An integrated ontology for multi-paradigm modelling for cyber-physical systems

Dominique Blouin, Rima Al-Ali, Holger Giese, Stefan Klikovits, Soumyadip Bandyopadhyay, Ankica Barisic, Ferhat Erata

for the next constructive phase following this work that will consist of developing a solution for implementing model management for MPM based on the knowledge captured by the ontology.

Chapter 5

An Integrated Ontology for Multi-Paradigm Modeling for Cyber-Physical Systems

Author(s): Dominique Blouin, Rima Al-Ali, Holger Giese, Stefan Klikovits, Soumyadip Bandyopadhyay, Ankica Barišić, Ferhat Erata

5.1 Introduction

This chapter presents the MPM4CPS ontology that integrates the Shared, CPS and MPM ontologies by providing cross-cutting concepts between these domains. In particular, it formalizes a new notion of viewpoint adapted from existing works. Such viewpoint integrates the concepts of megamodel fragments of the MPM ontology with stakeholder concerns about the system of the Shared ontology and with the CPS constituent elements under development of the CPS ontology. The CPS ontology was specified in Chapter 3 in the form of a metamodel complemented with a feature model. After conversion of the provided CPS metamodel and feature model to an OWL ontology, we are able to refer to classes of the CPS ontology represented in OWL form from our MPM4CPS viewpoint notion.

We also extend the core notions of workflow processes of the Shared ontology to capture model-based development processes. In particular, we relate these processes to their employed viewpoints. Finally, we refine the notion of engineering paradigm introduced in the Shared ontology to define our notion of modeling paradigms, which adapts existing work to take into account our new viewpoint notion.

In the following, we first briefly introduce the state of the art that served as starting point to develop the ontology in Section 5.2. Then, we detail the ontology that includes our extension of these state of the art works in Section 5.3.

Section 4.4 illustrates the usage of the ontology with the Ensemble-Based CPS and HPI CPSLab examples previously introduced in Chapter 2. We finally conclude this chapter and this first part of this book on foundations for MPM4CPS in Section 5.5.

5.2 State of the Art

Integration needs underlying the development of embedded real-time and cyber-physical systems have been outlined in [142]. These observations and state of the art work presented in this section will guide the development of the MPM4CPS ontology in order to cover the needs of the EBCPS and HPI CPSLab example development environments and their CPS case studies.

The MPM4CPS ontology is centered on a notion of viewpoint that relates parts of the CPS under study with the employed formalisms, languages and tools as informally defined in the framework of Broman et al. [48]. As already mentioned in Section 4.2 of Chapter 4, parts of the framework of Broman et al. also inspired the MPM ontology. Their framework is itself an adaptation of the ISO/IEEE standard 42010 standard on architecture descriptions [14], which we introduced in the Shared ontology presented in Chapter 2 except for its modeling-related notions that the MPM4CPS ontology redefines.

The MPM4CPS ontology enhances the framework of Broman et al. by enriching it with modeling notions of the MPM ontology such as megamodel and megamodel fragment. In addition, the ontology also considers model-based development processes to orchestrate the different modeling activities performed to develop specific parts of CPS using different megamodel fragments. Finally, the ontology also adapts an existing notion of modeling paradigms that characterizes the formalism and development process of these development environments.

Therefore, we divide this state of the art along these three paxis. The first part presents different existing works on viewpoint-related concepts including introducing the viewpoint concepts of the ISO/IEEE 42010 standard and that of the framework of Broman et al. [48]. The second part presents state of the art on the modeling of model-based development processes. Finally, the third part proposes a brief state of the art on modeling paradigms including the closely related topic of programming paradigms.

5.2.1 Viewpoints

We first complete the introduction of the ISO/IEEE 42010 standard by presenting its viewpoint-related notions. As depicted in Figure 2.9, the ISO/IEEE 42010 standard defines its own notion of viewpoint, which associates a set of stakeholder concerns with a set of model kinds employed to address these concerns in the design of an architecture. These model kinds serve as typing elements for the models employed for specifying the architecture. Architecture viewpoints are then used to govern the view on the system according to the concerns and the model kinds.

The framework of Broman et al. [48] adapts the ISO/IEEE 42010 standard to CPS design by enriching the provided viewpoint notion. This is achieved by replacing the Model Kind concept of the ISO/IEEE 42010 standard by the formalisms supporting the viewpoint to perform the different activities on the system models (Figure 4.2). In addition, the Broman et al. viewpoint notion adds a link from the viewpoint to the parts of the system under design relevant to the concerns framed by the viewpoint. This is illustrated in Figure 5.1 where the *Control Robustness Design*, *Control Performance Design* and *Software Design* example viewpoints are depicted for an advanced driver assistance system (ADAS) (e.g., adaptive cruise control) embedded control system. A matrix shows the parts of the system and their concerns along its two axis and cross-cutting points are defined and grouped to define viewpoints.

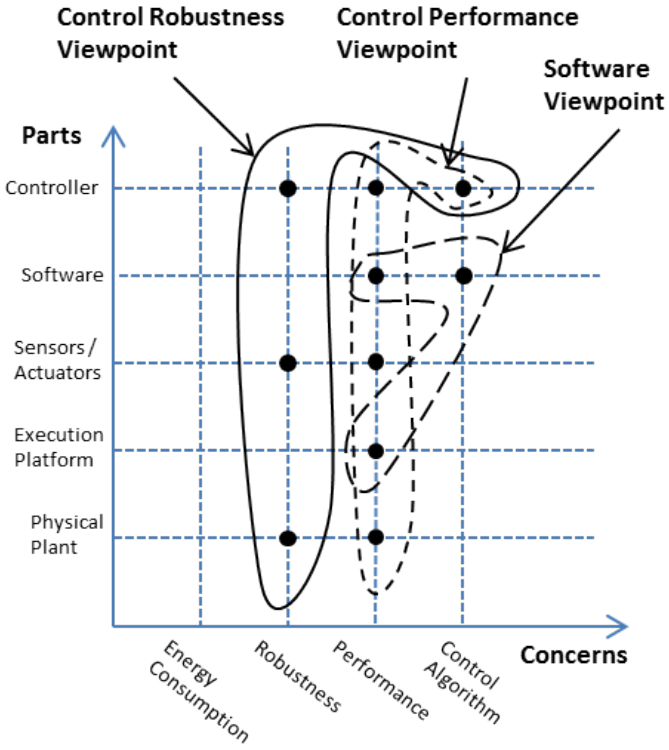

Figure 5.1: Example of viewpoints matrix (based on [48])

For instance, the *Software Design* viewpoint frames the *Performance* and the *ADAS Algorithm* concerns, which are impacted by the *Software* and *Computing Platform* parts of the system. As can be seen in Figure 5.2, such viewpoint employs the state machines, dataflow and discrete event formalism to address these concerns in the CPS design.

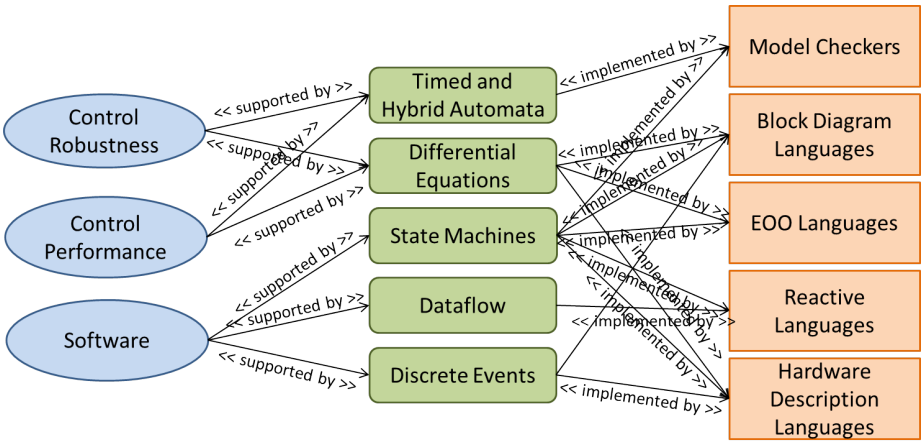

Figure 5.2: Example of viewpoints and their employed formalisms (based on [48])

These notions will be at the heart of our MPM4CPS ontology. Besides, we also want to relate these viewpoints to the development processes in which they are used, which is an important characteristic of CPS development approaches and environments. This is the topic of the next section.

5.2.2 Model-Based Development Process Modeling

MPM advocates to model everything, at the most appropriate level(s) of abstraction, using the most appropriate formalisms, for the modeling activities to be performed. This includes not only the modeling of the system to be developed (at the micro modeling scale as defined by the MPM ontology) and the modeling of the models and modeling languages employed to develop the system (at the megamodeling scale), but also the modeling of model-based development processes that orchestrate the various activities performed on models of the system.

Unfortunately, we do not find many work on this topic in the literature. This is also the result of a systematic mapping study of MPM for CPS initiated during the MPM4CPS COST action [156] A main result of this study is that among the studies that reported a development process, 38% of them describe it informally often in a partial way; 30% describe it semi-formally, step by step, but still giving only natural language descriptions; and only 32% provide a formal model for specifying their process. Therefore, development process modeling remains an issue in MPM4CPS.

Nevertheless, we find the FTG+PM [137], which as already mentioned in the state of the art of Chapter 4, is the only model management language that formalizes development processes. Therefore, it seems to be currently the most appropriate language to specify engineering processes in MPM. Figure 5.3 shows

the main concepts of the language. On the left hand side of the figure, we find the Formalism Transformation Graph (FTG) concepts, which describe a repository of transformation relations between formalisms. On the right hand side, we find the Process Model (PM) concepts. Those are based on the UML activity diagram, which has been adapted so that activities are typed by transformations defined in the FTG part. In addition, objects representing artifacts processed by the activities are typed by languages of the FTG side therefore representing models taken as input by the activities when executing transformations.

