# Chapter 9
# FTG+PM: Describing Engineering Processes in Multi-Paradigm Modelling

Moharram Challenger, Ken Vanherpen, Joachim Denil, and Hans Vangheluwe

**Abstract** Model-based System Engineering (MBSE) is a methodology that uses models throughout the engineering to replace the paper-based approach of communication among stakeholders. Multi-Paradigm Modelling (MPM) is at the core of this engineering approach as for each phase in the engineering process the most appropriate models at the most appropriate levels of abstraction is used. A design process includes a set of activities in which the design decisions or evaluations of the (sub-) system properties are done. Furthermore, the design artifacts are transformed by the design activities. We can define transformations as the manipulation of a model with a specific purpose. MPM approaches do not have a standard way of representing processes. A process model for MPM should focus on the languages, model instances and transformations between these models at different levels of abstraction. In this chapter, we propose the Formalism Transformation Graph and Process Model (FTG+PM) as a standard representation of MPM processes. The described process can be simulated for analysis and orchestration, as a set of (automatic) transformations.

## Learning Objectives

After reading this chapter, we expect you to be able to:

- Understand why modelling the design process is of importance.
- Represent the MPM processes using the Formalism Transformation Graph and Process Model (FTG+PM).
- Reason on the orchestration of a modelled design process to support the designers.

## 9.1 Introduction

To tackle the increasing complexity of today's systems, engineers already practice a Model-Based Systems Engineering (MBSE) methodology [95]. In MBSE, models are used to support requirements engineering,

Moharram Challenger
University of Antwerp, Belgium
e-mail: moharram.challenger@uantwerpen.be

Ken Vanherpen
University of Antwerp, Belgium
e-mail: ken.vanherpen@uantwerpen.be

Joachim Denil
Flanders Make, Belgium
e-mail: joachim.denil@uantwerp.be

Hans Vangheluwe
McGill University, Canada
e-mail: hans.vangheluwe@uantwerp.be



P. Carreira et al. (eds.), *Foundations of Multi-Paradigm Modelling for Cyber-Physical Systems*,
https://doi.org/10.1007/978-3-030-43946-0_9

design, verification, and validation activities of a system beginning in the conceptual design phase and continuing throughout development and later life cycle phases [153]. While these models often operate at different abstraction levels, model transformations are applied to manipulate models between different appropriate representations. They are typically used for code synthesis, integration, analysis, simulation, and optimization purposes. Multi-Paradigm Modelling (MPM) as a method consolidates these modelling methods and techniques, encouraging engineers to model each aspect of the system explicitly at the most appropriate level(s) of abstraction using the most appropriate formalism(s) [212].

While engineering a system, a design process is followed, in which a set of design activities are executed in a well-defined order. In each design activity a set of design artifacts are consumed and/or produced. These design artifacts explicitly model the informed design decisions taken by the engineers or the evaluations done by the engineer. As such, they can also be regarded as model transformations where information in a set of input models is consumed to produce a set of output models.

Although this series of design activities should result in an operational system that meets the predefined set of requirements, MPM approaches do not have a standard way of representing the process. This is in spite of the many advantages that the explicit modelling of the development process can offer, such as optimization, consistency analysis, time and risk analysis, etc. Generally, having well defined processes reduces the risk of failing a design project, as we can explicitly reason on problems that have to be overcome in the design of a system [143].

In this chapter, we propose the Formalism Transformation Graph and Process Model (FTG+PM) as a standard representation of MPM processes. As its name implies, it consists of two (related) parts: (a) the FTG where the focus is on the languages and transformations used throughout the process and (b) the flow of design activities and design artifacts. As tool-support is crucial in the MBSE (and MPM) life-cycle, we demonstrate how the orchestration of a modelled design process can support designers by scheduling transformations.

## 9.2 Model-based Systems Engineering

Document-centric approaches used to be at the core of communicating design choices between different engineers. However, as systems steadily enlarge in size, number of components, and features offered to the user, this is no longer appropriate. The added systems' complexity leads to an increase in the development time, number of errors, and development cost. As such, model-based systems engineering (MBSE) approaches have gained momentum to mend these shortcomings [95]. In MBSE, models are used as an integral part of the technical baseline of the system under design. This includes models in different development phases of a capability, system, and/or product [76].

MBSE is an engineering methodology in which the domain models and modeling techniques are used as the primary means of information exchange between engineers, rather than on document-based information exchange. The MBSE approach was outlined and popularized by INCOSE when it kicked off its MBSE initiative in January 2007 [154]. It focuses on distributed but integrated model management and the main goal is to increase the productivity and reducing the risk, by minimizing unnecessary manual transcription of concepts when coordinating the work of large teams. MBSE methodology provides several advantages:

- System Engineers focus on the technicalities of the problem rather than document structure
- Diagrammatic descriptions are often less ambiguous than textural descriptions
- Greater consistency across related documents
- Dependencies are explicitly captured across stovepipes resulting in less duplication and inconsistency

The MBSE methodology supports system requirement engineering, analysis, design, implementation, verification, and validation activities of a system beginning in the conceptual design phase and continuing throughout development and later life cycle phases [154]. There are a multitude of modelling techniques and approaches that fall within MBSE. Some of them include:

- Structured Analysis and Design
- Data Flow Diagramming
- State Transition Diagramming
- Behavioural Modelling
- Entity Relationship Modelling

- Process Modelling

Recently, the focus of MBSE is also the model execution in computer simulation experiment, to overcome the gap between the system model and the respective simulation software. As a result, the term Modeling and Simulation-based Systems Engineering (M&SBSE) has also been used along with MBSE [123].

