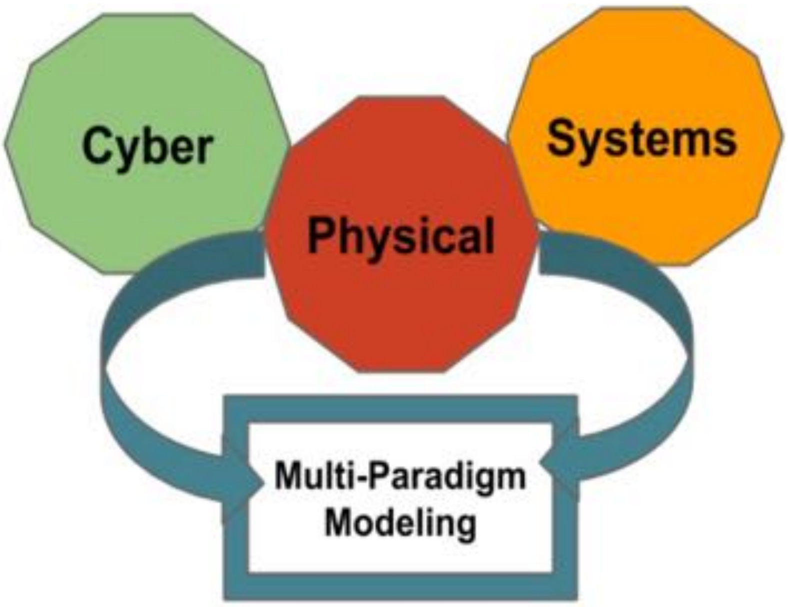

ICT COST Action IC1404

Framework to Relate / Combine Modeling Languages and Techniques

Rima Al-Ali, Ankica Barisic, Fernando Barros, Dominique Blouin, Etienne Borde, Maria Ganza, Holger Giese, Mauro Iacono, Peter Kieseberg, Bedir Tekinerdogan, Hans Vangheluwe, Constantin Bala Zamfirescu

Deliverable: WG1.2

Core Team	**Document Info**	
University of Antwerp	Deliverable	WG1.2
New University of Lisbon	Dissemination	Restricted
University of Malaga	Status	Final
Hasso-Plattner Inst., Potsdam	Doc's Lead Partner	Hasso-Plattner Inst.
University of Twente	Date	January 8, 2017
fortiss GmbH, München	Version	1.0
University of Lisbon	Pages	57

Contents

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 2 and then an ontology of Multi-Paradigm Modeling in chapter 3. Then, these ontologies are combined to define an ontology of Multi-Paradigm Modeling for Cyber Physical Systems presented in chapter 4. 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 (67) 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.

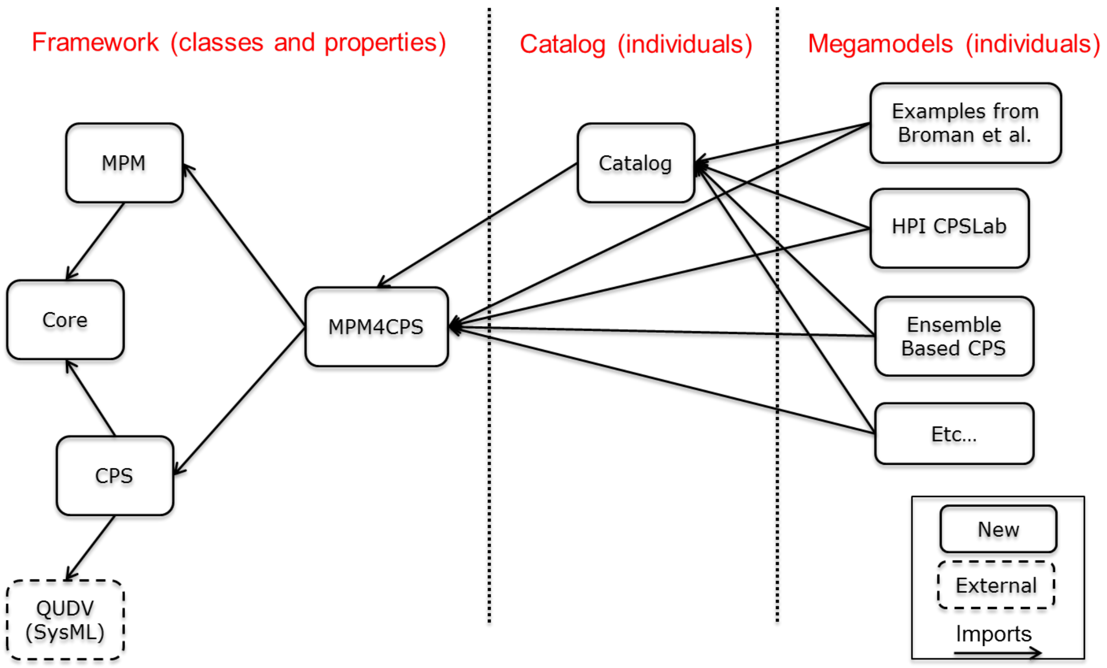

Figure 1.1: Overview of the structure of the MPM4CPS ontology

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 2, the ontology of Multi-Paradigm Modeling presented in chapter 3 and the combined ontology of Multi-Paradigm Modeling for Cyber Physical Systems presented in chapter 4.

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 (67) 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 (67) 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.

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 (67), 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 (83). 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 diagram (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 subfeatures) 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.
- A excludes B - A and B cannot be part of the same product.

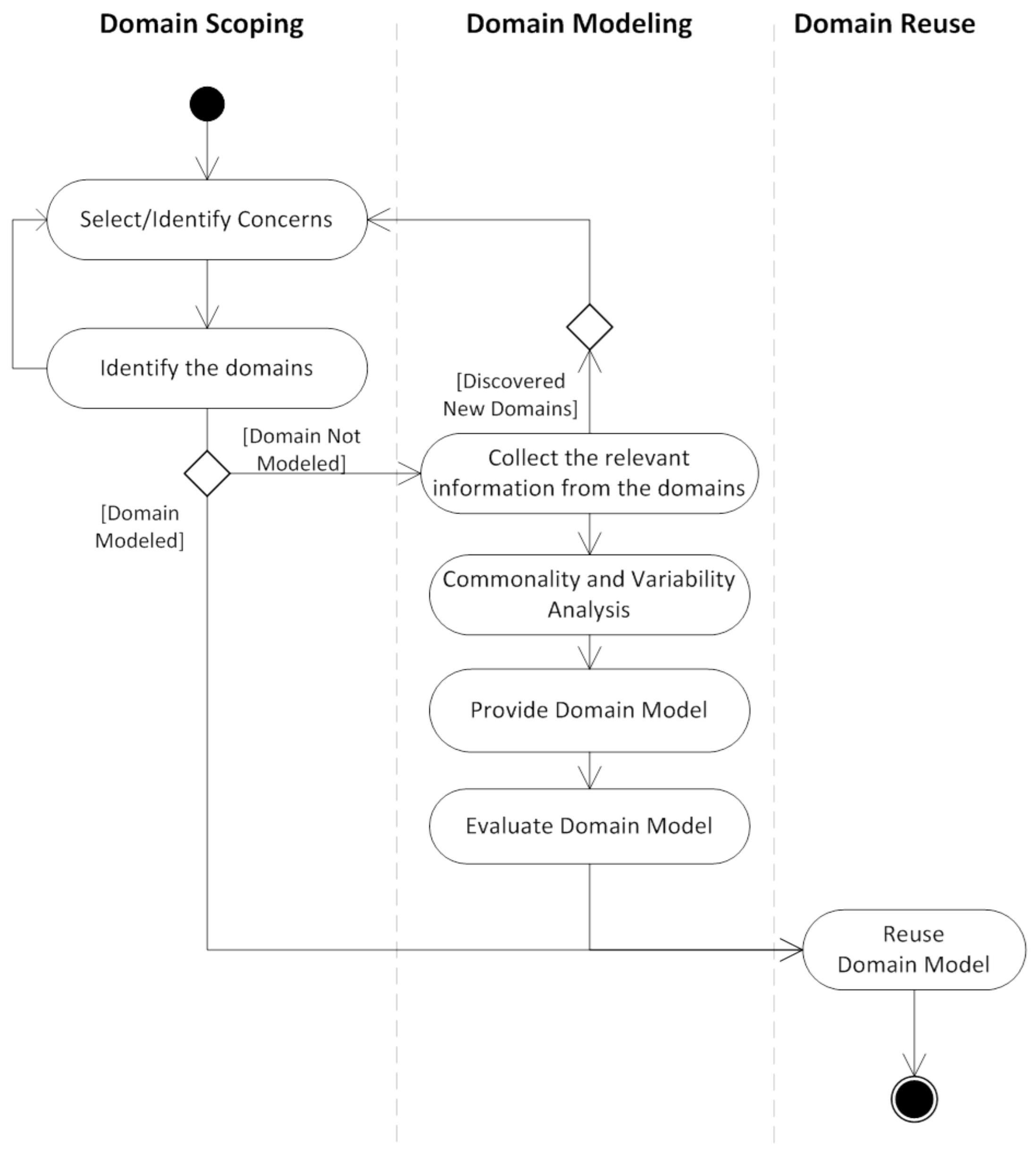

Figure 1.2: Common structure of domain analysis methods (adopted from: (110))

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" (73). 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 (48).

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. In the next year, it is planned to also integrate the CPS ontology into the Protege OWL technical space.

2 Ontology of Cyber-Physical Systems

2.1 State-of-the-art

Cyber-Physical Systems (CPS) are systems that integrate the physical world with the virtual, computational world. The components include software systems, communications technology, and sensors/actuators that interact with the real world. CPSs can provide an innovation for various kinds of industries, creating entirely new markets and platforms for growth. Example application domains of CPS include healthcare, transportation, precision agriculture, energy conservation, environmental control, avionics, critical infrastructure control (electric power, 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 would be enormous.

CPS requires transdisciplinary approaches merging different engineering disciplines. As such, an important challenge of CPS is to conjoin abstractions of the various engineering disciplines and the models for physical processes including differential equations, stochastic processes, etc. However, currently there is no common design and modeling approach yet to all the involved disciplines in CPS. For the design of CPS actually engineers from various disciplines need to be able to explore system designs collaboratively, allocating responsibilities to software and physical elements, and analyzing trade-offs between them. Recent advances show that coupling disciplines by using co-simulation will allow disciplines to cooperate without enforcing new tools or design methods. TODO: Develop the need for MPM?

