual surrogate. Virtual entities can have different representations such as 3D models, avatars, objects or even a social network account. Some Virtual Entities can also interact with other Virtual Entities to fulfill their goal.

3.2.2 Architecture

The architecture of a CPS represents the gross level structure of the system consisting of cyber-physical components. Current architecture design approaches for CPS seem to be primarily domain-specific and no standard reference architecture has been yet agreed upon. In this line, the development of an ontology for CPS also contributes to the efforts for designing a reference architecture.

A CPS reference architecture defines the generic structure of CPS architectures for particular application domains, laying the foundation for functionality, dependability, and other quality properties. An architecture organizes the functionality and the properties of a system to enable partitioning, verification, and management.

Figure 3.3 presents a layered view of a CPS architecture inspired on the IoT stack that arranges a CPS into successive layers of cohesive modules that share similar concerns. The four layers at the center include device layer, network layer, CPS layer, application layer, and business layer. The CPS component layer includes the capabilities for the CPS components to undertake sensing and actuation. The network layer provides functionality for networking connectivity and transport capabilities enabling the coordination of components. The Services layer consists of functionality for generic support services (such as data processing or data storage), and specific support capabilities for the particular applications that may already apply a degree of intelligence. The application layer orchestrates the services to provide emergent properties. Then, there are two main cross-cutting concerns. A Security layer captures the security functionality, while the management layer supports capabilities such as device management, traffic and congestion management.

The reference architecture can be used to derive concrete application architectures. A concrete architecture defines the boundaries and constraints for the implementation and is used to analyze risks, balance trade-offs, plan the implementation project and allocate tasks.

3.3 Ontology Overview

Figure 3.4 shows an overview of the CPS ontology. The details of each concept are provided in the following subsections.

3.4 Domain Concepts

This ontology of cyber-physical systems contains concepts divided into sub-domains as presented in the following subsections.

3.4.1 ApplicationDomainsDC

Various studies have addressed the domains and domain specific applications of CPS. Gunes et al. summarize a number of research efforts that address some of those domains, namely Smart Manufacturing, Emergency Response, Air Transportation, Critical Infrastructure, Health Care and Medicine, Intelligent Transportation, and Robotic for Service.

3.4.2 ArchitectureDC

3.4.3 QualityRequirementsDC

Cyber-Physical Systems revolutionize our interaction with the physical world. Of course, this revolution does not come free. Since even legacy embedded systems require higher standards than general-purpose computing, we need to pay special attention to this next generation physically-aware engineered system requirements if we really want to put our full trust in them. Therefore, we want to clarify the definitions of some common CPS system-level requirements /challenges.

3.4.3.1 Accuracy

Accuracy refers to the degree of closeness of a system's measured/observed outcome to its actual/calculated one. A highly accurate system should converge to the actual outcome as close as possible. High accuracy especially comes into play for CPS applications where even small imprecisions are likely to cause system failures. For example, a motion-based object tracking system under the presence of imperfect sensor conditions may take untimely control action based on incorrect object position estimation, which in return leads to the system failure.

Subclass of

- **Predictability** (see section 3.4.3.13)

3.4.3.2 Adaptability

Adaptability refers to the capability of a system to change its state to survive by adjusting its own configuration in response to different circumstances in the environment. A highly adaptable system should be quickly adaptable to evolving needs/circumstances. Adaptability is one of the key features in the next generation air transportation systems (e.g. NextGen). NextGen's capabilities enhance airspace performance with its computerized air transportation network which enables air vehicles immediately to accommodate themselves to evolving operational environment such as weather conditions, air vehicle routing and other pertinent flight trajectory patterns over satellites, air traffic congestion, and issues related to security.

Subclass of

- **Sustainability** (see section 3.4.3.21)

3.4.3.3 Availability

Availability refers to the property of a system to be ready for access even when faults occur. A highly available system should isolate malfunctioning portion from itself and continue to operate without it. Malicious cyber-attacks (e.g. denial of service attacks) hinder availability of the system services significantly. For example, in Cyber-Physical Medical Systems, medical data shed light on necessary actions to be taken in a timely manner to save a patient's life. Malicious attacks or system/component failure may cause services providing such data to become unavailable, hence, posing risk on the patient's life.

Subclass of

- **Dependability** (see section 3.4.3.7)
- **Security** (see section 3.4.3.20)

3.4.3.4 Composibility

Composibility refers to the property of several components to be merged within a system and their inter-relationships. A highly composable system should allow recombination of the system components repeatedly to satisfy specific system requirements. Composibility should be examined in different levels (e.g. device composibility, code composibility, service composibility, system composibility). Certainly, system composibility is more challenging, hence the need for well-defined composition methodologies that follow composition properties from the bottom up. Additionally, requirements and evaluations must be composable accordingly. In the future, it will probably be of paramount importance to incrementally add emerging systems to the system of systems (e.g. CPS) with some predictable confidence without degrading the operation of the resulting system.

Subclass of

- **Interoperability** (see section 3.4.3.11)

3.4.3.5 Compositonality

Compositionality refers to the property of how well a system can be understood entirely by examining every part of it. A highly compositional system should provide great insight about the whole from derived behaviors of its constituent parts/components. Achieving high compositionality in CPS design is very challenging especially due to the chaotic behavior of constituent physical subsystems. Designing highly compositional CPS involves strong reasoning about the behavior of all constituent cyber and physical subsystems/components and devising cyber-physical methodologies for assembling CPSs from individual cyber and physical components, while requiring precise property taxonomies, formal metrics and standard test benches for their evaluation, and well-defined mathematical models of the overall system and its constituents.

Subclass of

- **Predictability** (see section 3.4.3.13)

3.4.3.6 Confidentiality

Confidentiality refers to the property of allowing only the authorized parties to access sensitive information generated within the system. A highly confidential system should employ the most secure methods of protection from unauthorized access, disclosure, or tampering. Data confidentiality is an important issue that needs to be satisfied in most CPS applications. For example, in an emergency management sensor network, attacks targeting confidentiality of data transmitted may degrade effectiveness of an emergency management system. Confidentiality of data transmitted through attacked sensor nodes can be compromised and that can

cause data flow in the network to be directed over compromised sensors; critical data to be eavesdropped; or fake node identities to be generated in the network. Further, false/malicious data can be injected into the network over those fake nodes. Therefore, confidentiality of data circulation needs to be retained in a reasonable degree.

Subclass of

- **Security** (see section 3.4.3.20)

3.4.3.7 Dependability

Dependability refers to the property of a system to perform required functionalities during its operation without significant degradation in its performance and outcome. Dependability reflects the degree of trust put in the whole system. A highly dependable system should operate properly without intrusion, deliver requested services as specified and not fail during its operation. The words dependability and trustworthiness are often used interchangeably. Assuring dependability before actual system operation is a very difficult task to achieve. For example, timing uncertainties regarding sensor readings and prompt actuation may degrade dependability and lead to unanticipated consequences. Cyber and physical components of the system are inherently interdependent and those underlying components might be dynamically interconnected during system operation, which, in return, renders dependability analysis very difficult. A common language to express dependability related information across constituent systems/underlying components should be introduced in the design stage.

Subclass of

- **QualityRequirementsDC** (see section 3.4.3)

3.4.3.8 Efficiency

Efficiency refers to the amount of resources (such as energy, cost, time etc.) the system requires to deliver specified functionalities. A highly efficient system should operate properly under optimum amount of system resources. Efficiency is especially important for energy management in CPS applications. For example, smart buildings can detect the absence of occupants and turn off HVAC (Heating, Ventilation, and Air Conditioning) units to save energy. Further, they can provide automated pre-heating or pre-cooling services based on the occupancy prediction techniques.

Subclass of

- **Sustainability** (see section 3.4.3.21)

3.4.3.9 Heterogeneity

Heterogeneity refers to the property of a system to incorporate a set of different types of interacting and interconnected components forming a complex whole. CPSs are inherently heterogeneous due to constituent physical dynamics, computational elements, control logic, and deployment of diverse communication technologies. Therefore, CPSs necessitate heterogeneous composition of all system components. For example, incorporating heterogeneous computing and communication capabilities, future medical devices are likely to be interconnected in increasingly complex open systems with a plug-and-play fashion, which makes a heterogeneous control network and closed loop control of interconnected devices crucial. Configuration of such devices may be highly dynamic depending on patient-specific medical considerations. Enabled by the science and emerging technologies, medical systems of the future are expected to provide situation-aware component autonomy, cooperative coordination, real-time guarantee, and heterogeneous personalized configurations far more capable and complex than today's.

Subclass of

- **Interoperability** (see section 3.4.3.11)

3.4.3.10 Integrity

Integrity refers to the property of a system to protect itself or information within it from unauthorized manipulation or modification to preserve correctness of the information. A high integrity system should provide extensive authorization and consistency check mechanisms. High integrity is one of the important properties of a CPS. CPSs need to be developed with greater assurance by providing integrity check mechanisms on several occasions (such as data integrity of network packets, distinguishing malicious behaviors from the ambient noise, identifying false data injection and compromised sensor/actuator components etc.). Properties of the physical and cyber processes should be well-understood and thus can be utilized to define required integrity assurance.

Subclass of

- **Security** (see section 3.4.3.20)

3.4.3.11 Interoperability

Interoperability refers to the ability of the systems/components to work together, exchange information and use this information to provide specified services. A highly interoperable system should provide or accept services conducive to effective communication and interoperation among system components. Performing far-reaching battlefield operations and having more interconnected and potentially joint-service combat systems, Unmanned Air Vehicles (UAVs) call for seamless communication between each other and numerous ground vehicles in operation. The lack of interoperability standards often causes reduction in the effectiveness of complicated and critical missions. Likewise, according to changing needs, dynamic standards should be developed and tested for devices, systems, and processes used in the Smart Grid to ensure and certify the interoperability of those ones being considered for a specific Smart Grid deployment under realistic operating conditions.

Subclass of

- **QualityRequirementsDC** (see section 3.4.3)

3.4.3.12 Maintainability

Maintainability refers to the property of a system to be repaired in case a failure occurs. A highly maintainable system should be repaired in a simple and rapid manner at the minimum expenses of supporting resources, and free from causing additional faults during the maintenance process. With the close interaction among the system components (e.g. sensors, actuators, cyber components, and physical components) underlying CPS infrastructure, autonomous predictive /corrective diagnostic mechanisms can be proposed. Continuous monitoring and testing of the infrastructure can be performed through those mechanisms. The outcome of monitoring and testing facilities help finding which units need to be repaired. Some components, which happen to be the source of recurrent failures, can be redesigned or discarded and replaced with the ones with better quality

Subclass of

- **Dependability** (see section 3.4.3.7)

3.4.3.13 Predictability

Predictability refers to the degree of foreseeing of a system's state/behavior/functionality either qualitatively or quantitatively. A highly predicable system should guarantee the speci-

fied outcome of the system's behavior/functionality to a great extent every moment of time at which it is operating while meeting all system requirements. In Cyber-Physical Medical Systems (CPMS), smart medical devices together with sophisticated control technologies are supposed to be well adapted to the patient's conditions, predict the patient's movements, and change their characteristics based on context awareness within the surrounding environment. Many medical devices perform operations in real-time, satisfying different timing constraints and showing diverse sensitivity to timing uncertainties (e.g. delays, jitters etc.). However, not all components of CPMS are time-predictable. Therefore, in addition to new programming and networking abstractions, new policies of resource allocation and scheduling should be developed to ensure predictable end-to-end timing constraints.

Subclass of

- **QualityRequirementsDC** (see section 3.4.3)

3.4.3.14 Reconfigurability

Reconfigurability refers to the property of a system to change its configurations in case of failure or upon inner or outer requests. A highly reconfigurable system should be self-configurable, meaning able to fine-tune itself dynamically and coordinate the operation of its components at finer granularities. CPSs can be regarded as autonomously reconfigurable engineered systems. Remote monitoring and control mechanisms might be necessity in some CPS application scenarios such as international border monitoring, wildfire emergency management, gas pipeline monitoring etc. Operational needs (e.g. security threat level updates, regular code updates, efficient energy management etc.) may change for such scenarios, which calls for significant reconfiguration of sensor/actuator nodes being deployed or the entire network to provide the best possible service and use of resources.

Subclass of

- **Sustainability** (see section 3.4.3.21)

3.4.3.15 Reliability

Reliability refers to the degree of correctness which a system provides to perform its function. The certification of system capabilities about how to do things correctly does not mean that they are done correctly. So a highly reliable system makes sure that it does the things right. Considering the fact that CPSs are expected to operate reliably in open, evolving, and uncertain environments, uncertainty in the knowledge, attribute (e.g. timing), or outcome of a process in the CPS infrastructure makes it necessary to quantify uncertainties during the CPS design stage. That uncertainty analysis will yield to effective CPS reliability characterization. Besides, accuracy of physical and cyber components, potential errors in design/control flow, cross-domain network connections in an ad-hoc manner limit the CPS reliability.

Subclass of

- **QualityRequirementsDC** (see section 3.4.3)

3.4.3.16 Resilience

Resilience refers to the ability of a system to persevere in its operation and delivery of services in an acceptable quality in case the system is exposed to any inner or outer difficulties (e.g. sudden defect, malfunctioning components, rising workload etc.) that do not exceed its endurance limit. A highly resilient system should be self-healing and comprise early detection and fast recovery mechanisms against failures to continue to meet the demands for services. High resilience comes into play in delivering mission-critical services (e.g. automated brake control in vehicular CPS, air and oxygen flow control over an automated medical ventilator etc.). Mission-

critical CPS applications are often required to operate even in case of disruptions at any level of the system (e.g. hardware, software, network connections, or the underlying infrastructure). Therefore, designing highly resilient CPS requires thorough understanding of potential failures and disruptions, the resilience properties of the pertinent application, and system evolution due to the dynamically changing nature of the operational environment.

Subclass of

- **Sustainability** (see section 3.4.3.21)

3.4.3.17 Robustness

Robustness refers to the ability of a system to keep its stable configuration and withstand any failures. A highly robust system should continue to operate in the presence of any failures without fundamental changes to its original configuration and prevent those failures from hindering or stopping its operation. In addition to failures, the presence of disturbances possibly arising from sensor noises, actuator inaccuracies, faulty communication channels, potential hardware errors or software bugs may degrade overall robustness of CPS. Lack of modeling integrated system dynamics (e.g. actual ambient conditions in which CPSs operate), evolved operational environment, or unforeseen events are other particular non-negligible factors, which might be unavoidable in the run-time, hence the need for robust CPS design.

Subclass of

- **Reliability** (see section 3.4.3.15)

3.4.3.18 Safety

Safety refers to the property of a system to not cause any harm, hazard or risk inside or outside of it during its operation. A very safe system should comply with both general and application-specific safety regulations to a great extent and deploy safety assurance mechanisms in case something went wrong. For example, among the goals for Smart Manufacturing (SM), point-in-time tracking of sustainable production and real-time management of processes throughout the factory yield to improved safety. Safety of manufacturing plants can be highly optimized through automated process control using embedded control systems and data collection frameworks (including sensors) across the manufacturing enterprise. Smart networked sensors could detect operational failures/anomalies and help prevention of catastrophic incidents due to those failures/anomalies.

Subclass of

- **Dependability** (see section 3.4.3.7)

3.4.3.19 Scalability

Scalability refers to the ability of a system to keep functioning well even in case of change in its size/increased workload, and take full advantage of it. The increase in the system throughput should be proportional to the increase in the system resources. A highly scalable system should provide scatter and gather mechanisms for workload balancing and effective communication protocols to improve the performance. Depending on their scale, CPSs may comprise over thousands of embedded computers, sensors, and actuators that must work together effectively. Scalable embedded many-core architectures with a programmable interconnect network can be deployed to deliver increasing compute demand in CPS. Further, a high performance and highly scalable infrastructure is needed to allow the entities of CPS to join and leave the existing network dynamically. In the presence of frequent data dissemination among those entities, dynamic software updates (i.e. changing the computer program in run-time) can help update CPS applications dynamically and use CPS resources more productively.

Subclass of

- **Interoperability** (see section 3.4.3.11)

3.4.3.20 Security

Security refers to the property of a system to control access to the system resources and protect sensitive information from unauthorized disclosures. A highly secure system should provide protection mechanisms against unauthorized modification of information and unauthorized withholding of resources, and must be free from disclosure of sensitive information to a great extent. CPSs are vulnerable to failures and attacks on both the physical and cyber sides, due to their scalability, complexity, and dynamic nature. Malicious attacks (e.g. eavesdropping, man-in-the-middle, denial-of-service, injecting fake sensor measurements or actuation requests etc.) can be directed to the cyber infrastructure (e.g. data management layer, communication infrastructure, decision making mechanisms etc.) or the physical components with the intent of disrupting the system in operation or stealing sensitive information. Making use of a large-scale network (such as the Internet), adopting insecure communication protocols, heavy use of legacy systems or rapid adoption of commercial off-the-shelf (COTS) technologies are other factors which make CPSs easily exposed to the security threats.

Subclass of

- **QualityRequirementsDC** (see section 3.4.3)

3.4.3.21 Sustainability

Sustainability means being capable of enduring without compromising requirements of the system, while renewing the system's resources and using them efficiently. A highly sustainable system is a long lasting system which has self-healing and dynamic tuning capabilities under evolving circumstances. Sustainability from energy perspective is an important part of energy provision and management policies. For example, the Smart Grid facilitates energy distribution, management, and customization from the perspective of customers or service providers by incorporating green sources of energy extracted from the physical environment. However, intermittent energy supply and unknown/ill-defined load characterization hinders the efforts to maintain long-term operation of the Smart Grid. To maintain sustainability, the Smart Grid requires planning and operation under uncertainties, use of real-time performance measurements, dynamic optimization techniques for energy usage, environment-aware duty cycling of computing units, and devising self-contained energy distribution facilities (such as autonomous micro grids).

Subclass of

- **QualityRequirementsDC** (see section 3.4.3)

3.4.4 SystemDC

3.4.4.1 Component

Subclass of

- **SystemDC** (see section 3.4.4)

3.4.4.2 System

Subclass of

- **Component** (see section 3.4.4.1)
- **SystemDC** (see section 3.4.4)

3.4.4.3 SystemPart

Subclass of

- **SystemDC** (see section 3.4.4)

3.5 Properties

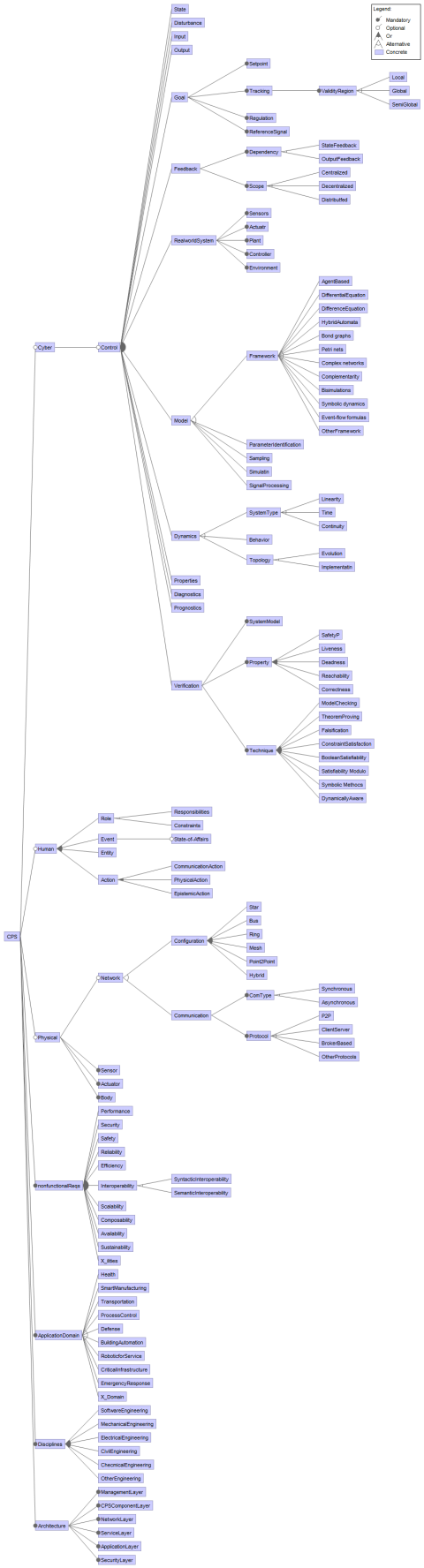

Figure 3.1: Feature Model of a CPS representing common and variant properties

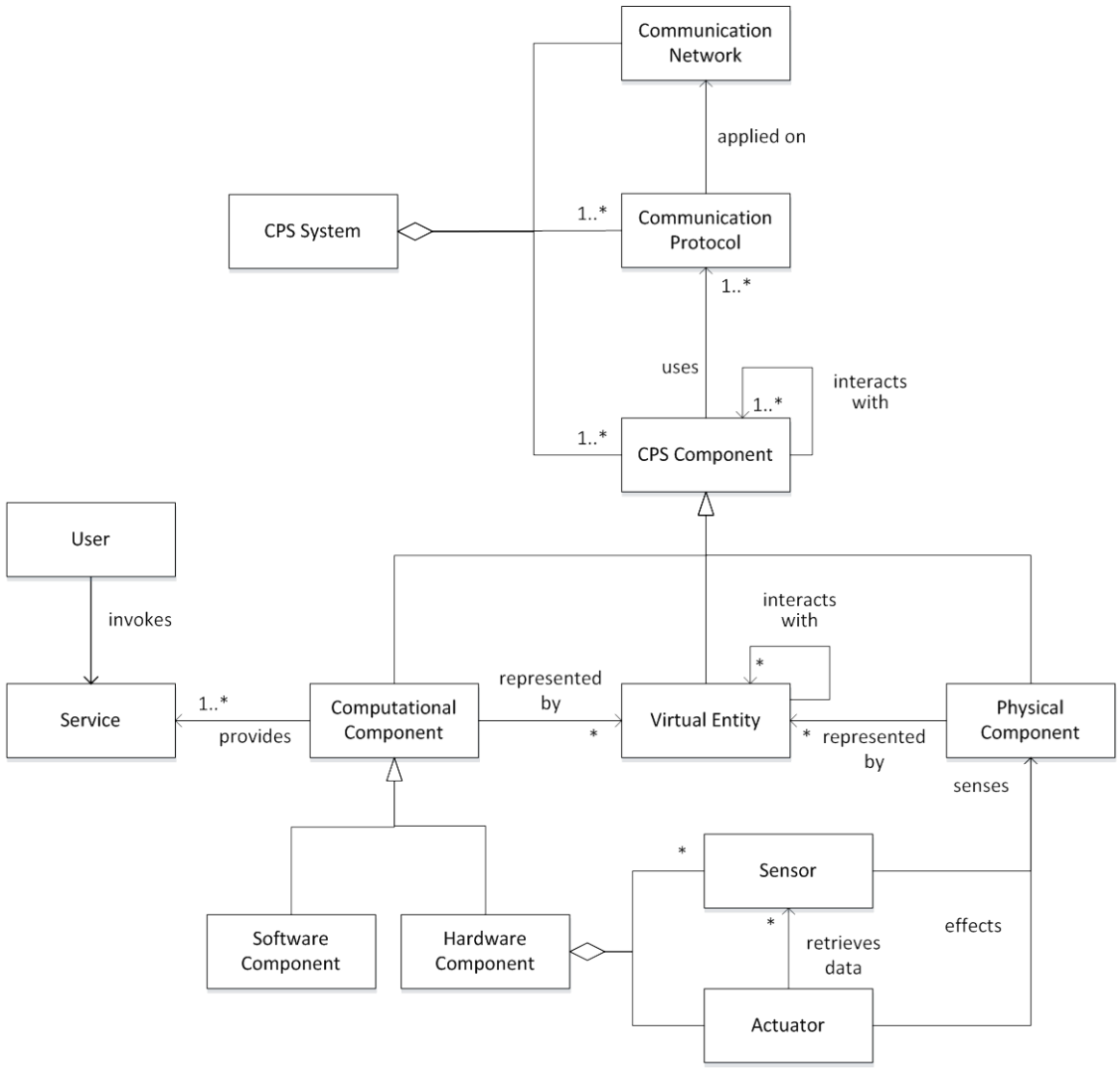

Figure 3.2: Basic concepts of CPS

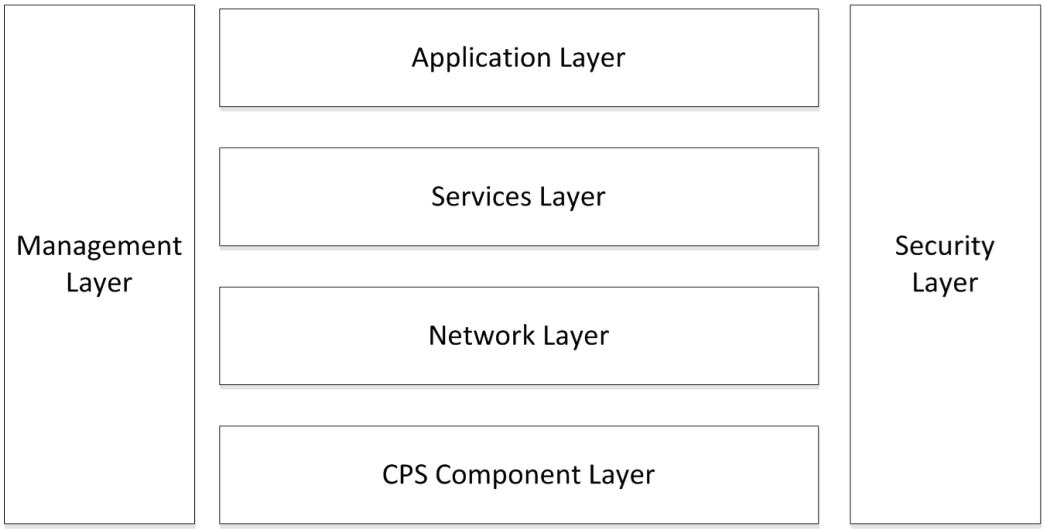

Figure 3.3: Layered view for CPS architectures

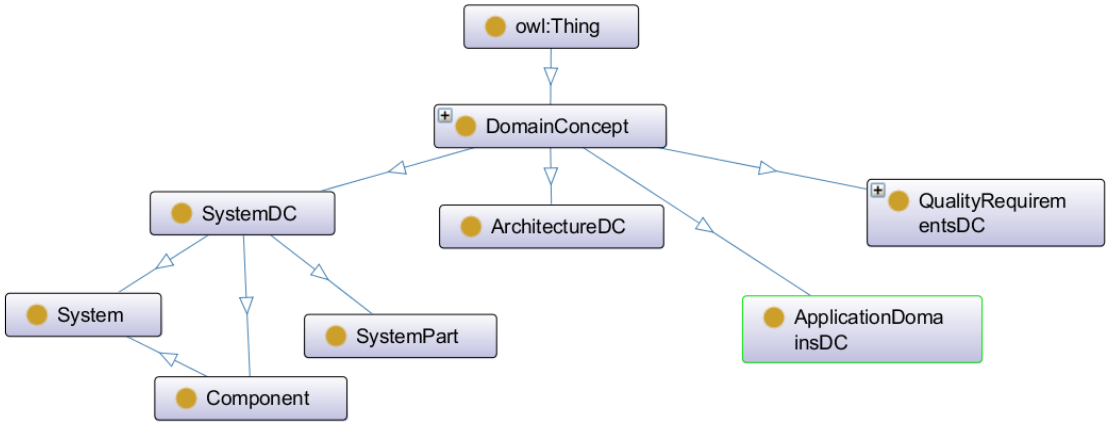

Figure 3.4: Overview of the CPS ontology

4 Ontology of Multi-Paradigm Modeling

This chapter presents the ontology of Multi-Paradigm Modeling (MPM), which consists of classes and properties to support a classification of MPM approaches. First, a state-of-the-art on MPM focusing on global model management, model integration and multi-formalism modelling is presented. Then an overview of the developed MPM ontology is presented followed by a detailed presentation of the proposed classes and properties.

4.1 State-of-the-art

Developing nowadays complex systems with Multi-Paradigm Modeling requires *Global Model Management* (*GMM*) (24; 67) to ensure that the models of different subsystems, of different views and of different domains are properly combined, even though the models might reside at different levels of abstraction. GMM must also ensure that the development activities that operate on the models are properly coordinated such that the models lead to a proper system as a whole, where the different elements and aspects covered by the different models are correctly integrated and are consistent with each other.

A classification of model integration problems and fundamental integration techniques has been introduced in (79). It highlights the techniques of decomposition and enrichment, which characterize two orthogonal dimensions of development where the system is decomposed into subsystems and domains (*horizontal* dimension) and into a set of models with increasing level of details (*vertical* dimension). Another approach to support interoperability among languages and tools is presented in (112). The approach is general, however it is exemplified on interoperability among architectural languages and tools (111). An approach to extend architectural languages is presented in (59). The technique is based on some operators to integrate models and it is generalized in (60). Model integration requires coordinating all activities operating on the models across these dimensions to ensure their consistency. A model-driven approach to automate the propagation of changes among Architecture Description Languages (111) is presented in (65). However, inconsistency management goes beyond simply identifying and resolving inconsistencies, since as pointed out in (69), inconsistencies may need to be tolerated at some stage of the development. Therefore, living with inconsistencies must be manageable and consequently, an approach is required to detect, resolve, but also tolerate inconsistencies for a significant amount of time during development. The work in (138) provides empirical evidence of how culture, processes, and organization impact traceability management and collaboration, and principles support practitioners with collaborative traceability management. The work shows that collaboration and traceability management have the potential to be mutually beneficial - when investing in one, also the other one is positively affected.