The models used in an MBSE approach pave the way to tackle system complexity by providing proper levels of abstraction where engineers make design choices or analyze the systems' capabilities or performance. Models operate at a similar or different levels of abstraction called horizontal and vertical abstractions, respectively [124][254]. The level of abstraction can be regarded as which properties of the system under design are taken into account in the models. Certain design approaches, such as the Model Driven Architecture (MDA) approach, explicitly introduce such levels of abstractions: Platform-specific Modeling (PSM) level or Platform Independent Modeling (PIM) level [166]. In this sense, the horizontal abstraction can be either among PIM models or PSM models. While the vertical abstraction can be considered among PIM and PSM models.

Model transformations can be applied to manipulate models from one abstraction to another. They are typically used for code synthesis, integration, analysis, simulation, and optimization purposes. By using model transformations, models in one formalism are automatically transformed to another formalism. This allows the possibility to benefit from the power of the target formalism, e.g. to simulate the system under design. Two types of transformations exist, Model-to-Model (M2M) and Model-to-Text (M2T) transformations.

Multi-paradigm Modelling (MPM) consolidates the modeling methods and techniques, such as MBSE, enabling engineers to model each aspect of the system explicitly at the most appropriate level(s) of abstraction using the most appropriate formalism(s), while modeling the development process(es) explicitly [213].

Cyber-physical Systems (CPS) are one of the inter-disciplinary systems which require different formalisms, levels of abstraction, tools, and viewpoints in their design and development. Therefore, they need for MPM and MBSE techniques to deal with the complexity and available risks. Also, the evolution in the requirements of different views during the CPS design and development needs for a MPM process management. The next sections deals with the life-cycle and design process in MPM for complex systems such as CPSs and IoT systems [307].

## 9.3 Development Lifecycle Models

As designing complex systems involves engineers from various disciplines, engineers follow a set of guidelines to ensure that the product implements the given system requirements. The order in which the guidelines are followed is called the design process. Inspired by the software engineering community, that has defined Software Development Life Cycle (SDLC) models over the past decades [246]. We can distinguish four models that are commonly used when designing complex systems: the waterfall model, the V model, the spiral model, and the agile model. They are conceptually shown in Figure 9.1.

### 9.3.1 Waterfall Model

The waterfall model is typed by a sequence of design steps/activities in which a step/activity can only be started if one deliverable from the previous step/activity is arrived, as such it is a gated process. The gate is a decision point to define if a next phase of the project can start. The process starts by defining a set of requirements, which are then refined to a set of technical requirements called specifications. Given these specification, engineers reason about a high-level architecture in which the structure of the system is defined. In the detailed design and implementation phase, the behaviour of the architectural elements is defined after which the system under design is tested. Finally, the designed system is deployed in its intended environment. There is often feed-back between steps/activities.

Note that the waterfall suits projects that are not subject to changes in one of the previous phases during the process e.g. the requirement should not change during architecture design. Otherwise, the design process must start over which may be costly in large engineering projects.