CPS builds on embedded systems, which are a self-contained systems that incorporates elements of control logic and real world interaction. An embedded system is typically a single device, while CPSs include many constituent systems. Further, embedded systems are specifically designed to achieve a limited number of tasks, often with limited resources. A CPS in contrast operates at a much larger scale, potentially including many embedded systems or other CPS elements including human and socio-technical systems. It is expected that further developments in engineering will improve the link between computational and physical elements by means of intelligent mechanisms, thereby further increasing the impact of cyber-physical systems.

2.2 Ontology

After a domain analysis to CPS we have derived the feature model as shown in figure 2.1. A CPS system can be designed together or separately. In the latter case the various different components which were separately designed need to be integrated. A CPS has different component types which can be computational or physical. Further, 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 CPSs these two features are considered mandatory and not alternative. Components in a CPS can be of different same type or different types thereby distinguishing between homogenous vs. heterogeneous CPSs. The final feature in the feature model presents the various application domains in which CPSs can be applied including manufacturing, healthcare, transportation etc.

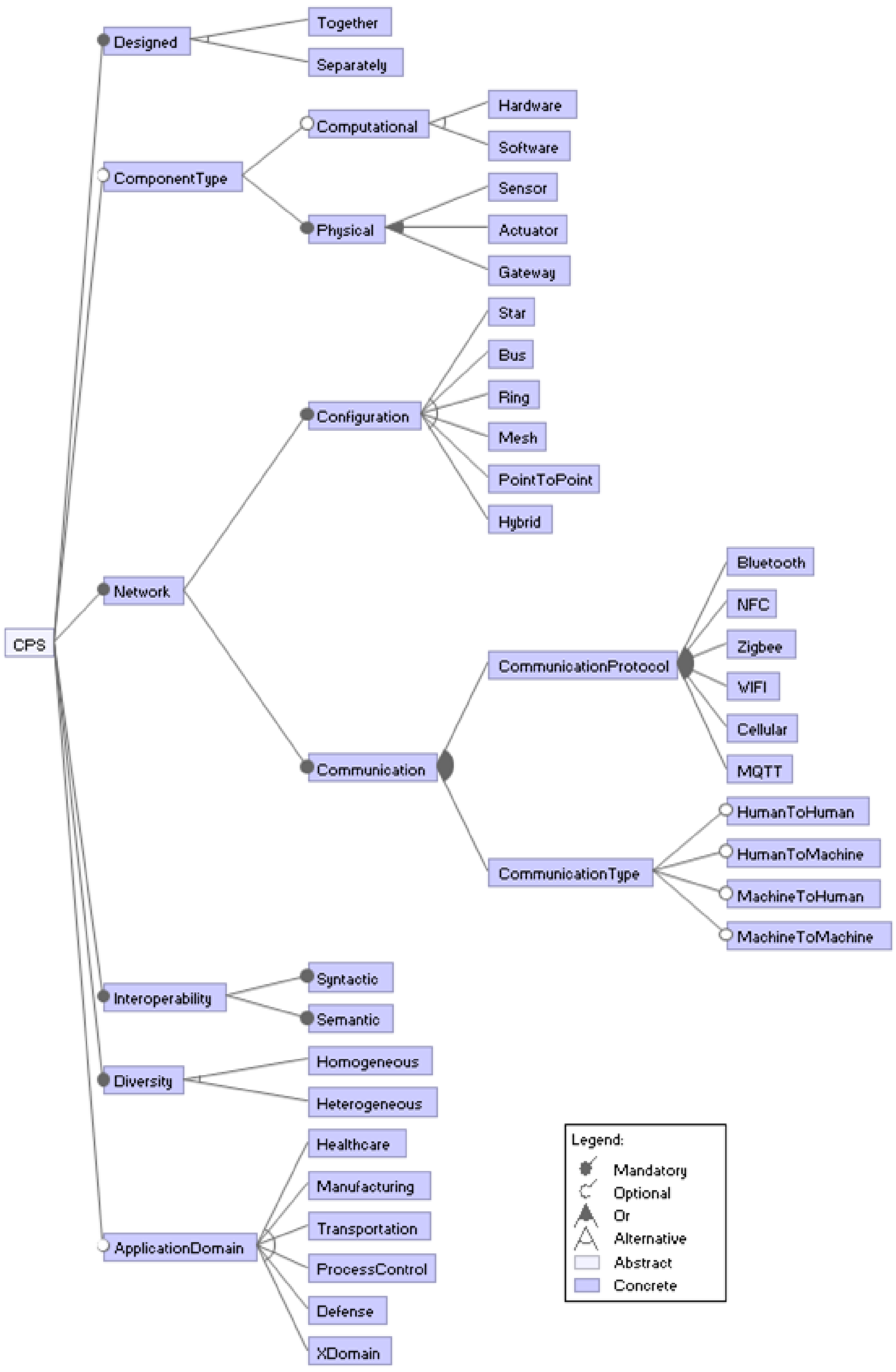

Figure 2.1: Feature Model of a CPS representing common and variant properties

2.2.1 Ontology Diagram

In this section we describe the metamodel **(TODO: convert to OWL)** for CPS which represents the concepts and their relations. The metamodel is shown in figure 2.2. A CPS system consists of CPS Components that interact using one or more Communication Protocols that run 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 affect them. Essentially, sensors take a mechanical, optical, magnetic or thermal signal and convert this 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.

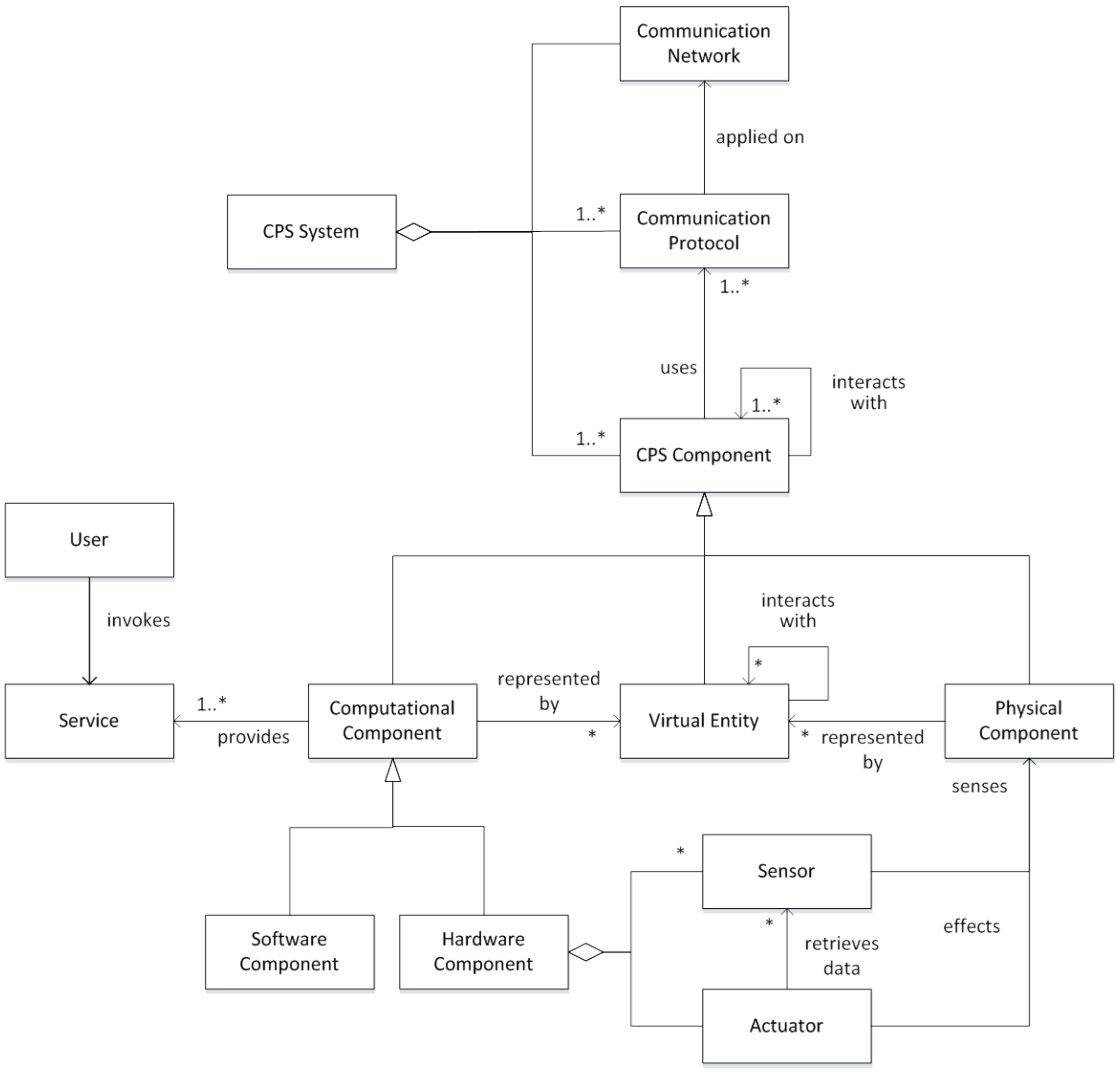

Figure 2.2: Basic concepts of CPSs

2.2.2 Architecture

The architecture of a CPS represents the gross level structure of the system consisting of cyber physical components. The current architecture design approaches for CPSs seem to be primarily domain-specific and no standard reference architecture has been defined yet. The development of an ontology for CPS also paves the way and supports the efforts for designing a reference architecture. A CPS reference architecture defines the generic structure of CPS architectures for particular application domains.

Figure 2.3 presents a layered view of a CPS architecture. Hereby, a layer simply represents a grouping of modules that offers a cohesive set of services. The reference architecture consists of four layers including device layer, network layer, CPS layer, application layer, and business layer. The CPS component layer includes the capabilities for the CPS components in the network. The network layer provides functionality for networking connectivity and transport capabilities. 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. The application layer contains the applications. The Security layer is a side-car layer relating to the other layers, and provides the security functionality. Finally, the management layer supports capabilities such as device management, local network topology management, and traffic and congestion management.