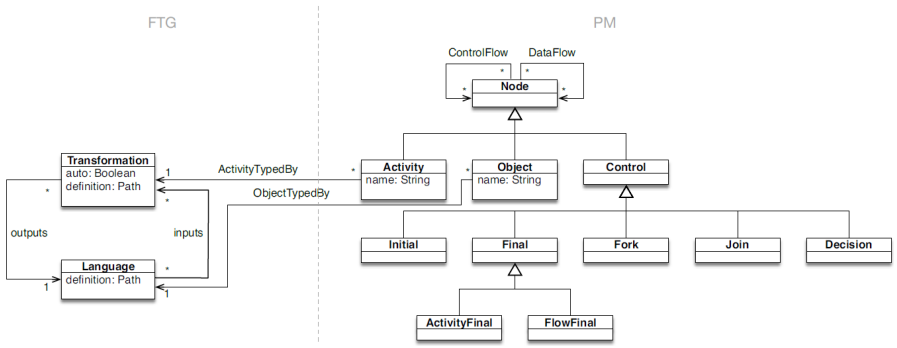

Figure 5.3: Overview of the FTG+PM concepts (from [137])

The FTG part can be compared with the megamodel notion of the MPM ontology of Chapter 4. A difference however is that megamodels can be logically organized into fragments, which is not the case of the FTG. Besides, only one kind of relation is defined between formalisms, while the MPM ontology captures several kinds such as synchronization, traceability and refinement relations. Finally, transformations relate formalisms and not modeling languages like for the megamodel relations of the MPM ontology. With our definitions of formalisms and modeling languages, we must relate modeling languages since they are concrete implementations of formalisms.

The FTG+PM languages, as well as the megamodeling notion of Chapter 4 and the workflow process concepts of the shared ontology of Chapter 2 will serve as basis for defining model-based process modeling in this MPM4CPS ontology.

5.2.3 Modeling Paradigms

Another important MPM-related notion is that of a *modeling paradigm*. We initially found that there was no precise definition of this notion despite that it originated as early as 1996 [157]. However, the notion of modeling paradigm can be seen as a generalization of the notion of *programming paradigm*, since programming languages can be seen as a subset of modeling languages. Programming paradigms, which originated as early as 35 years ago [16, 17] have been defined to categorize the different approaches or styles used by the different programming languages. Due to the growing heterogeneity of software systems,

which results in different kinds of problems to be solved, different approaches or paradigms to solve these problems had to be developed. Therefore, a plethora of programming languages have been created and categorized according to their underlying paradigms. However, the notion of programming paradigm, which varies from one author to another, was also never made very precise. The most precise definition that we find is that of [158], which informally defines a programming paradigm as "*...a set of programming concepts, organized into a simple core language called the paradigm's kernel language*". Yet, this definition remains vague.

Such lack of precise definition of programming and modeling paradigms was an important problem for our ontology of MPM, which triggered some work to precise the definition [21, 22]. Part of this work has already been introduced in Chapter 2 that presented the Shared ontology and its paradigm subdomain. However, only the part that is independent of modeling was presented as its level of abstraction is adequate for modeling the more general notion of engineering paradigms.

We will build on the modeling-specific part of this work to precise the notion of modeling paradigm for this MPM4CPS ontology. As can be seed from Figure 2.11, this work defines a modeling paradigm as a set of properties characterizing the languages (including their semantics) and workflows (development processes) employed to develop systems. This can be seen as an enlargement of the notion of a programming paradigm, which typically only characterizes programming languages and do not say anything about workflows. Examples of simple paradigms may be object orientation, which only pertains to formalisms, or agile development, which only characterizes workflows. More complex paradigms examples discussed in the work of [22] are Synchronous Data Flow (SDF) and Discrete Event Dynamic Systems (DEv). These will be further discussed in Section 5.3 where we will introduce our MPM4CPS ontology in details.

5.3 Ontology

In this section we provide an overview of the OWL MPM4CPS ontology that defines cross-cutting concepts between the Shared, MPM and CPS ontologies presented in the previous chapters. We first define the `MPM4CPSDC` domain concept class as a subclass of the `DomainConcept` class of the Shared ontology to organize the classes of the MPM4CPS integrating domain (Figure 5.4). All classes defined in this section are said to be part of the MPM4CPS domain and will therefore be made subclasses of this MPM4CPS subdomain class.

Following the state of the art of the previous section, we first define our viewpoint notion inspired from the framework of Broman et al., but adapted for the megamodel and megamodel fragments of the MPM ontology. Then, we refine the workflow process subdomain of the Shared ontology (see Section 2.5 of Chapter 2) to specialize it for model-based development. Finally, we refine the notion of engineering paradigms of the Shared ontology to define a more

specific notion of modeling paradigm.

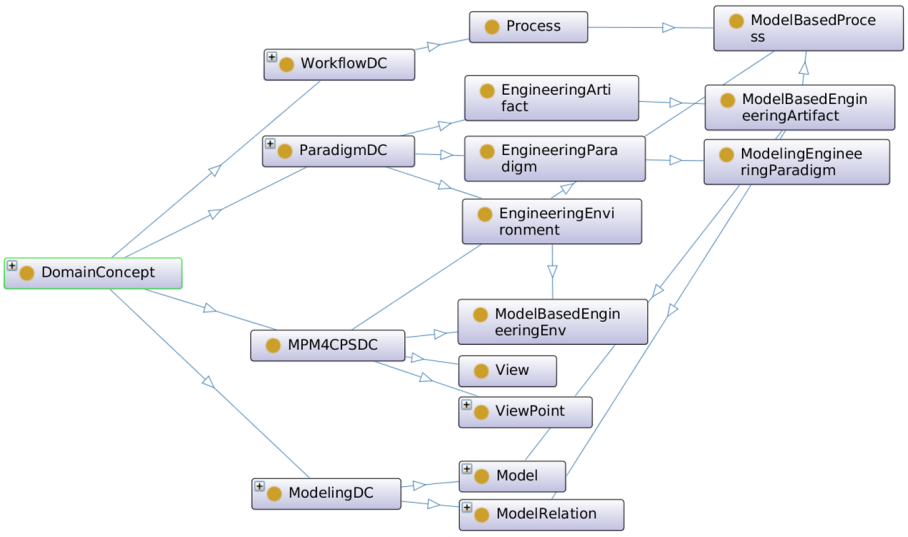

Figure 5.4: Overview of the OWL MPM4CPS ontology

5.3.1 Viewpoint

We start by defining our notion of `Viewpoint`, which is inspired from the definition from the framework of Broman et al. (Figure 4.2 and Figure 5.1). Their definition is itself an adaptation of the notion of viewpoint from the IEEE 42010 standard [14] (Figure 2.9). Compared to the notion of viewpoint from the IEEE 42010, the notion of the framework of Broman et al. replaces the `ModelKind` (Figure 2.9) representing the modeling languages supporting a viewpoint by a set of `Formalism`.

However, we have seen in the MPM ontology how we defined the notion of `MegamodelFragment` that organizes modeling languages conjointly used with their appropriate `ModelRelation`s to support an activity of a development process (e.g. the Model Test activity of the simulation stage of the HPI CPSLab development process in Chapter 4). Therefore, we replace the set of formalism of the Broman et al. framework by a megamodel fragment, and we define the `hasSupportingMegamodelFragments` object property to relate a viewpoint to a set of supporting megamodel fragments.

According to the IEEE 42010 standard, a viewpoint *frames* the concerns of stakeholders (Figure 2.9). Therefore we create the `hasFramedConcerns` object property to relate a viewpoint to a set of framed concerns as defined in the Shared ontology (Chapter 2).

In addition, a viewpoint *governs* an *architecture view*. We therefore define

the `View` class and the `hasGovernedView` object property to relate a viewpoint to its view. Finally, we create another object property to relate a view to its employed models.

An addition of the framework of Broman et al. to the IEEE 42010 standard is to define a reference from a viewpoint to the system parts being developed using views of the viewpoint. We adopt this feature and create the `hasSystemConstituentElements` object property between a viewpoint and its system constituents or parts. The range of the property is the `ConstituentElement` OWL class, which was generated from feature of the feature model of the CPS ontology (Figure 3.2).

We note that this makes our notion of viewpoint a cross-cutting concept between the MPM and CPS domains of our ontological framework. This explains why the viewpoint-related notions are part of the MPM4CPS integrating ontology, since they depend on all other ontologies. We also note that the link between a viewpoint and CPS constituent elements can be seen as being derived from the standard *representedBy* relation between a real system and its models [159].

This completes the definition of our viewpoint-related notions. We then consider how these viewpoints are used throughout development by introducing notions related to model-based workflows.

5.3.2 Model-Based Workflows

We have introduced core workflow process modeling notions inspired from the WFMC-TC-1025 standard in the Shared ontology of Chapter 2. On the other hand, the FTG+PM process modeling language introduced in Section 5.2.2 defines similar process modeling notions adapted from UML activity diagrams. An advantage of the WFMC-TC-1025 standard is its wider coverage of the workflow domain. For instance, embedded and external subprocesses can be modeled. Besides, the notion of activity performer is included to model tools or humans performing the activities. In FTG+PM the performer notion is implicitly embedded within the activity concept, the human / tool characteristic being declared in the transformation class (*auto* attribute of the *Transformation* class of Figure 5.3).

Such richness of the workflow subdomain is required to model industrial-strength processes such as the one of the HPI CPSLab example, which reuses an existing methodology from the automotive domain (Figure 2.19). However, as opposed to the FTG+PM language, the WFMC-TC-1025 standard does not say anything about modeling but only links activity performers to the processed data fields (Figure 2.6), such data fields being typed by associated data types declared in processes. We already covered the FTG-related notions in our MPM ontology via megamodels capturing modeling languages and their relationships, but in a finer way by providing different kinds of transformation relations. Besides our megamodels are hierarchical[1] allowing to capture contextual model

[1]This is not yet covered in this version of the MPM ontology.

transformation relations.

Therefore, the process notions of our MPM4CPS ontology will reuse the core process notions of the Shared ontology, reuse the link between the process model and the FTG of the FTG+PM language, but replace the FTG by the richer megamodel fragment notion. In the following, we provide an overview of model-based workflow process notions as part of this MPM4CPS ontology.

Model-Based Process

We first subclass the `Process` class of the Shared ontology into the `ModelBasedProcess` class to represent all processes that manipulate models instead of data fields. The choice of the model-based name follows the naming of the model-based engineering paradigm.

As opposed to the WFMC-TC-1025 standard, a `ModelBasedProcess` does not declare the types of the data fields processed by activity performers. Instead, a set of viewpoints can be linked to a process through an object property `hasViewpoint`, so that data types are declared as the elements grouped under the associated megamodel fragment. Similar to the FTG+PM language, such types are modeling languages and their relations declared in the megamodel fragments.

Activity Performers

The `ModelingTool` class of the MPM ontology was declared as a subclasses of the `Tool` class of the Shared ontology. This class was also made a subclass of `Application` activity performer class of the workflow subdomain of the Shared ontology. Therefore, a modeling tool can be directly used as activity performer by any activity. Similarly, the `ModelingHuman` resource of the MPM ontology was also made a subclass of the `Application` activity performer class of the workflow subdomain. Therefore, modeling human can be directly used as activity performer for representing modeling activities that are performed manually.