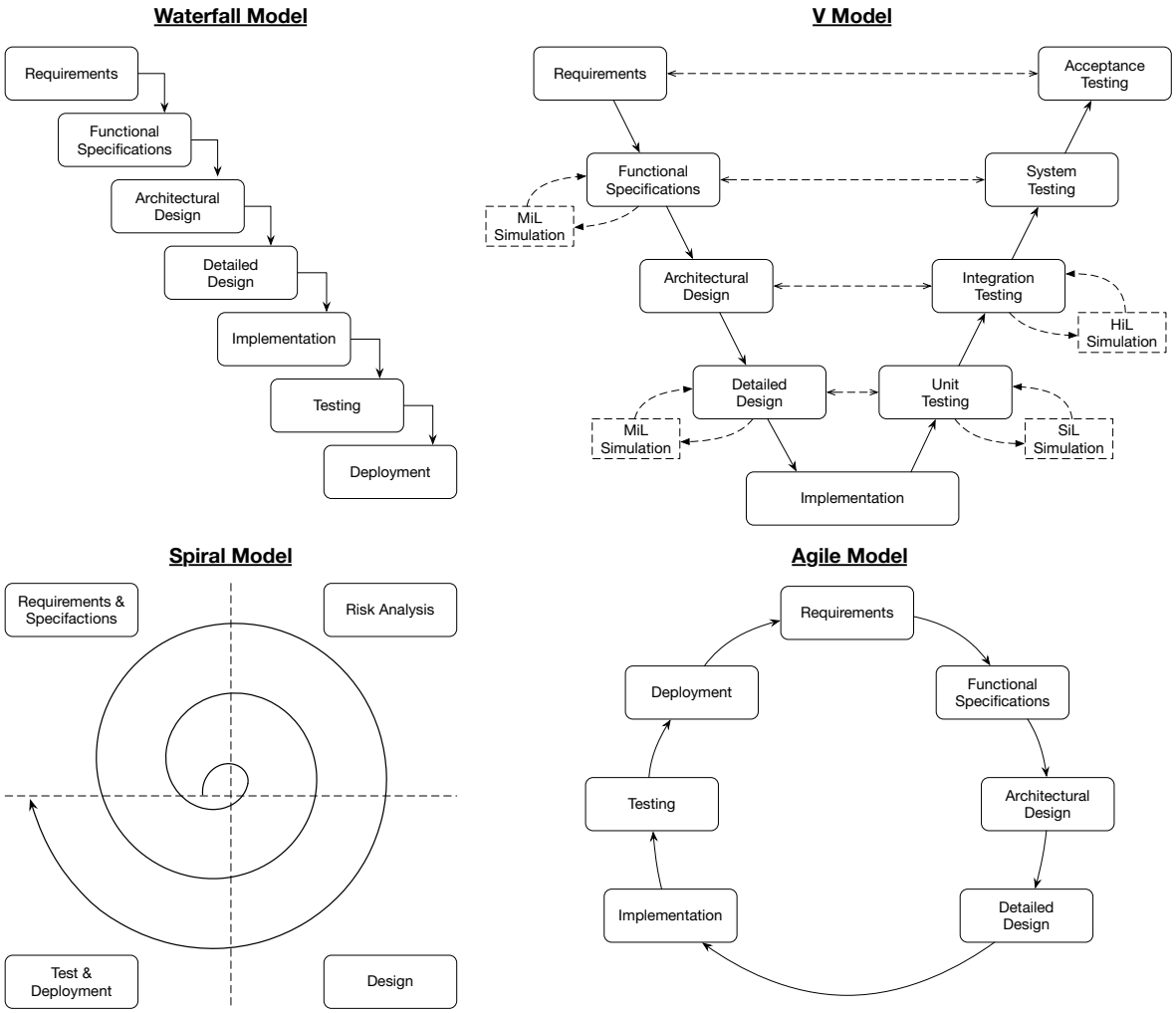


Fig. 9.1: Conceptual overview of commonly used MBSE design processes

### 9.3.2 V Model

Similar to the waterfall model, the V model is typed by a sequential design process. However, the V model distinguishes between *design* and *verification & validation* steps, respectively on the left-hand side and right-hand side of the V model, while keeping them tightly coupled as illustrated by the horizontal dashed lines, see Figure 9.1. While going from high-level requirements to a low-level implementation, the specifications of the design are verified by executing acceptance tests, ranging from low-level unit tests to high-level system tests. Ultimately, at the top of the V model, it is validated whether the system satisfies the requirements. Model-based techniques allow engineers to front-load a large set of the V&V activities.

Although the verification steps are executed while designing the system so that late detection of implementation errors can be avoided, a changing product requirement still requires one to redesign (parts of) the system which, again, may be costly. Nevertheless, in the context of designing (large) complex systems using a MBSE approach, the V model is widely accepted as the industry standard for designing and testing complex systems [95].

### 9.3.3 Spiral Model

Using an iterative design process, spiral model avoid these (costly) redesign by repeatedly going through four design phases: Requirements and specification election, risk analysis, design, and testing and deployment. In the first iteration, preliminary requirements and specifications are elected so that the first prototype of the system can be designed, tested and deployed.

In subsequent iterations, requirements and specifications are added and/or become more detailed so that the initial prototype further evolves towards the final product. During each iteration, the process is monitored such that possible project risks (e.g. technical feasibility and project cost) can be better assessed. Although this design model better mitigates changing requirements and design errors, one should note that an increasing number of iterations can lead to an increasing (cumulative) project cost.

### 9.3.4 Agile Model

Agile design process models are typed by so called sprints (i.e. design iterations) in which a small part of the overall system is designed, tested, and deployed. At the end of each sprint, a finalized, working, system is delivered to the stakeholders. This differs from a spiral design approach in which a prototype might be delivered. Moreover, the design iterations of the agile design approach are typically shorter than the ones of the spiral design approach. Agile design processes are therefore suitable when requirements are subject to change during the design and implementation phases, and when multidisciplinary teams need to cooperate. On the downside, the lack of a thorough system analysis may result in a more complex and more expensive design process.

## 9.4 Modelling the Design Process

A process for a complex system (such as CPS) should cover different aspects of the system and should make the different levels of abstraction clear. These levels of abstraction could be at the domain-specific design of the computational or physical components, the verification of the system at a specific abstraction level, and/or during deployment where different approximations can be used to obtain a better understanding of the parameters involved. At the defined level of abstraction, it has to be clear which languages and transformations are involved and how they are used together in the design and development process. Though, from a tool building perspective, this definition could be too restrictive, as some activities can require manual intervention before the activity are completed. Since tool-support is crucial in the MDE (and MPM) life-cycle, it is important that the described process can be executed automatically. One approach to do this is by means of a set of transformations scheduled after each other.