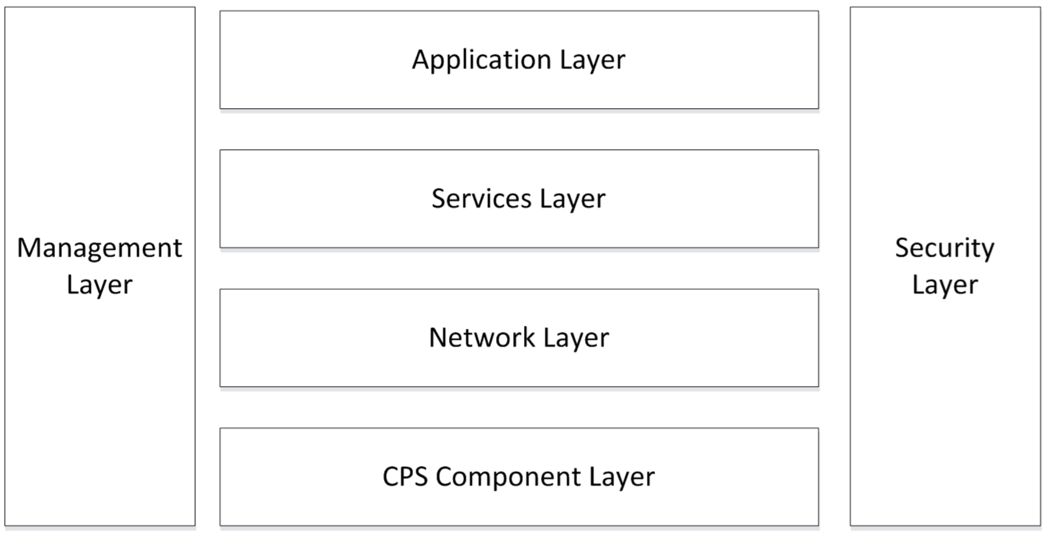

Figure 2.3: Layered view for CPS architectures

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 analyse risks, balance trade-offs, plan the implementation project and allocate tasks.

3 Ontology of Multi-Paradigm Modeling

3.1 State-of-the-art

Developing nowadays complex systems with MPM requires *Global Model Management* (*GMM*) (21; 58) 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 (68). 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). This requires coordinating all activities operating on the models across these dimensions to ensure their consistency. However, inconsistency management goes beyond simply identifying and resolving inconsistencies, since as pointed out in (60), inconsistencies may need to be tolerated at some stage of development. Therefore, living with inconsistencies must be manageable and consequently, an approach is required to detect, resolve, but also tolerate inconsistencies for a considerable amount of time during development.

The development activities for nowadays complex systems and in particular CPSs encompass multiple domains and teams, where each team is using its own set of modeling languages thus requiring proper integration of these languages. Indeed, it has been shown that using a single language to cover all domains would lead to very large monolithic languages not easily customizable for the development environment and tools needed by development organizations. These considerations lead to *Multi-Paradigm Modeling* (*MPM*) (114), 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 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 (21). *Mega modeling* (22; 58) 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 I) models and modeling languages, II) model operations, and III) mega models.

3.1.1 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.

3.1.1.1 Links

All approaches make use of some kind of *trace links* between models and their model elements to integrate models. In this project, we adopt the definitions of traceability proposed by the Center of Excellence for Software Traceability (CoEST) (CoEST). A *trace link* is *"...a specified association between a pair of artifacts, one comprising the source artifact and one comprising the target artifact..."*. Following the CoEST again, trace links are specialized into traces between the *vertical* and *horizontal* dimensions. Hence, a *vertical* trace *"...links artifacts at different levels of abstraction so as to accommodate lifecycle-wide or end-to-end traceability, such as from requirements to code..."*. An horizontal trace links *"...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"*.

There is a plethora of approaches (e.g., (AMW; Epsilon; 59; 92; 77; 108; 28) (MoTE)) making use of trace links to integrate models. 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 meta models of the models to be linked. Similarly, the Epsilon framework (Epsilon) provides a tool (ModeLink) to establish correspondences between models. MegaL Explorer (59) supports relating heterogeneous software development artifacts which do not necessary have to be models or model elements using predefined relation types. SmarfEMF (92) is another tool for linking models based on annotations of Ecore meta models 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 (77) 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 (108) and (28) 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 (107; 105)(20), 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 (107; 105)(20), all approaches suffer from being dependent on the set of integrated languages, thus requiring to better support modularity. Furthermore, only (MoTE)(107; 105)(20) supports automated management of traceability links.

3.1.1.1.1 Interfaces

In addition to links, a few more sophisticated approaches (e.g., (89; 78; 82)) introduce the concept of *model interface* (*int.* column in Table **??**) for specifying how models can be linked. In (89), 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 (78; SmartEMF) proposes a more generic concept of model interface to track dependencies between models and meta models and provide automated compatibility checks. Composite EMF Models (82; 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.

3.1.1.1.2 Meta Model Composition

Some approaches (e.g., (Kompren; Kompose; 57; EMF Views) (25)) consider the construction of view meta models in terms of other meta models or language fragments. In (57), 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. In (24), a language and tool (Kompren) (Kompren) are proposed to specify and generate slices of meta models via the selection of classes and properties of an input meta model. A reduced meta model is then produced from the input meta model. However the produced meta model must be completely regenerated when the input meta model is changed. Such is the case for the Kompose approach (Kompose), which on the contrary to Kompren proposes to create *compound meta models,* where a set of visible model elements from each combined meta models is selected, and optionally related. The EMF Views (EMF Views; 44) provides similar approach however without the need to duplicate the meta model elements as opposed to Kompose and Kompren where a new meta model is created. These virtual view meta models seem to be usable transparently by tools. Finally, the Global Model Management language (GMM*)[1] (25) 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 meta models already integrated within the GMM* language.

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

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 (23), 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; 112), a graph-based query language (EMF-IncQuery) relying on incremental pattern matching for improved performance is also proposed. In (53), 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 (72), this approach is extended for the case where the constraints themselves may change besides the constrained models. Finally in (37), 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.

3.1.2 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*).

3.1.2.1 Flow Composition Approaches

Formal United System Engineering Development (FUSED) (28) 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. Such is the case of 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 (115), 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 (103) 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 (93) 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* (25) also supports model transformation chaining, but through the specification of relations between models of specific meta models 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.

3.1.2.2 Context Composition Approaches

A few approaches allow context composition of model operations. In (57) as mentioned above, an approach is described to combine independent model transformations resulting in extended transformations for corresponding extended meta models. In (51), an approach is described for specifying the construction of view models 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 with bi-directionality the MoTCoF language (106) allows for both flow and fine grained context composition of model transformations. An advantage over (57) 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.

3.1.3 Mega Models 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.

3.1.3.1 Integration Language and other Approaches

The CyPhy (108) used in the GME modeling tool (GME) and FUSED (28; 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 (77) 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[2] 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, this approach necessitating the conversion of all models as a RDF graph is not incremental and will not scale for large models.

3.1.3.2 Mega Models

In this second strand, mega models 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 mega model 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 (93) as mentioned above is also a mega modeling language as well as MegaL Explorer (59) allowing to model the artifacts used in software development environments and their relations from a linguistic point of view. The involved software languages and 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 mega-model 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 (25) consists of a mega modeling language inspired from (74). Meta models 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 (26) 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 mega modeling languages are monolithic and as a result, predefined mega model fragments cannot be composed and reused to avoid rebuilding complete mega model specifications from scratch for new projects. Among these mega modeling approaches, only FTG+PM, GMM* and (107; 105) address the automated execution of mega models in response to model changes or modeling events from the tool's user interface. GMM* and (107; 105) pro-

[2]An automated method making use of Bayesian Belief Networks is also under study (76).

vide incremental execution of mega models to some extent by re-evaluating only the relations concerned with the detected model changes.

3.1.3.3 Semantic Integration

Multiformalism modeling

According to (96), 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 (49) 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 (102): 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) 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 (94).

Metamodeling is an important resource for both performance oriented approaches (88)(113) 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 multiformalism with both continuous and discrete formalisms, and are thus capable of modeling natural systems in a better way (119). 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 (118), (15) and (16), 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 (9)). In the second group, the approaches have been implemented in a number of different tools, with different backgrounds, such as SHARPE, SMART, DEDS, AToM, 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 (111) 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 (41)(42)(40) supports the specification and solution, by simulation or approximation, of complex discrete-state systems. DEDS (19) provides a common abstract notation in which submodels written in different formalisms are translated. Möbius (104)(43)(47)(50) 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. AToM (90)(52) exploits metamodeling to implement model transformations, used to solve models by its solver. OsMoSys (62)(116)(63)(64)(70) and SIMTHESys (11)(13)(80)(79) 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 (94), 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 (99)) 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 OsMoSys or in some SIMTHESys solvers(18)), or by transformation (e.g. in AToM, or in (27)). Noticeable are the approaches that exploit mean field analysis to cope with very large space states (e.g. (29) or (38)).

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 (91); performance evaluation of Service Oriented Architecture have been studied in (1) and (81); cardiovascular system and its regulation has been studied, with a hybrid approach, in (75); interdependencies in electric power systems have been studied in (39); the ERMTS/ETCS European standard for high speed trains has been studied in (61); security attacks have been studied in (71); exceptions aware systems have been studied in (17); effects of software rejuvenation techniques have been studied in (10); NoSQL systems have been studied in (14). Multiformalism has been also applied as an implementation technique to provide higher level tools or formalisms: in (101) a flexible, optimized Repairable Fault Tree modeling and solution approach is presented; a performance oriented model checking example is given in (12); an analysis framework for detecting inconsistencies in high level semantic relationships between models has been developed in (100).

3.2 Ontology Overview

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

Figure 3.1 shows an overview of the MPM ontology. The details of each concept are provided in the following subsections.