The development activities for nowadays complex systems and in particular CPSs encompass multiple domains and teams, with each team using a dedicated set of modelling languages, thus requiring their proper integration and management. Using a single "model-it-all" language to cover all domains would certainly lead to large, monolithic languages that become less efficient, not easily customisable for development environments and tools needed by development teams, therefore adding difficulties to the already demanding effort of developing CPS. These considerations lead to *Multi-Paradigm Modeling* (*MPM*) (135), which advocates the combination of reusable *modular* modeling languages instead of large monolithic languages. Hence, GMM must support integrating models and *modeling languages* with appropriate abstractions and modularity, but also coordinating all activities operating on the models and specified as *model operations* / *transformations*. The execution of these model operations

has to be *scalable* for being able to handle large models. This requires *incrementality*, where only the operations impacted by a model change are re-executed, thus avoiding the effort to recompute entire models, as in the case of incremental code compilers.

GMM is also known as *modeling-in-the-large*, which consists of establishing global relationships (e.g. model operations that generated one model from other models) between macroscopic entities (models and meta models), while ignoring the internal details of these entities (24). *Mega modeling* (25; 67) has been introduced for the purpose of describing these macroscopic entities and their relations. Nowadays only preliminary approaches exist that provide *ad-hoc* solutions for fragments of the sketched problem and a solid understanding of the underlying needs including new foundations to address this problem as proposed to be developed by WG1 of MPM4CPS. In particular, the current approaches do at most offer some modularity and/or incrementality for a single aspect as modeling languages or model operations. However, support for handling complex modeling landscapes as a whole in a modular and incremental fashion as required for the large-scale problems that exist in practice is not offered so far.

In the following, we will first look at existing solutions that address the construction and execution of models and modeling languages in Section 4.1.1, model operation sin Section 4.1.2, and mega models in Section 4.1.3.

4.1.1 Models and Modeling Languages: Construction and Execution

The construction of models and modeling languages is addressed in the current approaches in three main ways via (1) linking of models and model elements, (2) model interfaces and (3) meta model composition.

4.1.1.1 Model / Model Elements Links

Many approaches rely on *traceability links* between models and/or model elements to capture megamodelling relations/operations. We adopt here the definition proposed by the Center of Excellence for Software Traceability (CoEST): a *trace link* is *"[...] a specified association between a pair of artifacts, one comprising the source artifact and one comprising the target artifact..."*. The CoEST specialises those links into two dimensions: *vertical* trace links link *[...] artifacts at different levels of abstraction so as to accommodate lifecycle-wide or end-to-end traceability, such as from requirements to code [...]"*; while *horizontal* trace links associate *"[...] artifacts at the same level of abstraction, such as: (i) traces between all the requirements created by 'Mary', (ii) traces between requirements that are concerned with the performance of the system, or (iii) traces between versions of a particular requirement at different moments in time"*.

A plethora of approaches have been proposed that make use of trace links for model integration (cf. e.g., (AMW; Epsilon; 68; 109; 87; 129; 31) (MoTE)). The Atlas Model Weaving (AMW) language (AMW) provided one of the first approaches for capturing hierarchical traceability links between models and model elements. The purpose was to support activities such as automated navigation between elements of the linked models. In this approach, a generic core traceability language is made available and optionally extended to provide semantics specific to the metamodels of the models to be linked. Similarly, the Epsilon framework (Epsilon) provides a tool named ModeLink to establish correspondences between models. MegaL Explorer (68) supports relating heterogeneous software development artifacts using predefined relation types, linking elements that do not necessary have to be models or model elements. SmarfEMF (109) is another tool for linking models based on annotations of Ecore metamodels to specify simple relations between model elements through correspondence rules for attribute values. Complex relations are specified with ontologies relating the concepts of the linked languages. The whole set of combined models is converted into Prolog facts to support various activities such

as navigation, consistency and user guidance when editing models. The CONSYSTENT tool and approach (87) make use of a similar idea. However, graph structures and pattern matching are used to represent the combined models in a common formalism and to identify and manage inconsistencies instead of Prolog facts as in the case of SmartEMF.

There are also a number of approaches, such as (129) and (31), that build on establishing links between models through the use of integration languages developed for a specific set of integrated modeling languages, where the integration language embeds constructs specific to the linked languages. This is also the case for model weaving languages extending the core AMW language. However, AMW has the advantage of capturing the linking domain with a core common language. Other means for linking and integrating models are Triple Graph Grammars (TGG) such as the Model Transformation Engine (MoTE) tool (MoTE), which similarly requires the specification of some sort of integration language (correspondence meta model) specific to the integrated languages. However, an important asset of this approach is that it automatically establishes and manages the traceability links and maintains the consistency of the linked models (model synchronization) in a scalable, incremental manner. Finally, in (128; 126)(23), an approach is presented to automatically create and maintain traceability links between models in a scalable manner. While the approach focuses on traceability management rather than model integration, compared to integration languages, it relies on link types defined at the model level (and not at the meta model / language level), thus avoiding the need to update the integration language every time a new language must be integrated.

The comparison of these approaches shows that apart from the approach (128; 126)(23), all approaches suffer from being dependent on the set of integrated languages, thus requiring to better support modularity. Furthermore, only (MoTE)(128; 126)(23) supports automated management of traceability links.

4.1.1.1.1 Interfaces

In addition to links, a few more sophisticated approaches (e.g., (106; 88; 96)) introduce the concept of *model interface* for specifying how models can be linked. In (106), the Analysis Constraints Optimization Language (ACOL) is proposed, which has been designed to be pluggable to an Architecture Description Language (ADL). A concept of *interface* specific to ACOL is included so that constraints can refer to these interfaces to relate to the model elements expected from the ADL.

SmartEMF (88; SmartEMF) proposes a more generic concept of model interface to track dependencies between models and metamodels and provide automated compatibility checks. Composite EMF Models (96; Composite EMF Models) introduces *export* and *import* interfaces to specify which model elements of a main model (*body*) should be exposed to other models (i.e. are part of the public API), and which elements of a body model are to be required from an export interface.

However, these approaches are only preliminary and need to be enriched to cover a larger number of model integration use cases such as for example, specifying modification policies of the linked model elements required to ensure the models can be kept consistent. They also lack integration into GMM.

4.1.1.1.2 Metamodel Composition

Some approaches (e.g., (Kompren; Kompose; 66; EMF Views) (28)) consider the construction of metamodels for expressing views in terms of other metamodels or language fragments. In (66), an approach implemented in the Gaspard2 tool (Gaspard2) is presented where meta models are artificially extended for the purpose of combining independent model transformations resulting in an extended transformation for the extended meta models. The work in (60) presents

operators to compose metamodels while preserving specific properties. In (27), the language and tool (Kompren) (Kompren) is proposed to specify and generate slices of metamodels via the selection of classes and properties of an input metamodel. A reduced metamodel is then produced from the input metamodel. However the produced metamodel must be completely regenerated when the input metamodel is changed. Such is the case for the Kompose approach (Kompose), which on the contrary to Kompren, proposes to create *compound metamodels,* where a set of visible model elements from each combined metamodels is selected, and optionally related. The EMF Views (EMF Views; 50) provides similar approach however without the need to duplicate the meta model elements as opposed to Kompose and Kompren where a new metamodel is created. These virtual view metamodels seem to be usable transparently by tools. Finally, the Global Model Management language (GMM*)[1] (28) provides means to specify and interpret reusable language subsets as sets of constraints combined to form subsetted meta models. Like for EMF Views, these reduced meta models can to some extent be used transparently by tools.

While each of these approaches provides interesting support for modular modeling languages, their unification into a common formalism, the use of an explicit notion of a model interface and their integration into GMM is lacking, except for subsetted metamodels already integrated within the GMM* language.

The execution of integrated models concerns the evaluation of the well-formedness constraints of each combined model alone, but also of the combined models as a whole. To our knowledge, no approach addresses the incremental checking of well-formedness conditions across the different language fragments of compound models. However, some approaches on incremental constraints evaluation exist. In (26), changes on models are expressed as sequences of atomic model operations to determine which constraint is impacted by the changes, so that only these constraints need to be re-evaluated. In (EMF-IncQuery; 133), a graph-based query language (EMF-IncQuery) relying on incremental pattern matching for improved performance is also proposed. In (61), an approach is presented for incremental evaluation of constraints based on a scope of model elements referenced by the query and determined during the first query evaluation. This scope is stored into cache and used to determine which queries need to be re-evaluated according for some model changes. In (83), this approach is extended for the case where the constraints themselves may change besides the constrained models. Finally in (43), an incremental OCL checker is presented where a simpler OCL expression and reduced context elements set are computed from an OCL constraint and a given structural change event. Evaluating this simpler constraint for the reduced context is sufficient to assert the validity of the initial constraint and requires significantly less computation resources.

4.1.2 Model Operations: Construction and Execution

The construction of model operations is addressed in two ways in the literature. Most approaches combine model operations as *model transformations chains* (named (1) *flow composition*), where each chained transformation operates at the granularity of complete models. In order to support reuse and scalability for complex modeling languages, which are defined by composing them from simpler modeling languages, a few approaches have considered specifying model transformations as white boxes. Composed of explicit fine grained operations processing model elements for a given context, these operations are reusable across several model transformations (named (2) *context composition*).

[1] We use * to distinguish this existing language and tool from the generic Global Model Management (GMM) acronym.

4.1.2.1 Flow Composition Approaches

Formal United System Engineering Development (FUSED) (31) is an integration language to specify complex relationships between models of different languages. It supports model transformation chains, but only implicitly via execution of tools, without explicit representation of the involved transformations and processed data. On the contrary, there is a plethora of approaches allowing the explicit specification and construction of model transformation chains implementing a data flow paradigm. A popular one is the AtlanMod Mega Model Management (AM3) tool (AM3), for which the Atlas Transformation Language (ATL) (ATL) is used to specify the model transformations. Besides, a type system has been developed (136), which enables type checking and inference on artifacts related via model transformations. Another similar but less advanced tool is the Epsilon Framework (Epsilon), which provides model transformation chaining via ANT tasks. Wires (124) and ATL Flow (ATLFlow) are tools providing graphical languages for the orchestration of ATL model transformations. The Formalism Transformation Graph + Process Model (FTG+PM) formalism (110) implemented in the AToMPM (A Tool for Multi-Paradigm Modeling) tool (AToMPM) provides similar functionality. However, it has the advantage of also specifying the complete modeling process in addition to the involved model transformations. This is achieved via activity diagrams coupled with model transformation specifications executed automatically to support the development process. Finally, GMM* (28) also supports model transformation chaining, but through the specification of relations between models of specific metamodels that can be chained. One advantage of this approach is that automated incremental (re-)execution of the specified relations between models is provided in response to received model change events. Incrementality of the execution of the transformations is also made possible by the integration of the MoTE (MoTE) incremental model transformation tool into GMM*.

However, while chaining model transformations offers some degree of modularity of model transformation specifications, apart from GMM*, most approaches suffer from scalability issues for large models, since the used transformation tools do not support incremental execution. In addition, the case where a generated model is modified by hand to add information not expressible with the language of the original model(s) cannot easily be handled by these approaches, since regenerating the model modified by hand will destroy the user-specific information. This need is better supported by context composition approaches.

4.1.2.2 Context Composition Approaches

A few approaches allow context composition of model operations. In (66) as mentioned above, independent model transformations are combined, resulting in extended transformations for corresponding extended meta models. In (57), view models are built using contextual composition of model operations (derivation rules) encoded as annotations of queries of the EMF IncQuery (EMF-IncQuery) language. Traceability links between view and source model elements are automatically established and maintained. The use of EMF IncQuery natively provides incremental execution of the derivation rules to synchronize the view model with the source model. Some views may be derived from other views thus allowing flow composition as chains of view models. This approach achieves results similar to TGGs supporting incrementality, however with the drawback of being unidirectional. Similarly, but equpped with bi-directionality, the MoTCoF language (127) allows for both flow- and fine-grained context composition of model transformations. An advantage over (66) however is that model transformations are used as black boxes without the need to adapt the transformations according to the context.

As can be seen, most approaches only support flow type modularity for model operations with batch execution except for the GMM* language thanks to its integration of MoTE providing incremental execution. This will not scale and lead to information losses in case of partial

model information overlap. Only a few approaches allow context modularity, which better supports incremental application where only the impacted operations can be re-applied following a change in order to avoid the cost of re-computing complete transformations. Such is the case of MoTCoF, which theoretically permits incremental execution, but a concrete technical solution is still lacking for it.

4.1.3 Megamodels and other Global Model Management Approaches

Two strands can be identified for GMM. A first one makes use of (1) *model integration languages*, which are defined for a specific set of integrated modeling languages and tools meaning that the integration language must be updated every time a new language or tool is used. The second strand attempts to solve this problem by making use of (2) *mega models* providing configurable global model management.

4.1.3.1 Integration Language and other Approaches

The CyPhy (129) used in the GME modeling tool (GME) and FUSED (31; FUSED) are examples of model integration languages. But as mentioned above, these languages must be adapted as soon as a different set of integrated languages and tools must be used, thus requiring highly skilled developers. Integration languages are therefore not practical.

Open Services for Lifecycle Collaboration (OSLC) (OSLC) provides standards for tool integration through the Web. Many specifications are available for *change management*, *resource previews*, *linked data*, etc. It builds on the W3C *linked data* standard, which aims at providing best practices for publishing structured data on the Web based on the W3C Resource Description Framework (RDF). RDF is a model for data interchange on the Web where data is represented as graphs. However, OSLC is more services (and tools) oriented and inherits the problems of *linked data*, which is specific to the Web and therefore does not separate the concerns of data representation and persistence as opposed to Model-Driven Engineering (MDE) where an abstract syntax is used independently of the way the data is stored.

Another approach making use of these standards is (87) and is implemented in a the CONSYSTENT tool used to identify and resolve inconsistencies across viewpoints due to information overlapping. The information of all models involved during development is captured in a common RDF graph. The approach relies on a human (in parallel, an automated method making use of Bayesian Belief Networks is also under study (86)) to specify patterns representing semantic equivalence links (semantic connections) across the graph models. Inconsistency patterns based on these semantic connections are continuously checked over the RDF model for potential matches identifying inconsistencies. Means to automatically resolve inconsistencies are under development. However, since the conversion of all models as an RDF graph is required, this approach is not incremental and will not scale for large models.

4.1.3.2 Mega Models

In this second strand, megamodels serve to capture and manage MDE resources such as modeling languages, model transformations, model correspondences and tools used in modeling environments. There are several mega modeling approaches as already mentioned. AM3 (AM3) is one of the first initiatives where a megamodel is basically a registry for MDE resources. Model transformations are specified with ATL (ATL) and model correspondences with the Atlas Model Weaving (AMW) language [2]. Similarly, FTG+PM (110) as well as MegaL Explorer (68), allow to model the artifacts used in software development environments and their relations from a linguistic point of view. The involved software languages, related technologies and technological spaces can be captured with linguistic relationships between them such as membership, subset, conformance, input, dependency, definition, etc. Operations between entities can also be captured. The artifacts do not need to be represented as models, but each entity of the

megamodel can be linked to a Web resource that can be browsed and examined. However, the language seems to be used mostly for visualization providing a better understanding of the developments artifacts but cannot be executed to perform model management. The aforementioned GMM* infrastructure (28) consists of a megamodeling language inspired from (85). Metamodels can be declared, as well as relations between models of these meta models. In particular, synchronization relations can relate models of two different meta models making use of the MoTE TGG engine (MoTE) to transform or synchronize the models. As mentioned earlier, chains of model transformations can be specified and executed incrementally in response to model change events and *subsets* of modeling languages can be declared. GMM* is experimented within the Kaolin tool (29) making use of complex and rich industrial languages such as AADL and VHDL thus challenging GMM for realistic specifications.

However, most of these mega modeling approaches only cover to a certain degree the core ingredients of specifying MDE resources by means of meta models and model operations with appropriate modularity and incrementality. Only fragments of the problem are solved. Furthermore, all these megamodeling languages are monolithic and as a result, predefined megamodel fragments cannot be easily composed and reused to avoid rebuilding complete megamodel specifications from scratch for new projects. An attempt towards the reuse of megamodel fragments is presented in (90; 89). The work makes use of megamodeling techniques to propose an automated infrastructure to facilitate customization, composition and reuse of the architect's representational resources to meet project-, domain- and organization-specific needs. Among these megamodeling approaches, only FTG+PM, GMM* and (128; 126) address the automated execution of megamodels in response to model changes or modeling events from the tool's user interface. GMM* and (128; 126) provide incremental execution of mega models to some extent by re-evaluating only the relations concerned with the detected model changes.

4.1.4 Multiformalism Modelling Approaches

According to (117), multiformalism modeling is one of the three dimensions of the Computer Automated Multi-Paradigm Modeling framework, established to allow the representation, the analysis and the synthesis of intricate knowledge at various levels of abstraction, together with multilevel abstraction and metamodeling. Multiformalism, Multiresolution, Multiscale Modeling (M4) environments may provide (55) an important and manageable resource to fulfill the needs for modeling and simulation of modelers that have to deal with complex systems, where complexity derives from heterogeneity of components and relationships, multiple scales, multiple interacting requirements. Besides performance (or verification) oriented issues, multiformalism approaches may also deal with software architecture oriented issues, e.g. by integrating UML as one of the formalisms to assist the development cycle of large, complex software systems (123): in general, literature proposes very popular dedicated transformational approaches for computer automated or assisted software generation, that provide a formal framework to support the steps that lead from a formal or semiformal specification to code, but in the rest of this subsection the focus is on performance oriented approaches.

In multiformalism modeling, many formalisms may be used simultaneously in a model. This may or may not exploit compositionality in the modeling approach, as elements of the different formalisms may coexist in the model, or the model may be composed of submodels written in different (single and particular) formalisms, or the different formalisms may be used in different steps of the processing of the model, by means of model transformation or generation. A general introduction to these themes can be found in (113).

Metamodeling is an important resource for both performance oriented approaches (105)(134) and software oriented transformation based tools, consequently metamodeling-based multiformalism approaches can be considered a peculiar category. Another special category of

approaches is constituted by the ones that deal with hybrid systems, that support multiformalisms with both continuous and discrete nature, and are thus capable of modeling natural systems in a better way (141). These approaches should be able to describe and solve jointly and coherently differential equation like descriptions and state space-based descriptions, for a same complex system. While the problem has been popular in the 70s and 80s, there is currently a renovated interest in it from the point of view of cyberphysical systems: the interested reader can find specific general multiformalism approaches in (140), (18) and (19), that also provides an overview of selected previous, classical literature.

Approaches

With reference to multiformalism approaches oriented to performance evaluation, a number of different naive and structured approaches to the problem have been presented in literature (a survey is provided in (12)). In the second group, the approaches have been implemented in a number of different tools, with different backgrounds, such as SHARPE, SMART, DEDS, AToM3, Möbius, OsMoSys and SIMTHESys. These tools also differ in the solution strategy adopted for the evaluation of models, and are designed with different purposes (e.g. some of them are designed to be extensible, some for experimenting new formalism variants, some optimize the solution process).

SHARPE (132) supports the composition by submodels of some given different formalisms, solved by different solvers, but based on Markovian approaches. The composition consists in the exchange of probability distributions between submodels. SMART (47)(48)(46) supports the specification and solution, by simulation or approximation, of complex discrete-state systems. DEDS (22) provides a common abstract notation in which submodels written in different formalisms are translated. Möbius (125)(49)(53)(56) supports, by states and events superposition, a number of different formalisms (that can be extended by user provided code) and alternative solvers (that can be chosen by the modeler) in a very articulated modeling and solution process.

Other approaches exploit, in different ways, metamodeling too. AToM3 (107; 58) exploits metamodeling to implement model transformations, used to solve models by its solver. OsMoSys (71; 137; 72; 73; 81) and SIMTHESys (14; 16; 94; 93) use metamodeling to let different user-defined formalisms interact by founding them over common metaformalisms and using elements and formalism level inheritance, and to implement different compositional mechanisms: while OsMoSys implements ad-hoc operators for parameters exchange between submodels, and integrates external solvers by means of orchestration and adapters, SIMTHESys privileges the experimentation of user-defined formalisms and embeds into formalism elements the interactions between different formalisms implementing multiformalism by arcs superposition, allowing the automatic synthesis of proper solvers, according to the nature of the involved formalisms (with no claim for their optimality): there is an explicit specification of both syntax and semantics of every formalism element to allow high flexibility in the specification of custom, user defined formalisms. For more details, the reader may refer to (113), that provides a more detailed analysis on multiformalism features and implementation, solution processes, purposes, compositional and transformational mechanisms of these approaches.

Solution

Most of the approaches are backed up with state space analysis techniques. Both analytical- and simulation-based methods are applied to perform the analysis, eventually with specific solutions to cope with the state space explosion problem, such as folding, decomposition, product forms solutions. The most common way is to directly generate the whole state space, with (e.g. in Möbius or in (120)) or without (e.g. in SMART or in some SIMTHESys solvers) an intermediate step of simple translation or more sophisticated transformation towards a specific intermediate representation, or by using partial state spaces exploiting modularity (e.g. in Os-

MoSys or in some SIMTHESys solvers(21)), or by transformation (e.g. in AToM, or in (30)). Noticeable are the approaches that exploit mean field analysis to cope with very large space states (e.g. (32) or (44)).

Applications

The literature provides a conspicuous number of applications: here some significant examples are provided. The effect of cyber-exploits have on information sharing and task synchronization have been studied in (108); performance evaluation of Service Oriented Architecture have been studied in (1) and (95); cardiovascular system and its regulation has been studied, with a hybrid approach, in (**?**); interdependencies in electric power systems have been studied in (45); the ERMTS/ETCS European standard for high speed trains has been studied in (70); security attacks have been studied in (82); exceptions aware systems have been studied in (20); effects of software rejuvenation techniques have been studied in (13); NoSQL systems have been studied in (17). Multiformalism has been also applied as an implementation technique to provide higher level tools or formalisms: in (122) a flexible, optimized Repairable Fault Tree modeling and solution approach is presented; a performance oriented model checking example is given in (15); an analysis framework for detecting inconsistencies in high level semantic relationships between models has been developed in (121).

4.2 Ontology Overview

This ontology captures the Multi-Paradigm Modeling Domain (MPM). It includes concepts for the related modeling, linguistic and formal sub domains.

Figure 4.1 shows an overview of the MPM ontology. The details of each concept are provided in the following subsections.

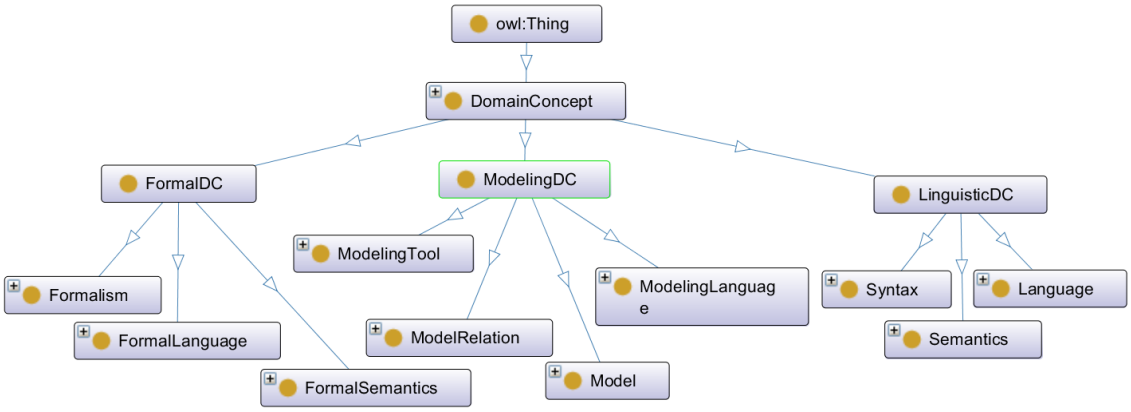

Figure 4.1: Overview of the MPM ontology

4.3 Domain Concepts

This ontology of multi-paradigm modeling contains concepts divided into sub-domains as presented in the following subsections.

4.3.1 FormalDC

This class groups domain concepts that are related to the formal aspects of MPM.

4.3.1.1 Architecture

A system architecture is a conceptual model that defines the structure, behavior, and more views of a system. An architecture description is a formal description and representation of a

system, organized in a way that supports reasoning about the structures and behaviors of the system.

Subclass of

- **FormalSemantics** (see section 4.3.3.14)

4.3.1.2 Behavioral

Subclass of

- **Architecture** (see section 4.3.3.2)

4.3.1.3 BehavioralConstraintLanguage

Subclass of

- **ConstraintLanguage** (see section 4.3.3.9)

4.3.1.4 Centralized

Subclass of

- **Deployment** (see section 4.3.3.10)

4.3.1.5 Communication

Communication is the act of conveying intended meanings from one entity or group to another through the use of mutually understood signs and semiotic rules.

Subclass of

- **Behavioral** (see section 4.3.3.4)

4.3.1.6 ConstraintLanguage

Subclass of

- **FormalLanguage** (see section 4.3.3.13)

4.3.1.7 Deployment

The deployment of a mechanical device, electrical system, computer program, etc., is its assembly or transformation from a packaged form to an operational working state. Deployment implies moving a product from a temporary or development state to a permanent or desired state.

Subclass of

- **FormalSemantics** (see section 4.3.3.14)

4.3.1.8 Distributed

Subclass of

- **Deployment** (see section 4.3.3.10)

4.3.1.9 FormalLanguage

A formal language is a language whose semantics is formally defined. This formal semantics relates to the formalism(s) the language realizes. A formal language has its abstract syntax formally defined.

Subclass of

- **FormalDC** (see section 4.3.1)

- **Language** (see section 4.3.3.16)

References

- TODO Moussa

4.3.1.10 FormalSemantics

In programming language theory, semantics is the field concerned with the rigorous mathematical study of the meaning of programming languages. It does so by evaluating the meaning of syntactically legal strings defined by a specific programming language, showing the computation involved. In such a case that the evaluation would be of syntactically illegal strings, the result would be non-computation. Semantics describes the processes a computer follows when executing a program in that specific language. This can be shown by describing the relationship between the input and output of a program, or an explanation of how the program will execute on a certain platform, hence creating a model of computation.

Subclass of

- **FormalDC** (see section 4.3.1)
- **Semantics** (see section 4.3.3.19)

4.3.1.11 Structural

Structure is an arrangement and organization of interrelated elements in a material object or system, or the object or system so organized.

Subclass of

- **Architecture** (see section 4.3.3.2)

4.3.1.12 StructuralConstraintLanguage

Subclass of

- **ConstraintLanguage** (see section 4.3.3.9)

4.3.2 FormalismDC

This class groups classes related to formalisms

4.3.2.1 AutomataBasedFormalism

The class of formalisms that are based on automata

Subclass of

- **Formalism** (see section 4.3.2.6)

4.3.2.2 BehavioralCharacteristic

Subclass of

- **FormalismCharacteristic** (see section 4.3.2.7)

4.3.2.3 ContinuousCharacteristic

Subclass of

- **BehavioralCharacteristic** (see section 4.3.2.2)

4.3.2.4 DiscreteCharacteristic

Subclass of

- **BehavioralCharacteristic** (see section 4.3.2.2)

4.3.2.5 FlowBasedFormalism

Subclass of

- **Formalism** (see section 4.3.2.6)

4.3.2.6 Formalism

Formalisms are mathematical objects consisting of an abstract syntax and a formal semantics. Languages are concrete implementations of formalisms. A language has a concrete syntax, may deviate slightly from the formalism in the semantics that it implements, or may implement multiple semantics (e.g., changing the type of numerical solver in a simulation tool may change the behavior of a model). Also, a language may implement more than one formalisms.

Subclass of

- **FormalismDC** (see section 4.3.2)

References

-

4.3.2.7 FormalismCharacteristic

Subclass of

- **FormalismDC** (see section 4.3.2)

4.3.2.8 FormalismFamily

Subclass of

- **FormalismDC** (see section 4.3.2)

4.3.2.9 HybridAutomataBasedFormalism

Subclass of

- **AutomataBasedFormalism** (see section 4.3.2.1)

4.3.2.10 LogicBasedFormalism

Subclass of

- **Formalism** (see section 4.3.2.6)

4.3.2.11 PetriNetBasedFormalism

Petri nets (also known as a place/transition net or P/T net) are formalisms for the description of distributed systems.

Subclass of

- **Formalism** (see section 4.3.2.6)

4.3.2.12 StructureCharacteristic

Subclass of

- **FormalismCharacteristic** (see section 4.3.2.7)

4.3.2.13 TimedAutomataBasedFormalism

Subclass of

- **HybridAutomataBasedFormalism** (see section 4.3.2.9)

4.3.2.14 TimedCharacteristic

Subclass of

- **BehavioralCharacteristic** (see section 4.3.2.2)

4.3.2.15 UncertaintyCharacteristic

Subclass of

- **BehavioralCharacteristic** (see section 4.3.2.2)

4.3.3 LinguisticDC

This class groups domain concepts related to the linguistic aspects of MPM, such as modeling, programming, formal and even natural languages and their syntax.

4.3.3.1 AbstractSyntax

metamodel

Subclass of

- **Syntax** (see section 4.3.3.22)

4.3.3.2 Architecture

A system architecture is a conceptual model that defines the structure, behavior, and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structures and behaviors of the system.

Subclass of

- **FormalSemantics** (see section 4.3.3.14)

4.3.3.3 ArchitectureDescriptionLanguage

Architecture description languages (ADLs) are used in several disciplines: system engineering, software engineering, and enterprise modelling and engineering.