Finally, the `hasTranformationSpecifications` object property of the MPM ontology can be used to relate an activity performer resource to the transformation relations it executes as declared in the activity's associated viewpoint.

5.3.3 Modeling Paradigms

In the Shared ontology, we have provided the notion of `EngineeringParadigm` defined as a characterization of environments in which engineering takes place. We refine this definition so that it characterizes model-based engineering environments and their employed model-based artifacts.

In order to define our modeling paradigm notion, we first subclass the `EngineeringEnvironment` class of the Shared ontology by the `ModelBasedEngineeringEnv` class. Similarly, we define the `ModelBasedEngineeringArtifact` as a subclass of the `EngineeringArtifact`. We also make the `Model` and `ModelRelation` classes of the kernel model of the MPM ontology subclasses of `ModelBasedEngineeringArtifact`.

Hence, all elements of the MPM ontology that are are subclass of `Model` are also `ModelBasedEngineeringArtifacts`. We also make the `ModelBasedProcess` class `ModelBasedEngineeringArtifacts` as well. Finally, we create the `hasModelingArtifacts` subproperty of the `hasArtifacts` object property having respectively the `ModelBasedEngineeringEnv` and `ModelBasedEngineeringArtifact` classes as domain and range.

We then define the `ModelingEngineeringParadigm` class as a subclass of the `EngineeringParadigm` class of the Shared ontology. As defined in the Shared ontology, an engineering paradigm declares characteristics of engineering artifacts that make the paradigm. Therefore in our refined definition of modeling paradigm, we refine the scope of the paradigm characteristics to modeling artifacts. Note that compared to the definition of [22], which was introduced in the state of the art section, our definition is wider as its scope consisting of any modeling artifact is not restricted to modeling languages and workflows.

Several ways can be thought of to express the characteristics of a paradigm. In the Shared ontology, those characteristics are represented as properties including their expressions and decision procedures. We have reused these notions from the work of [22]. We therefore present in greater details their way to express paradigm characteristics.

As can be seen from Figure 2.11, the properties expressing characteristics are actually meant to be evaluated over a *paradigmatic structure*, which expresses some of the paradigm characteristics as patterns to be checked over formalisms and workflows. Therefore, the first step in checking that a set of modeling artifacts implements a paradigm is to match a paradigmatic structure over the considered modeling artifacts. Once a match is found, paradigmatic properties can be evaluated against the structure to completely determine if the artifacts satisfy the paradigm.

In order to express paradigmatic structures, the authors define an adaptation of the metamodel language where classes are placeholders to be matched by other languages. Therefore, modeling artifacts such as languages and processes must also be expressed as metamodels for paradigm satisfaction checking. The approach also relies on the fact that language semantics are expressed as semantic domains whose languages are also expressed as metamodels so that they can also be characterized by paradigmatic structures. In the approach, it is expected that providers of modeling artifacts are responsible to *map* their artifacts to the paradigmatic structure. This is one drawback of the approach, which may make the evaluation of paradigms satisfaction over industrial heterogeneous modeling environments difficult. For languages, workflows and their semantic domains already expressed as metamodels, this is relatively easier. However for other metamodeling technical spaces such as grammars, a conversion is required. Besides, a language's semantics is often not formalized but embedded into programming code of the tool that executes the language, which makes the task of identifying paradigms even harder.

Nevertheless, the work of [22] is a first attempt and an ongoing work on defining modeling paradigms and is therefore subjected to be improved in the next future. Hence, at this stage, we will not yet precise in this ontology any approach for expressing paradigm characteristics. Besides, since the scope of

our modeling paradigm definition is wider, the paradigmatic structure proposed in [22] would be too restrictive. Nevertheless, we will present an example of paradigm characteristics expression for the HPI CPSLab in Section 5.4.3.

Another important remark is the different definitions of the formalism notion between the authors of [22] and that of our MPM ontology. The authors use the definition of [47] (Figure 4.1) where a formalism is defined as "*a language, a semantic domain and a semantic mapping function giving meaning to model in the language*". In the MPM ontology, we followed the framework of Broman et al. [48] where formalisms are "*mathematical objects consisting of an abstract syntax and a formal semantics*", and where modeling languages are "*concrete implementations of formalisms*". Broman et al. further note that a language's semantics may slightly deviate from the semantics of formalisms they realize.

The question is then to understand the difference between our definition of formalisms and the paradigms of [22], for those paradigms that only characterize formalisms (as per the definition of [47]). This is even more relevant as [22] use Synchronous Data Flow (SDF) and Discrete Event Dynamic Systems (DEv) as paradigms examples in their work. At some point, the authors even consider the SDF paradigm as a "*conceptual formalism*". In addition, these two paradigms are classified as formalisms in the Catalog of Formalisms, Modeling Languages and Tools [9], which was created during the MPM4CPS COST action project. An interesting future work would therefore consists of studying all other formalisms of the catalog such as Petri Nets, Abstract State Machines, Hybrid Automata using exploratory modeling to better understand the formalism notion.

5.4 Examples

In this section, we illustrate how the MPM4CPS integrated ontology presented in Section 5.3 covers the needs for a comprehensive modeling of development environments and their CPS case studies such as the EBCPS and HPI CPSLab examples. For each of these examples, we first start by modeling some of the employed viewpoints that we relate to their stakeholder concerns, the employed megamodel fragments of Chapter 4 and the parts of the CPS of Chapter 3.

Next, we show how the engineering methodologies of each example can be captured including their different stages and the activities employed at these stages. Each of these activities makes use of a workflow process that uses one of the previously defined viewpoints. The process decomposes the root activity into a sequence of subactivities whose activity performers are set as the proper modeling tools and humans described in Chapter 4. Note that we do not present a complete coverage of the engineering methodologies and processes here, but only the parts needed to illustrate the use of the MPM4CPS integrated ontology.

Finally, we illustrate the modeling of the Synchronous Data Flow (SDF) paradigm with the integrated ontology characterizing one of the example development environment through some of its activities making use of modeling languages based on SDF.

5.4.1 EBCPS

We present the modeling details of the EBCPS example with regard to the MPM4CPS ontology. The EBCPS methodology is a simple methodology that contains only one stage. A more comprehensive example is provided for the HPI CPSLab methodology that contains three stages. In this example, we use the division of the megamodel fragments as defined in Chapter 4, which is per modeling language rather than per activity as for the case of the HPI CPSLab. Thus, our viewpoints can employ more than one megamodel fragment, and activity processes can refer to more than one viewpoint.

Methodology

We define the EBCPSMethodology as an instance of the `EngineeringMethodology` class, which contains a set of stages. In our case, the scenario of autonomous vehicle presented in Chapter 2 employs only one stage that consists of a simulation stage.

Methodology

- EngineeringMethodology: EBCPSMethodology
  - hasStages EngineeringStage: EBCPSSimulationStage

Methodology Implementing Process

Per our definition from the workflow subdomain of the shared ontology, a methodology does not specify how it is implemented since several implementations could exist for a given methodology. Therefore we define the EBCPSProcess as one implementation of the EBCPSMethodology.

This process defines an activity set that specifies activities and transitions to orchestrate the different root modeling activities of the EBCPSProcess. Each of these activities also implements the single simulation stage of the EBCPS methodology. The root activities for the simulation stage are:

ActivitySet

- hasSetActivities SubFlow: RequirementsRootActivity
  - isImplementingStage: EBCPSSimulationStage
  - hasSubProcess: RequirementsProcess
- hasSetActivities SubFlow: DesignRootActivity
  - isImplementingStage: EBCPSSimulationStage
  - hasSubProcess: DesignProcess
- hasSetActivities SubFlow: RuntimeRootActivity

  - isImplementingStage: EBCPSSimulationStage
  - hasSubProcess: RuntimeProcess

- hasTransitions Transition: Requirements2DesignTransition, Design2RuntimeTransition

The processes presented for each root activity declare finer grained activities such as editing a model, simulating a model, checking simulation results, etc. Some activities are performed by humans (i.e. a designer) such as in case of capturing the requirements and implementing the components with self-adaptation support. Other activities may be performed by tools such as simulation tools.

Each process uses a viewpoint that specifies the concerns addressed by the process, the part(s) of the CPS that are being developed and their employed megamodel fragment(s). These megamodel fragments specify the modeling languages and their relationships that support the process activities. We first present the viewpoints in the following.

Viewpoints

- Viewpoint: ComponentAutonomyVP
  - hasFramedConcerns: SafetyConcern, AdaptabilityConcern, EfficiencyConcern
  - hasSystemConstituentElements: Controller, Plant, Sensors, Actuators.
  - hasSupportingMegaModelFragments: RequirementsMegaModelFragment, DesginMegaModelFragment, RuntimeMegaModelFragment, SelfAdaptationMegaModelFragment
- Viewpoint: ComponentsCooperationVP
  - hasFramedConcerns: SafetyConcern, EfficiencyConcern
  - hasSystemConstituentElements: Communication, Controller, Plant, Sensors, Actuators
  - hasSupportingMegaModelFragments: RequirementsMegaModelFragment, DesginMegaModelFragment, RuntimeMegaModelFragment, SimulationMegaModelFragment

Activity Subprocesses

- ModelBasedProcess: RequirementsProcess
  - ActivitySet: editing model - To create an IRM model, the developer needs to define the invariants in the system and their relations to the processes and knowledge exchange, which can have assumptions. The processes are associated to a component role that is also part of the IRM model. After finishing the modeling part, the next activity is to generate the skeleton of the components and ensembles in Java code from the IRM model.

  - hasViewPoint: ComponentAutonomyVP, ComponentsCooperationVP

- ModelBasedProcess: DesignProcess
  - ActivitySet: editing model - Using the skeleton, the developer implements the processes in the component and the ensembles. In this part, the developer can support mode-switching in the component. In the example, the vehicle component have a controller and a plant, which should be also implemented.
  - hasViewPoint: ComponentAutonomyVP, ComponentsCooperationVP

- ModelBasedProcess: RuntimeProcess
  - ActivitySet: running model, result model - The developer can execute the design model and have the components and ensembles running (i.e. DEECoRuntimeModel) that results with logs as an outputs. The runtime environment monitors the component and performs the mode-switching when needed. At the same time, the simulation models run and synchronous with DEECoRuntimeModel.
  - hasViewPoint: ComponentAutonomyVP, ComponentsCooperationVP

5.4.2 HPI CPSLab

The HPI CPSLab and its methodology including several stages supported by multi-formalisms settings provides a comprehensive example of a development process where the developed system is gradually built starting from pure models at the simulation stage and gradually integrating more and more of the real artifacts in the following prototyping and pre-production stages (Figure 2.19).