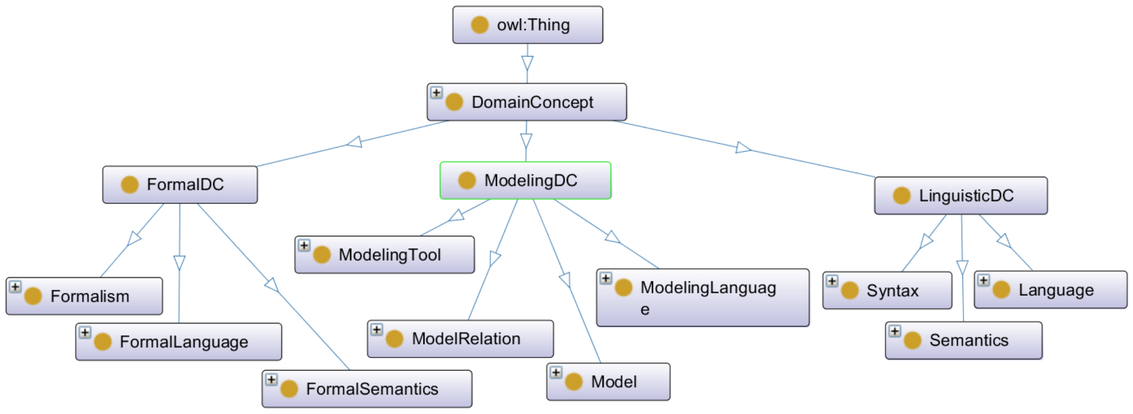

Figure 3.1: Overview of the MPM ontology

3.3 Domain Concepts

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

3.3.1 Formal Domain Concepts

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

3.3.1.1 Architecture

Subclass of:

- **FormalSemantics** (see section 3.3.2.14)

https://en.wikipedia.org/wiki/Systems_architecture

3.3.1.2 Behavioral

Subclass of:

- **Architecture** (see section 3.3.2.2)

«TODO: Provide rdfs:comment annotation assertion»

3.3.1.3 BehavioralConstraintLanguage

Subclass of:

- **ConstraintLanguage** (see section 3.3.2.9)

«TODO: Provide rdfs:comment annotation assertion»

3.3.1.4 Centralized

Subclass of:

- **Deployment** (see section 3.3.2.10)

«TODO: Provide rdfs:comment annotation assertion»

3.3.1.5 Communication

Subclass of:

- **Behavioral** (see section 3.3.2.4)

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.

3.3.1.6 ConstraintLanguage

Subclass of:

- **FormalLanguage** (see section 3.3.2.13)

«TODO: Provide rdfs:comment annotation assertion»

3.3.1.7 Deployment

Subclass of:

- **FormalSemantics** (see section 3.3.2.14)

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.

3.3.1.8 Distributed

Subclass of:

- **Deployment** (see section 3.3.2.10)

«TODO: Provide rdfs:comment annotation assertion»

3.3.1.9 FormalLanguage

Subclass of:

- **FormalDC** (see section 3.3.1)
- **Language** (see section 3.3.2.16)

In mathematics, computer science, and linguistics, a formal language is a set of strings of symbols together with a set of rules that are specific to it.

3.3.1.10 FormalSemantics

Subclass of:

- **FormalDC** (see section 3.3.1)
- **Semantics** (see section 3.3.2.19)

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.

3.3.1.11 Formalism

Subclass of:

- **FormalDC** (see section 3.3.1)
- **Principle** (see section **??**)

In foundations of mathematics, philosophy of mathematics, and philosophy of logic, formalism is a theory that holds that statements of mathematics and logic can be considered to be statements about the consequences of certain string manipulation rules.

3.3.1.12 Logic

Subclass of:

- **Formalism** (see section 3.3.1.11)

«TODO: Provide rdfs:comment annotation assertion»

3.3.1.13 Structural

Subclass of:

- **Architecture** (see section 3.3.2.2)

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

3.3.1.14 StructuralConstraintLanguage

Subclass of:

- **ConstraintLanguage** (see section 3.3.2.9)

«TODO: Provide rdfs:comment annotation assertion»

3.3.2 Linguistic Domain Concepts

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

3.3.2.1 AbstractSyntax

Subclass of:

- **Syntax** (see section 3.3.2.22)

metamodel

3.3.2.2 Architecture

Subclass of:

- **FormalSemantics** (see section 3.3.2.14)

https://en.wikipedia.org/wiki/Systems_architecture

3.3.2.3 ArchitectureDescriptionLanguage

Subclass of:

- **ModelingLanguage** (see section 3.3.3.12)

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 modelling 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.

3.3.2.4 Behavioral

Subclass of:

- **Architecture** (see section 3.3.2.2)

«TODO: Provide rdfs:comment annotation assertion»

3.3.2.5 BehavioralConstraintLanguage

Subclass of:

- **ConstraintLanguage** (see section 3.3.2.9)

«TODO: Provide rdfs:comment annotation assertion»

3.3.2.6 Centralized

Subclass of:

- **Deployment** (see section 3.3.2.10)

«TODO: Provide rdfs:comment annotation assertion»

3.3.2.7 Communication

Subclass of:

- **Behavioral** (see section 3.3.2.4)

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.

3.3.2.8 ConcreteSyntax

Subclass of:

- **Syntax** (see section 3.3.2.22)

textual/graphics as in AADL

3.3.2.9 ConstraintLanguage

Subclass of:

- **FormalLanguage** (see section 3.3.2.13)

«TODO: Provide rdfs:comment annotation assertion»

3.3.2.10 Deployment

Subclass of:

- **FormalSemantics** (see section 3.3.2.14)

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.

3.3.2.11 Distributed

Subclass of:

- **Deployment** (see section 3.3.2.10)

«TODO: Provide rdfs:comment annotation assertion»

3.3.2.12 DomainSpecificLanguage

Subclass of:

- **Language** (see section 3.3.2.16)

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.

3.3.2.13 FormalLanguage

Subclass of:

- **FormalDC** (see section 3.3.1)
- **Language** (see section 3.3.2.16)

In mathematics, computer science, and linguistics, a formal language is a set of strings of symbols together with a set of rules that are specific to it.

3.3.2.14 FormalSemantics

Subclass of:

- **FormalDC** (see section 3.3.1)
- **Semantics** (see section 3.3.2.19)

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.

3.3.2.15 Graphical

Subclass of:

- **ConcreteSyntax** (see section 3.3.2.8)

«TODO: Provide rdfs:comment annotation assertion»

3.3.2.16 Language

Subclass of:

- **LinguisticDC** (see section 3.3.2)

Language is the ability to acquire and use complex systems of communication

3.3.2.17 ModelingLanguage

Subclass of:

- **Language** (see section 3.3.2.16)
- **ModelingDC** (see section 3.3.3)

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.

3.3.2.18 ProgrammingLanguages

Subclass of:

- **Language** (see section 3.3.2.16)

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.

3.3.2.19 Semantics

Subclass of:

- **LinguisticDC** (see section 3.3.2)

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.

3.3.2.20 Structural

Subclass of:

- **Architecture** (see section 3.3.2.2)

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

3.3.2.21 StructuralConstraintLanguage

Subclass of:

- **ConstraintLanguage** (see section 3.3.2.9)

«TODO: Provide rdfs:comment annotation assertion»

3.3.2.22 Syntax

Subclass of:

- **LinguisticDC** (see section 3.3.2)

«TODO: Provide rdfs:comment annotation assertion»

3.3.2.23 Textual

Subclass of:

- **ConcreteSyntax** (see section 3.3.2.8)

«TODO: Provide rdfs:comment annotation assertion»

3.3.2.24 TransformationLanguage

Subclass of:

- **Language** (see section 3.3.2.16)

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.

3.3.3 Modeling Domain Concepts

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

3.3.3.1 ApplicationMegaModel

Subclass of:

- **Megamodel** (see section 3.3.3.7)

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

3.3.3.2 ArchitectureDescriptionLanguage

Subclass of:

- **ModelingLanguage** (see section 3.3.3.12)

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 modelling 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.

3.3.3.3 CapturingOperation

Subclass of:

- **TransformationOperation** (see section 3.3.3.15)

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

3.3.3.4 ConfigurationMegaModel

Subclass of:

- **Megamodel** (see section 3.3.3.7)

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

3.3.3.5 IntegrationOperation

Subclass of:

- **ModelOperation** (see section 3.3.3.10)

The process to integrate many models together.

3.3.3.6 MegaModelFragment

Subclass of:

- **Model** (see section 3.3.3.8)

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.)

TODO: Do we add also application and configuration division here?

3.3.3.7 Megamodel

Subclass of:

- **Model** (see section 3.3.3.8)

A model that contains models and relations between them.

3.3.3.8 Model

Subclass of:

- **ModelingDC** (see section 3.3.3)

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

3.3.3.9 ModelConstraint

Subclass of:

- **Constraint** (see section 5.2.2.1)
- **ModelingDC** (see section 3.3.3)

A restriction over model that is input or output for relation

3.3.3.10 ModelOperation

Subclass of:

- **ModelRelation** (see section 3.3.3.11)

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

3.3.3.11 ModelRelation

Subclass of:

- **Relation** (see section 5.2.2.2)
- **ModelingDC** (see section 3.3.3)
- **Process** (see section **??**)

A relation that connects models.

3.3.3.12 ModelingLanguage

Subclass of:

- **Language** (see section 3.3.2.16)
- **ModelingDC** (see section 3.3.3)

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.

3.3.3.13 ModelingTool

Subclass of:

- **Tool** (see section 5.2.5.1)
- **ModelingDC** (see section 3.3.3)

Tools to model the system.

3.3.3.14 TracabilityRelation

Subclass of:

- **ModelOperation** (see section 3.3.3.10)

«TODO: Provide rdfs:comment annotation assertion»

3.3.3.15 TransformationOperation

Subclass of:

- **ModelOperation** (see section 3.3.3.10)

The process to transform one model to another.

3.4 Properties

3.4.1 hasConcerns

A concern of stakeholder.

Subproperty of: None

Domains:

- **Stakeholder** (see section 5.2.3.6)

Ranges:

- **Concern** (see section 5.2.3.1)

3.4.2 hasConstraint

A constraint applied on relation.

Subproperty of: None

Domains:

- **Relation** (see section 5.2.2.2)

Ranges:

- **Constraint** (see section 5.2.2.1)

3.4.3 hasContext

«TODO: Provide rdfs:comment annotation assertion»

Subproperty of: None

Domains:

- **ModelOperation** (see section 3.3.3.10)

Ranges:

- **ModelOperation** (see section 3.3.3.10)

3.4.4 hasEvolvedTo

An updated version of an element. (e.g.Tool evolved to another)

Subproperty of: None

Domains: None

Ranges: None

3.4.5 hasFormalLanguage

The relation defines the formal language for a formalism. It is an inverse of hasFormalism.

Subproperty of: None

Domains:

- **Formalism** (see section 3.3.1.11)

Ranges:

- **FormalLanguage** (see section 3.3.2.13)

3.4.6 hasFormalism

The relation defines the formalism for a formal language. It is an inverse of hasFormalLanguage.

Subproperty of: None

Domains:

- **FormalLanguage** (see section 3.3.2.13)

Ranges:

- **Formalism** (see section 3.3.1.11)

3.4.7 hasInput

The model need inputs to do perform the determined task.

Subproperty of:

- **isConnecting** (see section 3.4.24)

Domains:

- **ModelOperation** (see section 3.3.3.10)

Ranges:

- **Model** (see section 3.3.3.8)

3.4.8 hasInputModel

It describes the input model for a model relation.