The system engineering community uses an architecture description language as a language and/or a conceptual model to describe and represent system architectures.

The software engineering community uses an architecture description language as a computer language to create a description of a software architecture. In the case of a so-called technical architecture, the architecture must be communicated to software developers; a functional architecture is communicated to various stakeholders and users. Some ADLs that have been developed are: Acme (developed by CMU), AADL (standardized by the SAE), C2 (developed by UCI), SBC-ADL (developed by National Sun Yat-Sen University), Darwin (developed by Imperial College London), and Wright (developed by CMU).

The up-to-date list of currently existing architectural languages might be found at Up-to-date list of ADLs.

The ISO/IEC/IEEE 42010 document, Systems and software engineering-Architecture description, defines an architecture description language as "any form of expression for use in architecture descriptions" and specifies minimum requirements on ADLs.

The enterprise modeling and engineering community have also developed architecture description languages catered for at the enterprise level. Examples include ArchiMate (now a

standard of The Open Group), DEMO, ABACUS (developed by the University of Technology, Sydney). These languages do not necessarily refer to software components, etc. Most of them, however, refer to an application architecture as the architecture that is communicated to the software engineers.

Most of the writing below refers primarily to the perspective from the software engineering community.

Subclass of

- **ModelingLanguage** (see section 4.3.4.13)

4.3.3.4 Behavioral

Subclass of

- **Architecture** (see section 4.3.3.2)

4.3.3.5 BehavioralConstraintLanguage

Subclass of

- **ConstraintLanguage** (see section 4.3.3.9)

4.3.3.6 Centralized

Subclass of

- **Deployment** (see section 4.3.3.10)

4.3.3.7 Communication

Communication is the act of conveying intended meanings from one entity or group to another through the use of mutually understood signs and semiotic rules.

Subclass of

- **Behavioral** (see section 4.3.3.4)

4.3.3.8 ConcreteSyntax

textual/graphics as in AADL

Subclass of

- **Syntax** (see section 4.3.3.22)

4.3.3.9 ConstraintLanguage

Subclass of

- **FormalLanguage** (see section 4.3.3.13)

4.3.3.10 Deployment

The deployment of a mechanical device, electrical system, computer program, etc., is its assembly or transformation from a packaged form to an operational working state. Deployment implies moving a product from a temporary or development state to a permanent or desired state.

Subclass of

- **FormalSemantics** (see section 4.3.3.14)

4.3.3.11 Distributed

Subclass of

- **Deployment** (see section 4.3.3.10)

4.3.3.12 DomainSpecificLanguage

A domain-specific language (DSL) is a computer language specialized to a particular application domain. This is in contrast to a general-purpose language (GPL), which is broadly applicable across domains.

Subclass of

- **Language** (see section 4.3.3.16)

4.3.3.13 FormalLanguage

A formal language is a language whose semantics is formally defined. This formal semantics relates to the formalism(s) the language realizes. A formal language has its abstract syntax formally defined.

Subclass of

- **FormalDC** (see section 4.3.1)
- **Language** (see section 4.3.3.16)

References

- TODO Moussa

4.3.3.14 FormalSemantics

In programming language theory, semantics is the field concerned with the rigorous mathematical study of the meaning of programming languages. It does so by evaluating the meaning of syntactically legal strings defined by a specific programming language, showing the computation involved. In such a case that the evaluation would be of syntactically illegal strings, the result would be non-computation. Semantics describes the processes a computer follows when executing a program in that specific language. This can be shown by describing the relationship between the input and output of a program, or an explanation of how the program will execute on a certain platform, hence creating a model of computation.

Subclass of

- **FormalDC** (see section 4.3.1)
- **Semantics** (see section 4.3.3.19)

4.3.3.15 Graphical

Subclass of

- **ConcreteSyntax** (see section 4.3.3.8)

4.3.3.16 Language

A language is a concrete realization of a set of formalisms. A language has a set of concrete syntaxes. The language may deviate slightly from the formalisms it realizes in the semantics that it realizes, or may realize multiple semantics.

Subclass of

- **LinguisticDC** (see section 4.3.3)

References

•

4.3.3.17 ModelingLanguage

A modeling language is any artificial language that can be used to express information or knowledge or systems in a structure that is defined by a consistent set of rules. The rules are used for interpretation of the meaning of components in the structure.

Subclass of

- **Language** (see section 4.3.3.16)
- **ModelingDC** (see section 4.3.4)

4.3.3.18 ProgrammingLanguages

A programming language is a formal computer language designed to communicate instructions to a machine, particularly a computer. Programming languages can be used to create programs to control the behavior of a machine or to express algorithms.

Subclass of

- **Language** (see section 4.3.3.16)

4.3.3.19 Semantics

Semantics (from Ancient Greek: "significant") is primarily the linguistic, and also philosophical study of meaning in language, programming languages, formal logics, and semiotics. It focuses on the relationship between signifiers-like words, phrases, signs, and symbols-and what they stand for, their denotation.

Subclass of

- **LinguisticDC** (see section 4.3.3)

4.3.3.20 Structural

Structure is an arrangement and organization of interrelated elements in a material object or system, or the object or system so organized.

Subclass of

- **Architecture** (see section 4.3.3.2)

4.3.3.21 StructuralConstraintLanguage

Subclass of

- **ConstraintLanguage** (see section 4.3.3.9)

4.3.3.22 Syntax

Subclass of

- **LinguisticDC** (see section 4.3.3)

4.3.3.23 Textual

Subclass of

- **ConcreteSyntax** (see section 4.3.3.8)

4.3.3.24 TransformationLanguage

A transformation language is a computer language designed to transform some input text in a certain formal language into a modified output text that meets some specific goal.

Subclass of

- **Language** (see section 4.3.3.16)

4.3.4 ModelingDC

This class groups domain concepts related to the modeling aspects of MPM.

4.3.4.1 ApplicationMegaModel

A representation of the real models in the CPS development environment.

Subclass of

- **Megamodel** (see section 4.3.4.6)

4.3.4.2 ArchitectureDescriptionLanguage

Architecture description languages (ADLs) are used in several disciplines: system engineering, software engineering, and enterprise modelling and engineering.

The system engineering community uses an architecture description language as a language and/or a conceptual model to describe and represent system architectures.

The software engineering community uses an architecture description language as a computer language to create a description of a software architecture. In the case of a so-called technical architecture, the architecture must be communicated to software developers; a functional architecture is communicated to various stakeholders and users. Some ADLs that have been developed are: Acme (developed by CMU), AADL (standardized by the SAE), C2 (developed by UCI), SBC-ADL (developed by National Sun Yat-Sen University), Darwin (developed by Imperial College London), and Wright (developed by CMU).

The up-to-date list of currently existing architectural languages might be found at Up-to-date list of ADLs.

The ISO/IEC/IEEE 42010 document, Systems and software engineering-Architecture description, defines an architecture description language as "any form of expression for use in architecture descriptions" and specifies minimum requirements on ADLs.

The enterprise modeling and engineering community have also developed architecture description languages catered for at the enterprise level. Examples include ArchiMate (now a standard of The Open Group), DEMO, ABACUS (developed by the University of Technology, Sydney). These languages do not necessarily refer to software components, etc. Most of them, however, refer to an application architecture as the architecture that is communicated to the software engineers.

Most of the writing below refers primarily to the perspective from the software engineering community.

Subclass of

- **ModelingLanguage** (see section 4.3.4.13)

4.3.4.3 CapturingOperation

The process of capturing information (e.g from users) into a model.

Subclass of

- **TransformationOperation** (see section 4.3.4.17)

4.3.4.4 ConfigurationMegaModel

Declares types for the models and relations contained in application megamodel. (language and relation types)

Subclass of

- **Megamodel** (see section 4.3.4.6)

4.3.4.5 IntegrationOperation

The process to integrate many models together.

Subclass of

- **ModelOperation** (see section 4.3.4.10)

4.3.4.6 Megamodel

A model that contains models and relations between them.

Subclass of

- **Model** (see section 4.3.4.8)

4.3.4.7 MegamodelFragment

The fragment is not a complete megamodel, and can't be used alone. It can be reused to take part in another megamodel. (e.g. physical part, view of the system, self-adaptation .... etc.)

Subclass of

- **Model** (see section 4.3.4.8)

4.3.4.8 Model

A representation of real artifacts regardless of the metamodeling technical space. e.g. xml file, equations...etc.

Subclass of

- **ModelingDC** (see section 4.3.4)

4.3.4.9 ModelConstraint

A restriction over model that is input or output for relation

Subclass of

- **ModelingDC** (see section 4.3.4)
- **Constraint** (see section 2.2.3.1)

4.3.4.10 ModelOperation

It is any kind of transformation from input set of models to output set of models.

Subclass of

- **ModelRelation** (see section 4.3.4.11)
- **Action** (see section 2.2.2.1)

4.3.4.11 ModelRelation

A relation that connects models.

Subclass of

- **ModelingDC** (see section 4.3.4)
- **Relation** (see section 2.2.3.2)

4.3.4.12 ModelingActivity

Subclass of

- **ModelingDC** (see section 4.3.4)
- **Activity** (see section 2.2.2.2)

4.3.4.13 ModelingLanguage

A modeling language is any artificial language that can be used to express information or knowledge or systems in a structure that is defined by a consistent set of rules. The rules are used for interpretation of the meaning of components in the structure.

Subclass of

- **Language** (see section 4.3.3.16)
- **ModelingDC** (see section 4.3.4)

4.3.4.14 ModelingParadigm

Subclass of

- **ModelingDC** (see section 4.3.4)

4.3.4.15 ModelingTool

Tools to model the system.

Subclass of

- **ModelingDC** (see section 4.3.4)
- **Tool** (see section 2.2.5.1)

4.3.4.16 TracabilityRelation

Subclass of

- **ModelOperation** (see section 4.3.4.10)

4.3.4.17 TransformationOperation

The process to transform one model to another.

Subclass of

- **ModelOperation** (see section 4.3.4.10)

4.4 Properties

5 Ontology of Multi-Paradigm Modeling for Cyber-Physical Systems

5.1 Introduction

This ontology of MPM for CPS provides cross-cutting concepts between the two domains of CPS and MPM. At the current stage of development, this ontology only contains a limited number of classes related to *viewpoints* inspired from (34).

5.2 Ontology Overview

Figure 4.1 shows an overview of the MPM4CPS ontology. The details of each concept are provided in the following subsections.

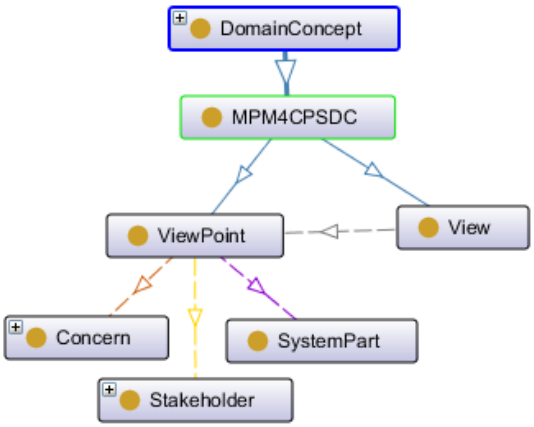

Figure 5.1: Overview of the MPM4CPS ontology

5.3 Domain Concepts

This ontology of multi-paradigm modeling contains concepts divided into sub-domains as presented in the following subsections.

5.3.1 MPM4CPSDC

5.3.1.1 View

Subclass of

- **MPM4CPSDC** (see section 5.3.1)
- **View** (see section 5.3.1.1)

5.3.1.2 ViewPoint

Subclass of

- **MPM4CPSDC** (see section 5.3.1)
- **ConcernedElement** (see section 2.2.4.2)
- **ViewPoint** (see section 5.3.1.2)
- **ViewPoint** (see section 5.3.1.2)

5.4 Properties

5.4.1 hasActions

Subproperty of: None

Domains:

- **Activity** (see section 2.2.2.2)

Ranges:

- **Action** (see section 2.2.2.1)

5.4.2 hasActivities

Subproperty of: None

Domains:

- **Process** (see section 2.2.2.3)

Ranges:

- **Activity** (see section 2.2.2.2)

5.4.3 hasCharacteristic

Subproperty of: None

Domains:

- **Formalism** (see section 4.3.2.6)

Ranges:

- **FormalismCharacteristic** (see section 4.3.2.7)

5.4.4 hasChildFormalism

Subproperty of: None

Domains:

- **FormalismFamily** (see section 4.3.2.8)

Ranges:

- **Formalism** (see section 4.3.2.6)

5.4.5 hasChildLanguage

Subproperty of: None

Domains:

- **Language** (see section 4.3.3.16)

Ranges:

- **Language** (see section 4.3.3.16)

5.4.6 hasConcerns

A concern of stakeholder. Subproperty of: None

Domains:

- **ConcernedElement** (see section 2.2.4.2)

Ranges:

- **Concern** (see section 2.2.4.1)

5.4.7 hasConstraint

A constraint applied on relation. Subproperty of: None

Domains:

- **Relation** (see section 2.2.3.2)

Ranges:

- **Constraint** (see section 2.2.3.1)

5.4.8 hasContext

Subproperty of: None

Domains:

- **ModelOperation** (see section 4.3.4.10)

Ranges:

- **ModelOperation** (see section 4.3.4.10)

5.4.9 hasEvolvedTo

An updated version of an element. (e.g.Tool evolved to another) Subproperty of: None

Domains: None

Ranges: None

5.4.10 hasInput

The model need inputs to do perform the determined task. Subproperty of:

- **isConnecting** (see section 5.4.34)

Domains:

- **ModelOperation** (see section 4.3.4.10)

Ranges:

- **Model** (see section 4.3.4.8)

5.4.11 hasInputModel

It describes the input model for a model relation. Subproperty of: None

Domains:

- **ModelRelation** (see section 4.3.4.11)

Ranges:

- **Model** (see section 4.3.4.8)

5.4.12 hasLanguage

The relation defines the formal language for a formalism. It is an inverse of hasFormalism. Subproperty of: None

Domains:

- **Formalism** (see section 4.3.2.6)

Ranges:

- **Language** (see section 4.3.3.16)

5.4.13 hasMegamodelFragment

It describes the relation between megamodel and its fragments. Subproperty of:

- **hasModel** (see section 5.4.14)

Domains:

- **Megamodel** (see section 4.3.4.6)

Ranges:

- **MegamodelFragment** (see section 4.3.4.7)

5.4.14 hasModel

It describes a recursive relation. e.g. MegaModelFragment hasModel xModel ... Subproperty of: None

Domains:

- **Model** (see section 4.3.4.8)

Ranges:

- **Model** (see section 4.3.4.8)

5.4.15 hasModelConstraint

Subproperty of:

- **hasConstraint** (see section 5.4.7)

Domains:

- **ModelOperation** (see section 4.3.4.10)

Ranges:

- **ModelConstraint** (see section 4.3.4.9)

5.4.16 hasModelOperation

It defines a model operation for a model. Subproperty of:

- **hasModelRelation** (see section 5.4.17)

Domains:

- **Model** (see section 4.3.4.8)

Ranges:

- **ModelOperation** (see section 4.3.4.10)

5.4.17 hasModelRelation

It defines a model relation for a model. Subproperty of: None

Domains:

- **Model** (see section 4.3.4.8)

Ranges:

- **ModelRelation** (see section 4.3.4.11)

5.4.18 hasNext

Subproperty of: None

Domains:

- **Action** (see section 2.2.2.1)

Ranges:

- **Action** (see section 2.2.2.1)

5.4.19 hasOutput

The model provides results in the following form. Subproperty of:

- **isConnecting** (see section 5.4.34)

Domains:

- **ModelOperation** (see section 4.3.4.10)

Ranges:

- **Model** (see section 4.3.4.8)

5.4.20 hasOutputModel

It describes the output model for a model relation. Subproperty of: None

Domains:

- **ModelRelation** (see section 4.3.4.11)

Ranges:

- **Model** (see section 4.3.4.8)

5.4.21 hasProvider

The company/university which the tool is developed by. Subproperty of: None

Domains: None

Ranges: None

5.4.22 hasPurpose

Subproperty of: None

Domains:

- **Model** (see section 4.3.4.8)

Ranges:

- **Purpose** (see section 2.2.4.4)

5.4.23 hasRelations

Subproperty of: None

Domains:

- **Model** (see section 4.3.4.8)

Ranges:

- **ModelRelation** (see section 4.3.4.11)

5.4.24 hasRole

A role of stakeholder Subproperty of: None

Domains:

- **Stakeholder** (see section 2.2.4.6)

Ranges:

- exactly 1 **Role** (see section 2.2.4.5)

5.4.25 hasStakeholders

Subproperty of: None

Domains:

- **ViewPoint** (see section 5.3.1.2)

Ranges:

- **Stakeholder** (see section 2.2.4.6)

5.4.26 hasSubFormalismFamily

Subproperty of: None

Domains:

- **FormalismFamily** (see section 4.3.2.8)

Ranges:

- **FormalismFamily** (see section 4.3.2.8)

5.4.27 hasSystemPart

Subproperty of: None

Domains:

- **ViewPoint** (see section 5.3.1.2)

Ranges:

- **SystemPart** (see section 3.4.4.3)

5.4.28 hasTool

The tool that represent the Language. It is an inverse of isToolFor. Subproperty of: None

Domains:

- **Language** (see section 4.3.3.16)

Ranges:

- **Tool** (see section 2.2.5.1)

5.4.29 hasViewpoint

Subproperty of: None

Domains:

- **View** (see section 5.3.1.1)

Ranges:

- **ViewPoint** (see section 5.3.1.2)

5.4.30 isAppliedTo

A constraint is applied to a some element. (e.g. relation) Subproperty of: None

Domains:

- **Constraint** (see section 2.2.3.1)

Ranges: None

5.4.31 isAppliedToModel

Subproperty of:

- **isAppliedTo** (see section 5.4.30)

Domains:

- **Constraint** (see section 2.2.3.1)

Ranges:

- **Model** (see section 4.3.4.8)

5.4.32 isBasedOnFormalism

The relation defines the formalism for a formal language. It is an inverse of hasLanguage. Subproperty of: None

Domains:

- **Language** (see section 4.3.3.16)

Ranges:

- **Formalism** (see section 4.3.2.6)

5.4.33 isCharacterizedBy

Subproperty of: None

Domains:

- **ViewPoint** (see section 5.3.1.2)

Ranges:

- **Concern** (see section 2.2.4.1)

5.4.34 isConnecting

A relation connects a model. Subproperty of: None

Domains:

- **ModelOperation** (see section 4.3.4.10)

Ranges:

- **Model** (see section 4.3.4.8)

5.4.35 isExtending

This property captures a language extensions. Typically, a language extends another one by adding constructs and possibly refining / overriding some ot the extended language. Subproperty of: None

Domains:

- **Language** (see section 4.3.3.16)

Ranges:

- **Language** (see section 4.3.3.16)

5.4.36 isExtendingFormalism

Subproperty of: None

Domains:

- **Formalism** (see section 4.3.2.6)

Ranges:

- **Formalism** (see section 4.3.2.6)

5.4.37 isPerformedBy

Subproperty of: None

Domains: None

Ranges: None

5.4.38 isReturningTo

Subproperty of: None

Domains:

- **IntegrationOperation** (see section 4.3.4.5)

Ranges:

- **IntegrationOperation** (see section 4.3.4.5)

5.4.39 isSpecializing

Subproperty of: None

Domains:

- **Formalism** (see section 4.3.2.6)

Ranges:

- **Formalism** (see section 4.3.2.6)

5.4.40 isSupportedBy

Subproperty of: None

Domains:

- **ViewPoint** (see section 5.3.1.2)

Ranges:

- **Formalism** (see section 4.3.2.6)

5.4.41 isToolFor

The language that is represented by a tool. It is an inverse of hasTool. Subproperty of: None

Domains:

- **Tool** (see section 2.2.5.1)

Ranges:

- **Language** (see section 4.3.3.16)

6 Examples

6.1 Ensemble-based CPS

6.1.1 Overview

An Ensemble-Based Cyber-Physical System (EBCPS) is an emergent system that is distributed, decentralized, dynamic, self-adaptive and scalable. It consists of autonomous components and forms ensembles of them depending on the context in the aim of achieving determined goals (i.e. organizing and decision-making). More specifically, the composition of components changes according to the fact of having the components appear and disappear dynamically, in addition to the unexpected changes in the environment and new requirements. There are many applications in different domains such as Traffic and Transport, Robotics, and Clouds. For instance, an ensemble of vehicle planning for optimal route with considering the road and traffic conditions.

The interest of building such systems was reflected in many European projects such as ASCENS[1], ALLOW Ensembles[2] and FoCAS [3], which are FP7 projects. ASCENS is oriented towards design and verification, while ALLOW Ensembles is more oriented towards performance. More specifically, ASCENS targets formalizing and modeling ensembles of autonomic-service components (SCs) that depend on knowledge units (K). It considers also the expression of self-adaption and provides tools and use cases. ALLOW Ensembles focuses on developing algorithms that improve ensembles utility and system dependability. Both previous projects are involved in FoCAS, which is a platform for communities that care about developing Collective Adaptive Systems (CASś).

As part from ASCENS project, a formalism language was introduced to express the ensembles called Service Component Ensemble Language (SCEL) Figure 6.1. The language allows to define Knowledge, Behaviors, Aggregations, and Policies. It allows the developer to define interfaces with attributes or knowledge for Service Components (SCs) and their behavior using processes. Also, the developer can define the Service Component Ensembles (SCEs) and the conditions or policies to form them. The ensembles are responsible for exchanging knowledge between the components. Each ensemble evaluates the constraints over the interface attributes of the involved components. The support of context-awareness comes from using attributes in the constraint evaluation of forming ensembles.

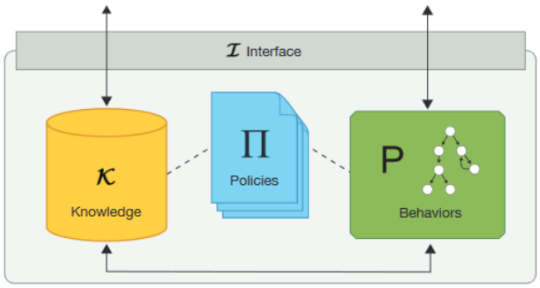

Figure 6.1: Service Component

Furthermore, ASCENS provides many tools for modeling (e.g. HELENA, KnowLang, DEECo) and for simulation (e.g. ARGoS, SPL, jDEECo Java, SimSOTA). Additionally, the concepts are

[1] http://www.ascens-ist.eu/
[2] http://www.allow-ensembles.eu/
[3] http://www.focas.eu/

presented in three use cases in different domains, which are: Clouds, Traffic and Transport, and Robotics.

6.1.2 Dependable Emergent Ensembles of Components (DEECo)

To manifest the new concepts of EBCPS, the Department of Distributed and Dependable Systems (d3s: http://d3s.mff.cuni.cz/) at Charles University in Prague developed an EBCPS toolchain that is used as development environment which merges the concepts of emergent systems with the CPS parts (i.e. physical, computational and network). More specifically, the parts that will be explained here are requirements, design, runtime, self-adaptation, analysis, composition and simulation. Moreover, there are many integrations and transformations between the models.

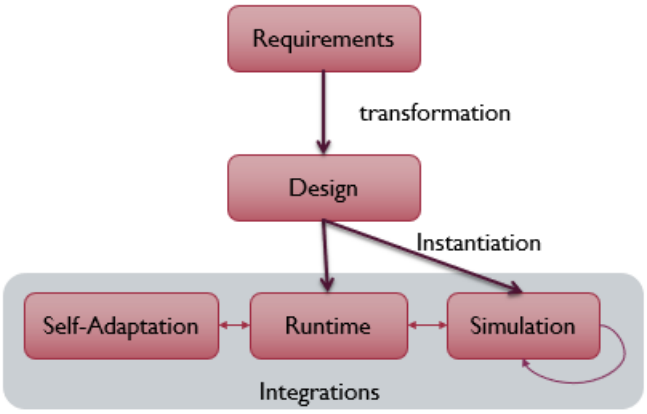

Figure 6.2: Overview of the models provided by DEECo and their relations

6.1.2.1 Requirements

Starting with modeling EBCPS requirements, Invariant Refinement Method (IRM) (98) (see Figure **??**, which is developed in Epsilon, is used to represent the tree of invariants and assumptions for the system. The tree ends with leaves of two types: 1) processes which are part of a component, 2) or exchange knowledge between components which are ensembles. Simply put, the IRM allows to connect the requirements to the architecture entities directly. Later on, the developer can transform the IRM model to code which is Java for the current represented tool-chain under DEECo specification.

6.1.2.2 Design and Runtime

Regarding the design and runtime parts (35)(2) , the team developed a runtime environment which is implemented in java and is known as jDEECo. The developer is able to design systems that respect the DEECo specification (i.e. SCEL specification), which is captured by specific Java annotations provided by jDEECo runtime Figure 6.4. More specifically, the designed system consists of roles, components, and ensembles. Each role has a set of attributes, which represent the knowledge related to this specific role. Furthermore, each component is defined as a set of processes featuring (a) role(s) having by that the role knowledge besides its local knowledge. Having knowledge in the role allows for preserving the encapsulation of the components that features the role and provides a separation in concerns. Hence, each ensemble forms depending on the context and perform knowledge exchange between different components with determined roles.

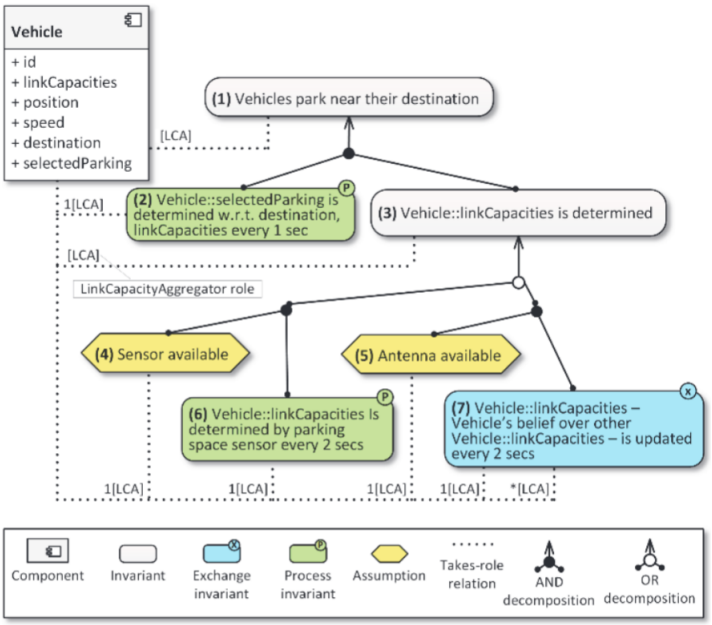

Figure 6.3: IRM tree for a smart parking senario

```
@Component
public class VehicleComponent {
   public String id;
   public Map<…> linkCapacities;
   public Coord position;
   public Double speed;
   public Link destination;
   public Parking selectedParking;

   @Process
   @PeriodicScheduling(period=500)
   public static void measureSpeed(
      @Out("speed") Double speed)
         {…}
}

@Ensemble
@PeriodicScheduling(period=1000)
public class CapacityExchangeEnsemble {

   @Membership
   public static boolean membership(
      @In("coord.position") Coord cPos,
      @In("member.position") Coord mPos)
         {…}

   @KnowledgeExchange
   public static void exchange(
      @InOut("coord.linksCapacities") Map<…> cLinksCapacities,
      @InOut("member.linksCapacities") Map<…> mLinksCapacities)
         {…}
}
```

Figure 6.4: Snippet of jDEECo code

It is worth mentioning that there is another framework that integrates with jDEECo, which is called Intelligent Ensemble framework[4] (103). It provides the developer with Ensemble Definition Language (EDL) that was developed using XText and XPand on Eclipse Modelling Framework. At runtime the formation of the ensemble is optimized by using Z3 SMT solver.

[4] http://d3s.mff.cuni.cz/software/deeco/files/seams-2017.zip or http://dx.doi.org/10.4230/DARTS.3.1.6

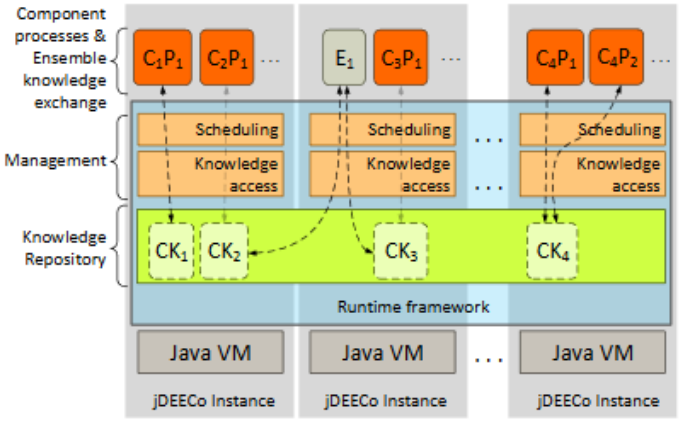

Figure 6.5: jDEECo runtime framework architecture

```
1  package SeamsDemo
2  data contract Car
3    priority : int
4    targetX : int
5    targetY : int
6    driveToX : int
7    driveToY : int
8  end
9
10 data contract ParkingLot
11   locationX : int
12   locationY : int
13   capacity : int
14 end
15
16 ensemble ParkingAssignment
17   id parking : ParkingLot
18   membership
19     roles
20       cars [0..20] : Car where (
(Abs(it.targetX - parking.locationX) +
Abs(it.targetY - parking.locationY)) < 10)
21     constraints
22       constraint sum cars 1 <=
parking.capacity
23     fitness sum cars it.priority
24   knowledge exchange
25       cars.driveToX = parking.locationX
26       cars.driveToY = parking.locationY
```

Figure 6.6: An example with an EDL specification

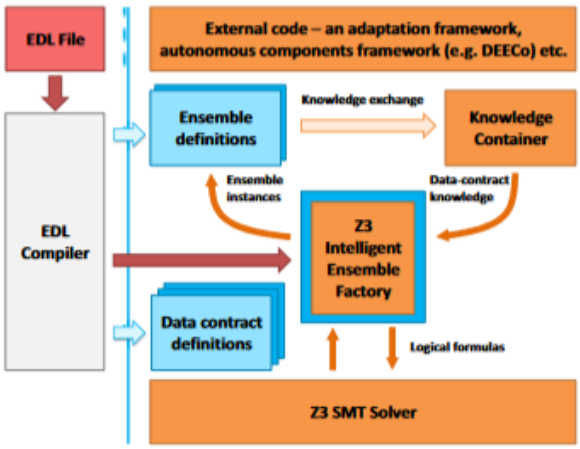

Figure 6.7: Framework high-level architecture that supports Ensemble Definition Language (EDL)

Not only is DEECo implemented in java, but also it is implemented in C++ [5] and Scala[6]. Nevertheless, the model operations that we mentioned here (i.e. transformation and integrations) are applied on jDEECo (i.e. java implementation).