The process model has to act as a guide for the design, verification and deployment of the system. This means that the design of a large set of applications has to be described using the modelling language. The language should focus on the constructs of MPM, mainly formalisms and transformations and how these interact with each other. There should also be an explicit representation of control- and data-flow since the output of a single phase in the process does not necessarily means that the produced models are the input of the next phase. This also includes the use of control structures that allow parallel design, iterations, etc.

### 9.4.1 Rationale

Over the years, the process engineering community has proposed various process modelling means for software development. Process modelling has a large research community, resulted in many modelling languages. Rolland [241] gives a definition of process modelling: *Process models are processes of the same nature that are classified together into a model. Thus, a process model is a description of a process at the type level. Since the process model is at the type level, a process is an instantiation of it. The same process model is used repeatedly for the development of many applications and thus, has many instantiations.* [241]

Generally, a process model is used for three different goals:

- **Descriptive:** The models are used by an external observer to look at what happens during a process. It can be used to improve the process.
- **Prescriptive:** The models define a set of rules to prescribe a desired process. If this process is followed, it would lead to a desired outcome.
- **Explanatory:** The goal of an explanatory model is to explain the rationale behind the process.

### 9.4.2 What to model?

The process models can describe different aspects of the system development. These aspects can be divided in four groups (see Figure 9.2), namely: Functional view, Dynamic view, Informational view and Organization view [78].

- **Functional view:** This view of a process model discusses the dependencies in the functions (i.e. the transformations) of the system under development. A well-known example of a modeling language employing this perspective is data flow diagrams.
- **Dynamic/Behavior view:** This view focuses on the activities, actions, and tasks of the system and their sequences. This aspect deals with the control-flow and the timing of the activities in the system.
- **Informational view:** This view deals with the informational entities (i.e. the data for process, activities, artifacts, and products) which are produced or manipulated by the process. This view includes both the structure of the informational entities and the relationships among them.
- **Organisational view:** This view presents the stakeholders of the system as well as who performs the process elements in what part of the organization.


Fig. 9.2: Describing Processes

### 9.4.3 Reasoning about maturity

When processes are repeated multiple times, it is important to extend, evolve and mature the process models as well. Adding more information to process models can ultimately lead to better (e.g. faster, more robust, etc.) processes. The term *maturity* relates to the degree of formality and optimization of processes, from ad-hoc practices, to formally defined steps, to managed result metrics, to active optimization of the processes.

The Capability Maturity Model (CMM[1]) is a methodology used to develop and refine a development process. The model's aim is to improve existing development processes. CMM was developed and is promoted by the Software Engineering Institute (SEI), a research and development center sponsored by the U.S. Department of Defense (DoD). CMM was originally developed as a tool for objectively assessing the ability of government contractors' processes to implement a contracted software project [303].

[1] CMM is a registered service mark of Carnegie Mellon University (CMU).

The CMM is similar to ISO 9001, one of the ISO 9000 series of standards specified by the International Organization for Standardization (ISO). The ISO 9000 standards specify an effective quality system for manufacturing and service industries; ISO 9001 deals specifically with software development and maintenance. The main difference between the two systems lies in their respective purposes: ISO 9001 specifies a minimal acceptable quality level for software processes, while the CMM establishes a framework for continuous process improvement and is more explicit than the ISO standard in defining the means to be employed to that end [198].

CMM describes a five-level evolutionary path, see Figure 9.3, of increasingly organized and systematically more mature processes. These maturity levels of processes are:

- At the *ad-hoc/initial* level, the processes are disorganized, even chaotic. Success is likely to depend on individual efforts, and is not considered to be repeatable, because processes would not be sufficiently defined and documented to allow them to be replicated.
- At the *repeatable* level, basic project management techniques are established, and successes could be repeated, because the requisite processes would have been made established, defined, and documented.
- At the *defined* level, an organization has developed its own standard process model through greater attention to documentation, standardization, and integration.
- At the *managed/capable* level, an organization monitors and controls its own processes through data collection and analysis.
- At the *optimized/efficient* level, processes are constantly being improved through monitoring feedback from current processes and introducing innovative processes to better serve the organization's particular needs.


Fig. 9.3: Capability Maturity Model (adopted from: http://performancexpress.org/)

## 9.5 Activities 2.0 for modelling processes

When engineering complex system using a MBSE approach, design artifacts will be created at different levels of abstraction within the same or between different engineering domains. A model of the design process should make the relation between different design artifacts, and their respective abstraction level, clear. This process

model has to act as a guide for the design, verification & validation, and implementation of the system. There should also be an explicit representation of control- and data-flow since the output of a single phase in the process does not necessarily means that the produced models are the input of the next phase. This also includes the use of control structures that allow parallel design, iterations, etc.

In [194, 196, 216] a Process Model (PM) language is proposed to describe the control and data flow between design activities. To this end, a subset of the Unified Modelling Language (UML) 2.0 activity diagrams are used. It enables to define the process as a descriptive and prescriptive model.

An example of a PM, describing the (partial) process of generating code from a control model, is shown in Figure 9.4.