Subproperty of: None

Domains:

- **ModelRelation** (see section 3.3.3.11)

Ranges:

- **Model** (see section 3.3.3.8)

3.4.9 hasLanguage

The model could be expressed in natural language or formal language that is why the range is Language class not the ModelingLanguage class.

Subproperty of: None

Domains:

- **Model** (see section 3.3.3.8)

Ranges:

- **Language** (see section 3.3.2.16)

3.4.10 hasMegaModelFragment

It describes the relation between megamodel and its fragments.

Subproperty of:

- **hasModel** (see section 3.4.11)

Domains:

- **Megamodel** (see section 3.3.3.7)

Ranges:

- **MegaModelFragment** (see section 3.3.3.6)

3.4.11 hasModel

It describes a recursive relation. e.g. MegaModelFragment hasModel xModel ...

Subproperty of: None

Domains:

- **Model** (see section 3.3.3.8)

Ranges:

- **Model** (see section 3.3.3.8)

3.4.12 hasModelConstraint

«TODO: Provide rdfs:comment annotation assertion»

Subproperty of:

- **hasConstraint** (see section 5.3.2)

Domains:

- **ModelOperation** (see section 3.3.3.10)

Ranges:

- **ModelConstraint** (see section 3.3.3.9)

3.4.13 hasModelOperation

It defines a model operation for a model.

Subproperty of:

- **hasModelRelation** (see section 3.4.14)

Domains:

- **Model** (see section 3.3.3.8)

Ranges:

- **ModelOperation** (see section 3.3.3.10)

3.4.14 hasModelRelation

It defines a model relation for a model.

Subproperty of: None

Domains:

- **Model** (see section 3.3.3.8)

Ranges:

- **ModelRelation** (see section 3.3.3.11)

3.4.15 hasNext

«TODO: Provide rdfs:comment annotation assertion»

Subproperty of: None

Domains:

- **Process** (see section **??**)

Ranges:

- **Process** (see section **??**)

3.4.16 hasOutput

The model provides results in the following form.

Subproperty of:

- **isConnecting** (see section 3.4.24)

Domains:

- **ModelOperation** (see section 3.3.3.10)

Ranges:

- **Model** (see section 3.3.3.8)

3.4.17 hasOutputModel

It describes the output model for a model relation.

Subproperty of: None

Domains:

- **ModelRelation** (see section 3.3.3.11)

Ranges:

- **Model** (see section 3.3.3.8)

3.4.18 hasProvider

The company/university which the tool is developed by.

Subproperty of: None

Domains: None

Ranges: None

3.4.19 hasPurpose

«TODO: Provide rdfs:comment annotation assertion»

Subproperty of: None

Domains:

- **Model** (see section 3.3.3.8)

Ranges:

- **Purpose** (see section 5.2.3.4)

3.4.20 hasRole

A role of stakeholder

Subproperty of: None

Domains:

- **Stakeholder** (see section 5.2.3.6)

Ranges:

- exactly 1 **Role** (see section 5.2.3.5)

3.4.21 hasTool

The tool that represent the Language. It is an inverse of isToolFor.

Subproperty of: None

Domains:

- **Language** (see section 3.3.2.16)

Ranges:

- **Tool** (see section 5.2.5.1)

3.4.22 isAppliedTo

A constraint is applied to a some element. (e.g. relation)

Subproperty of: None

Domains:

- **Constraint** (see section 5.2.2.1)

Ranges: None

3.4.23 isAppliedToModel

«TODO: Provide rdfs:comment annotation assertion»

Subproperty of:

- **isAppliedTo** (see section 5.3.6)

Domains:

- **Constraint** (see section 5.2.2.1)

Ranges:

- **Model** (see section 3.3.3.8)

3.4.24 isConnecting

A relation connects a model.

Subproperty of: None

Domains:

- **ModelOperation** (see section 3.3.3.10)

Ranges:

- **Model** (see section 3.3.3.8)

3.4.25 isExtending

Shows the extension of another language

Subproperty of: None

Domains:

- **Language** (see section 3.3.2.16)

Ranges:

- **Language** (see section 3.3.2.16)

3.4.26 isImplementedBy

«TODO: Provide rdfs:comment annotation assertion»

Subproperty of: None

Domains: None

Ranges: None

3.4.27 isPerformedBy

«TODO: Provide rdfs:comment annotation assertion»

Subproperty of: None

Domains: None

Ranges: None

3.4.28 isReturningTo

«TODO: Provide rdfs:comment annotation assertion»

Subproperty of: None

Domains:

- **IntegrationOperation** (see section 3.3.3.5)

Ranges:

- **IntegrationOperation** (see section 3.3.3.5)

3.4.29 isToolFor

The language that is represented by a tool. It is an inverse of hasTool.

Subproperty of: None

Domains:

- **Tool** (see section 5.2.5.1)

Ranges:

- **Language** (see section 3.3.2.16)

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

4.1 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 (31).

4.2 Ontology

Figure 4.1 shows the classes and properties of the MPM4CPS ontology. The MPM4CPSDC class groups the Viewpoint and View classes to constitute the MPM4CPS domain.

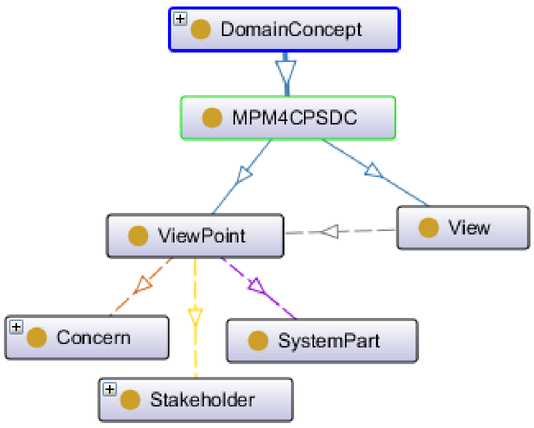

Figure 4.1: Overview of the MPM4CPS ontology

A viewpoint relates the *concerns* of *stakeholders* with *parts* of the system of interest. For example, a *control design performance* viewpoint may relate the controller designer with the concern of performance and the controller, software, sensors and actuators, computing platform and physical plant parts of the system.

In the next version of this ontology, other classes that relate the MPM and CPS domains will be added as needed.

5 Ontology of Core Common Concepts

5.1 Ontology Overview

The core ontology provides concepts that do not pertain to the CPS neither the MPM domain, but that are still required by one of the domains or both. It is a place to define concepts reusable by all the ontologies developed by WG1.

Figure 5.1 shows an overview of the core ontology. The details of each concept are provided in the following subsections.

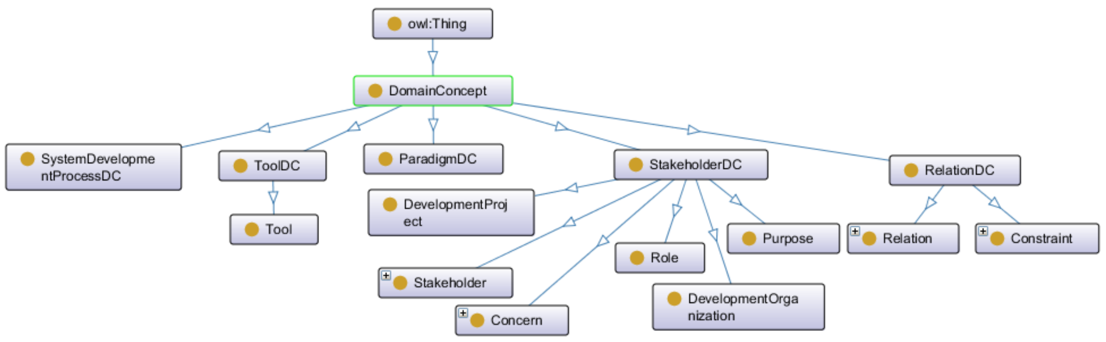

Figure 5.1: Overview of the core ontology

5.2 Domain Concepts

This ontology of shared concepts is divided into sub-domains as presented in the following subsections.

5.2.1 Paradigm Domain Concepts

This class groups domain concepts that are related to the paradigm aspects used by the MPM and / or CPS domains.

5.2.2 Relation Domain Concepts

This class groups domain concepts that are related to the relational aspects used by the MPM and / or CPS domains.

5.2.2.1 Constraint

Subclass of:

- **RelationDC** (see section 5.2.2)

A limitation or a restriction over a relation.

5.2.2.2 Relation

Subclass of:

- **RelationDC** (see section 5.2.2)

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

5.2.3 Stakeholder Domain Concepts

This class groups domain concepts that are related to the stakeholder aspects used by the MPM and / or CPS domains.

5.2.3.1 Concern

Subclass of:

- **StakeholderDC** (see section 5.2.3)

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.

5.2.3.2 DevelopmentOrganization

Subclass of:

- **StakeholderDC** (see section 5.2.3)

«TODO: Provide rdfs:comment annotation assertion»

5.2.3.3 DevelopmentProject

Subclass of:

- **StakeholderDC** (see section 5.2.3)

«TODO: Provide rdfs:comment annotation assertion»

5.2.3.4 Purpose

Subclass of:

- **StakeholderDC** (see section 5.2.3)

The object for which something exists

5.2.3.5 Role

Subclass of:

- **StakeholderDC** (see section 5.2.3)

A role (also rï£¡le 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.

5.2.3.6 Stakeholder

Subclass of:

- **StakeholderDC** (see section 5.2.3)
- **Stakeholder** (see section 5.2.3.6)

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

5.2.3.7 ToolProvider

Subclass of:

- **Stakeholder** (see section 5.2.3.6)

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

5.2.4 SystemDevelopmentProcess Domain Concepts

«TODO: Provide rdfs:comment annotation assertion»

TODO: To be reviewed. Should this be named SystemDevelopment or simply development?

5.2.5 Tool Domain Concepts

This class groups domain concepts that are related to the tooling aspects used by the MPM and / or CPS domains.

5.2.5.1 Tool

Subclass of:

- **ToolDC** (see section 5.2.5)

A programming tool or software development tool is a computer program that software developers use to create, debug, maintain, or otherwise support other programs and applications. The term usually refers to relatively simple programs, that can be combined together to accomplish a task, much as one might use multiple hand tools to fix a physical object. The ability to use a variety of tools productively is one hallmark of a skilled software engineer.

5.3 Properties

5.3.1 hasConcerns

A concern of stakeholder.