[5] http://d3s.mff.cuni.cz/projects/components_and_services/deeco/files/CDEECo.zip

6.1.2.3 Self-Adaptation

Self-adaptation is done in two methods. The first method is by transforming IRM tree to java code with annotations, then at runtime the annotations are processed by IRM-SA engine which uses SAT solver to make decisions between the different paths. The other method is by using modes and mode-switching annotations that are provided by DEECo. It is possible to associate a mode to a set of processes, thus when a mode-switch happens a different set of processes is activated.

There is a part that is presented as an extension of mode-switch transition logic, which are: 1) ordinary differential equation (ODE) to evaluate the inaccuracy boundaries of physical attribute s(6), 2) statistical testing to use prediction depending on trends of historical dat a(38).

Another essential point that DEECo focuses on is forming ensembles and exchanging knowledge. DEECo manifests those concepts and allows for using conditions that are a representation of context to form ensembles. The conditions are refined by using a Z3 SMT Solve r(103) and fitness, which filter out the less suitable compositions between all possible ones. Regarding exchanging knowledge between components, it could be determined by context constraints or by adding boundaries on gossip protocol that is responsible of spreading informatio n(36). Furthermore, DEECo allows a hierarchical composition for ensemble s (41)(40).

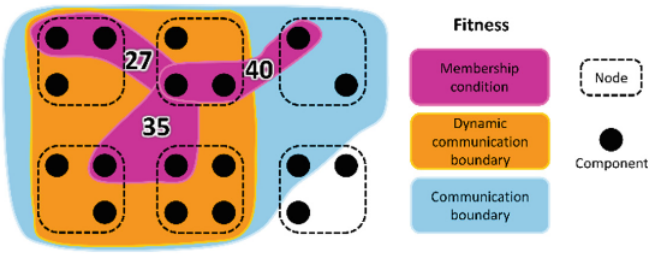

Figure 6.8: Membership vs. boundary conditions in ensemble formation

6.1.2.4 Simulation

Finally, the simulation part targets two domains for vehicles(99) and robots(114). Concerning vehicles, jDEECo has an integration with MATSim which allows to simulate vehicles and OMNET++ which simulates the network delays. Regarding the robots case, jDEECo has an integration with ROS which simulates movements of robots. While ROS has in its turn an integration with OMNET++ to simulate network delays and with Stage to simulate sensors and actuators.

To conclude, DEECo development environment provides a toolchain for modeling CPS system with an emergent behavior starting from requirement to runtime and simulation. To get the different views of system developmen,t many transformation and integration operations are involved which supports the idea of multi-paradigm modeling. These are further presented in the developed ontology of DEECo briefly presented below.

6.1.2.5 Applications

Many applications were introduced using DEECo concepts, which are in Automotive, Robotics, Clouds and Industry domains.

The Automotive example (92) is basically to present aspects of requirements, self-adaptation, networking, and simulation. The IRM tree represents finding a free parking and MATSim [7] simulates the traffic. The models involved in this example are IRM Model, DEECo Design time Model, DEECo Runtime Model, and MATSim. Furthermore, in (100), the example presents

[6] http://github.com/d3scomp/tcof
[7] https://matsim.org

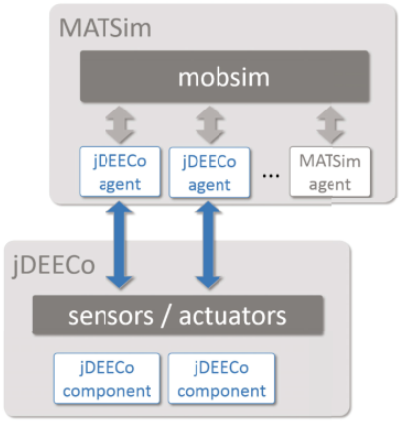

Figure 6.9: jDEECo integration with MATSim

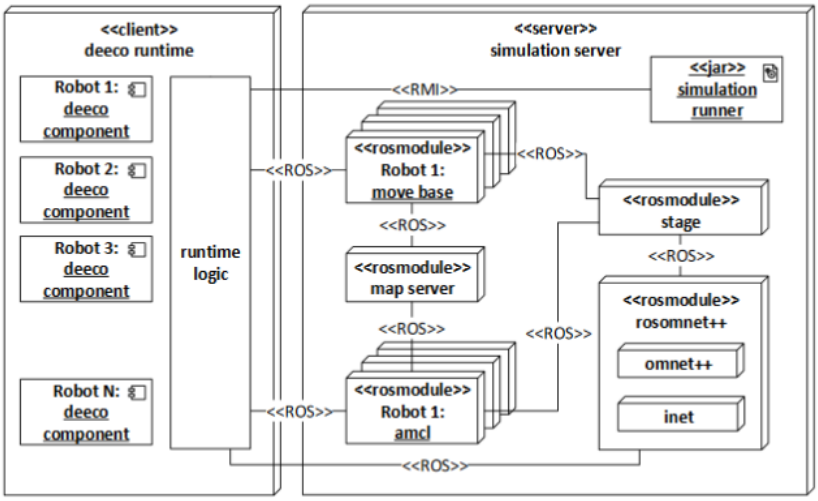

Figure 6.10: jDEECo integration with ROS

road trains that use MANETs network and utilizes the knowledge exchange by using bounded gossiping (37).

Figure 6.11: Illustration of communication groups; each is associated with an instance of SameDestination

Also, a railroad emergency response service example includes trucks helping damaged trains (e.g. because of low fuel quality) (104). The service should take into account the performance and the balance in serving between different clients. In other words, repairing the trains should be fast, but also the trucks should consider the situation in case another farther train was damaged at the same time (i.e. not only going to the closest train). Here, the development starts with defining EDL Files, compiling them, and then generating the DEECo Design time Model. At runtime, the DEECo Runtime Model integrates with Z3 SMT Solver to optimize forming the ensembles.

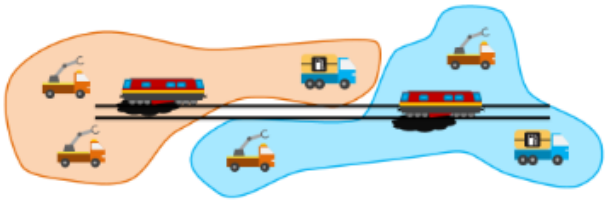

Figure 6.12: Example of two well-chosen emergency groups.

The FireFighter Coordination example is another example that represents the use of IRM-SA model and DEECo (76) (77). The example describes groups of firefighters that have to operate using low-power nodes and without communication guarantees. The work presents self-adaptation using multi-layered architecture, which are: Meta-Adaptation Layer, Adaptation Layer,and Business System. Thus, the example involves IRM-SA model, Self-Adaptation Model, DEECo Runtime Model and DEECo Design time Model.

Another example related to parking places but with edge cloud usage is presented in (42) Figure 6.13. The example describes vehicles that detect parking spots and informs other vehicles about it. The used simulation tool is Veins LTE, which is based on OMNeT++[8] (i.e. network simulator), INET[9] and SimuLTE[10] (i.e. radio simulators), and SUMO[11] (i.e. road traffic simulator). The main concern is performance model that is presented by Queueing Networks (QN), and it is planned to be integrated with DEECo.

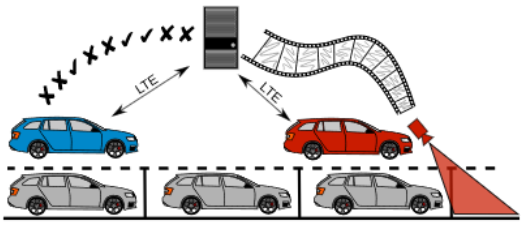

Figure 6.13: Scanning cars (on the right) are sending a photo stream to the edge cloud server reporting spot availability to the parking cars (on the left)

Regarding Clouds domain, the example concerns are performance and adaptation, and targets real-time guarantees for image processing application that runs on edge-cloud architecture [(91)]. The work is in progress and it will involve Statistical Model with Self-Adaptation Model from the current models.

As for predictability and adaptability, the use case present cleaners that detect the need for charging or cleaning, and the active mode is determined statistically to avoid premature mode-switch (39). This example involves the statistical mode-switching in Self-Adaptation Model, in

[8] https://www.omnetpp.org
[9] https://inet.omnetpp.org
[10] http://simulte.com
[11] http://sumo.dlr.de

addition to DEECo Design time Model and DEECo Runtime Model. The evaluation of statistical functions was done on STM32F4-DISCOVERY board.

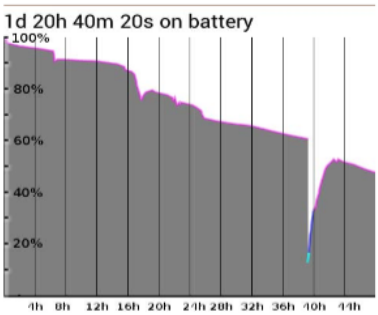

Figure 6.14: Sample battery energy level during continuous discharge

Another case is Autonomous Cleaning Robots Coordination (ACRC) (115) where robots have tasks of visiting and cleaning the offices. The problems that robots encounter here are imprecise localization, limited communication range, and latencies, which are solved by introducing self-healing in the self-adaptation logic. The testbed is done using ROS simulation for robots with camera (i.e. Stage) for navigation and IEEE 802.15.4 transceiver for communication (i.e. OMNet++). [12]

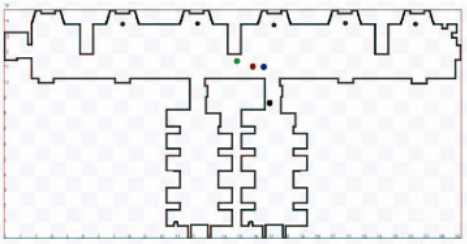

Figure 6.15: A visualization of the the area that cleaners should visit and clean

The scalability issue was targeted in an example about intelligent production line (IPL) [13] (116). The example also aims at preserving the safety conditions in the work place with taking into account the scalabe QoS grantees. To put it differently, the robots have safety zone and speed grantees, which change depending on the number of human and robots in the area. The used models are OMNeT++ and INET simulators with DEECo Design Model and DEECo Runtime Model.

The privacy and trust use case describes the exchange of the sensitive data between companies or inside the company itself (5),(4),(3) (i.e. as part of Trust 4.0 project[14]). More specifically, the system preserves information by preventing physical encounter between different teams, which is done through granting/denying access for the employees depending on the time table and existing people in the room (4). Another example is about production shifts, the foreman role has access only for employees' names in that shift. In case the system detects a possible delay of one of the employees, the system grants the foreman an access to name and phone number of a backup employee. Similarly, the confidentiality levels between companies depends on the context such as encountering incidents. These examples uses the basic concepts of DEECo (i.e. DEECo Design time Model and DEECo Runtime Model) in Scala [15]. It involves

[12] http://d3s.mff.cuni.cz/projects/components_and_services/deeco/files/seams-2016-artifact.zip
[13] https://github.com/d3scomp/scalable-reliability
[14] http://trust40.ipd.kit.edu/home/
[15] https://www.scala-lang.org/

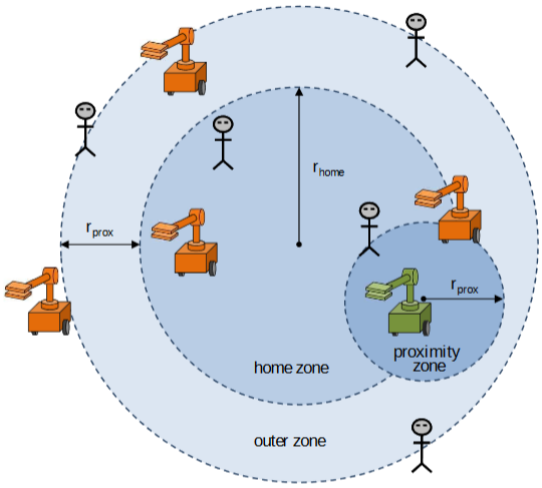

Figure 6.16: Intelligent production line: home, proximity, and outer zones

fitness model in the lunch room example (4), and ValueStreamer [16] and PCM [17] on industry 4.0 example.

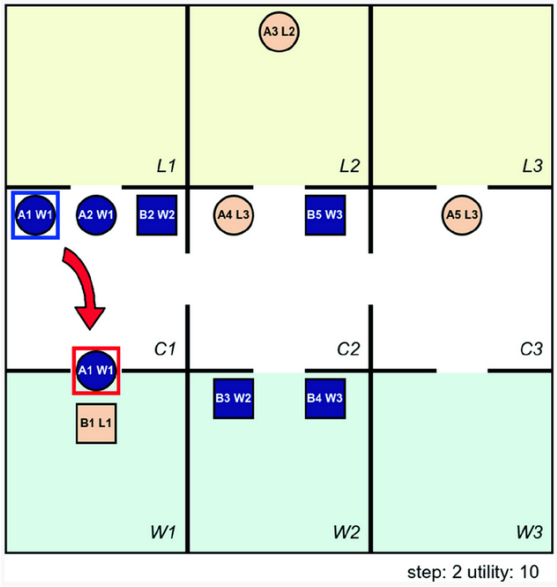

Figure 6.17: Entry for the developer A1 to room W1 is rejected due to the presence of the developer B1

6.1.3 Ontology

The structure of DEECo is described using the ontologies provided by MPM4CPS. This is done by instantiating its classes, so the created set of individuals represents DEECo.

6.1.3.1 CPS

In this part, the individuals represents the system domain and its parts, in addition to the its requirements.

- ApplicationDomainsDS: BuildingAutomation, IntelligentTransport, RoboticForService, SmartManufacturing

[16] https://www.valuestreamer.de/en/home/
[17] https://www.palladio-simulator.com/science/palladio_component_model/

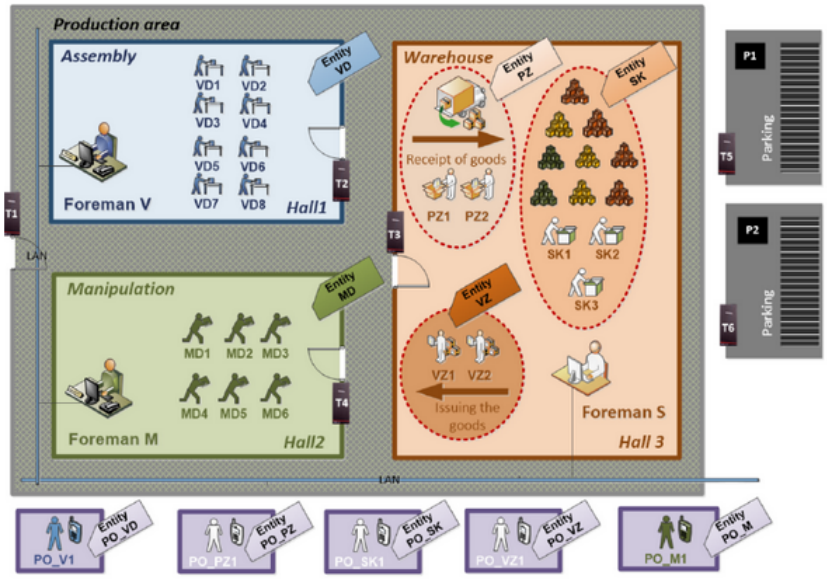

Figure 6.18: The use case illustrates a production area, which has many halls. For each hall, there is a single foreman who manages the workers.

- Architecture: Generic Architecture
- SystemDC:

  SystemPart: regarding the CommunicationProtocol, the short range communication (P2P) is used in most of the use cases (e.g. MANET, gossip protocol, ...).

  Component: entities which are modeled in CPS using DEECo are hardware/physical components. For instance, vehicle or cleaner are physical components with a need for representation of physical process dynamics (i.e. differential equations). Additionally, it is possible to model components in the environment such as human.

- QualityRequirementsDC

  Dependability: Safety, Availability

  Interoperability: Scalability

  Predictability: Accuracy

  Reliability: Robustness

  Security: Confidentiality

  Sustainability: Adaptability, Efficiency (i.e. performance), Resilience

6.1.3.2 MPM

In this part, the individuals includes a DEECo megamodel and its fragments defining all model operations supported by DEECo as outlined in Figure 6.2 and Figure 6.20. Model operations are of different kinds such as transformation and integrations operations. The first model operation is the capture of requirements followed by a transformation operation from the requirements model into a design model, then by an instantiation operation to create runtime model and simulation model. The runtime model integrates with the self-adaptation model Figure 6.19.

Formalism, Models, and Tools

- Formalism

  AutomataBasedFormalism: LTS (Labeled Transition System)

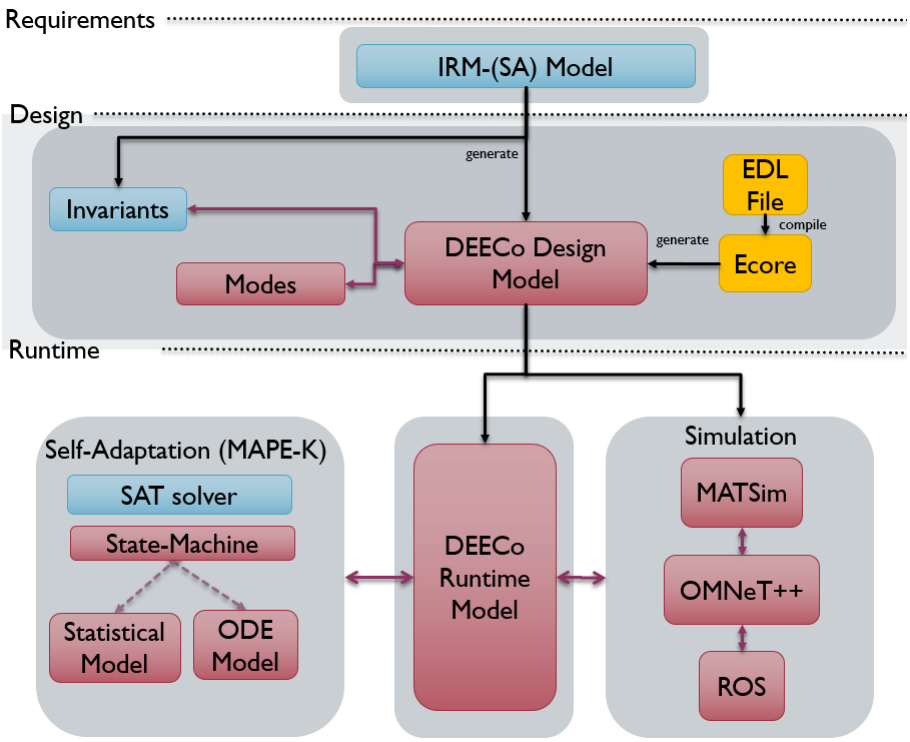

Figure 6.19: Models and Tools Transformations and integrations

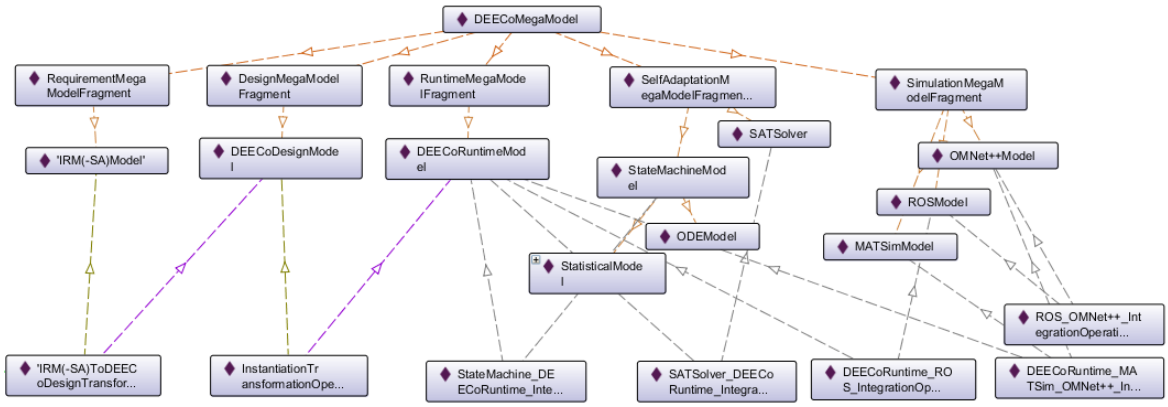

Figure 6.20: The example of DEECo represented by OntoGraf using Protoge

- Language

  FormalLanguage: SCEL (Service Component Ensemble Language)

  ArchitecturalDescriptionLanguage: DEECo realizes SCEL

  ModelingLanguage: IRM, IRM-SA

- Model

  MegaModel: DEECo (DEECo stands for Dependable Emergent Ensembles of Components)

  MegaModelFragment: *mentioned below in details*

- ModelRelation: *mentioned below in details*

- Tool

ModelingTool: IRM-SA (Invariant Refinement Method - Self-Adaptation) developed in D3S Charles University in Prague, EclipseEpsilon

RuntimeTool: jDEECo from Department of Distributed and Dependable Systems, Charles University in Prague

SimulationTool: ROS from Open Source Robotics Foundation, OMNet++ from OpenSim Ltd, MATSim from MATSim Community, Stage from Richard Vaughan and contributors 1998-2011 Part of the Player Project

MegaModel Fragments / Models for DEECo

- Requirements MegaModel Fragment

  Models: IRM Model

- Design MegaModel Fragment

  Models: DEECo Design Model

- Self-Adaptation MegaModel Fragment

  Models: IRM-SA Model, Mode-Switching Model

- Runtime MegaModel Fragment

  Models: DEECo Runtime Model

- Simulation MegaModel Fragment

  Models: MATSim, ROS, OMNET++, Stage

Tools / Model Operations for DEECo

- Model Operation: Requirements Capturing Operation - Modeling requirements refinements by Human

  Input: Human

  Output Model(s): IRM-SA Model

- Model Operation: Requirements-Design Transformation Operation - Capturing requirements refinements in design time

  Input Model(s): IRM-SA Model

  Output Model(s): DEECo Design Model

- Model Operation: Instantiation-Transformation Operation - Instantiation of the DEECo components and ensembles

  Input Model(s): DEECo Design Model

  Output Model(s): DEECo Runtime Model

- Model Operation: Instantiation-Transformation Operation2 - Instantiation of the Simulation Model

  Input Model(s): DEECo Design Model

  Output Model(s): ROS, MATSim , OMNet++

- Model Operation: Runtime-to-Vehicle Simulation Integration Operation - System Design Validation

  Input Model(s): DEECo Runtime Model

  Output Model(s): MATSim , OMNet++

- Model Operation: Runtime-to-Robot Simulation Integration Operation - System Design Validation

  Input Model(s): DEECo Runtime Model

  Output Model(s): ROS

- Model Operation: Simulation-to-Simulation Integration Operation - System Design Validation

  Input Model(s): ROS

  Output Model(s): Stage , OMNet++

Development Process

The development process starts with capturing requirements operation in which a human is responsible of defining the IRM-SA model. Afterwards, the IRM tree is transformed to DEECo Design Model, which its instantiation are DEECo Runtime Model and Simulation Models.

The DEECo Runtime Model is integrated with self-Adaptation model. It is possible also to define modes and their transitions in DEECo Design time model. Hence, the logic over transitions are extensible with ODE or statistical reasoning. Similarly, it is possible to define in the DEECo Design Model formation of ensembles by using gossiping boundary and fitness. The boundary defines limits on gossiping protocol and the fitness evaluates the history of forming ensembles and select the best fitting group of components that are involved to form the ensemble.

Finally, DEECo Runtime Model is also integrated with simulation models. For more information, please check the following links:

- DEECO website: http://d3s.mff.cuni.cz/software/deeco/
- jDEECo: https://github.com/d3scomp/JDEECo/tree/simulation
- IRM: http://d3s.mff.cuni.cz/software/irm/
- Modes: https://github.com/d3scomp/JDEECo/tree/master/jdeeco-modes
- Statistical Operations: https://github.com/d3scomp/TimeSeriesStatistics
- OMNet++: http://d3s.mff.cuni.cz/projects/components_and_services/deeco/files/jDEECo-OMNeT.zip
- MATSim: http://d3s.mff.cuni.cz/projects/components_and_services/deeco/files/jDEECo-MATSim.zip
- ROS: http://d3s.mff.cuni.cz/projects/components_and_services/deeco/files/seams-2016-artifact.zip
- Intelligent Ensemble framework: http://d3s.mff.cuni.cz/software/deeco/files/seams-2017.zip

6.1.3.3 MPM4CPS

In this part, the views and viewpoints for this system are described as following:

- View: Requirement Designer
- ViewPoint: The concerns in this viewpoint is related to resilience, and involves both roles and ensembles in the system. The designer determines the environmental assumptions to each choice to ensure preserving the required invariants. That makes this viewpoint related to the cyber part of the system (i.e. software).
- View: Component Designer

- ViewPoint: The concerns in this viewpoint is related to performances, accuracy, robustness and self-adaptation. It involves designing roles, processes and modes, in addition to relations between processes and the assumptions from requirement phase. Also, components contain controllers, sensors, and actuators, which means that this viewpoint is related to physical and cyber parts of the system.
- View: Ensemble Designer
- ViewPoint: The concerns in this viewpoint are related to knowledge exchange, security, optimization, scalability and performances. The designer should determine the roles involved in ensembles, the context constraints and what information to exchange. Additionally, the designer can optimize forming the ensembles by using fitting function and defining a boundary over gossip protocol. The design of ensembles could be also associated to system requirements as it is mentioned in the Requirement Designer view. This makes this viewpoint related to network and cyber (i.e. software) parts of the system.

6.2 HPI CPSLab[18]

To structure the presentation of this case study according to the efforts for the MPM4CPS project, we will at first in Section 6.2.1 provide an overview about the case study and lab, then in Section 6.2.2 review the technical setting and derive the required needs concerning the CPS ontology, thereafter in Section 6.2.3 we will outline how the models, tools, and tool chain employed in the case study can be captured as multi-paradigm modeling and derive the required needs concerning the MPM ontology, and finally in Section 6.2.4 we discuss how the CPS character of the case study is reflected in its multi-paradigm modeling and the use of the MPM4CPS ontology.

6.2.1 Overview

As outlined in more details from a conceptual point of view in (139), the presented *CPSLab*[19] at the Hasso Plattner Institute (HPI)[20] at the University of Potsdam applied, adapted, and evaluated an existing industrial-strength development methodology from the automotive domain (33) (see also Figure 6.21) for the robotic system application domain. We therefore evaluated and adapted a component-based approach using an MDE approach supporting the combination of soft and hard real-time behavior.

The resulting methodology for robotic systems supports several development activities such as modeling, simulation, verification/testing at different stages, prototyping and pre-production. The lab supports tools and related libraries in an integrated tool-chain that reflects physical and cyber aspects of distributed robotics systems.

We consider a robot system as depicted in Figure 6.22, where a single robot has the duty to transport pucks as advised by the overall factory automation. The regular behavior of the robot is to move around, transport pucks, or charge its batteries. The behavior must meet strict constraints, such as preventing complete discharge of the batteries and with a lower priority, ensure to transport pucks as requested. It must also perform reasonably well with respect to some soft goals to minimize energy consumption and maximize throughput.

Figure 6.23 depicts a structural overview of the robot system. The whole cyber-physical evaluation scenario consists of four different rooms. In the first room, the pucks are packed and dropped for transportation in area A_P. A robot R_P transports the puck to a second room and drops it within the sorting area A_S. Based on the current delivery status, the robot R_S chooses

[18]Acknowledgements: We thank Sebastian Wätzoldt, Stefan Neumann, Joachim Hänsel, and Falk Benke for their contributions to the lab described in this section and their contribution to the presented content and figures.