We present the modeling of this process and of one of its employed modeling paradigms with the MPM4CPS ontology. We first present the modeling of the methodology and its stages. Then, for each stage we present the detailed modeling of the different activities implementing the stage, including the employed viewpoints. Finally, we present the modeling of a simple modeling paradigm employed by the HPI CPSLab process and its viewpoints.

Methodology

We define the CPSLabMethodology instance of the `EngineeringMethodology` class of the workflow subdomain of shared ontology (Chapter 2) to capture the HPI CPSLab methodology and its set of stages as depicted in Figure 2.19. This is achieved by representing each stage as an instance of the `EngineeringStage` class and creating instances of the `hasNextStage` object property defining the order between the stages.

Methodology

- EngineeringMethodology: CPSLabMethodology

  - hasStages EngineeringStage: `CPSLabSimulationStage`
    - hasNextStage: `CPSLabPrototypingStage`
  - hasStages EngineeringStage: `CPSLabPrototypingStage`
    - hasNextStage: `CPSLabPreproductionStage`
  - hasStages EngineeringStage: `CPSLabPreproductionStage`

Methodology Implementing Process

We define the root CPSLabProcess instance of the `ModelBasedProcess` class to implement the CPSLabMethodology. This process defines an activity set defining root activities and transitions to orchestrate them. Each of these activities contributes to implementing a stage of the CPSLab methodology. In addition, each root activity is decomposed into finer grained activities as declared with an associated subprocess.

ActivitySet

- hasSetActivities SubFlow: MTActivity
  - isImplementingStage: CPSLabSimulationStage
  - hasSubProcess: MTProcess
- hasSetActivities SubFlow: MiLActivity
  - isImplementingStage: CPSLabSimulationStage
  - hasSubProcess: MiLProcess
- hasSetActivities SubFlow: RPActivity
  - isImplementingStage: CPSLabSimulationStage
  - hasSubProcess: RPProcess
- hasSetActivities SubFlow: SiLActivity
  - isImplementingStage: CPSLabPrototypingStage
  - hasSubProcess: SiLProcess
- hasSetActivities SubFlow: HiLActivity
  - isImplementingStage: CPSLabPrototypingStage
  - hasSubProcess: HiLProcess
- hasSetActivities SubFlow: STActivity
  - isImplementingStage: CPSLabPreproductionStage
  - hasSubProcess: STProcess

- hasTransitions Transition: MT2MiLTransition
- hasTransitions Transition: MiL2RPTransition
- hasTransitions Transition: RP2SiLTransition
- hasTransitions Transition: MiL2SiLTransition
- hasTransitions Transition: SiL2HiLTransition
- hasTransitions Transition: HiL2STTransition

Transitions are instantiated with appropriate conditions (not presented here) to define the order of execution of activities. Note that the declared order must be overall consistent with the ordering of the implemented stages as defined by the methodology. Overall consistency means that activities of a stage that follows another stage should never be executed before the activities of the preceding stage have already been executed at least once. Indeed, although not shown in this case, root activity transitions may return back to an activity of a preceding stage in case the errors discovered during the current stage were introduced at an earlier stage.

In the next sections, for each stage we present the viewpoints and the root activity processes that employ them.

Simulation Stage

The purpose of the simulation stage is to define the control laws of the system. As opposed to the two next stages, its activities only employ models as captured by the megamodel fragments of Chapter 4 to represent the system and its environment.

For the HPI CPSLab example, we define a set of viewpoints specific to each of the root activities. This differs from the EBCPS example where existing viewpoints (e.g. from a library) are reused to support the activities. In this case, we define three viewpoints to support each root modeling activity of the simulation stage as follows:

Viewpoints

- Viewpoint: CPSLabMTControlAlgorithmVP
  - hasFramedConcerns: ControlAlgorithm
  - hasSystemConstituentElements: Controller, Plant, Sensor, Actuator
  - hasSupportingMegaModelFragments: CPSLabMTMMF
- Viewpoint: CPSLabMiLControlAlgorithmVP
  - hasFramedConcerns: ControlAlgorithm, Stability, Safety, Reliability
  - hasSystemConstituentElements: Controller, Plant, Sensor, Actuator

  - hasSupportingMegaModelFragments: CPSLabMiLMMF

- Viewpoint: CPSLabRPControlAlgorithmVP
  - hasFramedConcerns: ControlAlgorithm, Stability, Safety, Reliability
  - hasSystemConstituentElements: Controller, Plant, Sensor, Actuator
  - hasSupportingMegaModelFragments: CPSLabRPMMF

Each of these viewpoints addresses the concern of the control algorithm of the system under design and is using a megamodel fragment defined in the example section of Chapter 4 of the MPM ontology to capture the employed modeling languages and their relations. In addition, the CPSLabMiLControlAlgorithmVP and CPSLabRPControlAlgorithmVP also address other concerns such as stability, safety, reliability, thanks to the plant model providing feedback to the controller as opposed to the static input data of the model test activity.

Each of these viewpoints describes a cyber-physical setting, at different levels of abstraction. The abstract control algorithm from the cyber domain captured by the Matlab/Simulink control model (ControlModel) is confronted with the physics as represented in the input data plus expected outcomes. This is model differently for each viewpoint as static data model (MT), plant model (MiL) and detailed robot model (RP). Therefore, all viewpoints cover the system constituents of interest represented by these models, which are the controller, plant, sensor and actuator elements. We further have a multi-formalism setting where the control is discrete while the input data is at least conceptually continuous.

Root Activity Subprocesses

Each root subflow activity is further described by a subprocess specifying its decomposition in terms of finer grained activities such a editing a model, executing a model transformation, etc. Besides, the process, which is responsible for defining the context for executing its activities is associated with a viewpoint providing such context. We list below the root activity sub processes and their associated viewpoints. As an example, we present the fine grained activities of the model test subprocess in the next section.

- ModelBasedProcess: CPSLabMTProcess
  - ActivitySet: ...
  - hasViewPoint: CPSLabMTControlAlgorithmVP

- ModelBasedProcess: CPSLabMiLProcess
  - ActivitySet: ...
  - hasViewPoint: CPSLabMiLControlAlgorithmVP

- ModelBasedProcess: CPSLabRPProcess
  - ActivitySet: ...

  - hasViewPoint: CPSLabRPControlAlgorithmVP

Model Test Subprocess (CPSLabMTProcess)

We describe here as an example the set of subactivities that constitute the CPSLabMTProcess. Like for the process orchestrating root activities, this is achieved by creating a block activity and its activity set. But we first define activity performers to perform the activities.

Activity Performers

- ModelingHuman: ControlEngineerPerf
  - hasTranformationSpecifications: : EditInputModelOperation, EditControlModelOperation, EditPlantModelOperation, EditValidityResultsModel...
- ModelingTool: SimulinkTool
  - hasTranformationSpecifications: SimulateModelOperation, ...

Then, we define the fine grained activities as per the list below.

ActivitySet

- hasSetActivities Activity: MTEditInputModel
  - hasActivityPerformer: ControlEngineerPerf
- hasSetActivities Activity: MTEditControlModel
  - hasActivityPerformer: ControlEngineerPerf
- hasSetActivities Activity: MTSimulateControlModel
  - hasActivityPerformer: SimulinkTool
- hasSetActivities Activity: MTCheckSimulationResults
  - hasActivityPerformer: ControlEngineerPerf
- hasTransitions Transition: EditInput2EditControlTransition
- hasTransitions Transition: EditControl2SimulateControlTransition
- hasTransitions Transition: CheckResults2EditControlTransition
  - hasCondition: ValidResults
- hasTransitions Transition: ...

Transitions are defined between the different activities. It should be noted that this is a simplified version of the real workflow as the order of the activities may depend on several conditions. For example, the transition CheckResults2EditControlTransition between the MTCheckSimulationResults and MTEditControlModel activities has a condition. Such condition evaluates some property of the ValidityResultsModel as was set by the designer during the MTCheckSimulationResults activity and indicating if the results are correct or not. If not correct, the control may be edited again. If correct, the process ends and by default returns to the calling subflow root activity.

At deployment, an application megamodel containing the models to be processed by activity performers can be bound to this structure for executing the process on real models.

The definition of the two other CPSLabMiLProcess and CPSLabRPProcess processes follows the same principles as that of the CPSLabMTProcess detailed above and is not presented. All details can be found in the ontology files accessible from [8].

Prototyping Stage

Compared to the simulation stage, which only uses models, the focus of the prototyping stage changes from design to implementation. In this stage, the source code plays a major role and is gradually incorporated into the system under development. The purpose is to ensure that implementation constraints such as discretization of variables and time due to the limited resources of the execution platform are handled appropriately to meet the system requirements. The concerns of this stage are therefore related to performance and accuracy and the activities of this stage are used to optimize related parameters such as data representation format, scheduling periods, sensor sampling rates, etc.

For this prototyping stage and the next pre-production stage of the HPI CPSLab methodology, we will only present the viewpoints specification. The modeling of root activity subprocesses is straight forward and follows the same principles as that of the simulation stage.

Like for the simulation stage, we define a viewpoint for each of the two root activities implementing the prototyping stage (Figure 2.19). Therefore for each stage activity, we first present the activity and then define its supporting viewpoint.

Software in the Loop (SiL)

For the Software in the Loop (`SiL`) activity, the tool TargetLink, which is fully integrated into MATLAB Simulink, is used to generate C code from the Simulink behavior model. This allows to seamlessly migrate the functions and control algorithm from continuous behavior of the model level to a discrete approximation implementation in software. Several parameters for code generation can be configured for the characteristics of the desired target platform. Different effects

can be analyzed and results can be compared to results obtained during the simulation stage.

The prototyping stage includes two forms of SiL activities. The first form consists of executing the developed software on a desktop computer against a simulator as depicted in Figure 4.29. The detailed control algorithm from the cyber domain captured by the Matlab/Simulink and AUTOSAR SystemDesk models (SystemModels) are brought together with the physics as present in the sophisticated robot model of the simulator (RobotModel). Therefore, we clearly have a cyber-physical setting. We again have a multi-formalism setting as the control is discrete while the sophisticated robot mode is at least conceptually continuous. Consistency is checked via co-simulation as the software for the robot control runs in parallel with the sophisticated robot simulator.