Subproperty of: None

Domains:

- **Stakeholder** (see section 5.2.3.6)

Ranges:

- **Concern** (see section 5.2.3.1)

5.3.2 hasConstraint

A constraint applied on relation.

Subproperty of: None

Domains:

- **Relation** (see section 5.2.2.2)

Ranges:

- **Constraint** (see section 5.2.2.1)

5.3.3 hasEvolvedTo

An updated version of an element. (e.g.Tool evolved to another)

Subproperty of: None

Domains: None

Ranges: None

5.3.4 hasProvider

The company/university which the tool is developed by.

Subproperty of: None

Domains: None

Ranges: None

5.3.5 hasRole

A role of stakeholder

Subproperty of: None

Domains:

- **Stakeholder** (see section 5.2.3.6)

Ranges:

- exactly 1 **Role** (see section 5.2.3.5)

5.3.6 isAppliedTo

A constraint is applied to a some element. (e.g. relation)

Subproperty of: None

Domains:

- **Constraint** (see section 5.2.2.1)

Ranges: None

6 Catalog of Megamodel Examples

This chapter presents examples of CPS development environments that have served for the development of the ontologies presented in the previous chapters. Classes of the provided ontologies have been instantiated to create individuals representing the example environments and analyzed to discover and fix shortcomings of the ontologies in an iterative manner.

6.1 Ensemble-based CPS

6.1.1 Overview

An Ensemble-Based Cyber-Physical System (EBCPS) is an emergent system that has autonomous components which form ensembles together depending on the context and in the aim of achieving determined goals (i.e. organizing and decision-making). Furthermore, the developers that target building such systems should consider as well the system nature of being distributed, decentralized, dynamic, self-adaptive and scalable.

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 tools.

Starting with modeling EBCPS requirements, Invariant Refinement Method (IRM) [(84)], 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 is part of a component, 2) or exchange knowledge between components which is an ensemble. Simply put, the IRM allows to connect the requirements to the architecture entities. Later on, the developer can transform the IRM model to code which is Java for the current represented tool-chain under DEECo specification (stands for Dependable Emergent Ensembles of Components).

Regarding the design and runtime parts (32)(2) , the team developed a runtime environment which is implemented in java and is known as jDEECo. The defined system respects the DEECo specification for each of its entities for both components and ensembles. Therefore, each role has a set of fields which represent the knowledge related to this specific role and it also includes a set of modes and their transitions. Thus, 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. Finally, each ensemble forms depending on the context and perform knowledge exchange between different components with determined roles.

Self-adaptation and analysis parts are reusable parts, where the self-adaptation is done by using IRM-SA engine that uses SAT solver to make decisions between the different paths. The analysis part is presented as an extension of mode-switch transition logic which includes three basic ideas: 1) using ordinary differential equation (ODE) to evaluate the inaccuracy boundaries of physical attributes(3), 2) statistical testing to use prediction depending on trends of historical data(34).

Another essential point that the EBCPS development environment 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 fitness model that filters out the less suitable compositions between all possible ones. Regarding the exchanging knowledge between components, it could be determined by context constraints or by adding boundaries on gossip protocol that is responsible of spreading information(33). Furthermore, DEECo allows a hierarchical composition for components and ensembles (36)(35).

Finally, the simulation part targets two domains for vehicles(85) and robots(95). 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, EBCPS 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 development many transformation and integration operations are involved which supports the idea of multi-paradigm modeling. These are further presented in the developed ontology of EBCPS briefly presented below.

6.1.2 Ontology

The structure of EBCPS is described using the ontologies provided by MPM4CPS by instantiating its classes to create a set of individuals representing EBCPS. It includes an EBCPS megamodel and its fragments defining all model operations for EBCPS as outlined in Figure 6.1. 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 a runtime model. The integration operations are basically links between the runtime model, the analysis models, the self-adaptation model, and the simulation models.

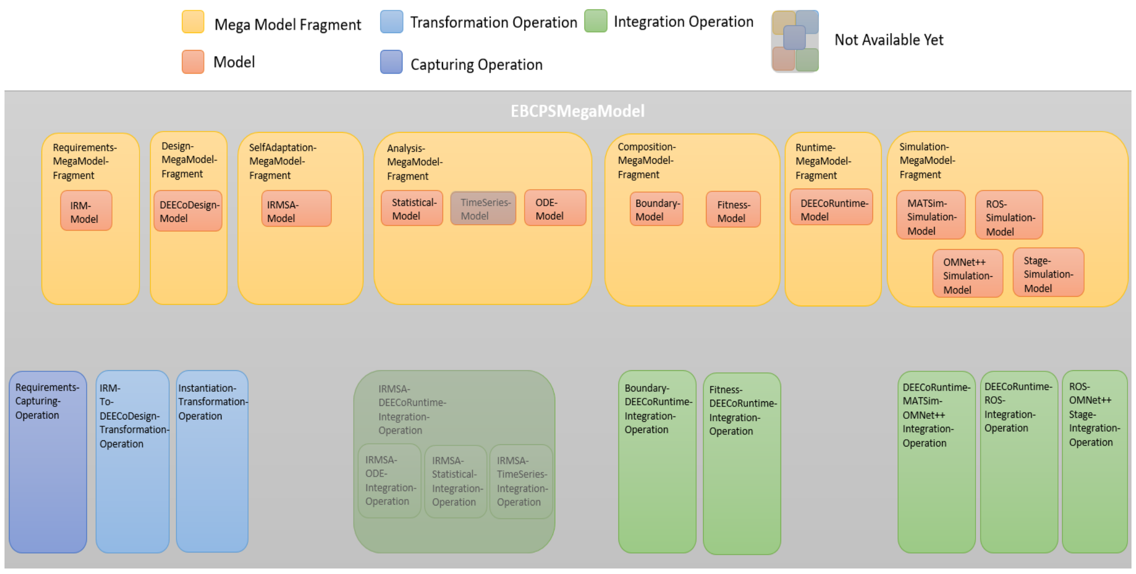
Mega Model Fragment
Model
Transformation Operation
Capturing Operation
Integration Operation
Not Available Yet
EBCPSMegaModel
Requirements-MegaModel-Fragment
IRM-Model
Design-MegaModel-Fragment
DEECoDesign-Model
SelfAdaptation-MegaModel-Fragment
IRMSA-Model
Analysis-MegaModel-Fragment
Statistical-Model
TimeSeries-Model
ODE-Model
Composition-MegaModel-Fragment
Boundary-Model
Fitness-Model
Runtime-MegaModel-Fragment
DEECoRuntime-Model
Simulation-MegaModel-Fragment
MATSim-Simulation-Model
ROS-Simulation-Model
OMNet++-Simulation-Model
Stage-Simulation-Model
Requirements-Capturing-Operation
IRM-To-DEECoDesign-Transformation-Operation
Instantiation-Transformation-Operation
IRMSA-DEECoRuntime-Integration-Operation
IRMSA-ODE-Integration-Operation
IRMSA-Statistical-Integration-Operation
IRMSA-TimeSeries-Integration-Operation
Boundary-DEECoRuntime-Integration-Operation
Fitness-DEECoRuntime-Integration-Operation
DEECoRuntime-MATSim-OMNet++-Integration-Operation
DEECoRuntime-ROS-Integration-Operation
ROS-OMNet++-Stage-Integration-Operation

Figure 6.1: EBCPS Mega-Model, which includes mega-model fragments, models and model operations

All this information and much more details of EBCPS have been captured in the EBCPS ontology. It is planned to generate a detailed specification of EBCPS automatically from this ontology as future work of the MPM4CPS action.

6.2 HPI CPS Lab[1]

As outlined in more detail from a conceptual point of view in (117), the presented *CPS Lab*[2] at the Hasso Plattner Institute (HPI)[3] at the University of Potsdam applied, adapted, and evaluated an existing industrial-strength development methodology from the automotive domain (30) (see also Figure 6.2) for the robotic system application domain. We therefore evaluated and adapted component-based approach using a MDE approach supporting also the combination of soft and hard real-time behavior.

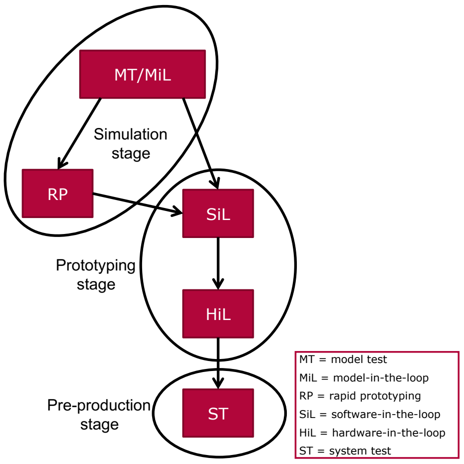

Figure 6.2: Overview of the methodology for modeling, verification, and validation employing simulation and testing (see (30))

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

We consider a robot system, 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 the batteries. The behavior must ensure strict constraints, such as it excludes that the battery is exhausted and with a lower priority ensures to transport pucks as requested. It also must perform reasonable with respect to some soft goals to minimize energy consumption and maximize throughput.