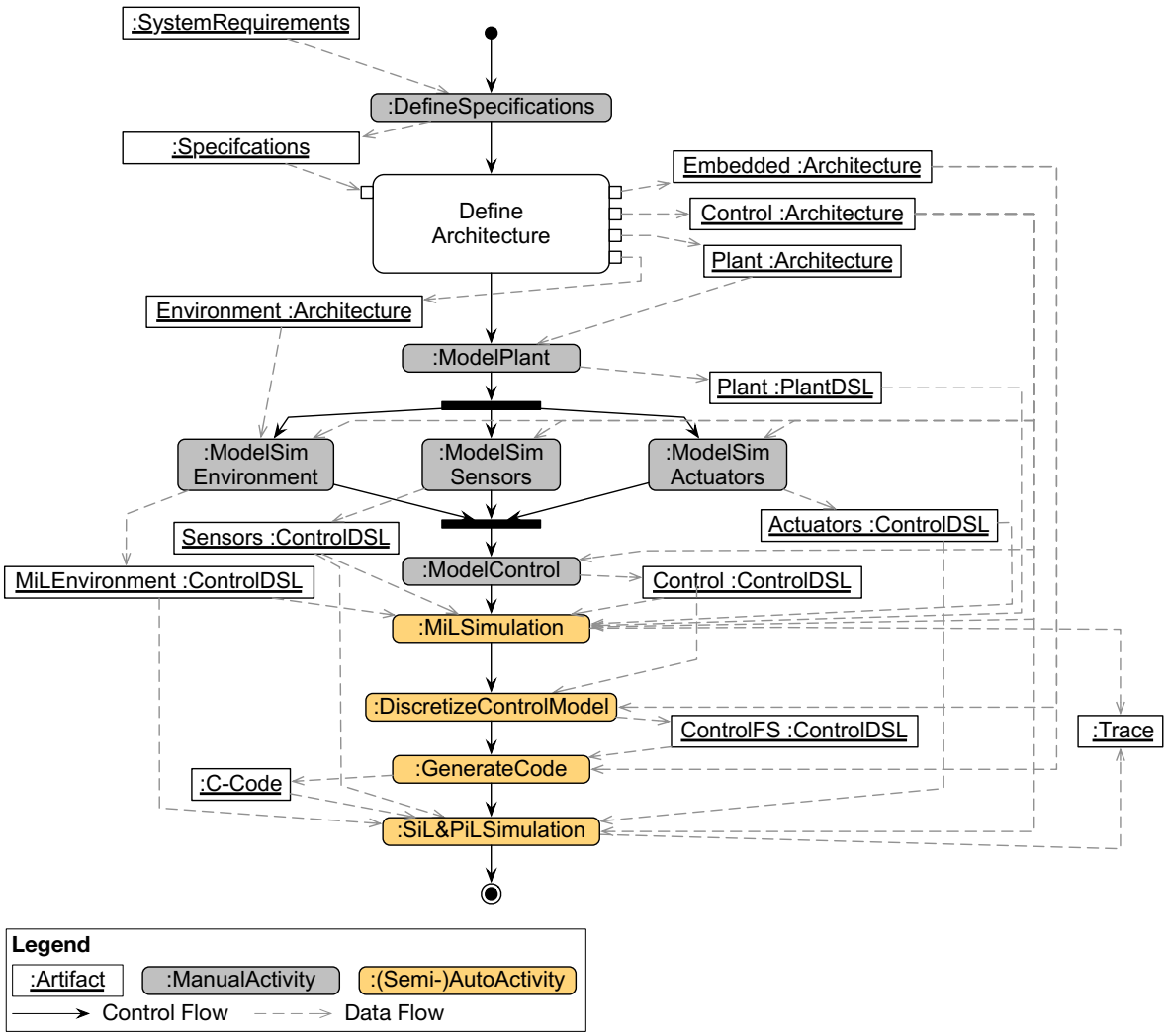


Fig. 9.4: Example of a Process Model (PM) for generating code from a control model

The labelled square edged rectangles represent the data objects (design artifacts) that flow throughout the design process. In a MBSE approach, these objects corresponds to models that are consumed or produced by design activities, represented by the labelled rectangles with rounded corners. For example, to verify the closed loop behaviour of a control model, a model of the physical system (i.e., a plant model) is required such that a Model-in-the-Loop (MiL) simulation can be executed. Note there the process is typed by a control flow and a data flow represented by solid and dashed edges, respectively. Furthermore, a distinction is made between manual and (semi-)automated activities. Finally, the join and fork Activity Diagram flow constructs, represented in Figure 9.4 as horizontal bars, allow one to represent concurrent design activities.

## 9.6 The tool perspective: Formalism Transformation Graph

In each phase of the Process Model, it has to be clear what formalisms and transformations are involved. Corresponding to mega-model principles [19, 31, 142], the Formalism Transformation Graph (FTG) [194, 196, 216] is a language that allows one to declare those formalisms and transformations. It models the relations between the different languages using transformations.

As can be seen in Figure 9.5, domain-specific formalisms are represented as labelled rectangles in an FTG model.