The second form of SiL also consists of executing the software on a desktop computer but in this case against a real robot remotely controlled as depicted in Figure 4.30. In this case, real sensor values are read from the robot and real actuators are controlled therefore including other effects such as sensor noise. The same detailed control algorithm from the cyber domain is this time brought together with the physics of the real remotely controlled robot. Consistency is checked via co-execution as the software for the robot control runs in parallel with the remotely controlled robot.

Hardware in the Loop (HiL)

The hardware in the loop (HiL) activity consists of executing the software either on the robot as depicted in Figure 4.30 or on special evaluation boards with debugging and calibration interfaces, which are similar to the final hardware execution platform. The almost unlimited execution resource of the desktop computer is replaced by the constrained resources of the final platform. Therefore concerns such as resources consumption could be added to this activity.

With its megamodel fragment, this activity ensures that the detailed control algorithm from the cyber domain captured by the Matlab/Simulink model (ControlModel) is brought together with the physics as present in the robot and thus we have clearly a cyber-physical setting. Consistency is checked via executing the software on the robot.

All these SiL and HiL activities actually address similar concerns about the system. The difference resides in the fact that different models at different levels of abstraction (including real hardware) are used. Therefore we define the viewpoints of the following list:

Viewpoints

- Viewpoint: CPSLabSiLSoftwareDesignVP
  - hasFramedConcerns: ControlAlgorithm, Stability, Safety, Reliability
  - hasSystemConstituentElements: Software (Cyber in feature model), ExecutionPlatform (Control in feature model)

  - hasSupportingMegaModelFragments: CPSLabSiLMMF

- Viewpoint: CPSLabHiLSoftwareDesignVP
  - hasFramedConcerns: ControlAlgorithm, Stability, Safety, Reliability, Resources Consumption
  - hasSystemConstituentElements: Software (Cyber in feature model), ExecutionPlatform (Control in feature model)
  - hasSupportingMegaModelFragments: CPSLabHiLMMF

5.4.3 Modeling Paradigms

We present here an example the modeling of one of the paradigms employed by the HPI CPSLab, which is Synchronous Data Flow (SDF), following its description in [22]. Then we present the modeling of the overall HPI CPSLab model-based development environment employing this paradigm within the MATLAB/Simulink tool captured in its megamodel.

SDF is a special case of the Data Flow paradigm [160]. It specifies a directed graph of computations nodes (also called *blocks*) exchanging signals representing infinite streams of data. Computation units execute whenever input data becomes available. Such units without input can fire at any time. They can be atomic or composite by encapsulating a subgraph. Arcs connect nodes together and describe how streams of data flow through the computation nodes. Execution consists of accumulating enough samples within the system as produced by blocks without inputs and performing the subsequent nodes computations thus consuming sample data on inputs and concurrently producing outputs.

The SDF Paradigm [161] is a specialization of Data Flow where all computation nodes are *synchronous*, meaning that each block explicitly defines how many samples are consumed and produced. In their work, [22] describe the SDF paradigm as exhibiting the following characteristics:

- SignalProperty: Signals composed of an infinite ordered stream of Samples are present.
- DirectedGraphProperty: A *directed* graph with Blocks as nodes and Arcs are present.
- BlocksPortsProperty: Blocks possess Ports that explicitly define how many Samples are used (consumed by Inputs, or produced by Outputs).
- ArcsProperty: Arcs connect Ports and instantaneously transmit Signals. Note that a Port may be plugged to several Arcs but shortcuts are prohibited. Arcs are forbidden to connect as source and target Ports of the same Type.
- MemoryFullProperty: A MemoryFull Block should always define an extra Port corresponding to initial conditions.

Following this, we define an engineering paradigm to represent SDF as follows:

Engineering Paradigms

- ModelingEngineeringParadigm: SDFParadigm
  - hasCharacteristics: SDFParadigmCharacteristics
    - hasProperties: SignalProperty, DirectedGraphProperty, BlocksPortsProperty, ArcsProperty, MemoryFullProperty

There are several ways such properties could be specified. If all languages and their semantics expressed as semantic domains are encoded using the same technical space such as Ecore metamodels, then these properties could be encoded as graph patterns using tools such as Henshin[2] or SDM[3].

It should be noted that the aforementioned Catalog of Formalisms, Modeling Languages and Tools already declares SDF as being a formalism and not a paradigm. Besides, several modeling languages of the catalog such as Simulink are declared as being based on SDF. Therefore, the question of whether SDF should be classified as a formalism or a paradigm remains. A deeper study of the catalog's formalisms could help answer that question.

Next, we model the HPI CPSLab model-based engineering environment, which uses the CPSLabMM megamodel presented in the HPI CPSLab example of Chapter 4 and the CPSLabProcess development process defined in the previous section:

Engineering Environment

- ModelBasedEngineeringEnv: CPSLabEngineeringEnv
  - hasModelingArtifacts: CPSLabMM, CPSLabProcess
  - isBasedOnParadigms: SDFParadigm

It should be noted that the `isBasedOnParadigms` property at the engineering environment level is derived from the same property of Simulink language captured in the mega model. This can be determined from the SDFParadigm and its properties evaluated onver the language and its semantics.

5.5 Conclusion

This chapter presented an integrated ontology for MPM4CPS that captures cross-cutting concepts between the Shared, CPS and MPM ontologies respectively presented in Chapters 2, 3 and 4. It defines notions such as model-based development processes, their employed viewpoints supported by megamodel

[2] `https://www.eclipse.org/henshin/`

[3] `https://www.hpi.uni-potsdam.de/giese/public/mdelab/mdelab-projects/story-diagram-tools/`

fragments of the MPM ontology and the CPS parts under development covered by these viewpoints and defined in the CPS ontology. It finally introduced modeling paradigm notions at the heart of MPM4CPS as a refinement of the more general notions of engineering paradigms define in the Shared ontology. All these elements are captured under the model-based engineering environment concept defined in this integrating ontology.

As much as possible, these notions of the ontological framework have been grounded on existing works and standards such as the Workflow Management Coalition (WfMC) WFMC-TC-1025 [10] and the IEEE 42010 [14] standards, which have been extended / adapted for the MPM4CPS domains. Besides benefiting from the maturity of these works, this also allows stakeholders already familiar with these standards to understand and use the framework with less effort.

In addition, the adopted exploratory modeling approach based on the characterization of existing development settings such as the EBCPS and HPI CPSLab with their CPSs case studies triggered several adjustments of the ontologies to account for existing setups. For instance, the comparison of the modeling of the two examples shows that the framework needs to be flexible enough to be able to capture the heterogeneous practices of industry. Despite that both examples come from the academic world, we have seen that yet they do not organize their megamodel fragments according to the same criteria; per language for the EBCPS and per root process activity for the HPI CPSLab. Besides, viewpoints can be constructed with different objectives such as those of the EBCPS that pre-exist the development process and that must be used as is, while for the HPI CPSLab, the viewpoints are specially constructed to support specific process activities. An even larger heterogeneity can be expected from legacy industrial settings and therefore, being able to cover a large spectrum of practices is essential for this framework to be useful for industry, otherwise disrupting existing industrial settings to adjust to the framework would limit its adoption.

Like for the case of biological science, the classification proposed in this work is not final and will evolve as new MPM4CPS environments are discovered. In particular, the notion of engineering paradigm, its modeling paradigm specialization, the notion of formalism versus paradigm and in multi-paradigm modeling further needs to be investigated, first to be able to discover and understand paradigms from existing development settings and second to support a constructive way of building new MPM4CPS engineering settings based on a set of given paradigms. We hope that our framework can form a solid foundation for implementing a model management solution to relate and combine modeling languages and tools supporting MPM4CPS, as per the original goal of the MPM4CPS COST action project. This will be considered in future work where constructive modeling will be used to build the envisaged solution using the ontology as foundation.

Bibliography

[1] Thomas Kühne. Unifying explanatory and constructive modeling: Towards removing the gulf between ontologies and conceptual models. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, MODELS '16, page 95–102, New York, NY, USA, 2016. Association for Computing Machinery.

[2] K. Kang, S. Cohen, J. Hess, W. Nowak, and A. Spencer Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, 1990.

[3] B. Tekinerdogan and K. Öztürk. *Feature-Driven Design of SaaS Architectures*, pages 189–212. Springer London, London, 2013.

[4] K. Czarnecki, Chang Hwan, P. Kim, and K. T. Kalleberg. Feature models are views on ontologies. In *10th International Software Product Line Conference (SPLC'06)*, pages 41–51, 2006.

[5] Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing? *International Journal of Human-Computer Studies*, 43(5):907 – 928, 1995.

[6] Hai H. Wang, Yuan Fang Li, Jing Sun, Hongyu Zhang, and Jeff Pan. Verifying feature models using owl. *Journal of Web Semantics*, 5(2):117 – 129, 2007. Software Engineering and the Semantic Web.

[7] Rima Al-Ali, Moussa Amrani, Soumyadip Bandyopadhyay, Ankica Barisic, Fernando Barros, Dominique Blouin, Ferhat Erata, Holger Giese, Mauro Iacono, Stefan Klikovits, Eva Navarro, Patrizio Pelliccione, Kuldar Taveter, Bedir Tekinerdogan, and Ken Vanherpen. COST IC1404 WG1 Deliverable WG1.2: Framework to Relate / Combine Modeling Languages and Techniques. Technical report, 2020.

[8] Multi-paradigm modeling for cyber-physical systems website. `http://mpm4cps.eu/`, 2020.

[9] Stefan Klikovits, Rima Al-Ali, Moussa Amrani, Ankica Barisic, Fernando Barros, Dominique Blouin, Etienne Borde, Didier Buchs, Holger Giese,

Miguel Goulao, Mauro Iacono, Florin Leon, Eva Navarro, Patrizio Pelliccione, and Ken Vanherpen. COST IC1404 WG1 Deliverable WG1.1: State-of-the- art on Current Formalisms used in Cyber-Physical Systems Development. Technical report, 2020.

[10] WFMC-TC-1025 Workflow Management Coalition Workflow Standard: Process Definition Interface – XML Process Definition Language, 2005.

[11] Mathias Weske. *Business Process Management: Concepts, Languages, Architectures (Third Edition)*. Springer-Verlag, Berlin, Heidelberg, 2019.

[12] ISO 21500:2012: Guidance on Project Management, 2012.