The upper drawing of Figure 6.3 depicts a structural overview of the robot system, while the photo in the middle of Figure 6.3 show the concrete setup. 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 AP. A robot RP transports the puck to a second room and drops it within the sorting area AS. Based on the current delivery status, the robot RS chooses one of the two booths and a band-conveyor transports the puck to the customer or stock delivery area (ACD, ASD) afterwards. In a third step, the robot RSt 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 CPS Lab robot laboratory consisting of three Robotino robots (see bottom of Figure 6.3). The robots can be equipped with several

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

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

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

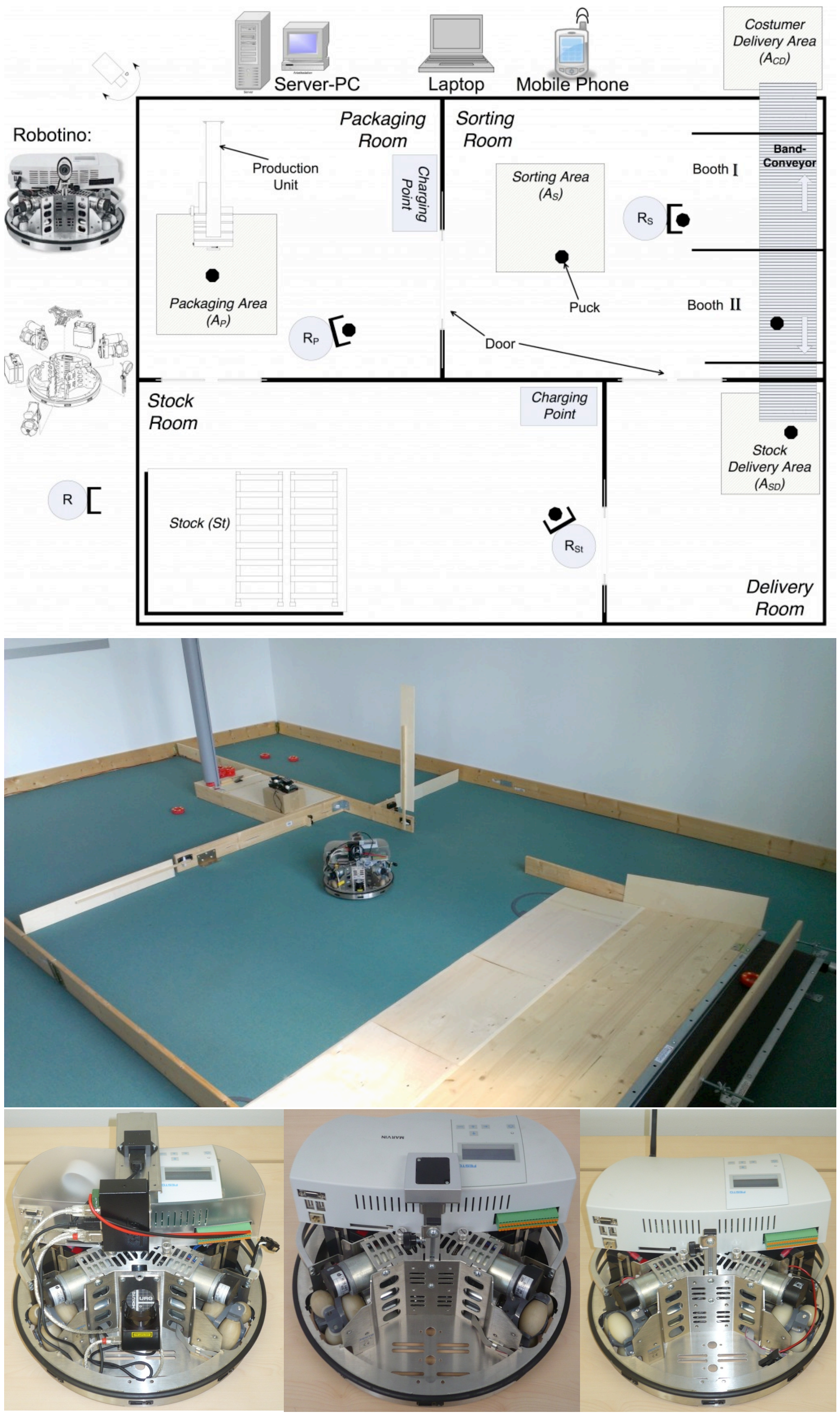

Figure 6.3: Overview of the employed evaluation scenario (see (117))

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). 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. Robots must fulfill different requirement, 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 reflect 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.

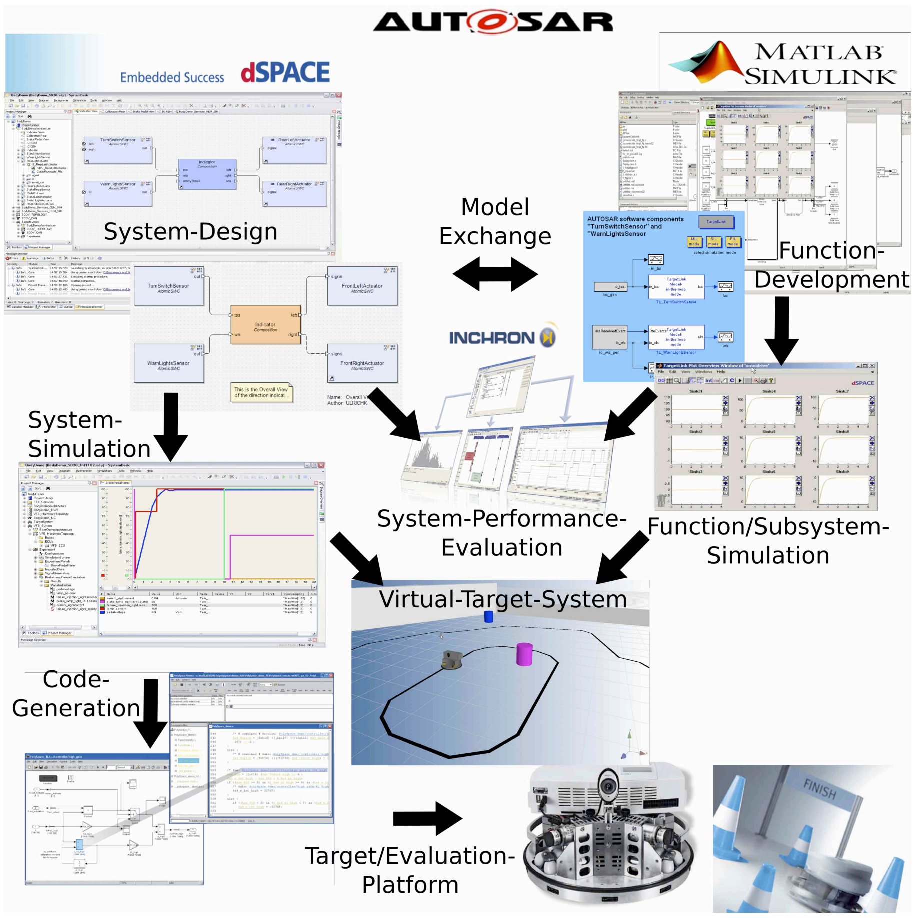

Figure 6.4: Overview of the employed tool chain (see (117))

In Figure the toolchain consisting 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 is depicted.

An additional element depicted in Figure is a prototypical tool we developed to map AUTOSAR models to a real-time simulator from INCRON to permit detailed performance evaluation early on in parallel to SiL activities. Another extension not depicted in Figure is a link between the SysML tool TOPCASED and the dSPACE tool SystemDesk for AUTOSAR that permits to derive the software and hardware architecture in AUTOSAR from the SysML system engineering models. However, to keep things simpler we in the following omit both extensions in our detailed description and the sketches of the megamodels.

6.2.1 Simulation Stage

The first stage supported is the 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 part 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.

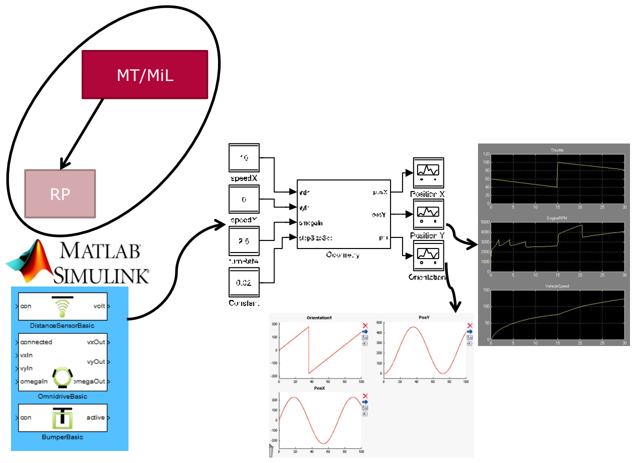

Figure 6.5: Overview of the model testing in the simulation stage of (30)

In a first activity named *model testing* (MT) (see Figure 6.5), 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 work as intended.

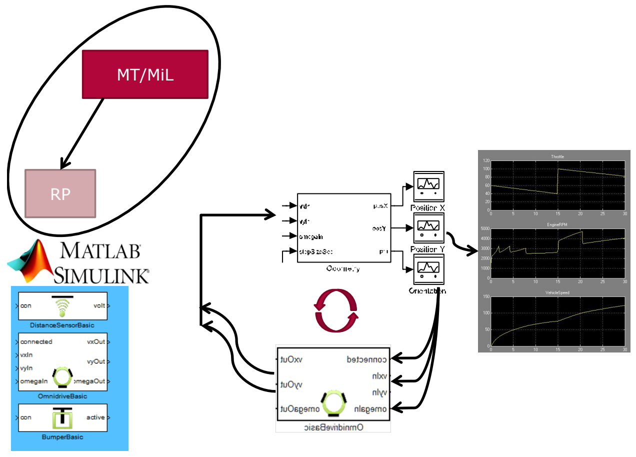

Figure 6.6: Overview of the model in loop simulation in the simulation stage of (30)

In a second step then 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) simulationas shown in Figure 6.6, which uses the feedback provided by the plant model to evaluate that the control behavior is as expected.

As the validity of plant models is often only rather limited when it comes to sophisticated aspects pf the physical behavior, as an additional step *rapid prototyping* as depicted in Figure 6.7 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

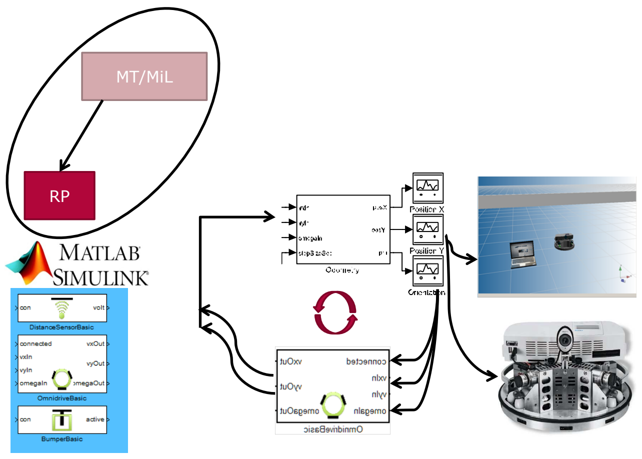

Figure 6.7: Overview of the rapid prototyping in the simulation stage of (30)

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.

While several tools are employed at several stages and activities, the models developed for each of these activities are quite different as visible in the megamodel depicted in Figure 6.8. For the model testing, 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 visualize the outcome with FESTO Robotino©View, a specific block set compatible with the FESTO Robotino-Library has to be employed.

6.2.2 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 take into account.

6.2.2.1 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.

As depicted in Figure 6.9, this is done by first define components and their communication via port types, messages, interfaces, and data types with AUTOSAR and map the beforehand considered functional partson 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 6.10, an important element of this refinement are 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 level must be defined including a mapping to hard and soft real-time task with proper levels for the priorities.