[19]http://www.cpslab.de

[20]http://www.hpi.de

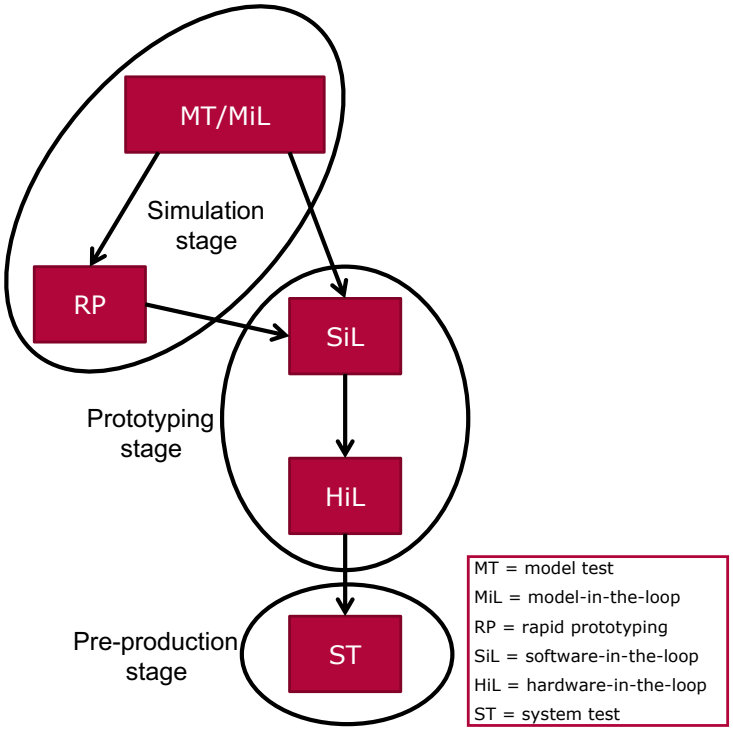

Figure 6.21: Overview of the methodology for modeling, verification, and validation employing simulation and testing (see (33))

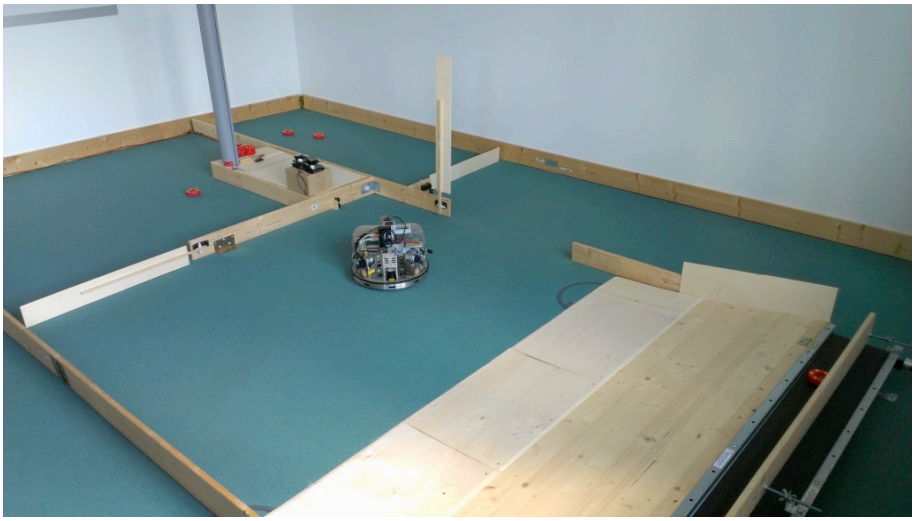

Figure 6.22: Photo of the lab (see (139))

one of the two booths and a band-conveyor transports the puck to the customer or stock delivery area (A_{CD}, A_{SD}) afterwards. In a third step, the robot R_{St} transfers the puck to stock in St. The doors can be opened or closed dynamically to vary the scenario. A robot can charge its battery at one of the two charging points. Each robot acts as an autonomous unit. Therefore, the tasks transportation, sorting and stocking are independent from each other.

For the evaluation of our research activities, we use our CPSLab robot laboratory consisting of three Robotino robots (see Figure 6.24). The robots can be equipped with several sensors (e.g., laser scanner, infrared (IR) distance sensors, GPS-like indoor navigation systems) as well as different actuators (e.g., servo motors, omnidirectional drive, gripper).

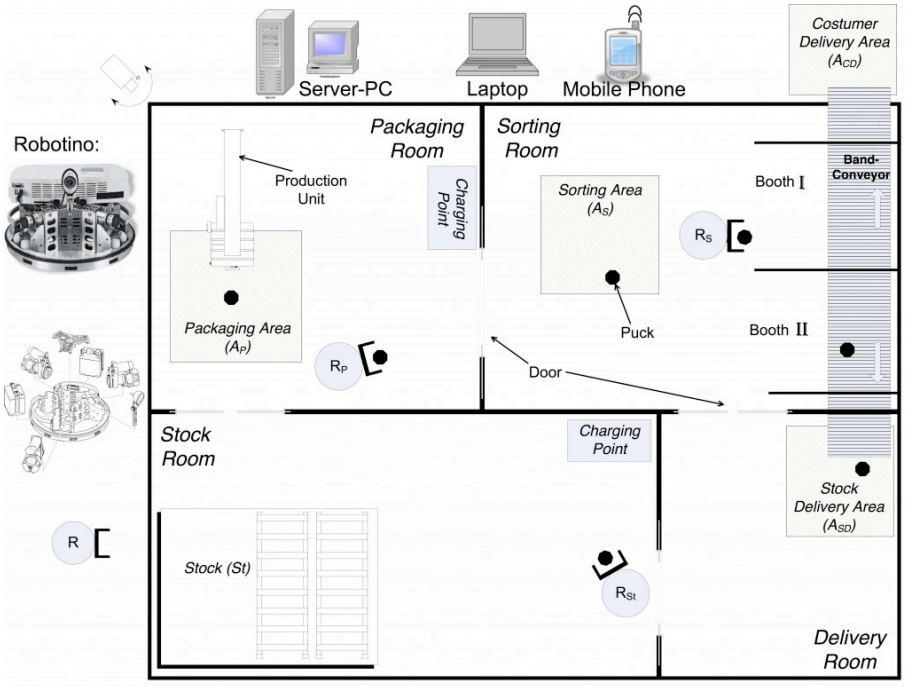

Figure 6.23: Structural overview of the employed evaluation scenario (see (139))

Figure 6.24: Photo of the employed robots (see (139))

The general idea of our evaluation scenario is the realization of a variable production setting, where robots can transport small pucks (representing goods in a production system) to different locations. The robots must fulfill different requirements, e.g., they must provide basic functionality like moving and avoiding obstacles in hard real-time (reacting on obstacles within a few milliseconds). Further, the robots must achieve high level goals, e.g., energy saving of the battery, short routing to the destination points and optimizing the throughput while transporting the pucks. While basic functionalities, such as obstacle avoidance,must be realized in hard real-time, we use existing libraries to realize higher functionalities such as path planning or creating a map by evaluating measured distance values. The latter can rarely be realized under hard real-time constraints because of insufficient libraries. Furthermore, we run a RTAI Linux operating system on the robot to enable hard real-time execution.

6.2.2 CPS

In the following, we will use the details of the different development steps to outline how the development is linked to CPS and the concepts of the CPS ontology introduced in Chapter 3.

6.2.2.1 Simulation Stage

The first step of the development process of Figure 6.21 is a simulation stage that focus on the model development resp. functional development for the employed control laws. At this stage, many details resulting from the physical and cyber parts of the system are ignore resp. simplified such as real sensor values with noise, specific effects of scheduling, the impact of communication interaction and messages, and timing/memory/computation constraints.

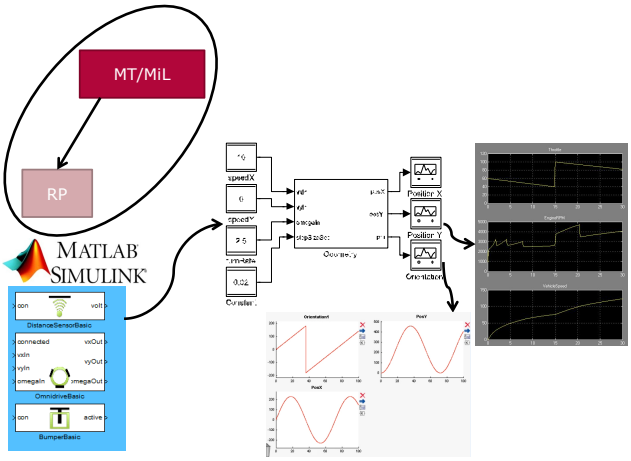

Figure 6.25: Overview of the model test in the simulation stage of (33)

Model Test

In a first activity named *model test* (MT) (see Figure 6.25), a so-called one-way/one-shot simulation with MATLAB/Simulink is supported where the model of the control behavior can be stimulated by inputs to see that they react properly. This provides some confidence that the setup control behavior works as intended.

Model Test - CPS Ontology

In the model test as outlined in Figure 6.25, the abstract control algorithm from the cyber domain for a specific function is confronted with the physics as present in the input data plus expected outcomes relevant for the function and thus we have a very simple cyber-physical setting. Thus we cover the elements CyberPart from the CPS ontology for a particular function.

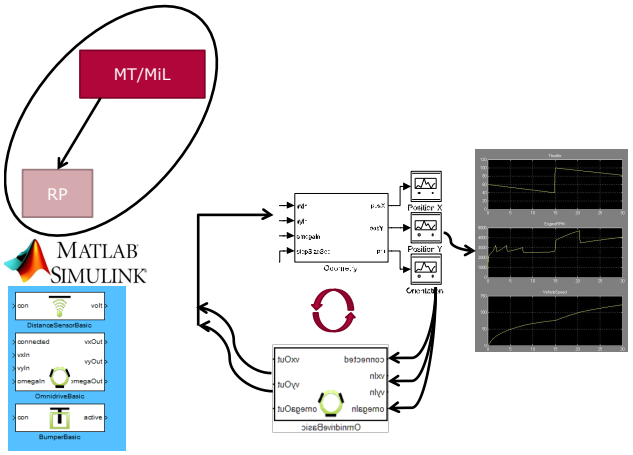

Figure 6.26: Overview of the model in loop simulation in the simulation stage of (33)

Model-in-the-Loop

In a second step, the model of the control behavior is combined with a MATLAB/Simulink model of the plant by means of a *model-in-the-loop* (MiL) simulation as shown in Figure 6.26, which uses the feedback provided by the plant model to evaluate that the control behavior is as expected.

Model-in-the-Loop - CPS Ontology

The model in the loop depicted in Figure 6.26 in contrast, the abstract control algorithm from the cyber domain is combined with the idealized physics as present in the plant model and thus we have a simple cyber-physical setting. Here we cover thus the elements CyberPart and PhysicalPart from the CPS ontology for a particular function.

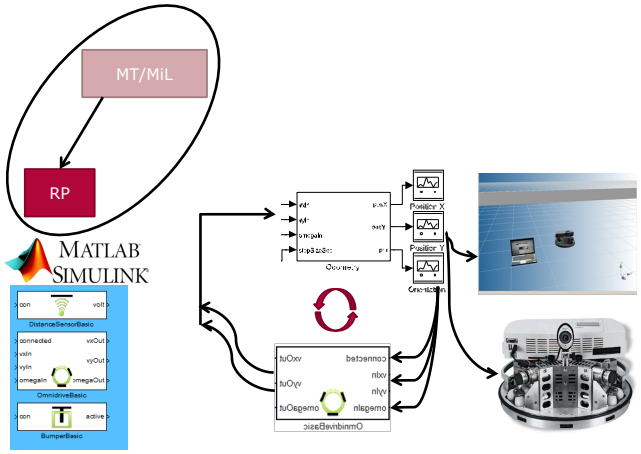

Figure 6.27: Overview of the rapid prototyping in the simulation stage of (33)

Rapid Prototyping

As the validity of plant models is often only rather limited when it comes to sophisticated aspects of the physical behavior, as an additional step *rapid prototyping* as depicted in Figure 6.27 is supported. For smaller control behavior, the model of the control behavior is linked to the real robot such that real sensor values with noise and timing constraints of the environment and platform can be covered. However, specific effects of scheduling, the impact of communication interaction and messages, and memory/computation constraints remain uncovered. For larger scenarios and for the multi robot scenarios a link to a real hardware setup is not feasible here. Instead we employ a model-in-the-loop (MiL) simulation where a complex environment and the communication between the robots can be explored. While this covers the impact of communication interaction and messages, other aspects like real sensor values with noise, specific effects of scheduling and timing/memory/computation constraints are, however, not covered.

Rapid Prototyping - CPS Ontology

The rapid prototyping against the robot as depicted in Figure 6.27, the abstract control algorithm from the cyber domain is brought together with the real physics of the robot and thus we have clearly a cyber-physical setting.

Our rapid prototyping based on a sophisticated robot simulator, again the abstract control algorithm from the cyber domain is brought together with the physics as present in the sophisticated robot model of the simulator and thus we have clearly a cyber-physical setting.

In both cases we cover thus the elements CyberPart from the CPS ontology for a particular function and the elements PhysicalPart from the CPS ontology for the part of the robot relevant for a particular function that is either simulated or considered directly.

6.2.2.2 Prototyping Stage

The second stage supported is the prototyping stage where the focus changes from models to their implementation in software or hardware and where besides the individual functions also the system architecture is covered. Due to this refined view, in particular discretization effects of the cyber part that are absent in the abstract mathematical models employed in the former stage now become visible. At this stage, less details are ignore resp. simplified as step by step specific effects of scheduling, the impact of communication interaction and messages, and timing/memory/computation constraints are taken into account.

More Detailed Modeling

To consider the more detailed view outlined, at the prototyping stage the models must be refined such that besides the individual functions also the system architecture is defined.

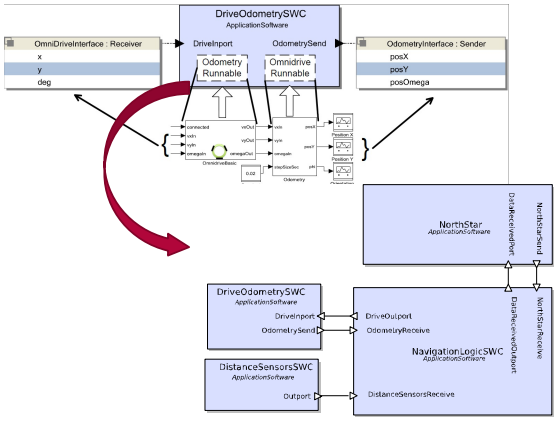

Figure 6.28: Overview of the definition of the software architecture in the prototyping stage of (33)

As depicted in Figure 6.28, this is done by first defining components and their communication via port types, messages, interfaces, and data types with AUTOSAR and map the beforehand considered functional parts on them. In this step, we also have to map the functionality extending the existing models and where necessary add custom implementation files. In a second step, we then define the overall architecture using AUTOSAR including besides the components and their communication also task specification and the hardware configuration.

As depicted in Figure 6.29, an important element of this refinement is also real-time constraints, e.g. to guarantee safety constraints. A combination of hard and soft real-time aspects at functional as well as architectural levels must be defined including a mapping to hard and soft real-time task with proper levels for the priorities.

Concerning the verification, we employ code generation at the prototyping stage and try to step by step add more and more details of the software and hardware to the picture in the following steps.

Software in the Loop (SiL)

The *software-in-the-loop* (SiL) simulation at the prototyping stage as depicted in Figure 6.30 requires that code generation is employed to derive code for the functional models and architectural models. In special cases, also additional manually developed code has to be integrated. Then, the code is executed and run against the available simulation of the robot and its environment.

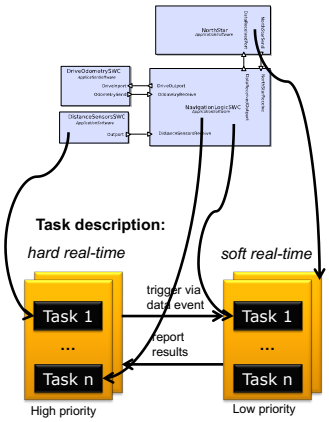

Figure 6.29: Overview of the mapping of the architecture to tasks and communication in the prototyping stage of (33)

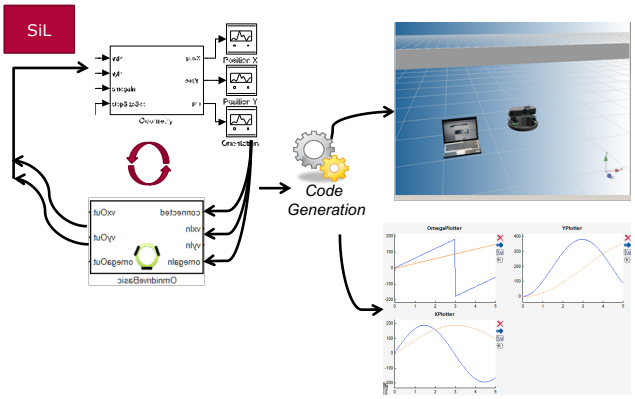

Figure 6.30: Overview of software-in-the-loop (sil) simulation in the prototyping stage of (33)

As we still do not always use the real hardware, we still ignore resp. simplify elements such as real sensor values with noise and not by the simulator covered timing constraints of the environment or platform, while specific effects of scheduling, the impact of communication interaction and messages, and by the simulator covered timing constraints of the environment or platform, and timing/memory/computation constraints of the software.

Software in the Loop (SiL) - CPS Ontology

The first form of software in the loop (SiL) executing the software on a desktop computer against a simulator features that the detailed control algorithm from the cyber domain is brought together with the physics as present in the sophisticated robot model of the simulator and thus we have clearly a cyber-physical setting.

The second form of software in the loop (SiL) executing the software on a desktop computer against a remote controlled robot ensures that the detailed control algorithm from the cyber domain is brought together with the physics as present in the remote controlled robot and thus we have clearly a cyber-physical setting.

In both cases we cover thus the elements CyberPart from the CPS ontology for the combination of all function and the elements PhysicalPart from the CPS ontology for the whole robot.

Hardware in the Loop (HiL)

By moving on to the lab itself, we can then also consider a *hardware-in-the-loop* (HiL) simulation at the prototyping stage as sketched in Figure 6.31, where besides the software that is

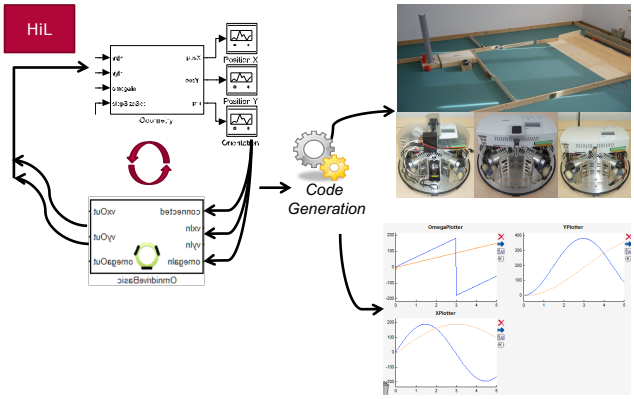

Figure 6.31: Overview of hardware-in-the-loop (HiL) testing in the prototyping stage of (33)

generated or integrated also the specific characteristics of the robot hardware and lab environment and its hardware can be experienced.

As we now employ the real hardware, we no longer ignore resp. simplify elements. Therefore, now real sensor values with noise, specific effects of scheduling, the impact of communication interaction and messages, and timing/memory/computation constraints are all considered.

Hardware in the Loop (HiL) - CPS Ontology

The hardware in the loop (HiL) executing the software on the robot depicted in Figure 6.31 ensures that the detailed control algorithm from the cyber domain is brought together with the physics as present in the robot and thus we have clearly a cyber-physical setting.

Thus we cover the elements CyberPart from the CPS ontology for the combination of all function and the elements PhysicalPart from the CPS ontology for the whole robot.

6.2.2.3 Pre-Production Stage

In our specific setting and due to our focus on the software development, the *system test* in the pre-production stage is not really different as we do not produce any system we want to sell later. In a commercial setting the robots in the prototyping, the robots in the lab would likely be equipped with more testing hardware or prototypical hardware, while in our lab only one level exists here.

The outlined methodology and tool chain adjusted from the automotive domain, provides suitable guidance due to the different focus in stages and follows where possible an MDE approach where tools and libraries are integrated such that the models drive the development. Only later the code and configuration data automatically generated from the models are employed to consider more details concerning the verification, simulation, and testing. We put special focus on supporting also hard and soft real-time considerations, which are oftentimes ignored in robotic development scenarios.

6.2.3 MPM

For the MPM ontology introduced in Chapter 4, we will in the following use the details of the different development steps to outline how the development is linked to MPM.

6.2.3.1 Overview

In Figure 6.32 the tool landscape for developing the aforementioned robot CPS is depicted. It consists of MATLAB/Simulink for modeling and simulation, dSPACE SystemDesk for modeling software architecture, hardware configuration, and task mapping, dSPACE TargetLink for code generation and the FESTO Robotino-Library with the FESTO Robotino©Sim simulator.

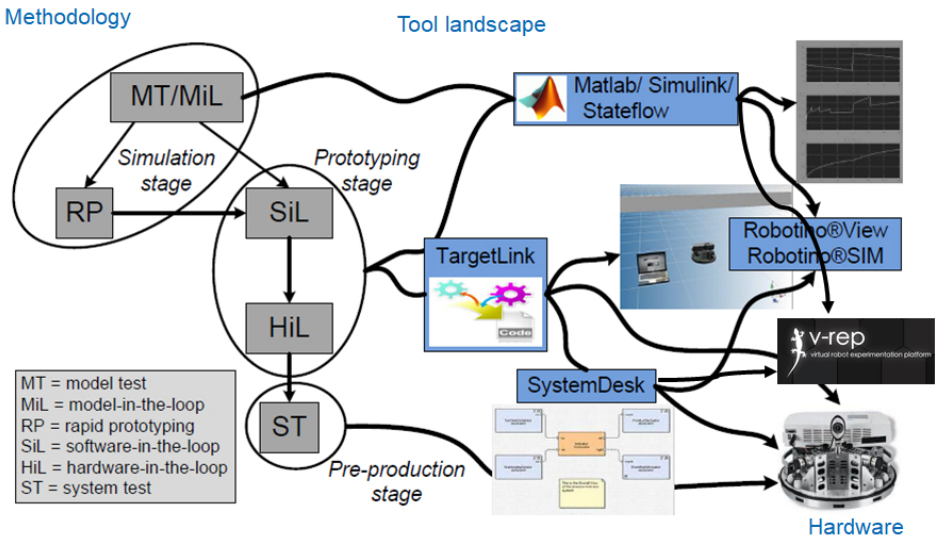

Figure 6.32: Tool landscape and its relation to the development methdology

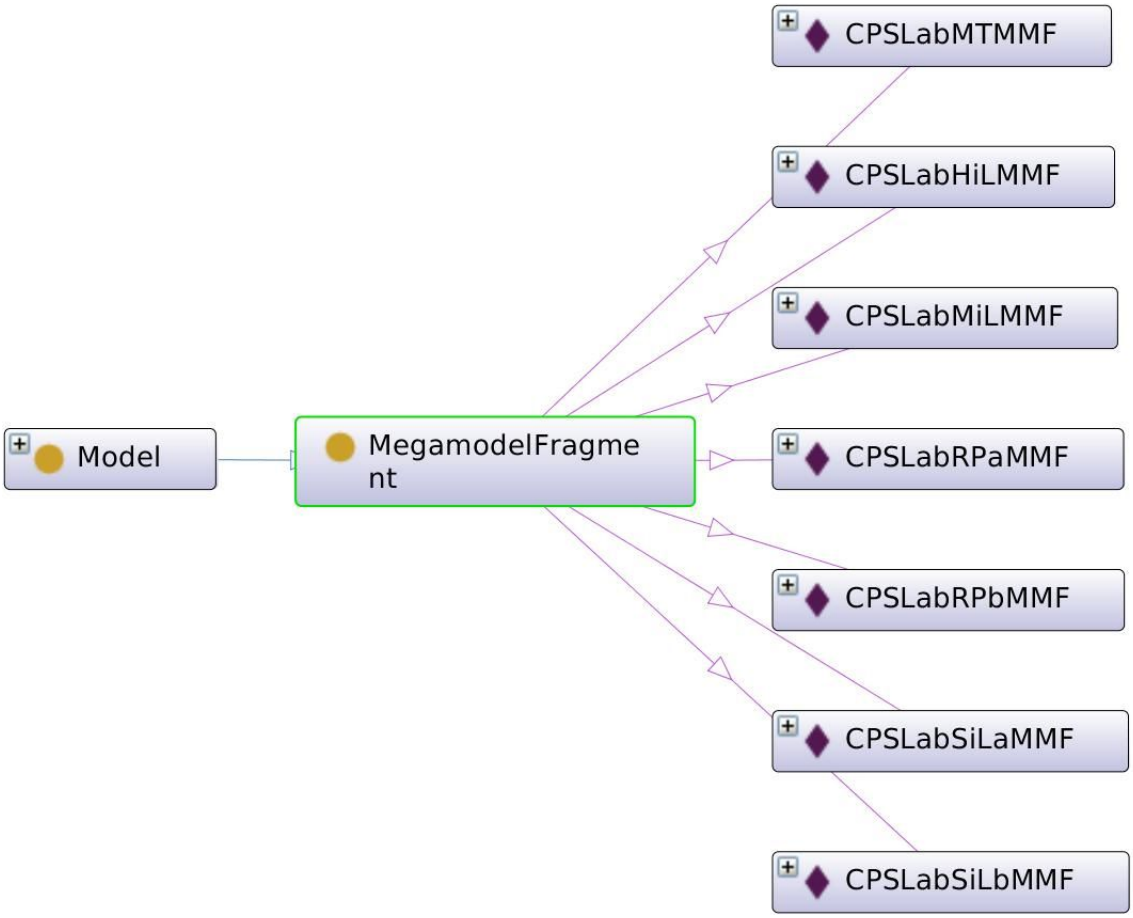

Figure 6.33: All MegaModelFragments of the CPSLabMM MegaModel

On overview about the stages and activities with an emphasis on models, tools, and multi-paradigm modeling is depicted in Figure 6.34.

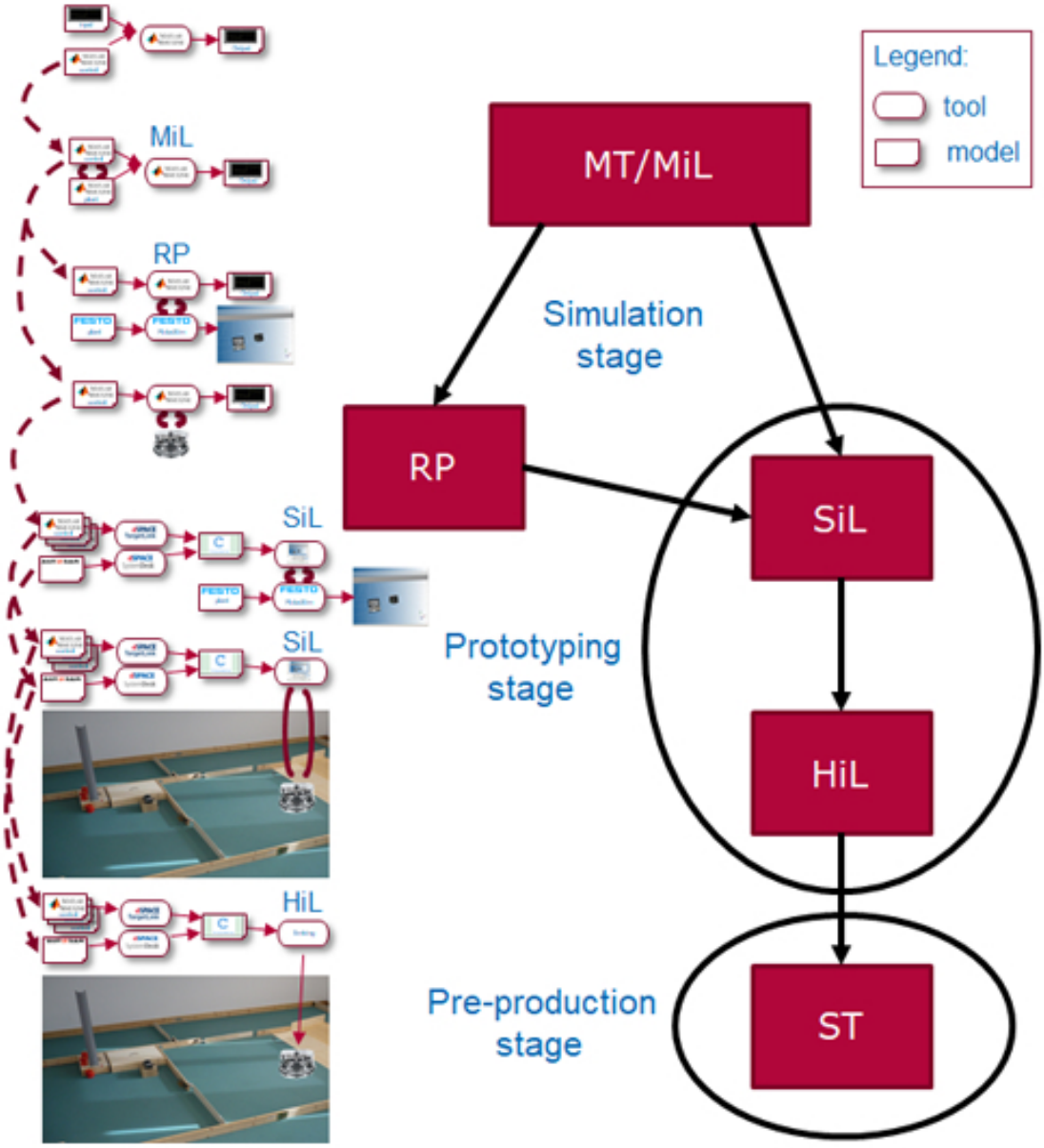

Figure 6.34: Overview over the megamodel fragments of the CPSLab megamodel and how the models are related (dashed arrows)

In the following, we outline to which elements of the MPM ontology of Chapter 4 the elements of the megamodel refer and which megamodel fragements cover the scenarios introduced in the last subsection.