[13] H. G. Gurbuz and B. Tekinerdogan. Analyzing systems engineering concerns in architecture frameworks – a survey study. In *2018 IEEE International Systems Engineering Symposium (ISSE)*, pages 1–8, 2018.

[14] ISO/IEC/IEEE 42010:2011. Systems and software engineering - Architecture description, the latest edition of the original IEEE Std 1471:2000, Recommended Practice for Architectural Description of Software-intensive Systems, 2011.

[15] Cees Lanting and Antonio Lionetto. Smart systems and cyber physical systems paradigms in an iot and industrie/y4.0 context. page S5002, 11 2015.

[16] Brent Hailpern. Guest editor's introduction multiparadigm languages and environments. *IEEE Software*, 3(01):6–9, jan 1986.

[17] Pamela Zave. A compositional approach to multiparadigm programming. *IEEE Software*, 6(05):15–18, sep 1989.

[18] Thomas Kuhn. *The Structure of Scientific Revolutions*. Chicago Press, 2012.

[19] Erik W. Aslaksen. The engineering paradigm. *International Journal of Engineering Studies*, 5(02):29–154, 2013.

[20] Johannes Halbe, Jan Adamowski, and Claudia Pahl-Wostl. The role of paradigms in engineering practice and education for sustainable development. *Journal of Cleaner Production*, 106:272 – 282, 2015. Bridges for a more sustainable future: Joining Environmental Management for Sustainable Universities (EMSU) and the European Roundtable for Sustainable Consumption and Production (ERSCP) conferences.

[21] M. Amrani, D. Blouin, R. Heinrich, A. Rensink, H. Vangheluwe, and A. Wortmann. Towards a formal specification of multi-paradigm modelling. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 419–424, 2019.

[22] M. Amrani, D. Blouin, R. Heinrich, A. Rensink, H. Vangheluwe, and A. Wortmann. Multi-Paradigm Modeling for Cyber-Physical Systems: A Descriptive Framework. Manuscript submitted for publication. 2020.

[23] Rima Al Ali, Tomas Bures, Ilias Gerostathopoulos, Petr Hnetynka, Jaroslav Keznikl, Michal Kit, and Frantisek Plasil. Deeco: An ecosystem for cyber-physical systems. In *Companion Proceedings of the 36th International Conference on Software Engineering*, ICSE Companion 2014, pages 610–611, New York, NY, USA, 2014. ACM.

[24] Rocco De Nicola, Diego Latella, Alberto Lluch Lafuente, Michele Loreti, Andrea Margheri, Mieke Massink, Andrea Morichetta, Rosario Pugliese, Francesco Tiezzi, and Andrea Vandin. *The SCEL Language: Design, Implementation, Verification*, pages 3–71. Springer International Publishing, Cham, 2015.

[25] Rima Al-Ali. Uncertainty-Aware Self-Adaptive Component Design in Cyber-Physical System. Technical Report D3S-TR-2019-02, Department of Distributed and Dependable Systems, Charles University, 2019.

[26] M. Kit, F. Plasil, V. Matena, T. Bures, and O. Kovac. Employing domain knowledge for optimizing component communication. In *2015 18th International ACM SIGSOFT Symposium on Component-Based Software Engineering (CBSE)*, pages 59–64, May 2015.

[27] Nicklas Hoch, Henry-Paul Bensler, Dhaminda Abeywickrama, Tomáš Bureš, and Ugo Montanari. *The E-mobility Case Study*, pages 513–533. Springer International Publishing, Cham, 2015.

[28] Filip Krijt, Zbynek Jiracek, Tomas Bures, Petr Hnetynka, and Frantisek Plasil. Automated dynamic formation of component ensembles - taking advantage of component cooperation locality. In *Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development - Volume 1: MODELSWARD,*, pages 561–568. INSTICC, SciTePress, 2017.

[29] Tomas Bures, Vladimir Matena, Raffaela Mirandola, Lorenzo Pagliari, and Catia Trubiani. Performance modelling of smart cyber-physical systems. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, ICPE '18, pages 37–40, New York, NY, USA, 2018. ACM.

[30] Tomas Bures, Ilias Gerostathopoulos, Petr Hnetynka, Jaroslav Keznikl, Michal Kit, and Frantisek Plasil. *Gossiping Components for Cyber-Physical Systems*, pages 250–266. Springer International Publishing, Cham, 2014.

[31] F. Krijt, Z. Jiracek, T. Bures, P. Hnetynka, and I. Gerostathopoulos. Intelligent ensembles - a declarative group description language and java framework. In *2017 IEEE/ACM 12th International Symposium on Software*

Engineering for Adaptive and Self-Managing Systems (SEAMS), pages 116–122, May 2017.

[32] Rima Al Ali, Tomas Bures, Ilias Gerostathopoulos, Jaroslav Keznikl, and Frantisek Plasil. Architecture adaptation based on belief inaccuracy estimation. *2014 IEEE/IFIP Conference on Software Architecture (WICSA)*, 00(undefined):87–90, 2014.

[33] Jaroslav Keznikl, Tomas Bures, Frantisek Plasil, Ilias Gerostathopoulos, Petr Hnetynka, and Nicklas Hoch. Design of ensemble-based component systems by invariant refinement. In *Proceedings of the 16th International ACM Sigsoft Symposium on Component-based Software Engineering*, CBSE '13, pages 91–100, New York, NY, USA, 2013. ACM.

[34] Tomas Bures, Ilias Gerostathopoulos, Petr Hnetynka, Jaroslav Keznikl, Michal Kit, and Frantisek Plasil. Deeco: An ensemble-based component system. In *Proceedings of the 16th International ACM Sigsoft Symposium on Component-based Software Engineering*, CBSE '13, pages 81–90, New York, NY, USA, 2013. ACM.

[35] Michal Kit, Ilias Gerostathopoulos, Tomas Bures, Petr Hnetynka, and Frantisek Plasil. 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*, SEAMS '15, pages 93–96, Piscataway, NJ, USA, 2015. IEEE Press.

[36] Vladimir Matena, Tomas Bures, Ilias Gerostathopoulos, and Petr Hnetynka. 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*, SEAMS '16, pages 82–88, New York, NY, USA, 2016. ACM.

[37] Bart Broekman and Edwin Notenboom. *Testing Embedded Software*. Addison Wesley, 2003.

[38] Sebastian Wätzoldt, Stefan Neumann, Falk Benke, and Holger Giese. Integrated Software Development for Embedded Robotic Systems. In Itsuki Noda, Noriaki Ando, Davide Brugali, and James Kuffner, editors, *Proceedings of the 3rd International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, volume 7628 of *Lecture Notes in Computer Science*, pages 335–348. Springer Berlin Heidelberg, October 2012.

[39] Bedir Tekinerdogan and Ömer Köksal. *Pattern based integration of internet of things systems*, volume 10972 of *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pages 19–33. Springer Verlag, 6 2018.

[40] G. Giray, B. Tekinerdogan, and E. Tüzün. *IoT System Development Methods*, pages 141–159. CRC Press/Taylor & Francis, 1 2018.

[41] Freek van den Berg, Vahid Garousi, Bedir Tekinerdogan, and Boudewijn R. Haverkort. Designing cyber-physical systems with adsl: A domain-specific language and tool support. In *13th System of Systems Engineering Conference, SoSE 2018*, pages 225–232. Institute of Electrical and Electronics Engineers Inc., 8 2018.

[42] David Sinreich. An architectural blueprint for autonomic computing. 2006.

[43] P.J. Mosterman and H. Vangheluwe. Computer automated multi-paradigm modeling: An introduction. *Simulation*, 80(9):433–450, 2004. cited By 0.

[44] Jean Bézivin, Frédéric Jouault, Peter Rosenthal, and Patrick Valduriez. Modeling in the Large and Modeling in the Small. In *Model Driven Architecture*, volume 3599/2005 of *Lecture Notes in Computer Science (LNCS)*, pages 33–46. Springer-Verlag, 2005.

[45] Jean-Marie Favre. 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*, 2004.

[46] D. Harel and B. Rumpe. Modeling languages: Syntax, semantics and all that stuff, part i: The basic stuff. Technical report, ISR, 2000.

[47] Holger Giese, Tihamer Levendovszky, and Hans Vangheluwe, editors. *Summary of the Workshop on Multi-Modelling Paradigms: Concepts and Tools*, 2006.

[48] David Broman, Edward A. Lee, Stavros Tripakis, and Martin Törngren. Viewpoints, formalisms, languages, and tools for cyber-physical systems. In *Proceedings of the 6th International Workshop on Multi-Paradigm Modeling*, MPM '12, pages 49–54, New York, NY, USA, 2012. ACM.

[49] F. Dandashi, V. Lakshminarayan, and N. Schult. Multiformalism, multiresolution, multiscale modeling. volume 2016-February, pages 2622–2631, 2016. cited By 0.

[50] H. Reza and E. Grant. Model oriented software architecture. volume 2, pages 4–5, 2004. cited By 0.

[51] Mauro Iacono Marco Gribaudo. An introduction to multiformalism modeling. In Marco Gribaudo and Mauro Iacono, editors, *Theory and Application of Multi-Formalism Modeling*, pages 1–16. IGI Global, Hershey, 2014.

[52] S. Lacoste-Julien, H. Vangheluwe, J. De Lara, and P.J. Mosterman. Meta-modelling hybrid formalisms. pages 65–70, 2004. cited By 6.

[53] H. Vangheluwe and J. De Lara. Computer automated multi-paradigm modelling: Meta-modelling and graph transformation. volume 1, pages 595–603, 2003. cited By 14.

[54] B.P. Zeigler and H. Praehofer. Interfacing continuous and discrete models for simulation and control. *SAE Technical Papers*, 1998. cited By 0.

[55] B.P. Zeigler. Embedding dev&dess in devs: Characteristic behaviors of hybrid models. pages 125–132, 2006. cited By 0.

[56] Enrico Barbierato, Marco Gribaudo, and Mauro Iacono. Modeling hybrid systems in {SIMTHESys}. *Electronic Notes in Theoretical Computer Science*, 327:5 – 25, 2016.

[57] Enrico Barbierato, Marco Gribaudo, and Mauro Iacono. *Simulating Hybrid Systems Within SIMTHESys Multi-formalism Models*, pages 189–203. Springer International Publishing, Cham, 2016.

[58] S. Balsamo, G. Dei Rossi, and A. Marin. A survey on multi-formalism performance evaluation tools. pages 15–23, 2012.