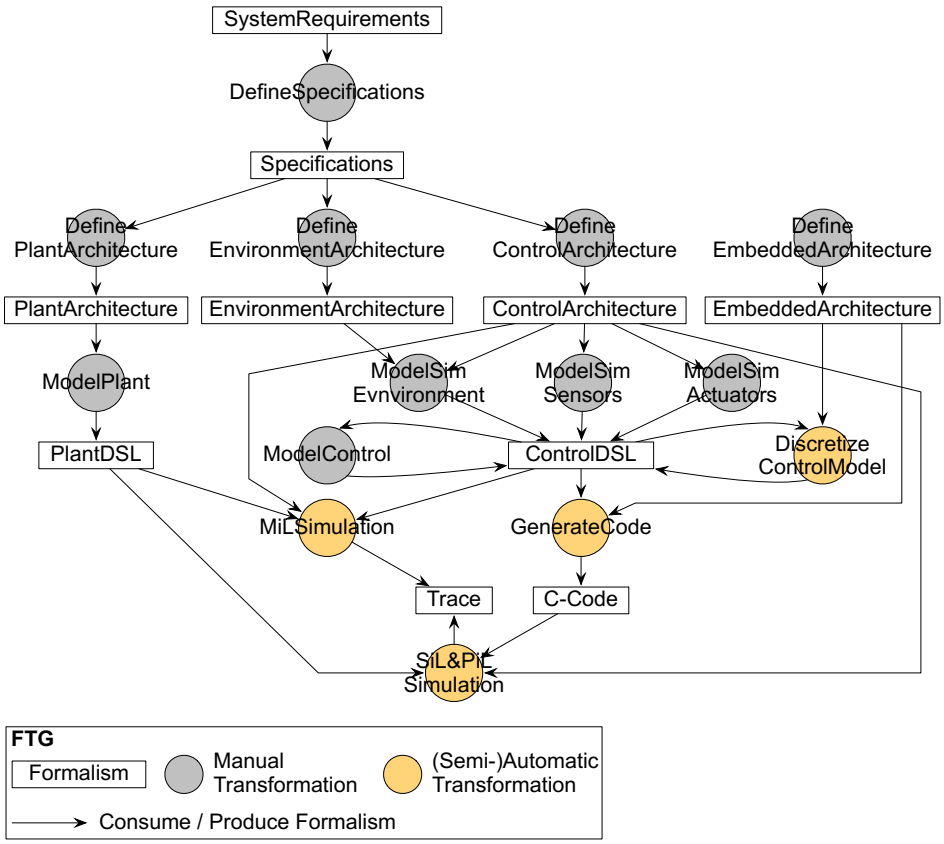


Fig. 9.5: Example of a Formalism Transformation Graph (FTG) for generating code from a control model

Transformations between those formalisms are depicted as labelled circles. Similar to the Process Model, these transformations can be either manual or (semi-) automated. The arrows from formalisms into transformations describe the inputs to the transformations and the arrows from the transformations into formalisms describe the outputs of the transformation. The FTG model is thus a graph describing the modelling languages and the transformations statically available to the engineers of a given domain. In the example of generating code from a control model, one can observe that a MiL Simulation is a (semi-)automated transformation from models expressed in a control, plant, and architectural formalism to a trace model represented in its appropriate formalism.

## 9.7 FTG+PM: Formalism Transformation Graph and Process Model

In the previous sections we have introduced models that enable engineers to describe their design activities, the formalisms that are used, and the transformations that exist between formalisms. One may have noticed that there exists a relation between the Formalism Transformation Graph and the Process Model. Indeed, data objects

(i.e., models) and design activities in the PM are instances of the formalisms and transformations defined in the FTG, respectively. These typing relations are made clear by the colon defined in the rectangles of the PM.

As such, we define the unified metamodel of the Formalism Transformation Graph and Process Model as depicted in Figure 9.6.