Formalism, Languages, Models, and Tools

- Used Language and Models
  - MATLAB/Simulink Language: ControlModel, PlantModel
  - FESTO Robotino©Sim Language: RobotModel
  - AUTOSAR Language: SystemModel
- MegaModel
  - MegaModel: CPSLabMM
  - MegaModelFragments: CPSLabMTMMF, CPSLabMiLMMF, CPSLabRPaMMF, CPSLabRPbMMF, CPSLabSiLaMMF, CPSLabSiLbMMF, CPSLabHiLMMF
  - ModelRealtions: (see detailed definition of the megamodel fragments)
- Tool
  - SimulationTool: MATLAB/Stateflow Simulator
  - TransformationTool: dSPACE TargetLink
  - ModelingTool: dSPACE SystemDesk
  - SimulationTool: FESTO Robotino©Sim
  - VisualizationTool: FESTO Robotino©View
  - ExecutionTool: Execution on a Desktop computer
  - ExecutionTool: Remote execution on a Robotino Robot
  - ExecutionTool: Local execution on a Robotino Robot

The added elements for the CPSLab ontology outlined in the text are depicted also in Figure 6.33.

6.2.3.2 Simulation Stage

In the simulation stage we have two development activities: model test and model-in-the loop. In the following, we will outline how they can be captured with mega-model fragments consisting of model instances and tool applications.

Model Test

The model test introduced in Figure 6.25, is rather trivial as it only employ a single model of the planned control algorithm plus some auxiliary models for test inputs. Then, as depicted in Figure 6.35 the model of the control algorithm is simulated by employing the test inputs.

Model Test - MPM Ontology

MegaModel Fragment CPSLabMTMMF

- MegaModelFragment: CPSLabMTMMF
  - Model(s): ControlModel

Tools / Models Operations of CPSLabMiLMMF

- ModelOperation: One Shot Simulation
  - Input Model(s): ControlModel, Input data (entered with MATLAB/Stateflow Simulator)
  - Output Model(s): Output data (visualized with MATLAB/Stateflow Simulator)

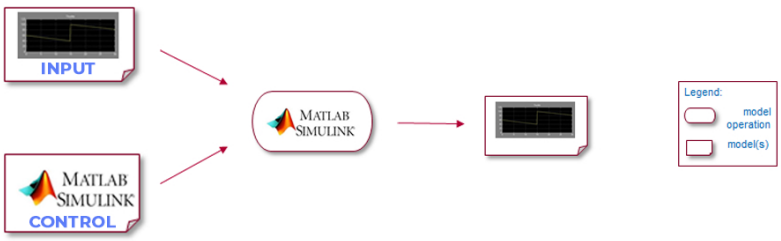

Figure 6.35: Model Test

- Employed Tool: MATLAB/Stateflow Simulator

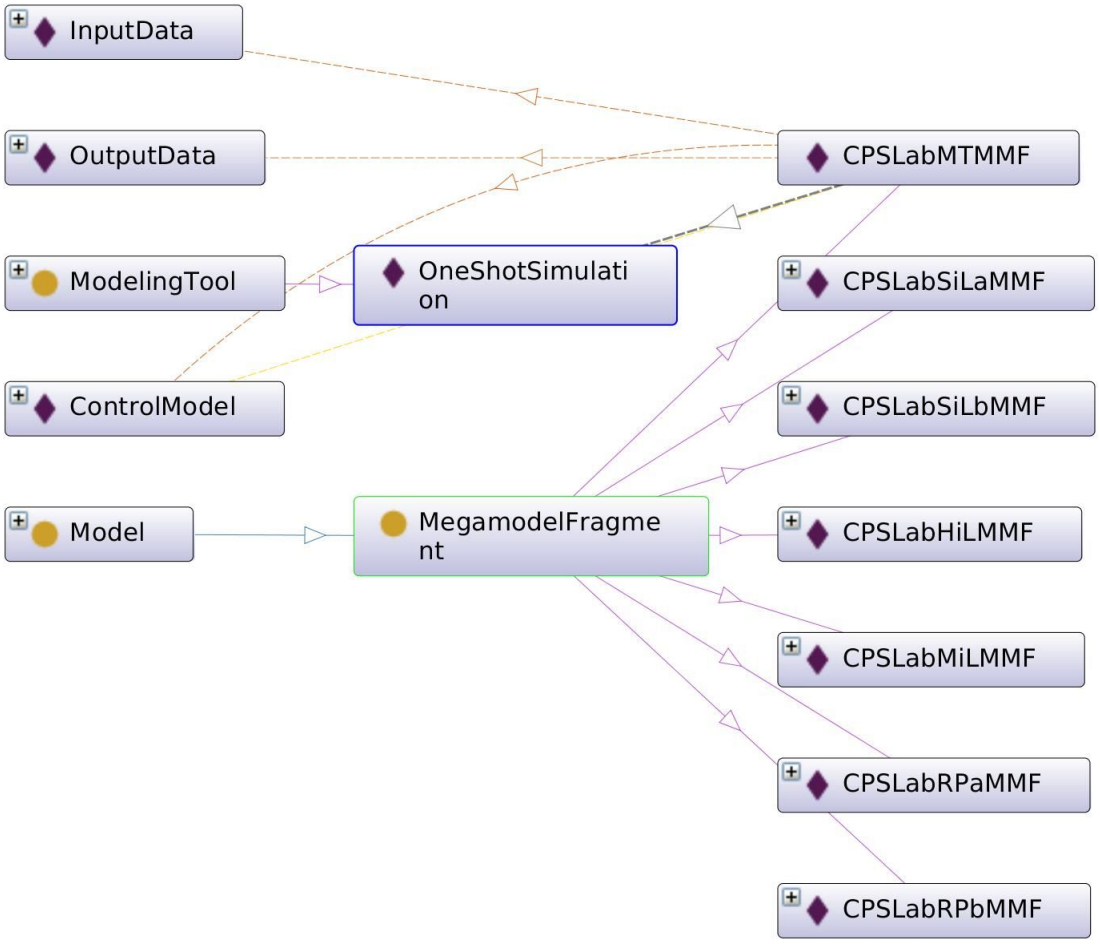

Figure 6.36: Part of the ontology for the MegaModelFragment CPSLabMTMMF covering Model Test

In Figure 6.36, the added MegaModel Fragment CPSLabMTMMF and its elements for the CPSLab ontology outlined in the text are presented.

Model-in-the-Loop

In contrast to model test, model-in-the-loop simulation as introduced in Figure 6.26 employed besides a model of the control algorithm also a model of the plant and uses as depicted in Figure 6.37 simulation to explore how well both fit together.

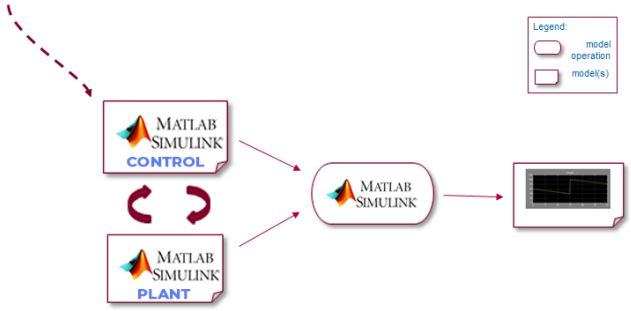

Figure 6.37: Model in the Loop

Model-in-the-Loop - MPM Ontology

MegaModel Fragment CPSLabMiLMMF

- MegaModelFragment: CPSLabMiLMMF
  - Model(s): ControlModel, PlantModel

Tools / Models Operations of CPSLabMiLMMF

- ModelOperation: Model-in-the-Loop Simulation
  - Input Model(s): ControlModel, PlantModel
  - Output Model(s): Output data (visualized with MATLAB/Stateflow Simulator)
  - Employed Tool: MATLAB/Stateflow Simulator

The added MegaModel Fragment CPSLabMiLMMF and its elements for the CPSLab ontology outlined in the text are depicted also in Figure 6.38.

Rapid Prototyping

The rapid prototyping as introduced in Figure 6.27 is supported in two forms. At first rapid prototyping can be done employing a sophisticated simulator for the robot as depicted in Figure 6.39. While not necessary exposing the control algorithm to physical reality as far as captured by the sophisticated model of the robot, the simulator already capture much more details than the plan model while still allow to analyze the behavior much easier then in the case of using the real robot.

The second case depicted in Figure 6.40 connects the abstract control algorithm with the real robot and therefore expose the algorithm to all physical effects. However, analysis might be difficult as running the algorithm against the robot is less easy to analyze then when running it against a simulator.

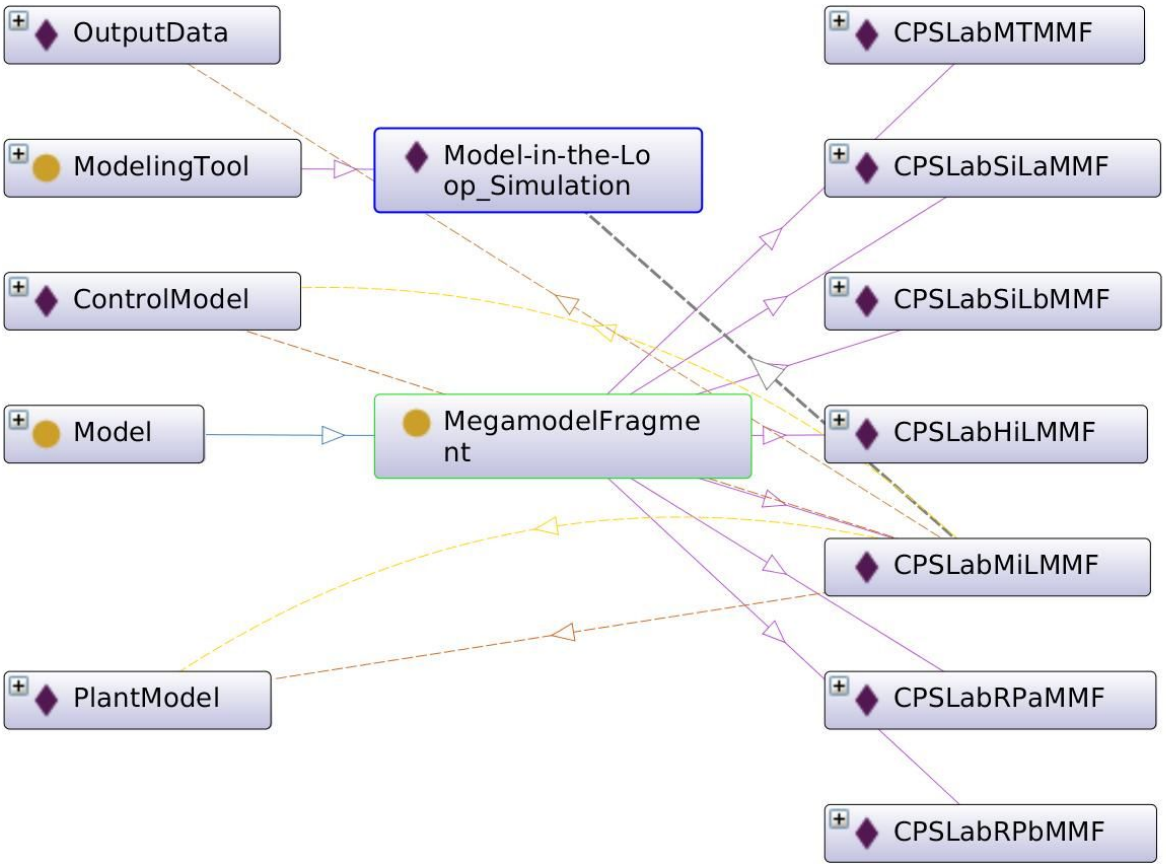

Figure 6.38: Part of the ontology for the MegaModelFragment CPSLabMiLMMF covering Model-in-the-Loop (MiL)

Figure 6.39: Rapid Prototyping (RP) with a detailed robot simulation

Rapid Prototyping - MPM Ontology

MegaModel Fragment CPSLabRPaMMF

- MegaModelFragment: CPSLabRPaMMF
  - Model(s): ControlModel, RobotModel

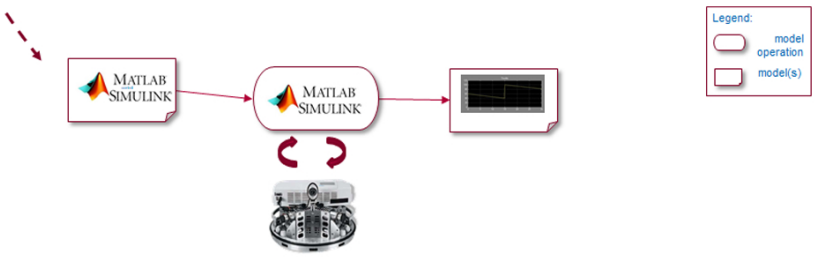

Figure 6.40: Rapid Prototyping (RP) with a remote controlled robot

Tools / Models Operations of CPSLabRPaMMF

- ModelOperation: Rapid Prototyping with Robot Simulation
  - Input Model(s): ControlModel, RobotModel
  - Output Model(s): Output data (visualized with MATLAB/Stateflow Simulator and/or FESTO Robotino©View)
  - Employed Tool: MATLAB/Stateflow Simulator, FESTO Robotino©Sim

In Figure 6.41, the added MegaModel Fragment CPSLabRPaMMF and its elements for the CPSLab ontology outlined in the text are presented.

MegaModel Fragment CPSLabRPbMMF

- MegaModelFragment: CPSLabRPbMMF
  - Model(s): ControlModel

Tools / Models Operations of CPSLabRPbMMF

- ModelOperation: Rapid Prototyping with Robot Execution
  - Input Model(s): ControlModel
  - Output Model(s): Output data (visualized with MATLAB/Stateflow Simulator and/or observed)
  - Employed Tool: MATLAB/Stateflow Simulator, Remote execution on a Robotino Robot

The added MegaModel Fragment CPSLabRPbMMF and its elements for the CPSLab ontology outlined in the text are depicted also in Figure 6.42.

Simulation Stage - MPM Ontology

Some issues are no yet covered by the MPM ontology and the employed megamodel and megamodel fragments: While the same types of models and tools are employed at several stages and activities as visible in the megamodel depicted in Figure 6.34, the models developed for each of these activities are quite different in the simulation stage. For the model test, only simply MATLAB Simulink models with the standard block set and input signals are usually employed. For the model-in-the-loop simulation, both the model of the control behavior and of the related fragment of the plant are modeled and evaluated using MATLAB/Simulink models with the standard block set. To link the behavior to the FESTO Robotino©Sim Simulator and

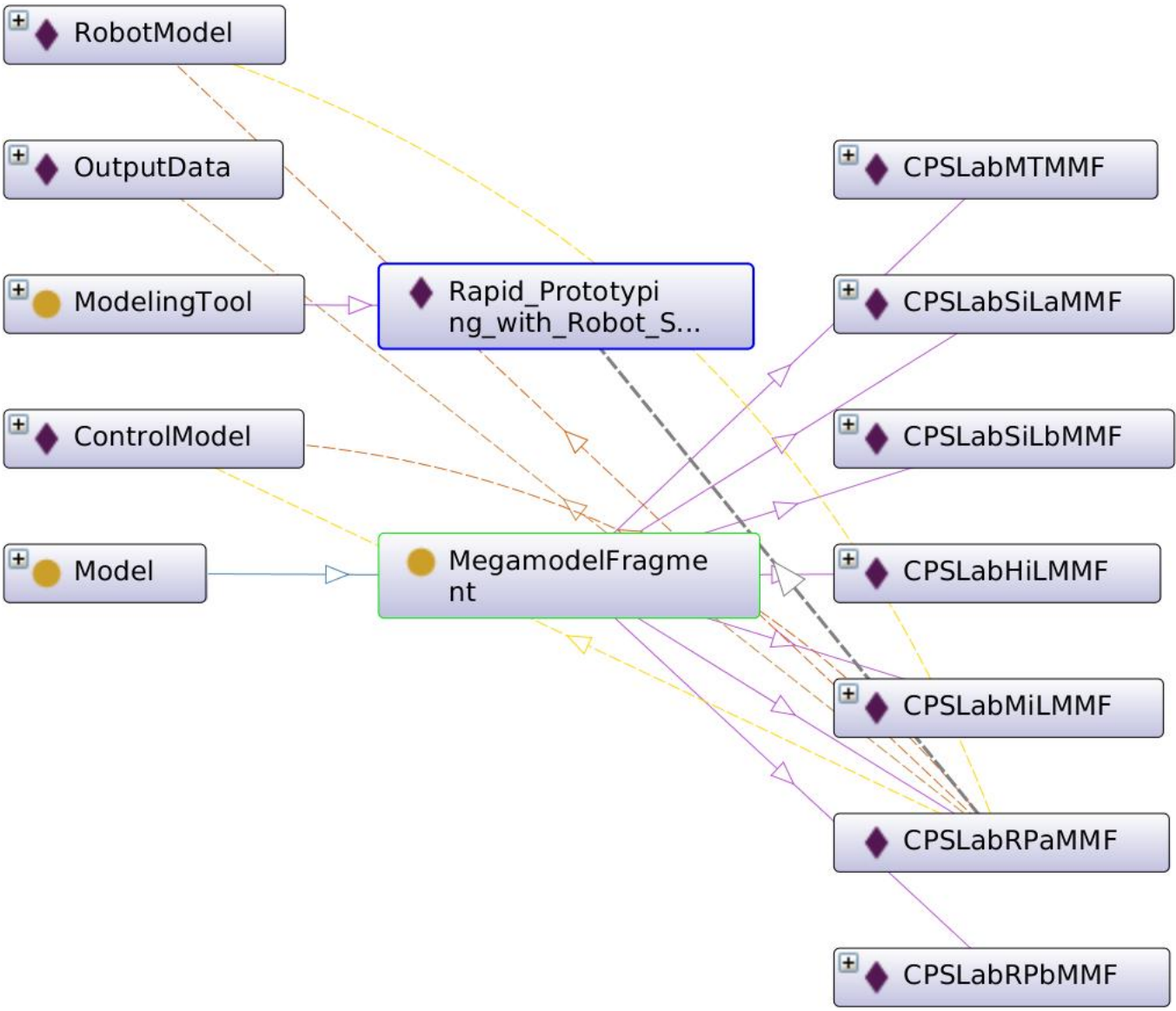

Figure 6.41: Part of the ontology for the MegaModelFragment CPSLabRPaMMF covering Rapid Prototyping with Robot Simulation

visualize the outcome with FESTO Robotino©View, a specific block set compatible with the FESTO Robotino-Library has to be employed.

6.2.3.3 Prototyping

6.2.3.4 Software in the Loop (SiL)

Software in the Loop (SiL) as introduced in Figure 6.30, can actually be done in different ways: A first version executes the generated software on a desktop computer and run it against a simulator as depicted in Figure 6.43.

A second form executes the generated software in contrast on a desktop computer and links it to the robot as depicted in Figure 6.44.

Software in the Loop (SiL) - MPM Ontology

MegaModel Fragment CPSLabSiLaMMF

- MegaModelFragment: CPSLabSiLaMMF

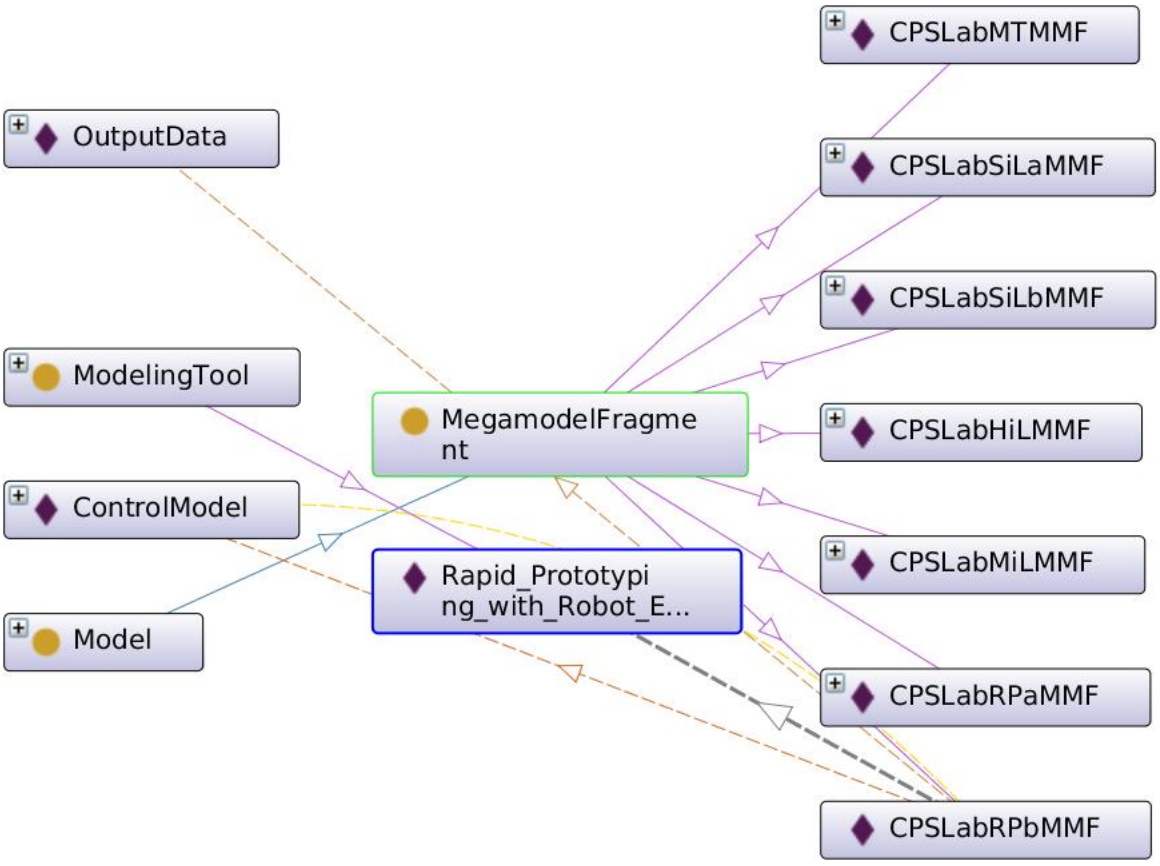

Figure 6.42: Part of the ontology for MegaModelFragment CPSLabRPbMMF covering Rapid Prototyping with Robot Execution

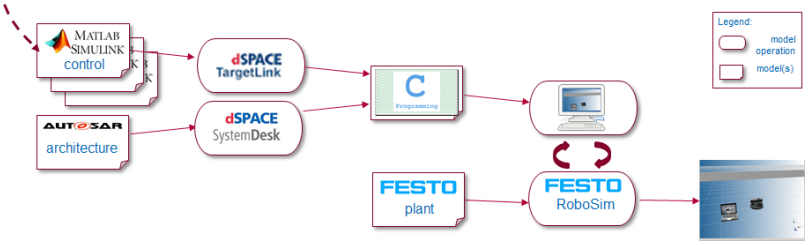

Figure 6.43: Software in the Loop (SiL) vs. Desktop + Sim

- Model(s): ControlModel *, SystemModel, RobotModel

Tools / Models Operations of CPSLabSiLaMMF

- ModelOperation: FunctionCodeGeneration*
  - Input Model(s): ControlModel
  - Output Model(s): ControlCode
  - Employed Tool: dSPACE TargetLink

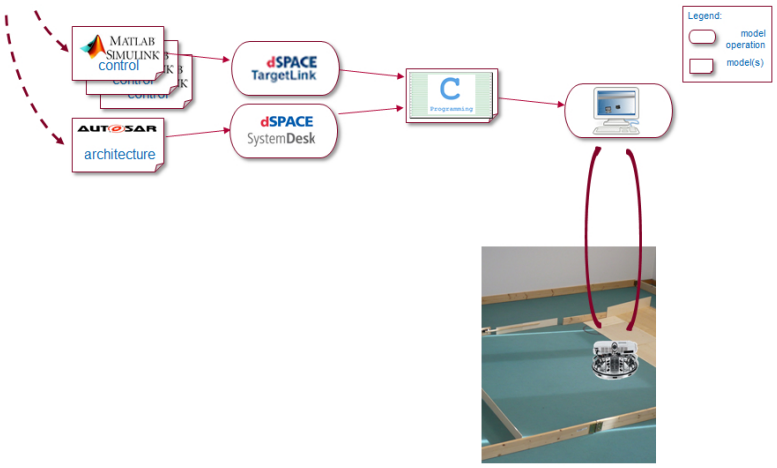

Figure 6.44: Software in the Loop (SiL) vs. Desktop + Robot

- ModelOperation: SystemCodeGeneration
  - Input Model(s): SystemModel
  - Output Model(s): SystemCode
  - Employed Tool: dSPACE SystemDesk
- ModelOperation: Software-in-the-Loop Simulation
  - Input Model(s): ControlCode *, SystemCode, RobotModel
  - Output Model(s): Output data (visualized with MATLAB/Stateflow Simulator and/or FESTO Robotino©View)
  - Employed Tool: Execution on a Desktop computer, FESTO Robotino©Sim

The added MegaModel Fragment CPSLabSiLaMMF and its elements for the CPSLab ontology outlined in the text are depicted also in Figure 6.45.

MegaModel Fragment CPSLabSiLbMMF

- MegaModelFragment: CPSLabSiLbMMF
  - Model(s): ControlModel *, SystemModel

Tools / Models Operations of CPSLabSiLbMMF

- ModelOperation: FunctionCodeGeneration*
  - Input Model(s): ControlModel
  - Output Model(s): ControlCode
  - Employed Tool: dSPACE TargetLink
- ModelOperation: SystemCodeGeneration
  - Input Model(s): SystemModel
  - Output Model(s): SystemCode
  - Employed Tool: dSPACE SystemDesk
- ModelOperation: Software-in-the-Loop Execution
  - Input Model(s): ControlCode *, SystemCode
  - Output Model(s): Output data (visualized with MATLAB/Stateflow Simulator and/or observed)

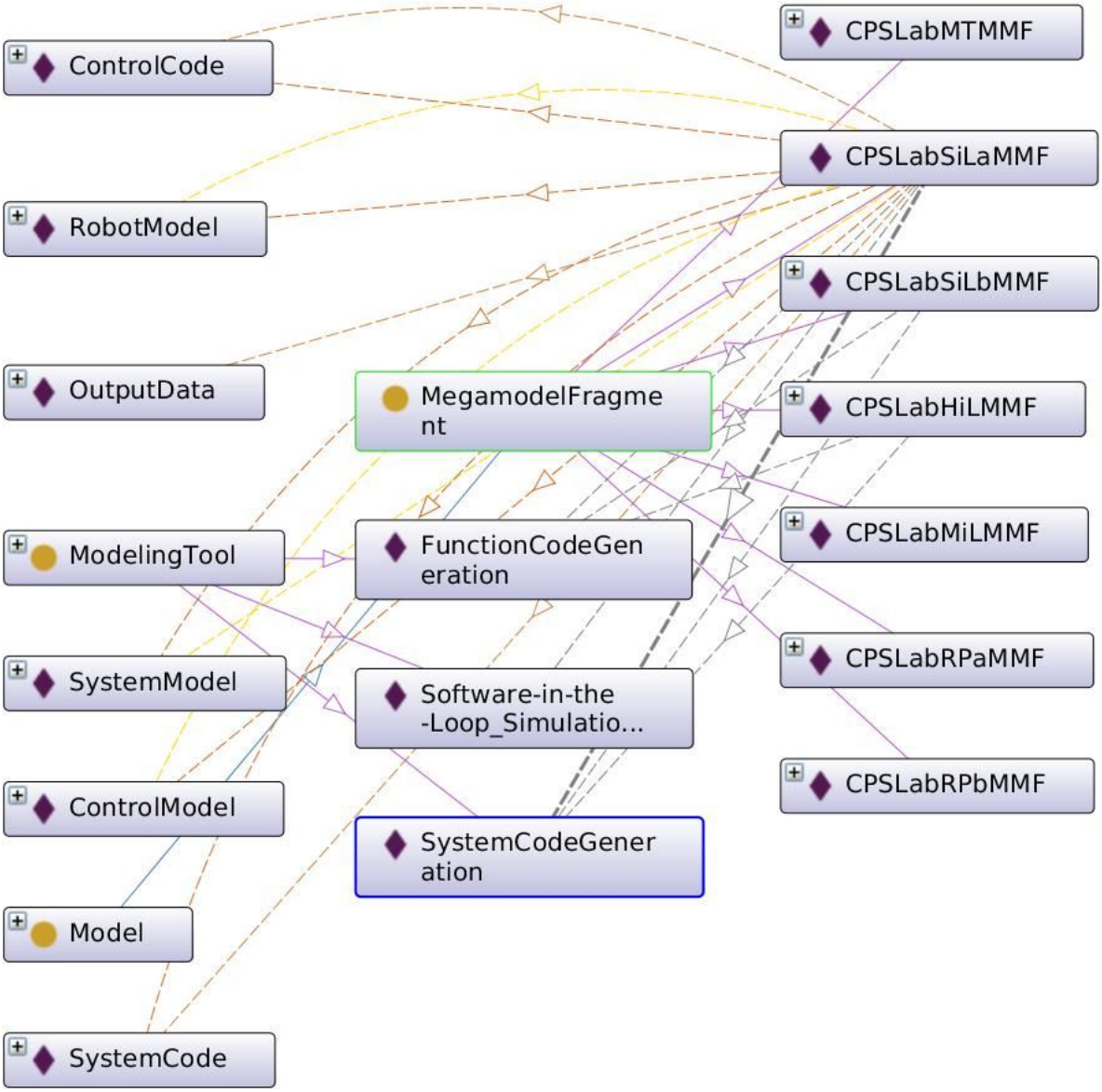

Figure 6.45: Part of the ontology for the MegaModelFragment CPSLabSiLaMMF covering Sil with Simulation

- Employed Tool: Remote execution on a Robotino Robot, Execution on a Desktop computer

The added MegaModel Fragment CPSLabSiLbMMF and its elements for the CPSLab ontology outlined in the text are depicted also in Figure 6.46.