[59] Kishor S. Trivedi. Sharpe 2002: Symbolic hierarchical automated reliability and performance evaluator. In *DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks*, page 544, Washington, DC, USA, 2002. IEEE Computer Society.

[60] G. Ciardo and A. S. Miner. Smart: the stochastic model checking analyzer for reliability and timing. In *Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings. First International Conference on the*, pages 338–339, Sept 2004.

[61] Gianfranco Ciardo, Andrew S. Miner, and Min Wan. Advanced features in smart: The stochastic model checking analyzer for reliability and timing. *SIGMETRICS Perform. Eval. Rev.*, 36(4):58–63, March 2009.

[62] G. Ciardo, R. L. Jones, III, A. S. Miner, and R. I. Siminiceanu. Logic and stochastic modeling with smart. *Perform. Eval.*, 63:578–608, June 2006.

[63] Falko Bause, Peter Buchholz, and Peter Kemper. A toolbox for functional and quantitative analysis of deds. In *Proceedings of the 10th International Conference on Computer Performance Evaluation: Modelling Techniques and Tools*, TOOLS '98, pages 356–359, London, UK, 1998. Springer-Verlag.

[64] W. H. Sanders. Integrated frameworks for multi-level and multi-formalism modeling. In *Petri Nets and Performance Models, 1999. Proceedings. The 8th International Workshop on*, pages 2–9, 1999.

[65] Graham Clark, Tod Courtney, David Daly, Dan Deavours, Salem Derisavi, Jay M. Doyle, William H. Sanders, and Patrick Webster. The mobius modeling tool. In *Proceedings of the 9th international Workshop on Petri Nets and Performance Models (PNPM'01)*, pages 241–, Washington, DC, USA, 2001. IEEE Computer Society.

[66] Tod Courtney, Shravan Gaonkar, Ken Keefe, Eric Rozier, and William H. Sanders. Möbius 2.3: An extensible tool for dependability, security, and performance evaluation of large and complex system models. In *DSN*, pages 353–358. IEEE, 2009.

[67] Daniel D. Deavours, Graham Clark, Tod Courtney, David Daly, Salem Derisavi, Jay M. Doyle, William H. Sanders, and Patrick G. Webster. The Möbius framework and its implementation, 2002.

[68] Juan de Lara and Hans Vangheluwe. Atom3: A tool for multi-formalism and meta-modelling. In Ralf-Detlef Kutsche and Herbert Weber, editors, *FASE*, volume 2306 of *Lecture Notes in Computer Science*, pages 174–188. Springer, 2002.

[69] M. Del V. Sosa, S.T. Acuna, and J. De Lara. Metamodeling and multiformalism approach applied to software process using atom [enfoque de metamodelado y multiformalismo aplicado al proceso software usando atom3]. pages 367–374, 2007.

[70] F. Franceschinis, M. Gribaudo, M. Iacono, N. Mazzocca, and V. Vittorini. 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.)*, pages 47–66. Technical Report DAIMI PB-561, aug 2002.

[71] V. Vittorini, G. Franceschinis, M. Gribaudo, M. Iacono, and N. Mazzocca. 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, April 2002.

[72] Giuliana Franceschinis, Marco Gribaudo, Mauro Iacono, Stefano Marrone, Nicola Mazzocca, and Valeria Vittorini. Compositional modeling of complex systems: Contact center scenarios in OsMoSys. In *ICATPN'04*, pages 177–196, 2004.

[73] G. Franceschinis, M. Gribaudo, M. Iacono, S. Marrone, F. Moscato, and V. Vittorini. Interfaces and binding in component based development of formal models. In *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*, VALUETOOLS '09, pages 44:1–44:10, ICST, Brussels, Belgium, Belgium, 2009.

ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

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

[75] Enrico Barbierato, Marco Gribaudo, and Mauro Iacono. Defining Formalisms for Performance Evaluation With SIMTHESys. *Electr. Notes Theor. Comput. Sci.*, 275:37–51, 2011.

[76] Enrico Barbierato, Marco Gribaudo, and Mauro Iacono. A performance modeling language for big data architectures. In Webjorn Rekdalsbakken, Robin T. Bye, and Houxiang Zhang, editors, *ECMS*, pages 511–517. European Council for Modeling and Simulation, 2013.

[77] Mauro Iacono and Marco Gribaudo. Element based semantics in multi formalism performance models. In *MASCOTS*, pages 413–416, 2010.

[78] M. Iacono, E. Barbierato, and M. Gribaudo. The SIMTHESys multiformalism modeling framework. *Computers and Mathematics with Applications*, (64):3828–3839, 2012.

[79] Mauro Pezze and Michal Young. Constructing multi-formalism state-space analysis tools: Using rules to specify dynamic semantics of models. pages 239–249, 1997. cited By 0.

[80] Enrico Barbierato, Gian-Luca Dei Rossi, Marco Gribaudo, Mauro Iacono, and Andrea Marin. Exploiting product forms solution techniques in multiformalism modeling. *Electronic Notes in Theoretical Computer Science*, 296(0):61 – 77, 2013.

[81] C.-V. Bobeanu, E.J.H. Kerckhoffs, and H. Van Landeghem. Modeling of discrete event systems: A holistic and incremental approach using petri nets. *ACM Transactions on Modeling and Computer Simulation*, 14(4):389–423, 2004. cited By 0.

[82] J.T. Bradley, M.C. Guenther, R.A. Hayden, and A. Stefanek. *GPA: A multiformalism, multisolution approach to efficient analysis of Large-Scale population models.* 2013. cited By 0.

[83] Aniello Castiglione, Marco Gribaudo, Mauro Iacono, and Francesco Palmieri. Exploiting mean field analysis to model performances of big data architectures. *Future Generation Computer Systems*, 37(0):203–211, 2014.

[84] A.H. Levis and B. Yousefi. Multi-formalism modeling for evaluating the effect of cyber exploits. pages 541–547, 2014. cited By 0.

[85] A.M. Abusharekh and A.H. Levis. Performance evaluation of soa in clouds. pages 614–620, 2016. cited By 0.

[86] Mauro Iacono and Stefano Marrone. Model-based availability evaluation of composed web services. *Journal of Telecommunications and Information Technology*, (4):5–13, 2014.

[87] A.I. Hernandez, V. Le Rolle, A. Defontaine, and G. Carrault. 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):4923–4940, 2009. cited By 0.

[88] S. Chiaradonna, P. Lollini, and F.D. Giandomenico. On a modeling framework for the analysis of interdependencies in electric power systems. pages 185–194, 2007. cited By 0.

[89] Francesco Flammini, Stefano Marrone, Mauro Iacono, Nicola Mazzocca, and Valeria Vittorini. A multiformalism modular approach to ERTMS/ETCS failure modelling. *International Journal of Reliability, Quality and Safety Engineering*, 21(01):1450001–1–1450001–29, 2014.

[90] Marco Gribaudo, Mauro Iacono, and Stefano Marrone. Exploiting bayesian networks for the analysis of combined attack trees. *Electronic Notes in Theoretical Computer Science*, 310(0):91 – 111, 2015. Proceedings of the Seventh International Workshop on the Practical Application of Stochastic Modelling (PASM).

[91] Enrico Barbierato, Marco Gribaudo, Mauro Iacono, and Stefano Marrone. Performability modeling of exceptions-aware systems in multiformalism tools. In *ASMTA*, pages 257–272, 2011.

[92] Enrico Barbierato, Andrea Bobbio, Marco Gribaudo, and Mauro Iacono. Multiformalism to support software rejuvenation modeling. In *ISSRE Workshops*, pages 271–276. IEEE, 2012.

[93] Enrico Barbierato, Marco Gribaudo, and Mauro Iacono. Performance evaluation of NoSQL big-data applications using multi-formalism models. *Future Generation Computer Systems*, 37(0):345–353, 2014.

[94] Daniele Codetta Raiteri, Mauro Iacono, Giuliana Franceschinis, and Valeria Vittorini. Repairable fault tree for the automatic evaluation of repair policies. In *DSN*, pages 659–668, 2004.

[95] Enrico Barbierato, Marco Gribaudo, and Mauro Iacono. 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*, 2011.

[96] A. Qamar, S.J.I. Herzig, C.J.J. Paredis, and M. Torngren. Analyzing semantic relationships between multiformalism models for inconsistency management. pages 84–89, 2015.

[97] Jean Bézivin, Frédéric Jouault, and Patrick Valduriez. 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*, 2004.

[98] CoEST Project Homepage. `http://www.coest.org/`, accessed 2016.

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

[100] Epsilon Project Homepage. `http://eclipse.org/epsilon/`, 2020.

[101] Jean-Marie Favre, Ralf Lämmel, and Andrei Varanovich. Modeling the linguistic architecture of software products. In *Proceedings of the 15th international conference on Model Driven Engineering Languages and Systems*, MODELS'12, pages 151–167, Berlin, Heidelberg, 2012. Springer-Verlag.

[102] Henrik Lochmann and Anders Hessellund. An Integrated View on Modeling with Multiple Domain-Specific Languages. In *Proceedings of the IASTED International Conference Software Engineering (SE 2009)*, pages 1–10. ACTA Press, February 2009.

[103] Sebastian J.I. Herzig, Ahsan Qamar, and Christiaan J.J. Paredis. An Approach to Identifying Inconsistencies in Model-based Systems Engineering. *Procedia Computer Science*, 28(0):354 – 362, 2014. 2014 Conference on Systems Engineering Research.

[104] Gabor Simko, Tihamer Levendovszky, Sandeep Neema, Ethan Jackson, Ted Bapty, Joseph Porter, and Janos Sztipanovits. Foundation for model integration: Semantic backplane. In *ASME 2012 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 1077–1086. American Society of Mechanical Engineers, 2012.

[105] Mark Boddy, Martin Michalowski, August Schwerdfeger, Hazel Shackleton, Steve Vestal, and Adventium Enterprises. FUSED: A Tool Integration Framework for Collaborative System Engineering. In *Analytic Virtual Integration of Cyber-Physical Systems Workshop*, 2011.

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

[107] Andreas Seibel, Stefan Neumann, and Holger Giese. Dynamic hierarchical mega models: comprehensive traceability and its efficient maintenance. *Software and Systems Modeling*, 9(4):493–528, 2010.

[108] Andreas Seibel, Regina Hebig, and Holger Giese. Traceability in Model-Driven Engineering: Efficient and Scalable Traceability Maintenance. In Jane Cleland-Huang, Orlena Gotel, and Andrea Zisman, editors, *Software and Systems Traceability*, pages 215–240. Springer London, 2012.