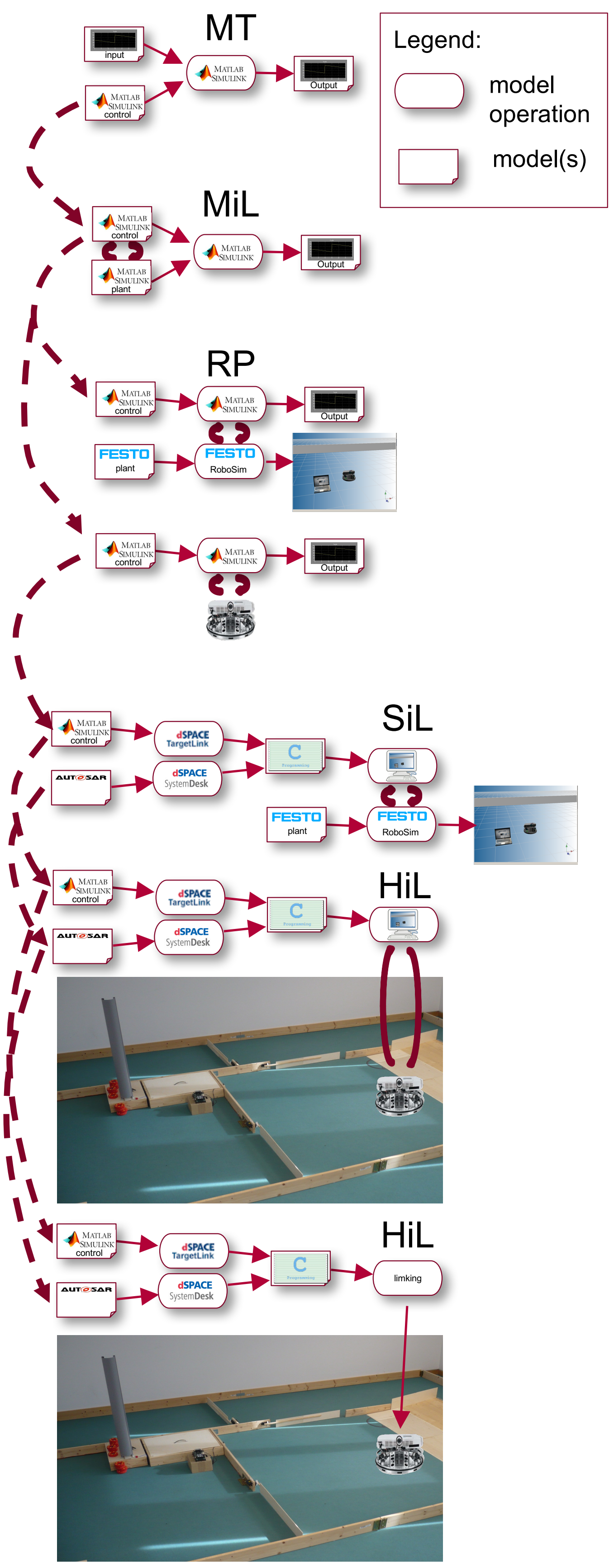

Figure 6.8: Sketch of the megamodel of the different stage of (30)

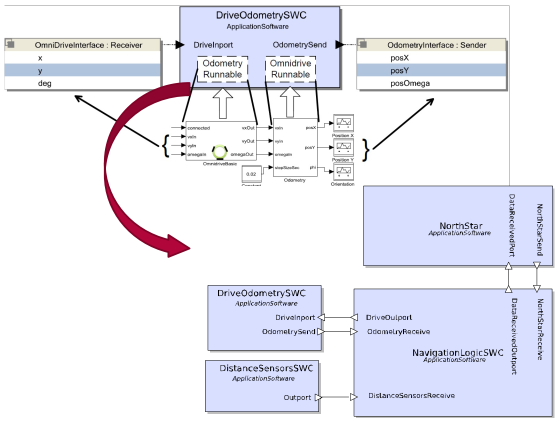

Figure 6.9: Overview of the definition of the software architecture in the prototyping stage of (30)

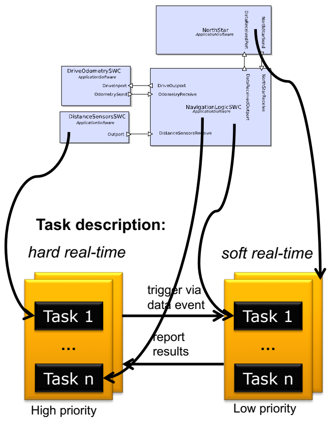

Figure 6.10: Overview of the mapping of the architecture to tasks and communication in the prototyping stage of (30)

6.2.2.2 Verification

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.

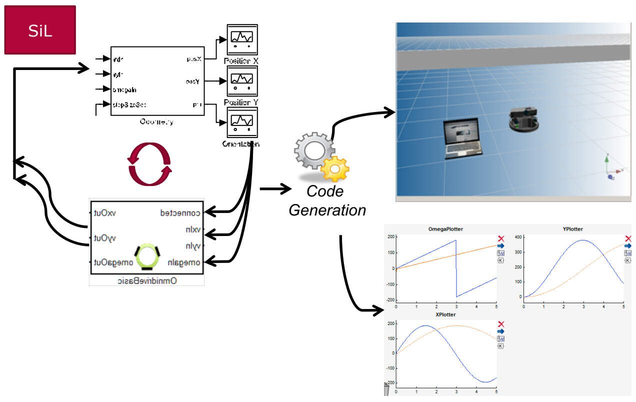

Figure 6.11: Overview of software-in-the-loop (sil) simulation in the prototyping stage of (30)

The *software-in-the-loop* (SiL) simulation at the prototyping stage as depicted in Figure 6.11, 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.

As we still do not use the real hardware, we still ignore resp. simplified 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.

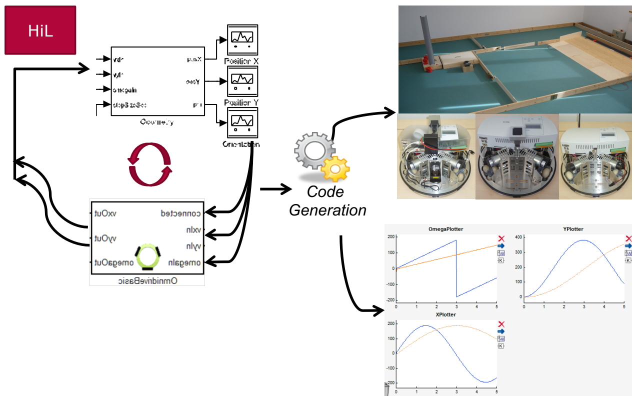

Figure 6.12: Overview of hardware-in-the-loop (hil) testing in the prototyping stage of (30)

By moving on to the lab itself, we can then also consider a *hardware-in-the-loop* (SiL) simulation at the prototyping stage as sketched in Figure 6.12, where besides the software that is generated or integrated also the specific characteristics of the robot hardware and labe environment and its hardware can be experienced.

As we now employ the real hardware, we no longer ignore resp. simplified 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.

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 6.8. 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 reconfigures such that either the complied software runs on a host computer and controls the Robotino robots remotely or the compiled and linked software runs directly on the Robotino robots.

6.2.3 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 sale 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.

6.2.4 Summary

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 a MDE approach where tool and library are integrated such that models drive the development only later code and configuration data automatically generated from the models is employed to consider more details concerning the verification, simulation, and testing. We put special focus on supporting also hard and soft real-time consideration which are oftentimes ignored in robotic development scenarios.

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 2 and then an ontology of Multi-Paradigm Modeling in chapter 3. Then, we presented an MPM4CPS ontology that combine these ontologies to cover Multi-Paradigm Modeling for Cyber Physical Systems presented in chapter 4. Finally, we presented two examples for CPS development cases and sketched how they fit into our framework and how their megamodels look like in chapter 6. These examples instantiate the MPM4CPS ontology and also make use of the catalog of languages and tools.

As future work, we plan to enrich the CPS ontology such that the catalog of languages can be better classified, to enrich the MPM ontology such that all needs of the example cases for CPS development approaches are covered, and to improve MPM4CPS ontology. Furthermore, we plan to develop documentation generators that will produce detailed specifications of CPS development environment examples to complete the overview descriptions provided in this document. Finally, besides improving the examples it would be also helpful to support more examples such that due to a higher heterogeneity in the group of examples a better coverage of the different aspects of CPS and MPM can be achieved.

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] 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.

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

[10] 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.

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

[12] 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*.

[13] 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

[14] 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.

[15] 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.

[16] 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.

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

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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*.

[23] 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.

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

[25] 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.

[26] 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.

[27] 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

[28] 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*.

[29] 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

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

[31] 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

[32] 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

[33] 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

[34] 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.

[35] 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

[36] 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

[37] 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.

[38] 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.

[39] 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

[40] 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.

[41] 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.

[42] 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

[43] 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–.

[44] 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/.

[47] 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.

[48] 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.

[49] 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

[50] 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.

[51] 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.

[52] Del V. Sosa, M., S. Acuï£¡a 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

[53] 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/.

[57] 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.

[58] 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*.

[59] 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.

[60] 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.

[61] 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.

[62] 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.

[63] 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.

[64] 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/.

[67] 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).

[68] 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/.

[70] 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*.

[71] 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

[72] 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.

[73] 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

[74] 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*.

[75] Hernï£¡ndez, A., V. Le Rolle, A. Defontaine and G. Carrault (2009), A multiformalism and multiresolution modelling environment: Application to the cardiovascular system and its regulation, *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, **367**, 1908, pp. 4923–4940, doi:10.1098/rsta.2009.0163, cited By 0.
https://www.scopus.com/inward/record.uri?eid=2-s2.0-73349122901&partnerID=40&md5=f4b53a48fb93b2be4739a21e840d621c

[76] 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.

[77] 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.

[78] 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.

[79] 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.

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

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

[82] 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.

[83] 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.

[84] 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

[85] 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

[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.

[88] 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

[89] 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.

[90] 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.

[91] 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

[92] 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.

[93] 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.

[94] 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.

[95] 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

[96] 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/.

[99] 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

[100] Qamar, A., S. Herzig, C. Paredis and M. Tï£¡rngren (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

[101] 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.

[102] 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

[103] 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.

[104] 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.

[105] 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.

[106] 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*.

[107] 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.

[108] 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/.

[110] 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

[111] 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.

[112] 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).

[113] 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

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

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

[116] 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.

[117] 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.

[118] 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

[119] 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