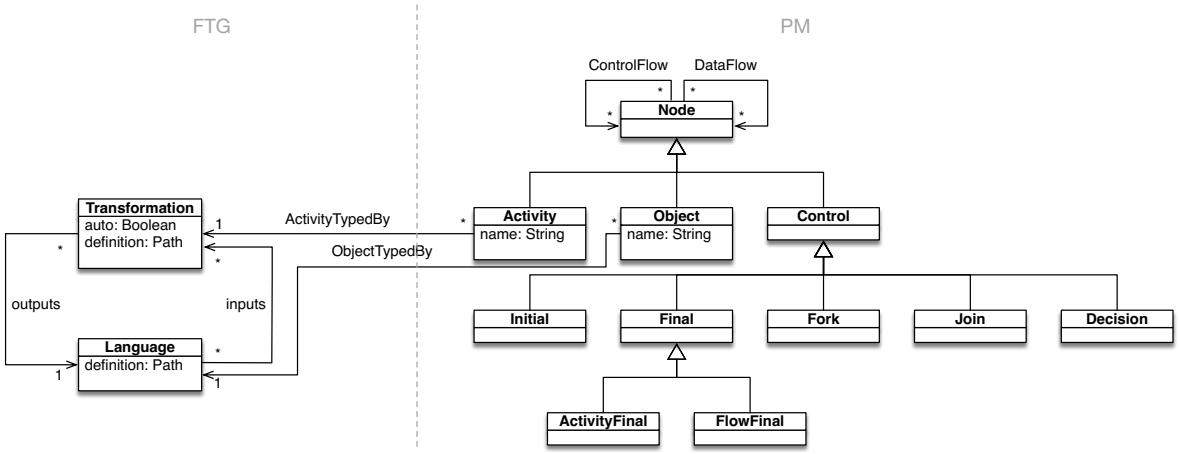


Fig. 9.6: Formalism Transformation Graph and Process Model (FTG+PM) metamodel

On the left-hand side the FTG metamodel can be observed in which the concepts *Language* and *Transformation* are depicted. Both concepts have a *definition* attribute that refers to the definition of the language or transformation. Additionally, an *auto* attribute defines whether the transformation is automatic or not. On the right-hand side the metamodel of the PM language can be seen with its *Activity*, *Object*, and *Control* nodes. Note the typing relation between an *Activity* and an *Object* in the PM, and an FTG *Transformation* and *Language*, respectively.

### 9.7.1 Reasoning about appropriateness of formalisms and heterogeneous modelling

The formalism transformation graph can be used for much more than typing alone. The mega-model describes the relations between different formalisms. As such, we can attach properties to formalisms and transformations. We can query the extended FTG to look for appropriate formalisms that satisfy a set of these properties.

The same applies for giving semantics to languages. The relations between the different formalisms can be used to define a path towards a certain analysis goal. For example, if we are interested in getting behavioral traces, we might use a simulator that creates behavioural traces from the model. However, you could also leverage the properties (and capabilities) of other formalisms and define another path in the FTG to create the state trajectories. For example, if a certain formalism can apply symbolic transformations to create a more run-time performant model, you can leverage these transformation and define a path in the FTG that goes over this formalism. As such the different translational semantics of the formalisms can be reused. This also has drawbacks as it is necessary to keep enough traceability information available to allow the engineer to e.g. debug the models. Figure 9.7 shows the FTG from [292]. Here two different properties are attached to the relations: (a) translational semantics (full line) versus simulation (dashed line) and (b) no approximation (blue) versus allowing approximation e.g. sampling (green). The vertical dashed line in the middle shows the split between the continuous-time domain (left) and the discrete-time domain (right).

The FTG also allows us to reason on combining different formalisms together. Multi-paradigm modelling advocates to use the most appropriate levels of abstraction using the most appropriate formalisms. When applying this principle to the design of complex cyber-physical systems, we have to combine both discrete and continuous-time formalisms. The cyber-part is much more naturally described in the discrete-time domain, while models-of-the-physics are much more naturally described in the continuous-time domain. However, to

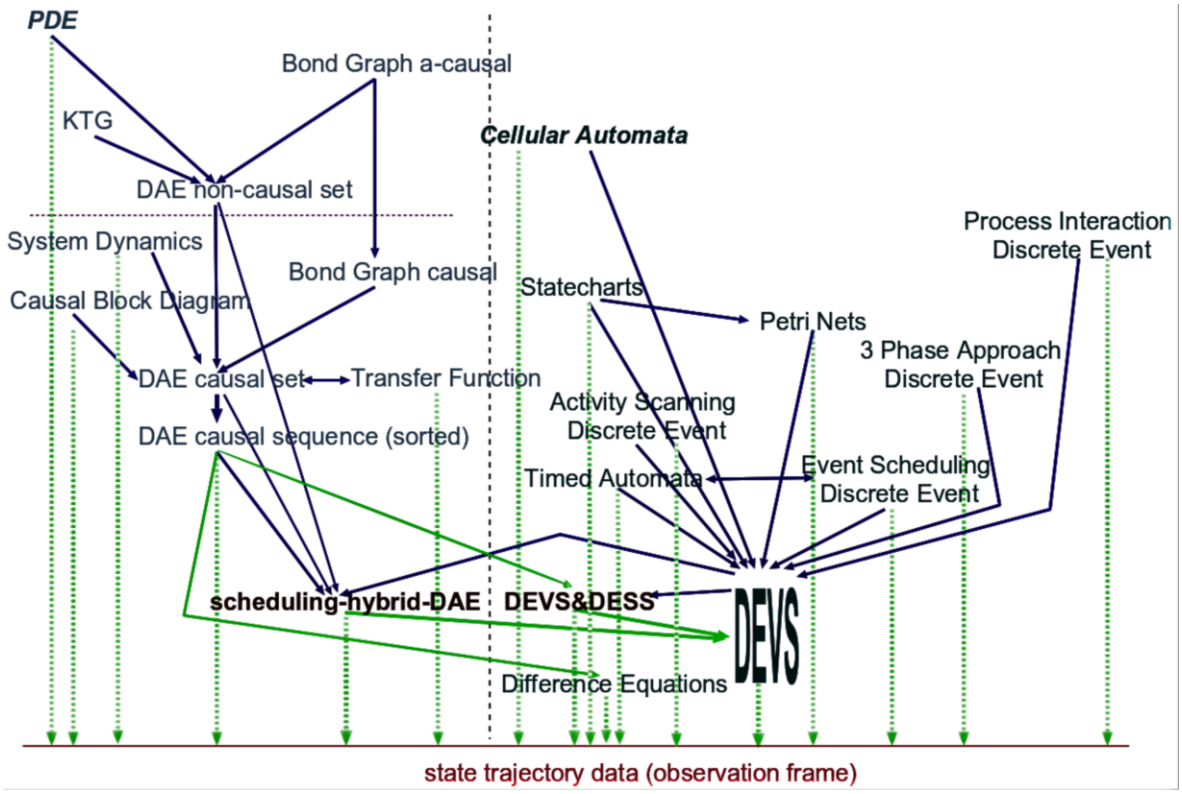


Fig. 9.7: Formalism Transformation Graph from [292]

evaluate the full system behaviour, we have to combine these models. Therefore, we need to reason on what it means to combine formalisms together, and how we can achieve this.