[109] Thomas Beyhl, Regina Hebig, and Holger Giese. A Model Management Framework for Maintaining Traceability Links. In Stefan Wagner and Horst Lichter, editors, *Software Engineering 2013 Workshopband*, volume P-215 of *Lecture Notes in Informatics (LNI)*, pages 453–457, Aachen, February 2013. Gesellschaft für Informatik (GI).

[110] D. Langsweirdt, N. Boucke, and Yolande Berbers. 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*, pages 138–144, May 2010.

[111] Anders Hessellund and Andrzej Wasowski. Interfaces and Metainterfaces for Models and Metamodels. In Krzysztof Czarnecki, Ileana Ober, Jean-Michel Bruel, Axel Uhl, and Markus Wolter, editors, *Model Driven Engineering Languages and Systems*, volume 5301 of *Lecture Notes in Computer Science*, pages 401–415. Springer Berlin Heidelberg, 2008.

[112] Stefan Jurack and Gabriele Taentzer. 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*, pages 187–202, 2010.

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

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

[115] Arvid Butting, Robert Eikermann, Oliver Kautz, Bernhard Rumpe, and Andreas Wortmann. Systematic composition of independent language features. *Journal of Systems and Software*, 152:50 – 69, 2019.

[116] MontiCore Project Homepage. `http://www.monticore.de/`, 2008.

[117] Arvid Butting, Robert Eikermann, Oliver Kautz, Bernhard Rumpe, and Andreas Wortmann. Modeling language variability with reusable language components. In *Proceedings of the 22nd International Systems and Software Product Line Conference - Volume 1*, SPLC '18, page 65–75, New York, NY, USA, 2018. Association for Computing Machinery.

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

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

[120] Anne Etien, Alexis Muller, Thomas Legrand, and Xavier Blanc. Combining Independent Model Transformations. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 2237–2243, New York, NY, USA, 2010. ACM.

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

[122] Dominique Blouin, Yvan Eustache, and Jean-Philippe Diguet. 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,* -, pages 43–52, 2014.

[123] Davide Di Ruscio, Ivano Malavolta, Henry Muccini, Patrizio Pelliccione, and Alfonso Pierantonio. Model-driven techniques to enhance architectural languages interoperability. In Juan de Lara and Andrea Zisman, editors, *Fundamental Approaches to Software Engineering*, pages 26–42, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[124] Arnaud Blouin, Benoit Combemale, Benoît Baudry, and Olivier Beaudoux. Kompren: modeling and generating model slicers. *Software & Systems Modeling*, 14(1):321–337, 2015.

[125] Hugo Bruneliere, Jokin Garcia Perez, Manuel Wimmer, and Jordi Cabot. Emf views: A view mechanism for integrating heterogeneous models. In Paul Johannesson, Mong Li Lee, Stephen W. Liddle, Andreas L. Opdahl, and Óscar Pastor López, editors, *Conceptual Modeling*, pages 317–325, Cham, 2015. Springer International Publishing.

[126] Xavier Blanc, Alix Mougenot, Isabelle Mounier, and Tom Mens. 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, pages 32–46, Berlin, Heidelberg, 8-12 June 2009. Springer Verlag.

[127] VIATRA Project Homepage. `https://www.eclipse.org/viatra/`, accessed 2020.

[128] Zoltan Ujhelyi, Gabor Bergmann, Abel Hegedus, Akos Horvath, Benedek Izso, Istvan Rath, Zoltan Szatmari, and Daniel Varro. EMF-IncQuery: An integrated development environment for live model queries. *Science of Computer Programming*, 98, Part 1:80–99, 2015. Fifth issue of Experimental Software and Toolkits (EST): A special issue on Academics Modelling with Eclipse (ACME2012).

[129] Alexander Egyed. Instant Consistency Checking for the UML. In *ICSE '06: Proceedings of the 28th International Conference on Software Engineering*, pages 381–390, Shanghai, China, 20-28 May 2006.

[130] Iris Groher, Alexander Reder, and Alexander Egyed. Incremental Consistency Checking of Dynamic Constraints. In DavidS. Rosenblum and Gabriele Taentzer, editors, *Fundamental Approaches to Software Engineering*, volume 6013 of *Lecture Notes in Computer Science*, pages 203–217. Springer Berlin Heidelberg, 2010.

[131] Jordi Cabot and Ernest Teniente. 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)*, pages 81–95. Springer Verlag, 5-9 June 2006.

[132] AM3 Project Homepage. `https://wiki.eclipse.org/AM3`, 2014.

[133] ATL Project Homepage. `https://eclipse.org/atl/`, 2015.

[134] Andrés Vignaga, Frédéric Jouault, MaríaCecilia Bastarrica, and Hugo Brunelière. Typing artifacts in megamodeling. *Software & Systems Modeling*, 12:105–119, 2013.

[135] José E. Rivera, Daniel Ruiz-Gonzalez, Fernando Lopez-Romero, José Bautista, and Antonio Vallecillo. Orchestrating ATL Model Transformations. In Frédéric Jouault, editor, *In Proc. of MtATL 2009: 1st International Workshop on Model Transformation with ATL*, pages 34–46, Nantes, France, July 2009.

[136] ATLFlow Project Homepage. `http://opensource.urszeidler.de/ATLflow/`, accessed 2020.

[137] Moharram Challenger, Ken Vanherpen, Joachim Denil, and Hans Vangheluwe. *FTG+PM: Describing Engineering Processes in Multi-Paradigm Modelling*, pages 259–271. Springer International Publishing, Cham, 2020.

[138] AToMPM Project Homepage. `https://atompm.github.io/`, accessed 2020.

[139] AMW Project Homepage. `https://wiki.eclipse.org/ATL/EMFTVM/`, accessed 2020.

[140] Csaba Debreceni, Akos Horvath, Abel Hegedus, Zoltan Ujhelyi, Istvan Rath, and Daniel Varro. Query-driven Incremental Synchronization of View Models. In *Proceedings of the 2Nd Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling*, VAO '14, pages 31:31–31:38, New York, NY, USA, 2014. ACM.

[141] Andreas Seibel, Regina Hebig, Stefan Neumann, and Holger Giese. 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*, July 2011.

[142] Holger Giese, Stefan Neumann, Oliver Niggemann, and Bernhard Schätz. Model-Based Integration. In Holger Giese, Gabor Karsai, Edward Lee, Bernhard Rumpe, and Bernhard Schätz, editors, *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*, pages 17–54. Springer, 2011.

[143] GME Project Homepage. `http://www.isis.vanderbilt.edu/projects/gme/`, accessed 2020.

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

[145] OSLC Project Homepage. `http://open-services.net/`, accessed 2020.

[146] Sebastian JI Herzig and Christiaan JJ Paredis. Bayesian Reasoning Over Models. In *11th Workshop on Model Driven Engineering, Verification and Validation MoDeVVa 2014*, page 69, 2014.

[147] An Eclipse-based workbench for INTeractive Model Management. `https://github.com/adisandro/MMINT`, 2014.

[148] Alessio Di Sandro, Rick Salay, Michalis Famelis, Sahar Kokaly, and Marsha Chechik. Mmint: A graphical tool for interactive model management. In *P&D@ MoDELS*, pages 16–19, 2015.

[149] MegaM@Rt2 framework. `https://megamart2-ecsel.eu/`, 2017.

[150] Wasif Afzal, Hugo Bruneliere, Davide Di Ruscio, Andrey Sadovykh, Silvia Mazzini, Eric Cariou, Dragos Truscan, Jordi Cabot, Abel Gómez, Jesús Gorroñogoitia, et al. The megam@ rt2 ecsel project: Megamodelling at runtime–scalable model-based framework for continuous development and runtime validation of complex systems. *Microprocessors and microsystems*, 61:86–95, 2018.

[151] Regina Hebig, Andreas Seibel, and Holger Giese. On the Unification of Megamodels. In Vasco Amaral, Hans Vangheluwe, Cécile Hardebolle, Laszlo Lengyel, Tiziana Magaria, Julia Padberg, and Gabriele Taentzer, editors, *Proceedings of the 4th International Workshop on Multi-Paradigm Modeling (MPM 2010)*, volume 42 of *Electronic Communications of the EASST*, 2011.

[152] Dominique Blouin, Gilberto Ochoa Ruiz, Yvan Eustache, and Jean-Philippe Diguet. 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, June 2015.

[153] R. Hilliard, I. Malavolta, H. Muccini, and P. Pelliccione. On the composition and reuse of viewpoints across architecture frameworks. In *2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*, pages 131–140, Aug 2012.

[154] Rich Hilliard, Ivano Malavolta, Henry Muccini, and Patrizio Pelliccione. Realizing architecture frameworks through megamodelling techniques. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE2010)*, pages 305–308, New York, NY, USA, 2010. ACM.

[155] T. Bures, P. Hnetynka, J. Kofron, R. A. Ali, and D. Skoda. Statistical approach to architecture modes in smart cyber physical systems. In *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, pages 168–177, April 2016.

[156] Ankica Barišić, Dušan Savić, Rima Al-Ali, Ivan Ruchkin, Dominique Blouin, Antonio Cicchetti, Raheleh Eslampanah, Oksana Nikiforova, Mustafa Abshir, Moharram Challenger, Claudio Gomes, Ferhat Erata, Bedir Tekinerdogan, Vasco Amaral, and Miguel Goulao. *Systematic Literature Review on Multi-Paradigm Modeling for Cyber-Physical Systems*, December 2018.

[157] Hans Vangheluwe, Ghislain Vansteenkiste, and Eugene Kerckhoffs. Simulation for the Future: Progress of the ESPRIT Basic Research working group 8467. In *European Simulation Symposium (ESS)*. SCS, 1996.

[158] Peter Van Roy. *Concepts, Techniques, and Models of Computer Programming*, chapter Programming Paradigms for Dummies: What Every Programmer Should Know, pages 9–47. MIT Press, 04 2012.

[159] Jean Bézivin. On the Unification Power of Models. *Software and Systems Modeling*, 4(2):171–188, 2005.

[160] Ian Watson and John G. Gürd. A Practical Data Flow Computer. *IEEE Computer*, 15:51–57, 1982.

[161] Edward A. Lee and David G. Messerschmitt. Static Scheduling of Synchronous Data Flow Programs For Digital Signal Processing. *IEEE Transactions on Computers*, 36(1):24–35, 1987.