6.2.3.5 Hardware in the Loop (HiL)

Hardware in the Loop (HiL) as introducted in Figure 6.31 in contrast links the generated software such that it can be executed on the robot as depicted in Figure 6.47.

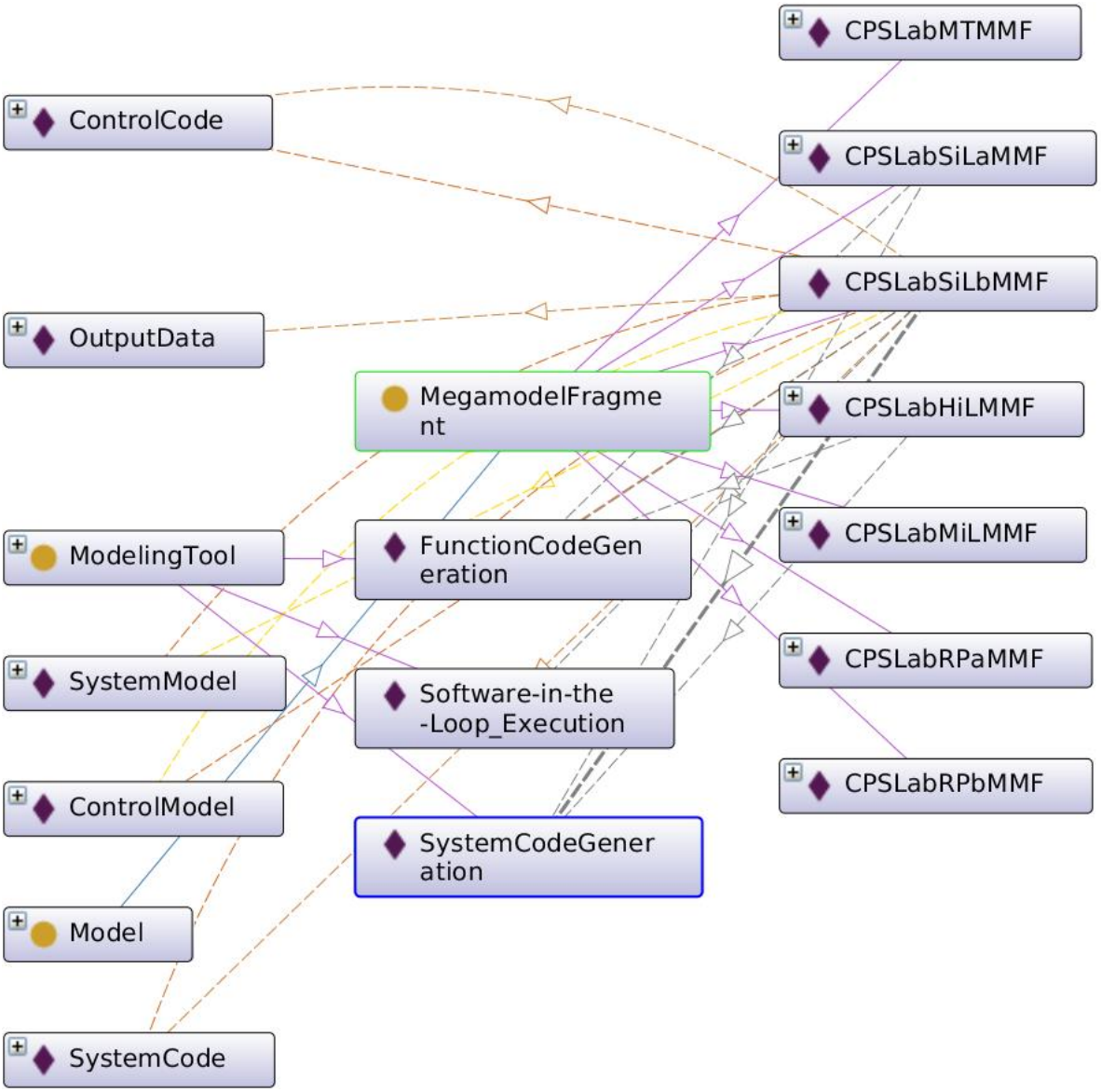

Figure 6.46: Part of the ontology for the MegaModelFragment CPSLabSiLbMMF covering Sil with Execution

Hardware in the Loop (HiL) - MPM Ontology

MegaModel Fragment CPSLabHiLMMF

- MegaModelFragment: CPSLabHiLMMF
  - Model(s): ControlModel *, SystemModel

Tools / Models Operations of CPSLabHiLMMF

- ModelOperation: FunctionCodeGeneration*
  - Input Model(s): ControlModel
  - Output Model(s): ControlCode
  - Employed Tool: dSPACE TargetLink
- ModelOperation: SystemCodeGeneration
  - Input Model(s): SystemModel

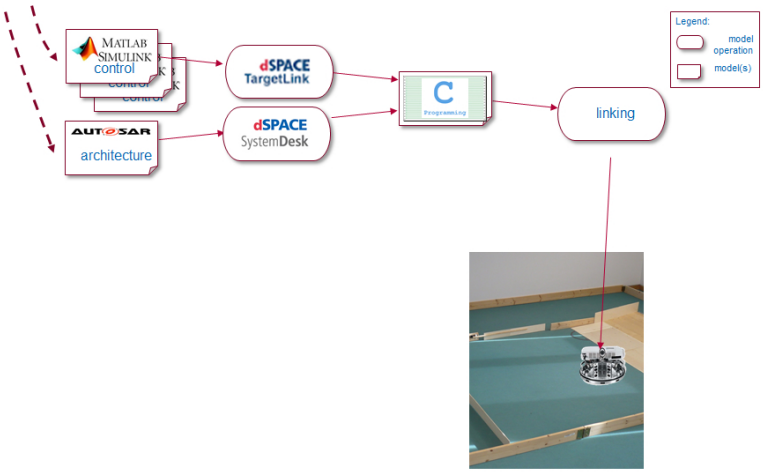

Figure 6.47: Hardware in the Loop (HiL)

- Output Model(s): SystemCode
- Employed Tool: dSPACE SystemDesk

- ModelOperation: Hardware-in-the-Loop Execution
  - Input Model(s): ControlCode *, SystemCode
  - Output Model(s): Output data (observed)
  - Employed Tool: Local execution on a Robotino Robot

The added MegaModel Fragment CPSLabHiLMMF and its elements for the CPSLab ontology outlined in the text are depicted also in Figure **??**.

Prototyping - MPM Ontology

Again, the MPM ontology and the employed megamodel and megamodel fragments do not cover all issues: In contrast to the restriction to MATLAB/Simulink during the simulation stage, for the prototyping stage also dSPACE SystemDesk for describing a component-based architecture with AUTOSAR must be considered as well as outlined above and depicted in the megamodel presented in Figure 6.34. For the software-in-the-loop simulation, it is necessary to adjust the functional models to the specific dSPACE TargetLink block set such that the dSPACE TargetLink for code generation can be employed. In addition, dSPACE SystemDesk is employed to define the software architecture, hardware configuration, and task mapping with AUTOSAR. Then, this combination of models is linked via the blocks for the FESTO Robotino-Library and simulated by linking the MATLAB/Simulink and FESTO Robotino©Sim simulators and visualize the outcome with FESTO Robotino©View. In case of the hardware-in-the-loop testing, the same block set for the FESTO Robotino-Library can be reconfigured such that either the compiled software runs on a host computer and controls the Robotino robots remotely or the compiled and linked software runs directly on the Robotino robots.

6.2.4 MPM4CPS

In order to discuss the role of MPM for the development CPS as present in the case study, we refer to the inherent integration needs underlying the development of embedded real-time systems and cyber-physical systems in particular as outlined in (79). Furthermore, we link these observations to the MPM4CPS ontology as presented in Chapter 5 and further needs to extend it.

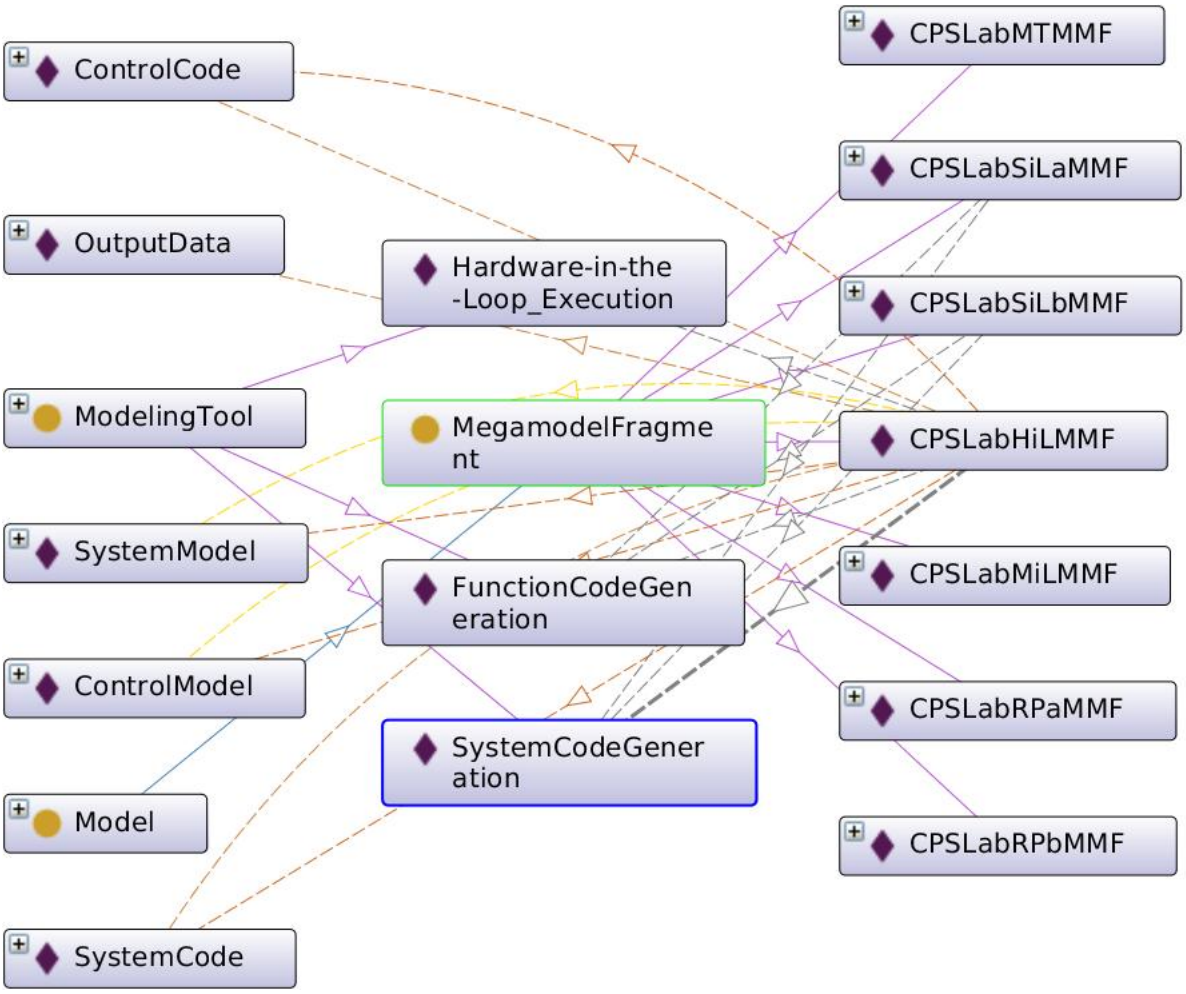

Figure 6.48: Part of the ontology for the MegaModelFragment CPSLabHiLMMF covering Hil

6.2.4.1 Simulation Stage

Model Test

In the model test as outlined in Figure 6.35 and in its megamodel fragment, the abstract control algorithm from the cyber domain captured by the Matlab/Simulink model (ControlModel) for the control is confronted with the physics as present in the input data plus expected outcomes and thus we have a very simple cyber-physical setting. We further often have a multi-formalism setting as the control is discrete while the input is at least conceptually continuous.

Model-in-the-Loop

The model in the loop depicted in Figure 6.37 and in its megamodel fragment in contrast, the abstract control algorithm from the cyber domain captured by the Matlab/Simulink model (ControlModel) is combined with the idealized physics as present in the plant captured by the Matlab/Simulink model (PlantModel) and thus we have a simple cyber-physical setting. We again often have a multi-formalism setting as the control is discrete while the input is at least conceptually continuous.

Rapid Prototyping

Our rapid prototyping based on a sophisticated robot simulator as depicted in Figure 6.39 and in its megamodel fragment, again the abstract control algorithm from the cyber domain captured by the Matlab/Simulink model (ControlModel) is brought together with the physics

as present in the sophisticated robot model (RobotModel) of the simulator and thus we have clearly a cyber-physical setting. We again often have a multi-formalism setting as the control is discrete, while the sophisticated model of the robot simulation is at least conceptually continuous. Consistency is checked via co-simulation as the simulator for the robot model runs in parallel with the sophisticated robot simulator.

The rapid prototyping against the robot as depicted in Figure 6.40 and in its megamodel fragment, the abstract control algorithm from the cyber domain captured by the Matlab/Simulink model (ControlModel) is brought together with the real physics of the robot and thus we have clearly a cyber-physical setting. Consistency is checked via co-simulation as the simulator for the robot model runs in parallel with the robot.

6.2.4.2 Prototyping

6.2.4.3 Software in the Loop (SiL)

The first form of software in the loop (SiL) executing the software on a desktop computer against a simulator as depicted in Figure 6.43 and in its megamodel fragment features that the detailed control algorithm from the cyber domain captured by the Matlab/Simulink and AUTOSAR SystemDesk models (multiple ControlModels and one SystemModel) are brought together with the physics as present in the sophisticated robot model of the simulator (RobotModel) and thus we have clearly a cyber-physical setting. We again often have a multi-formalism setting as the control is discrete while the sophisticated robot mode is at least conceptually continuous. Consistency is checked via co-simulation as the software for the robot control runs in parallel with the sophisticated robot simulator.

The second form of software in the loop (SiL) executing the software on a desktop computer against a remote controlled robot depicted in Figure 6.44 and in its megamodel fragment ensures that the detailed control algorithm from the cyber domain captured by the Matlab/Simulink model and AUTOSAR SystemDesk models (multiple ControlModels and one SystemModel) are brought together with the physics as present in the remote controlled robot and thus we have clearly a cyber-physical setting. Consistency is checked via co-execution as the software for the robot control runs in parallel with the remote controller robot.

6.2.4.4 Hardware in the Loop (HiL)

The hardware in the loop (HiL) executing the software on the robot depicted in Figure 6.44 and in its megamodel fragment ensures that the detailed control algorithm from the cyber domain aptured by the Matlab/Simulink model (ControlModel) is brought together with the physics as present in the robot and thus we have clearly a cyber-physical setting. Consistency is checked via executing the software on the robot.

6.2.5 Summary

We have demonstrated that the framework introduced in this report is suitable to capture the needs concerning this case study for CPS in form of the extensions of the CPS ontology discussed in Section 6.2.2, covering the needs concerning this case study for MPM in form of the extensions of the MPM ontology and the catalog discussed in Section 6.2.3, and captures the cyber-physical aspects of the development quite well as outlined in Section 6.2.4.

7 Summary and Future Work

In this report on the Framework to Relate / Combine Modeling Languages and Techniques of Working Group1 on Foundations of the ICT COST Action IC1404 Multi-Paradigm Modelling for Cyber-Physical Systems (MPM4CPS), we first presented an ontology of Cyber Physical Systems in Chapter 3 and then an ontology of Multi-Paradigm Modeling in Chapter 4. Then, we presented an MPM4CPS ontology that combine these ontologies to cover Multi-Paradigm Modeling for Cyber Physical Systems presented in Chapter 5. Finally, we presented two examples for CPS development cases and sketched how they fit into our framework and how their megamodels resp. megamodel fragments covering development scenarios look like in Chapter 6. These examples instantiate the MPM4CPS ontology and also make use of the catalog of languages and tools.

We developed a CPS ontology such that the catalog of languages can be well classified, developed a MPM ontology such that all needs of the example cases for CPS development approaches are covered, and developed a first MPM4CPS ontology that integrate both based on viewpoints. Furthermore, we developed documentation generators that produce detailed specifications from the ontologies.

As future work, we plan to further enrich the CPS ontology, the MPM ontology, and the MPM4CPS ontology as ongoing community effort. Furthermore, we plan to further develop documentation generators such that they can also produce detailed specifications for development environment examples.

Bibliography

[1] Abusharekh, A. and A. Levis (2016), Performance evaluation of SoA in clouds, pp. 614–620, cited By 0.
https://www.scopus.com/inward/record.uri?eid=2-s2.0-84978636762&partnerID=40&md5=5d11c77c537d51c2bc8fa4696f30258c

[2] Al Ali, R., T. Bures, I. Gerostathopoulos, P. Hnetynka, J. Keznikl, M. Kit and F. Plasil (2014), DEECo: An Ecosystem for Cyber-physical Systems, in *Companion Proceedings of the 36th International Conference on Software Engineering*, ACM, New York, NY, USA, ICSE Companion 2014, pp. 610–611, ISBN 978-1-4503-2768-8, doi: 10.1145/2591062.2591140.
http://doi.acm.org/10.1145/2591062.2591140

[3] Al-Ali, R., T. Bures, B.-O. Hartmann, J. Havlik, R. Heinrich, P. Hnetynka, A. Juan-Verdejo, P. Parizek, S. Seifermann and M. Walter (2018), Use Cases in Dataflow-Based Privacy and Trust Modeling and Analysis in Industry 4.0 Systems, doi: 10.5445/IR/1000085169.

[4] Al Ali, R., T. Bures, P. Hnetynka, F. Krijt, F. Plasil and J. Vinarek (2018), Dynamic Security Specification Through Autonomic Component Ensembles, in *Leveraging Applications of Formal Methods, Verification and Validation. Distributed Systems*, Eds. T. Margaria and B. Steffen, Springer International Publishing, Cham, pp. 172–185, ISBN 978-3-030-03424-5.

[5] Al-Ali, R., R. Heinrich, P. Hnetynka, A. Juan-Verdejo, S. Seifermann and M. Walter (2018), Modeling of Dynamic Trust Contracts for Industry 4.0 Systems, in *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings*, ACM, New York, NY, USA, ECSA '18, pp. 45:1–45:4, ISBN 978-1-4503-6483-6, doi: 10.1145/3241403.3241450.
http://doi.acm.org/10.1145/3241403.3241450

[6] Ali, R. A., T. Bures, I. Gerostathopoulos, J. Keznikl and F. Plasil (2014), Architecture Adaptation Based on Belief Inaccuracy Estimation, *2014 IEEE/IFIP Conference on Software Architecture (WICSA)*, **00**, undefined, pp. 87–90, doi: doi.ieeecomputersociety.org/10.1109/WICSA.2014.20.

[AM3] AM3 (2014), AM3 Project Homepage, https://wiki.eclipse.org/AM3.

[AMW] AMW (2015), AMW Project Homepage, https://projects.eclipse.org/projects/modeling.gmt.amw/.

[ATL] ATL (2015), ATL Project Homepage, https://eclipse.org/atl/.

[ATLFlow] ATLFlow (2013), ATLFlow Project Homepage, http://opensource.urszeidler.de/ATLflow/.

[AToMPM] AToMPM (accessed 2012), AToMPM Project Homepage, http://www-ens.iro.umontreal.ca/~syriani/atompm/atompm.htm.

[12] Balsamo, S., G. D. Rossi and A. Marin (2012), A Survey on Multi-Formalism Performance Evaluation Tools, pp. 15–23, ISBN 9789077381731.

[13] Barbierato, E., A. Bobbio, M. Gribaudo and M. Iacono (2012), Multiformalism to Support Software Rejuvenation Modeling., in *ISSRE Workshops*, IEEE, pp. 271–276, ISBN 978-1-4673-5048-8.

[14] Barbierato, E., M. Gribaudo and M. Iacono (2011), Defining Formalisms for Performance Evaluation With SIMTHESys, *Electr. Notes Theor. Comput. Sci.*, **275**, pp. 37–51.

[15] Barbierato, E., M. Gribaudo and M. Iacono (2011), Exploiting multiformalism models for testing and performance evaluation in SIMTHESys, in *Proceedings of 5th International*

ICST Conference on Performance Evaluation Methodologies and Tools - VALUETOOLS 2011.

[16] Barbierato, E., M. Gribaudo and M. Iacono (2013), A Performance Modeling Language For Big Data Architectures., in *ECMS*, Eds. W. Rekdalsbakken, R. T. Bye and H. Zhang, European Council for Modeling and Simulation, pp. 511–517, ISBN 978-0-9564944-6-7. http://dblp.uni-trier.de/db/conf/ecms/ecms2013.html#BarbieratoGI13

[17] Barbierato, E., M. Gribaudo and M. Iacono (2014), Performance evaluation of NoSQL big-data applications using multi-formalism models, *Future Generation Computer Systems*, **37**, 0, pp. 345–353, ISSN 0167-739X, doi: http://dx.doi.org/10.1016/j.future.2013.12.036.

[18] Barbierato, E., M. Gribaudo and M. Iacono (2016), Modeling Hybrid Systems in {SIMTHESys}, *Electronic Notes in Theoretical Computer Science*, **327**, pp. 5 – 25, ISSN 1571-0661, doi: http://dx.doi.org/10.1016/j.entcs.2016.09.021, the 8th International Workshop on Practical Application of Stochastic Modeling, {PASM} 2016.

[19] Barbierato, E., M. Gribaudo and M. Iacono (2016), *Simulating Hybrid Systems Within SIMTHESys Multi-formalism Models*, Springer International Publishing, Cham, pp. 189–203, ISBN 978-3-319-46433-6, doi: 10.1007/978-3-319-46433-6_13.

[20] Barbierato, E., M. Gribaudo, M. Iacono and S. Marrone (2011), Performability Modeling of Exceptions-Aware Systems in Multiformalism Tools, in *ASMTA*, pp. 257–272.

[21] Barbierato, E., G.-L. D. Rossi, M. Gribaudo, M. Iacono and A. Marin (2013), Exploiting product forms solution techniques in multiformalism modeling, *Electronic Notes in Theoretical Computer Science*, **296**, 0, pp. 61 – 77, ISSN 1571-0661, doi: http://dx.doi.org/10.1016/j.entcs.2013.07.005.

[22] Bause, F., P. Buchholz and P. Kemper (1998), A Toolbox for Functional and Quantitative Analysis of DEDS, in *Proceedings of the 10th International Conference on Computer Performance Evaluation: Modelling Techniques and Tools*, Springer-Verlag, London, UK, TOOLS '98, pp. 356–359, ISBN 3-540-64949-2.

[23] Beyhl, T., R. Hebig and H. Giese (2013), A Model Management Framework for Maintaining Traceability Links, in *Software Engineering 2013 Workshopband*, volume P-215 of *Lecture Notes in Informatics (LNI)*, Eds. S. Wagner and H. Lichter, Gesellschaft für Informatik (GI), Aachen, volume P-215 of *Lecture Notes in Informatics (LNI)*, pp. 453–457.

[24] Bézivin, J., F. Jouault, P. Rosenthal and P. Valduriez (2005), Modeling in the Large and Modeling in the Small, in *Model Driven Architecture*, volume 3599/2005 of *Lecture Notes in Computer Science (LNCS)*, Springer-Verlag, volume 3599/2005 of *Lecture Notes in Computer Science (LNCS)*, pp. 33–46.

[25] Bézivin, J., F. Jouault and P. Valduriez (2004), On the Need for Megamodels, in *Proceedings of the OOPSLA/GPCE: Best Practices for Model-Driven Software Development workshop, 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications*.

[26] Blanc, X., A. Mougenot, I. Mounier and T. Mens (2009), Incremental Detection of Model Inconsistencies Based on Model Operations, in *CAiSE '09: Proceedings of the 21st International Conference on Advanced Information Systems Engineering, Amsterdam, The Netherlands*, volume 5565/2009, Springer Verlag, Berlin, Heidelberg, volume 5565/2009, pp. 32–46.

[27] Blouin, A., B. Combemale, B. Baudry and O. Beaudoux (2015), Kompren: modeling and generating model slicers, *Software & Systems Modeling*, **14**, 1, pp. 321–337.

[28] Blouin, D., Y. Eustache and J.-P. Diguet (2014), Extensible Global Model Management with Meta-model Subsets and Model Synchronization, in *Proceedings of the 2nd International Workshop on The Globalization of Modeling Languages co-located with ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems, GEMOC@Models 2014, Valencia, -*, pp. 43–52.

[29] Blouin, D., G. Ochoa Ruiz, Y. Eustache and J.-P. Diguet (2015), Kaolin: a System-level AADL Tool for FPGA Design Reuse, Upgrade and Migration, in *NASA/ESA International Conference on Adaptive Hardware and Systems (AHS)*, Montréal, Canada.

[30] Bobeanu, C.-V., E. Kerckhoffs and H. Van Landeghem (2004), Modeling of discrete event systems: A holistic and incremental approach using Petri nets, *ACM Transactions on Modeling and Computer Simulation*, **14**, 4, pp. 389–423, doi: 10.1145/1029174.1029178, cited By 0.
https://www.scopus.com/inward/record.uri?eid=2-s2.0-10944258434&partnerID=40&md5=1ede5ce69882dd45f896ca087824d1af

[31] Boddy, M., M. Michalowski, A. Schwerdfeger, H. Shackleton, S. Vestal and A. Enterprises (2011), FUSED: A Tool Integration Framework for Collaborative System Engineering, in *Analytic Virtual Integration of Cyber-Physical Systems Workshop*.

[32] Bradley, J., M. Guenther, R. Hayden and A. Stefanek (2013), *GPA: A multiformalism, multisolution approach to efficient analysis of Large-Scale population models*, doi: 10.4018/978-1-4666-4659-9.ch008, cited By 0.
https://www.scopus.com/inward/record.uri?eid=2-s2.0-84956815020&partnerID=40&md5=8896757c2d6f87b4bef9f776af11f866

[33] Broekman, B. and E. Notenboom (2003), *Testing Embedded Software*, Addison Wesley.

[34] Broman, D., E. A. Lee, S. Tripakis and M. Törngren (2012), Viewpoints, Formalisms, Languages, and Tools for Cyber-physical Systems, in *Proceedings of the 6th International Workshop on Multi-Paradigm Modeling*, ACM, New York, NY, USA, MPM '12, pp. 49–54, ISBN 978-1-4503-1805-1, doi: 10.1145/2508443.2508452.
http://doi.acm.org/10.1145/2508443.2508452

[35] Bures, T., I. Gerostathopoulos, P. Hnetynka, J. Keznikl, M. Kit and F. Plasil (2013), DEECO: An Ensemble-based Component System, in *Proceedings of the 16th International ACM Sigsoft Symposium on Component-based Software Engineering*, ACM, New York, NY, USA, CBSE '13, pp. 81–90, ISBN 978-1-4503-2122-8, doi: 10.1145/2465449.2465462.
http://doi.acm.org/10.1145/2465449.2465462

[36] Bures, T., I. Gerostathopoulos, P. Hnetynka, J. Keznikl, M. Kit and F. Plasil (2014), *Gossiping Components for Cyber-Physical Systems*, Springer International Publishing, Cham, pp. 250–266, ISBN 978-3-319-09970-5, doi: 10.1007/978-3-319-09970-5_23.
http://dx.doi.org/10.1007/978-3-319-09970-5_23

[37] Bures, T., I. Gerostathopoulos, P. Hnetynka, J. Keznikl, M. Kit and F. Plasil (2014), Gossiping Components for Cyber-Physical Systems, in *Software Architecture*, Eds. P. Avgeriou and U. Zdun, Springer International Publishing, Cham, pp. 250–266, ISBN 978-3-319-09970-5.

[38] Bures, T., P. Hnetynka, J. Kofron, R. A. Ali and D. Skoda (2016), Statistical Approach to Architecture Modes in Smart Cyber Physical Systems, in *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, pp. 168–177, doi: 10.1109/WICSA.2016.33.

[39] Bures, T., P. Hnetynka, J. Kofron, R. A. Ali and D. Skoda (2016), Statistical Approach to Architecture Modes in Smart Cyber Physical Systems, in *2016 13th Working IEEE/IFIP*

Conference on Software Architecture (WICSA), pp. 168–177, doi: 10.1109/WICSA.2016.33.