There are three techniques available to simulate the behaviour of hybrid systems.

- Hybrid modelling: A new modelling language is created based on the formalisms and the connections between the formalisms. The modelling language is operationalized with a simulator.
- Co-simulation: Co-simulation reasons on the combination of models by combining the state trajectories. For this, the simulator of the different formalisms is used and orchestrated explicitly.
- Mapping to a common formalism: Finally, the different formalisms and their connection are mapped to a common formalism. For example, In Figure 9.7 you can see that DEVS can be used as a common formalisms to map both discrete-time and continuous-time models to. However, it also shows that there are other possibilities, for example, Bond graphs and Causal-block diagrams can be combined using DAE causal equations.

### 9.7.2 Orchestrating Processes

Orchestration of a process means that the engineers are supported in their design process. This support can take various forms, from showing the engineers a dashboard with various metrics about the process and the steps to follow (e.g. in [193]) to automatically setting up modelling environments, opening the correct models and automatically saving and transforming models when possible. We assume the latter and reason about the orchestration of a process model, more specifically an FTG+PM, in a multi-paradigm modelling environment.

The orchestration of the FTG+PM language requires that the action nodes, denoting a transformation, are properly executed. The order of execution of these nodes is based on the mapping of the activity diagram to coloured Petri nets. The mapping of the activity diagram to the Petri net is described in [168]. When an

action node is encountered the action has to be executed. Depending on the state of the auto attribute in the transformation, the framework has to:

- false: open a modelling environment containing the input model(s) in the specified language(s) and the modelling environment of the output language
- true: automatically execute the transformation with the desired input models

For this end, the tool should be able to capture commands from the engineers, e.g. to indicate when the designer is done with her/his task and the process can continue. Furthermore, the FTG part is very important as it gives the information to the MPM environment to open a certain which modelling environment should be opened to aid the designer. Automating the model transformations that are already defined, results in the consecutive execution of different transformations. A multi-paradigm modelling environment, such as AToMPM [259], has dedicated languages to execute model transformations. We can use these languages to enact the FTG+PM by transforming the process model to such a language. Other modelling environments have similar tools available that can be used to setup such an orchestration, e.g. [240, 145, 131].

## 9.8 Summary

This section discusses the use of Model-based System Engineering (MBSE) in the development life-cycle of Cyber-Physical Systems (CPS). It focuses specifically on the design processes and transformation of paradigm in the Multi-Paradigm Modelling (MPM) of CPSs.

Generally, MPM approaches do not have a standard way of representing processes. A process model for MPM should focus on the languages, model instances and transformations between these models at different levels of abstraction. These levels of abstraction could be at the domain-specific design of the computational or physical components, the verification of the system at a high abstraction level, but also during deployment where different approximations can be used to obtain a better understanding of the parameters involved.

In this regard, a standard representation for MPM processes of CPS is introduced called Formalism Transformation Graph and Process Model (FTG+PM). The automation and tooling is crucial in MPM lifecycle and it is key that the described process can be simulated for analysis and orchestrated to support the designers. To support the designers, the reasoning on the orchestration of a modelled design process is elaborated.

## 9.9 Literature and Further Reading

The interested readers can study the following papers and books for more details. To elaborate on the development/design processes, you can read [246] for software development life cycle models, [95] for the V model which is widely used for MBSE, and the book in [276] for agile development process.

To investigate more on FTG+PM, you can study [194] and [196] for the use of FTG in MDE, as well as the paper in [216] for a case study which uses FTM-PM in automotive. The background studies for FTG+PM can be also found in multiple references [81][41][292][295][291][293][290][178][288][289].

Finally, to read more on the reasoning and orchestration using FTG+PM, you can study tthe work of István et. al [79].

## 9.10 Self-Assessment

1. Create an FTG-PM for a design process or development process of a system of your choice.
2. For the development of the system indicated in exercise (1), discuss which design process is the most appropriate?
3. Formulate the difference between waterfall and agile development processes?
4. Considering your design process in exercise (1), where is it (regarding its maturity) according to the levels of CMM?
5. How you can combine partial differential equations with a a casual bond graphs models?

## Acknowledgements

The authors would like to thank the followings: (1) The COST Action networking mechanisms and support of IC1404 Multi-Paradigm Modeling for Cyber-Physical Systems (MPM4CPS). COST is supported by the EU Framework Programme Horizon 2020. (2) Flanders Make, Belgium. Flanders Make is the strategic research centre for the manufacturing industry in the Flanders area, Belgium.