[40] Bures, T., P. Hnetynka, F. Krijt, V. Matena and F. Plasil (2016), *Smart Coordination of Autonomic Component Ensembles in the Context of Ad-Hoc Communication*, Springer International Publishing, Cham, pp. 642–656, ISBN 978-3-319-47166-2, doi: 10.1007/978-3-319-47166-2_45.
http://dx.doi.org/10.1007/978-3-319-47166-2_45

[41] Bures, T., F. Krijt, F. Plasil, P. Hnetynka and Z. Jiracek (2015), Towards Intelligent Ensembles, in *Proceedings of the 2015 European Conference on Software Architecture Workshops*, ACM, New York, NY, USA, ECSAW '15, pp. 17:1–17:4, ISBN 978-1-4503-3393-1, doi: 10.1145/2797433.2797450.
http://doi.acm.org/10.1145/2797433.2797450

[42] Bures, T., V. Matena, R. Mirandola, L. Pagliari and C. Trubiani (2018), Performance Modelling of Smart Cyber-Physical Systems, in *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, ACM, New York, NY, USA, ICPE '18, pp. 37–40, ISBN 978-1-4503-5629-9, doi: 10.1145/3185768.3186306.
http://doi.acm.org/10.1145/3185768.3186306

[43] Cabot, J. and E. Teniente (2006), Incremental Evaluation of OCL Constraints, in *CAiSE'06: 18th International Conference on Advanced Information Systems Engineering, Luxembourg, Luxembourg*, volume 4001/2006 of *Lecture Notes in Computer Science (LNCS)*, Springer Verlag, volume 4001/2006 of *Lecture Notes in Computer Science (LNCS)*, pp. 81–95.

[44] Castiglione, A., M. Gribaudo, M. Iacono and F. Palmieri (2014), Exploiting mean field analysis to model performances of big data architectures, *Future Generation Computer Systems*, **37**, 0, pp. 203–211, ISSN 0167-739X, doi: http://dx.doi.org/10.1016/j.future.2013.07.016.

[45] Chiaradonna, S., P. Lollini and F. Giandomenico (2007), On a modeling framework for the analysis of interdependencies in electric power systems, pp. 185–194, doi: 10.1109/DSN.2007.68, cited By 0.
https://www.scopus.com/inward/record.uri?eid=2-s2.0-36049035971&partnerID=40&md5=b71e5b88b6b900d845d599c07906f32b

[46] Ciardo, G., R. L. Jones, III, A. S. Miner and R. I. Siminiceanu (2006), Logic and stochastic modeling with SMART, *Perform. Eval.*, **63**, pp. 578–608, ISSN 0166-5316.

[47] Ciardo, G. and A. S. Miner (2004), SMART: the stochastic model checking analyzer for reliability and timing, in *Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings. First International Conference on the*, pp. 338–339, doi: 10.1109/QEST.2004.1348056.

[48] Ciardo, G., A. S. Miner and M. Wan (2009), Advanced Features in SMART: The Stochastic Model Checking Analyzer for Reliability and Timing, *SIGMETRICS Perform. Eval. Rev.*, **36**, 4, pp. 58–63, ISSN 0163-5999, doi: 10.1145/1530873.1530885.
http://doi.acm.org/10.1145/1530873.1530885

[49] Clark, G., T. Courtney, D. Daly, D. Deavours, S. Derisavi, J. M. Doyle, W. H. Sanders and P. Webster (2001), The Mobius Modeling Tool, in *Proceedings of the 9th international Workshop on Petri Nets and Performance Models (PNPM'01)*, IEEE Computer Society, Washington, DC, USA, pp. 241–.

[50] Clasen, C., F. Jouault and J. Cabot (2011), Virtual Composition of EMF Models, in *7emes Journees sur l'Ingenierie Dirigee par les Modeles (IDM 2011)*, Lille, France.

[CoEST] CoEST (accessed 2016), CoEST Project Homepage, http://www.coest.org/.

[Composite EMF Models] Composite EMF Models (accessed 2015), Composite EMF Models Project Homepage, http://www.mathematik.uni-marburg.de/~swt/compoemf/.

[53] Courtney, T., S. Gaonkar, K. Keefe, E. Rozier and W. H. Sanders (2009), Möbius 2.3: An extensible tool for dependability, security, and performance evaluation of large and complex system models., in *DSN*, IEEE, pp. 353–358.

[54] Czarnecki, K., C. Hwan, P. Kim and K. T. Kalleberg (2006), Feature models are views on ontologies, in *10th International Software Product Line Conference (SPLC'06)*, pp. 41–51, doi: 10.1109/SPLINE.2006.1691576.

[55] Dandashi, F., V. Lakshminarayan and N. Schult (2016), Multiformalism, Multiresolution, Multiscale Modeling, volume 2016-February, pp. 2622–2631, doi: 10.1109/WSC.2015.7408370, cited By 0. https://www.scopus.com/inward/record.uri?eid=2-s2.0-84962885309&partnerID=40&md5=a4db2cfc506ce5622925eb9e345d02ec

[56] Deavours, D. D., G. Clark, T. Courtney, D. Daly, S. Derisavi, J. M. Doyle, W. H. Sanders and P. G. Webster (2002), The Möbius Framework and Its Implementation.

[57] Debreceni, C., A. Horvath, A. Hegedus, Z. Ujhelyi, I. Rath and D. Varro (2014), Query-driven Incremental Synchronization of View Models, in *Proceedings of the 2Nd Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling*, ACM, New York, NY, USA, VAO '14, pp. 31:31–31:38.

[58] Del V. Sosa, M., S. Acuna and J. De Lara (2007), Metamodeling and multiformalism approach applied to software process using AToM [Enfoque de Metamodelado y Multiformalismo Aplicado al Proceso Software usando AToM3], pp. 367–374, cited By 0. https://www.scopus.com/inward/record.uri?eid=2-s2.0-84883063089&partnerID=40&md5=9d2c10013885274018144a23e10464c4

[59] Di Ruscio, D., I. Malavolta, H. Muccini, P. Pelliccione and A. Pierantonio (2010), Developing Next Generation ADLs Through MDE Techniques, in *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ACM, New York, NY, USA, ICSE '10, pp. 85–94, ISBN 978-1-60558-719-6, doi: 10.1145/1806799.1806816. http://doi.acm.org/10.1145/1806799.1806816

[60] Di Ruscio, D., I. Malavolta, H. Muccini, P. Pelliccione and A. Pierantonio (2012), Model-Driven Techniques to Enhance Architectural Languages Interoperability, in *Fundamental Approaches to Software Engineering*, Eds. J. de Lara and A. Zisman, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 26–42, ISBN 978-3-642-28872-2.

[61] Egyed, A. (2006), Instant Consistency Checking for the UML, in *ICSE '06: Proceedings of the 28th International Conference on Software Engineering*, Shanghai, China, pp. 381–390.

[EMF-IncQuery] EMF-IncQuery (2015), EMF-IncQuery Project Homepage, https://www.eclipse.org/incquery/.

[EMF Views] EMF Views (accessed 2015), EMF Views Project Homepage, http://atlanmod.github.io/emfviews/.

[Epsilon] Epsilon (2014), Epsilon Project Homepage, http://eclipse.org/epsilon/.

[65] Eramo, R., I. Malavolta, H. Muccini, P. Pelliccione and A. Pierantonio (2012), A model-driven approach to automate the propagation of changes among Architecture Description Languages, *Software & Systems Modeling*, **11**, 1, pp. 29–53, ISSN 1619-1374,

doi: 10.1007/s10270-010-0170-z.
https://doi.org/10.1007/s10270-010-0170-z

[66] Etien, A., A. Muller, T. Legrand and X. Blanc (2010), Combining Independent Model Transformations, in *Proceedings of the 2010 ACM Symposium on Applied Computing*, ACM, New York, NY, USA, SAC '10, pp. 2237–2243.

[67] Favre, J.-M. (2004), Foundations of Model (Driven) (Reverse) Engineering – Episode I: Story of The Fidus Papyrus and the Solarus, in *Post-Proceedings of Dagstuhl Seminar on Model Driven Reverse Engineering*.

[68] Favre, J.-M., R. Lämmel and A. Varanovich (2012), Modeling the linguistic architecture of software products, in *Proceedings of the 15th international conference on Model Driven Engineering Languages and Systems*, Springer-Verlag, Berlin, Heidelberg, MODELS'12, pp. 151–167.

[69] Finkelstein, A. C. W., D. Gabbay, A. Hunter, J. Kramer and B. Nuseibeh (1994), Inconsistency Handling in Multiperspective Specifications, *IEEE Transactions on Software Engineering*, **20**, 8, pp. 569–578.

[70] Flammini, F., S. Marrone, M. Iacono, N. Mazzocca and V. Vittorini (2014), A Multiformalism Modular Approach to ERTMS/ETCS Failure Modelling, *International Journal of Reliability, Quality and Safety Engineering*, **21**, 01, pp. 1450001–1–1450001–29, doi: 10.1142/S0218539314500016.

[71] Franceschinis, F., M. Gribaudo, M. Iacono, N. Mazzocca and V. Vittorini (2002), Towards an Object Based Multi-Formalism Multi-Solution Modeling Approach., in *Proc. of the Second International Workshop on Modelling of Objects, Components, and Agents (MOCA'02), Aarhus, Denmark, August 26-27, 2002 / Daniel Moldt (Ed.)*, Technical Report DAIMI PB-561, pp. 47–66.

[72] Franceschinis, G., M. Gribaudo, M. Iacono, S. Marrone, N. Mazzocca and V. Vittorini (2004), Compositional Modeling of Complex Systems: Contact Center Scenarios in OsMoSys, in *ICATPN'04*, pp. 177–196.

[73] Franceschinis, G., M. Gribaudo, M. Iacono, S. Marrone, F. Moscato and V. Vittorini (2009), Interfaces and binding in component based development of formal models, in *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, VALUETOOLS '09, pp. 44:1–44:10, ISBN 978-963-9799-70-7.

[FUSED] FUSED (2015), FUSED Project Homepage, http://www.adventiumlabs.com/our-work/products-services/fused-informational-video/.

[Gaspard2] Gaspard2 (2008), Gaspard2 Project Homepage, http://www.lifl.fr/west/gaspard/.

[76] Gerostathopoulos, I., T. Bures, P. Hnetynka, A. Hujecek, F. Plasil and D. Skoda (2017), Strengthening Adaptation in Cyber-Physical Systems via Meta-Adaptation Strategies, *ACM Trans. Cyber-Phys. Syst.*, **1**, 3, pp. 13:1–13:25, ISSN 2378-962X, doi: 10.1145/2823345.
http://doi.acm.org/10.1145/2823345

[77] Gerostathopoulos, I., T. Bures, P. Hnetynka, J. Keznikl, M. Kit, F. Plasil and N. Plouzeau (2016), Self-adaptation in software-intensive cyberâĂŞphysical systems: From system goals to architecture configurations, *Journal of Systems and Software*, **122**, pp. 378 – 397, ISSN 0164-1212, doi: https://doi.org/10.1016/j.jss.2016.02.028.
http://www.sciencedirect.com/science/article/pii/S0164121216000601

[78] Giese, H. and D. Blouin (2016), State-of-the-art on Current Formalisms used in Cyber-Physical Systems Development, Technical Report D1.1 (Version 1), ICT COST Action IC1404 Multi-Paradigm Modelling for Cyber-Physical Systems (MPM4CPS).

[79] Giese, H., S. Neumann, O. Niggemann and B. Schätz (2011), Model-Based Integration, in *Model-Based Engineering of Embedded Real-Time Systems - International Dagstuhl Workshop, Dagstuhl Castle, Germany, November 4-9, 2007. Revised Selected Papers*, volume 6100 of *Lecture Notes in Computer Science*, Eds. H. Giese, G. Karsai, E. Lee, B. Rumpe and B. Schätz, Springer, volume 6100 of *Lecture Notes in Computer Science*, pp. 17–54.

[GME] GME (2011), GME Project Homepage, http://www.isis.vanderbilt.edu/projects/gme/.

[81] Gribaudo, G., M. Iacono, M. Mazzocca and V. Vittorini (2003), The OsMoSys/DrawNET Xe! Languages System: A Novel Infrastructure for Multi-Formalism Object-Oriented Modelling, in *ESS 2003: 15th European Simulation Symposium And Exhibition*.

[82] Gribaudo, M., M. Iacono and S. Marrone (2015), Exploiting Bayesian Networks for the Analysis of Combined Attack Trees, *Electronic Notes in Theoretical Computer Science*, **310**, 0, pp. 91 – 111, ISSN 1571-0661, doi: http://dx.doi.org/10.1016/j.entcs.2014.12.014, proceedings of the Seventh International Workshop on the Practical Application of Stochastic Modelling (PASM). http://www.sciencedirect.com/science/article/pii/S157106611400098X

[83] Groher, I., A. Reder and A. Egyed (2010), Incremental Consistency Checking of Dynamic Constraints, in *Fundamental Approaches to Software Engineering*, volume 6013 of *Lecture Notes in Computer Science*, Eds. D. Rosenblum and G. Taentzer, Springer Berlin Heidelberg, pp. 203–217.

[84] Gruber, T. R. (1995), Toward principles for the design of ontologies used for knowledge sharing?, *International Journal of Human-Computer Studies*, **43**, 5, pp. 907 – 928, ISSN 1071-5819, doi: http://dx.doi.org/10.1006/ijhc.1995.1081. http://www.sciencedirect.com/science/article/pii/S1071581985710816

[85] Hebig, R., A. Seibel and H. Giese (2011), On the Unification of Megamodels, in *Proceedings of the 4th International Workshop on Multi-Paradigm Modeling (MPM 2010)*, volume 42 of *Electronic Communications of the EASST*, Eds. V. Amaral, H. Vangheluwe, C. Hardebolle, L. Lengyel, T. Magaria, J. Padberg and G. Taentzer, volume 42 of *Electronic Communications of the EASST*.

[86] Herzig, S. J. and C. J. Paredis (2014), Bayesian Reasoning Over Models, in *11th Workshop on Model Driven Engineering, Verification and Validation MoDeVVa 2014*, p. 69.

[87] Herzig, S. J., A. Qamar and C. J. Paredis (2014), An Approach to Identifying Inconsistencies in Model-based Systems Engineering, *Procedia Computer Science*, **28**, 0, pp. 354 – 362, ISSN 1877-0509, 2014 Conference on Systems Engineering Research.

[88] Hessellund, A. and A. Wasowski (2008), Interfaces and Metainterfaces for Models and Metamodels, in *Model Driven Engineering Languages and Systems*, volume 5301 of *Lecture Notes in Computer Science*, Eds. K. Czarnecki, I. Ober, J.-M. Bruel, A. Uhl and M. Wolter, Springer Berlin Heidelberg, pp. 401–415.

[89] Hilliard, R., I. Malavolta, H. Muccini and P. Pelliccione (2010), Realizing Architecture Frameworks Through Megamodelling Techniques, in *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE2010)*, ACM, New York, NY, USA, pp. 305–308, ISBN 978-1-4503-0116-9, doi: 10.1145/1858996.1859057.

http://doi.acm.org/10.1145/1858996.1859057

[90] Hilliard, R., I. Malavolta, H. Muccini and P. Pelliccione (2012), On the Composition and Reuse of Viewpoints across Architecture Frameworks, in *2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*, pp. 131–140, doi: 10.1109/WICSA-ECSA.212.21.

[91] Hnetynka, P., P. Kubat, R. Al-Ali, I. Gerostathopoulos and D. Khalyeyev (2018), Guaranteed Latency Applications in Edge-cloud Environment, in *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings*, ACM, New York, NY, USA, ECSA '18, pp. 43:1–43:4, ISBN 978-1-4503-6483-6, doi: 10.1145/3241403.3241448.
http://doi.acm.org/10.1145/3241403.3241448

[92] Hoch, N., H.-P. Bensler, D. Abeywickrama, T. Bureš and U. Montanari (2015), *The E-mobility Case Study*, Springer International Publishing, Cham, pp. 513–533, ISBN 978-3-319-16310-9, doi: 10.1007/978-3-319-16310-9_17.
https://doi.org/10.1007/978-3-319-16310-9_17

[93] Iacono, M., E. Barbierato and M. Gribaudo (2012), The SIMTHESys multiformalism modeling framework, *Computers and Mathematics with Applications*, , 64, pp. 3828–3839, ISSN 0898-1221, doi: 10.1016/j.camwa.2012.03.009.

[94] Iacono, M. and M. Gribaudo (2010), Element Based Semantics in Multi Formalism Performance Models, in *MASCOTS*, pp. 413–416.

[95] Iacono, M. and S. Marrone (2014), Model-Based Availability Evaluation of Composed Web Services, *Journal of Telecommunications and Information Technology*, , 4, pp. 5–13.

[96] Jurack, S. and G. Taentzer (2010), A Component Concept for Typed Graphs with Inheritance and Containment Structures, in *Graph Transformations - 5th International Conference, ICGT 2010 Enschede, The Netherlands, September 27 - - October 2, 2010. Proceedings*, pp. 187–202.

[97] Kang, K., S. Cohen, J. Hess, W. Nowak and A. S. Peterson (1990), Feature-Oriented Domain Analysis (FODA) Feasibility Study, Technical Report CMU/SEI-90-TR-21, Software Engineering Institute.

[98] Keznikl, J., T. Bures, F. Plasil, I. Gerostathopoulos, P. Hnetynka and N. Hoch (2013), Design of Ensemble-based Component Systems by Invariant Refinement, in *Proceedings of the 16th International ACM Sigsoft Symposium on Component-based Software Engineering*, ACM, New York, NY, USA, CBSE '13, pp. 91–100, ISBN 978-1-4503-2122-8, doi: 10.1145/2465449.2465457.
http://doi.acm.org/10.1145/2465449.2465457

[99] Kit, M., I. Gerostathopoulos, T. Bures, P. Hnetynka and F. Plasil (2015), An Architecture Framework for Experimentations with Self-adaptive Cyber-physical Systems, in *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, IEEE Press, Piscataway, NJ, USA, SEAMS '15, pp. 93–96.
http://dl.acm.org/citation.cfm?id=2821357.2821374

[100] Kit, M., F. Plasil, V. Matena, T. Bures and O. Kovac (2015), Employing domain knowledge for optimizing component communication, in *2015 18th International ACM SIGSOFT Symposium on Component-Based Software Engineering (CBSE)*, pp. 59–64, doi: 10.1145/2737166.2737172.

[Kompose] Kompose (2009), Kompose Project Homepage,
http://www.kermeta.org/mdk/kompose.

[Kompren] Kompren (2014), Kompren Project Homepage,
http://people.irisa.fr/Arnaud.Blouin/software_kompren.html.

[103] Krijt, F., Z. Jiracek, T. Bures, P. Hnetynka and I. Gerostathopoulos (2017), Intelligent Ensembles - A Declarative Group Description Language and Java Framework, in *2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pp. 116–122, doi: 10.1109/SEAMS.2017.17.

[104] Krijt, F., Z. Jiracek, T. Bures, P. Hnetynka and F. Plasil (2017), Automated Dynamic Formation of Component Ensembles - Taking Advantage of Component Cooperation Locality, in *Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development - Volume 1: MODELSWARD,*, INSTICC, SciTePress, pp. 561–568, ISBN 978-989-758-210-3, doi: 10.5220/0006273705610568.

[105] Lacoste-Julien, S., H. Vangheluwe, J. De Lara and P. Mosterman (2004), Meta-modelling hybrid formalisms, pp. 65–70, cited By 6.
https://www.scopus.com/inward/record.uri?eid=2-s2.0-20344363389&partnerID=40&md5=dce9d1471bc0989c81e4dfd7734945cd

[106] Langsweirdt, D., N. Boucke and Y. Berbers (2010), Architecture-Driven Development of Embedded Systems with ACOL, in *Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 2010 13th IEEE International Symposium on*, pp. 138–144.

[107] de Lara, J. and H. Vangheluwe (2002), AToM3: A Tool for Multi-formalism and Meta-modelling., in *FASE*, volume 2306 of *Lecture Notes in Computer Science*, Eds. R.-D. Kutsche and H. Weber, Springer, volume 2306 of *Lecture Notes in Computer Science*, pp. 174–188, ISBN 3-540-43353-8.

[108] Levis, A. and B. Yousefi (2014), Multi-formalism modeling for evaluating the effect of cyber exploits, pp. 541–547, cited By 0.
https://www.scopus.com/inward/record.uri?eid=2-s2.0-84905758024&partnerID=40&md5=621e2f7f79f7645829cd972e7632d441

[109] Lochmann, H. and A. Hessellund (2009), An Integrated View on Modeling with Multiple Domain-Specific Languages, in *Proceedings of the IASTED International Conference Software Engineering (SE 2009)*, ACTA Press, pp. 1–10.

[110] Lúcio, L., S. Mustafiz, J. Denil, H. Vangheluwe and M. Jukss (2013), FTG+PM: An Integrated Framework for Investigating Model Transformation Chains, in *SDL 2013: Model-Driven Dependability Engineering*, volume 7916 of *Lecture Notes in Computer Science*, Eds. F. Khendek, M. Toeroe, A. Gherbi and R. Reed, Springer Berlin Heidelberg, pp. 182–202.

[111] Malavolta, I., P. Lago, H. Muccini, P. Pelliccione and A. Tang (2013), What Industry Needs from Architectural Languages: A Survey, *IEEE Transactions on Software Engineering*, **39**, 6, pp. 869–891, ISSN 0098-5589, doi: 10.1109/TSE.2012.74.

[112] Malavolta, I., H. Muccini, P. Pelliccione and D. Tamburri (2010), Providing Architectural Languages and Tools Interoperability through Model Transformation Technologies, *IEEE Transactions on Software Engineering*, **36**, 1, pp. 119–140, ISSN 0098-5589, doi: 10.1109/TSE.2009.51.

[113] Marco Gribaudo, M. I. (2014), An introduction to multiformalism modeling, in *Theory and Application of Multi-Formalism Modeling*, Eds. M. Gribaudo and M. Iacono, IGI Global, Hershey, pp. 1–16.

[114] Matena, V., T. Bures, I. Gerostathopoulos and P. Hnetynka (2016), Model Problem and Testbed for Experiments with Adaptation in Smart Cyber-physical Systems, in *Proceedings of the 11th International Symposium on Software Engineering for Adaptive*

and Self-Managing Systems, ACM, New York, NY, USA, SEAMS '16, pp. 82–88, ISBN 978-1-4503-4187-5, doi: 10.1145/2897053.2897065.
http://doi.acm.org/10.1145/2897053.2897065

[115] Matena, V., T. Bures, I. Gerostathopoulos and P. Hnetynka (2016), Model Problem and Testbed for Experiments with Adaptation in Smart Cyber-Physical Systems, in *2016 IEEE/ACM 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pp. 82–88, doi: 10.1109/SEAMS.2016.017.

[116] Matena, V., A. Masrur and T. Bures (2017), An Ensemble-Based Approach for Scalable QoS in Highly Dynamic CPS, in *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pp. 234–238, doi: 10.1109/SEAA.2017.62.

[117] Mosterman, P. and H. Vangheluwe (2004), Computer automated multi-paradigm modeling: An introduction, *Simulation*, **80**, 9, pp. 433–450, doi: 10.1177/0037549704050532, cited By 0.
https://www.scopus.com/inward/record.uri?eid=2-s2.0-11844295391&partnerID=40&md5=c7c25773d89faecf261ca09095f00b04

[MoTE] MoTE (2015), MoTE Project Homepage, http://www.mdelab.org/mdelab-projects/mote-a-tgg-based-model-transformation-engine/.

[OSLC] OSLC (accessed 2015), OSLC Project Homepage, http://open-services.net/.

[120] Pezze, M. and M. Young (1997), Constructing multi-formalism state-space analysis tools: Using rules to specify dynamic semantics of models, pp. 239–249, doi: 10.1109/ICSE.1997.610261, cited By 0.
https://www.scopus.com/inward/record.uri?eid=2-s2.0-0030615480&partnerID=40&md5=4801673c307437dddecc1867abf58a3e

[121] Qamar, A., S. Herzig, C. Paredis and M. Torngren (2015), Analyzing semantic relationships between multiformalism models for inconsistency management, pp. 84–89, doi: 10.1109/SYSCON.2015.7116733, cited By 0.
https://www.scopus.com/inward/record.uri?eid=2-s2.0-84941265770&partnerID=40&md5=03008fd98b3c5a0bf83fbe3f72716c92

[122] Raiteri, D. C., M. Iacono, G. Franceschinis and V. Vittorini (2004), Repairable Fault Tree for the Automatic Evaluation of Repair Policies, in *DSN*, IEEE Computer Society, pp. 659–668, ISBN 0-7695-2052-9.

[123] Reza, H. and E. Grant (2004), Model oriented software architecture, volume 2, pp. 4–5, doi: 10.1109/CMPSAC.2004.1342651, cited By 0.
https://www.scopus.com/inward/record.uri?eid=2-s2.0-18844462472&partnerID=40&md5=d0ecfcca48d4b45b6945cec427b1ec26

[124] Rivera, J. E., D. Ruiz-Gonzalez, F. Lopez-Romero, J. Bautista and A. Vallecillo (2009), Orchestrating ATL Model Transformations, in *In Proc. of MtATL 2009: 1st International Workshop on Model Transformation with ATL*, Ed. F. Jouault, Nantes, France, pp. 34–46.

[125] Sanders, W. H. (1999), Integrated frameworks for multi-level and multi-formalism modeling, in *Petri Nets and Performance Models, 1999. Proceedings. The 8th International Workshop on*, pp. 2–9, ISSN 1063-6714, doi: 10.1109/PNPM.1999.796527.

[126] Seibel, A., R. Hebig and H. Giese (2012), Traceability in Model-Driven Engineering: Efficient and Scalable Traceability Maintenance, in *Software and Systems Traceability*, Eds. J. Cleland-Huang, O. Gotel and A. Zisman, Springer London, pp. 215–240.

[127] Seibel, A., R. Hebig, S. Neumann and H. Giese (2011), A Dedicated Language for Context Composition and Execution of True Black-Box Model Transformations, in *4th International Conference on Software Language Engineering (SLE 2011) , Braga, Portugal*.

[128] Seibel, A., S. Neumann and H. Giese (2010), Dynamic hierarchical mega models: comprehensive traceability and its efficient maintenance, *Software and Systems Modeling*, **9**, 4, pp. 493–528.

[129] Simko, G., T. Levendovszky, S. Neema, E. Jackson, T. Bapty, J. Porter and J. Sztipanovits (2012), Foundation for model integration: Semantic backplane, in *ASME 2012 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, American Society of Mechanical Engineers, pp. 1077–1086.

[SmartEMF] SmartEMF (2008), SmartEMF Project Homepage, http://www.itu.dk/~hessellund/smartemf/.

[131] Tekinerdogan, B. and K. Öztürk (2013), *Feature-Driven Design of SaaS Architectures*, Springer London, London, pp. 189–212, ISBN 978-1-4471-5031-2, doi: 10.1007/978-1-4471-5031-2_9. http://dx.doi.org/10.1007/978-1-4471-5031-2_9

[132] Trivedi, K. S. (2002), SHARPE 2002: Symbolic Hierarchical Automated Reliability and Performance Evaluator, in *DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks*, IEEE Computer Society, Washington, DC, USA, p. 544, ISBN 0-7695-1597-5.

[133] Ujhelyi, Z., G. Bergmann, A. Hegedus, A. Horvath, B. Izso, I. Rath, Z. Szatmari and D. Varro (2015), EMF-IncQuery: An integrated development environment for live model queries, *Science of Computer Programming*, **98, Part 1**, pp. 80–99, fifth issue of Experimental Software and Toolkits (EST): A special issue on Academics Modelling with Eclipse (ACME2012).

[134] Vangheluwe, H. and J. De Lara (2003), Computer Automated Multi-Paradigm Modelling: Meta-Modelling and Graph Transformation, volume 1, pp. 595–603, cited By 14. https://www.scopus.com/inward/record.uri?eid=2-s2.0-1642478929&partnerID=40&md5=fac4e33882516aff8f15a69428f482ff

[135] Vangheluwe, H., J. de Lara and P. J. Mosterman (2002), An introduction to multiparadigm modelling and simulation, in *Proceedings of the AIS2002 Conference (2002)*.

[136] Vignaga, A., F. Jouault, M. Bastarrica and H. Brunelière (2013), Typing artifacts in megamodeling, *Software & Systems Modeling*, **12**, pp. 105–119.

[137] Vittorini, V., G. Franceschinis, M. Gribaudo, M. Iacono and N. Mazzocca (2002), DrawNet++: Model Objects to Support Performance Analysis and Simulation of Complex Systems, in *Proc. of the 12th Int. Conference on Modelling Tools and Techniques for Computer and Communication System Performance Evaluation (TOOLS 2002)*, London, UK.

[138] Wohlrab, R., E. Knauss, J.-P. Steghöfer, S. Maro, A. Anjorin and P. Pelliccione (2018), Collaborative traceability management: a multiple case study from the perspectives of organization, process, and culture, *Requirements Engineering*, ISSN 1432-010X, doi:

10.1007/s00766-018-0306-1.
https://doi.org/10.1007/s00766-018-0306-1

[139] WÃd'tzoldt, S., S. Neumann, F. Benke and H. Giese (2012), Integrated Software Development for Embedded Robotic Systems, in *Proceedings of the 3rd International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, volume 7628 of *Lecture Notes in Computer Science*, Eds. I. Noda, N. Ando, D. Brugali and J. Kuffner, Springer Berlin Heidelberg, volume 7628 of *Lecture Notes in Computer Science*, pp. 335–348.

[140] Zeigler, B. (2006), Embedding DEV&DESS in DEVS: Characteristic behaviors of hybrid models, pp. 125–132, cited By 0.
https://www.scopus.com/inward/record.uri?eid=2-s2.0-84890259506&partnerID=40&md5=822222361e7255f743360e9943966ba4

[141] Zeigler, B. and H. Praehofer (1998), Interfacing continuous and discrete models for simulation and control, *SAE Technical Papers*, doi: 10.4271/981725, cited By 0.
https://www.scopus.com/inward/record.uri?eid=2-s2.0-84877510757&partnerID=40&md5=09796ecb5de062e13b3dad7a950df982