MDE & MDA in a Multi-Paradigm Modeling Perspective

MDE & MDA in a Multi-Paradigm Modeling Perspective

Ambra Molesini
DISI, Alma Mater Studiorum-Università di Bologna
Viale Risorgimento 2, 40136 Bologna, Italy
ambra.molesini@unibo.it
http://ambramolesini.apice.unibo.it

Enrico Denti
DISI, Alma Mater Studiorum-Università di Bologna
Viale Risorgimento 2, 40136 Bologna, Italy
enrico.denti@unibo.it
http://enricodenti.apice.unibo.it

Andrea Omicini
DISI, Alma Mater Studiorum-Università di Bologna
Via dell'Università,50, 47521 Cesena (FC), Italy
andrea.omicini@unibo.it
http://andreaomicini.apice.unibo.it

ABSTRACT

Since nowadays most complex software systems are intrinsically multi-paradigm, their engineering is a challenging issue. Multi-Paradigm Modeling (MPM) aims at facing the challenge by providing concepts and tools promoting the integration of models, abstractions, technologies, and methods originating from diverse computational paradigms. In this paper we overview the main MPM approaches in the literature, evaluate their strengths and weaknesses, and compare them according to three main criteria—namely, (i) the software development process, (ii) the adoption of meta-model techniques, (iii) the availability of adequate supporting tools. We also explore the adoption of other promising approaches for the engineering of multi-paradigm systems, such as Multi-Agent Systems (MAS) and Systems of Systems (SoS), and discuss the role of Situational Process Engineering (SPE) in the composition of multi-paradigm software processes.

***Keywords**:* Multi-Paradigm Modeling; Development Process; Meta-Modeling; MDE

INTRODUCTION

Complex software systems are nowadays acknowledged to be inherently *multi-paradigm* – that is, built out from with diverse components, differing in terms of the basic abstractions they are based on, the technologies they exploit, the way in which they behave, the autonomy degree, the intelligence, to name just a few. So, their engineering is a challenge, calling for a multiplicity of concepts and tools, as well as the suitable integration of models, abstractions, technologies, and methods originating from diverse computational paradigms. In turn, these also mandate for the adequate integration of heterogeneous software development processes, and the related methodologies (and tools). Since each of the aspects bringing diversity in systems is typically best modelled and developed according to its own specific paradigm (Kuhn, 2012) (Zambonelli & Parunak, 2003), non-trivial software systems end up with being intrinsically multi-paradigm in their very nature.

Along this line, Multi-Paradigm Modeling (MPM) approaches (Vangheluwe, 2002) aim at developing usable, effective, and coherent methods and techniques for the engineering of multi-paradigm systems – both physical systems, software systems, and their combinations.

Model Driven Engineering (MDE) moves precisely from the idea of "*using different models at different levels of abstraction for developing systems*" (Fondement & Silaghi, 2004): in fact, most MPM approaches build precisely on MDE notions, techniques and supporting tools.

To this end, MDE technologies introduce and combine two key ingredients:

- Domain Specific Modeling Languages (DSML) for "*formalizing the application structure, behavior, and requirements within given domains*" (Schmidt, 2006). DSML are often defined by meta-models, with transformations aimed at automatically generating (more specific) models from previous (more abstract) models.
- Model transformations to "*represent the automatic manipulation of a model with a specific intention*" (Syriani, 2011). The rules and mappings to do so are expressed at the meta-model layer, i.e. the layer where the transformation rules conceptually belong.

Because so many MPM approaches have been proposed over the years, finding one's way in such a field is anything but trivial: the differences among the available approaches range over several aspects and dimensions – from the more conceptual (i.e., developing processes, adopted abstractions, using of meta-modeling techniques) to the most practical ones (i.e., availability of effective tools, their learning curve, to name just some). Accordingly, choosing the adequate approach for a given application does not merely amount at listing the features, pros, and cons of the possible choices: what is needed is a key for understanding and choosing advisedly, possibly highlighting a uniform (pre-selected) set of *key aspects*, that enable a comparative analysis on a homogeneous basis. In order to work as an effective "choice helper" tool, the result of such a classification should be available not only in textual (verbose) form, but also in a more effective, concise form of eye-catching tables, for immediate reference at the reader's convenience.

This chapter is therefore organized as follows. First, we summarize the essential background as effectively as possible (Section **Background**), and overview the most relevant MPM approaches in the MDE&MDA perspective (Section **Multi-Paradigm Modeling: Overview**). Then (Section **Comparison and Discussion**) we discuss, compare and classify MPM approaches according to *the three main criteria*:

1. the software development process
2. the adoption of meta-model techniques
3. the availability/unavailability of adequate supporting tools

We also provide highlights on possible alternative views/approaches for engineering multi-paradigm systems beyond MPM (Section **Engineering Multi-Paradigm Systems beyond MPM**) and finally draw some conclusions.

BACKGROUND

This section summarizes the basic concepts and aspects about development processes, methodologies, meta-models, modeling and meta-modeling languages.

The *development process* is likely the single issue in software engineering where the simultaneous exploitation of multiple paradigms impacts the most: so, it is also our main comparison criterion. Subsection **Developing Process** provides some background on the different notions of development process in the literature, and their relationship with the methodology notion.

The next key ingredient for the definition of new processes (and therefore another main classification criterion) is *Meta-models*, discussed in Subsection **Meta-models**. Because these are mostly defined via Unified Modeling Language (UML Home Page, 1997), Subsection **Meta-modeling Languages: UML & MOF** recaps its infrastructure and relationship with Meta Object Facility (MOF Home Page, 2002), the OMG-standard meta-meta-model used for defining the UML meta-model. Finally**,** Subsection **Model-Driven Engineering** presents the model-driven engineering approach.

Development Processes

Defining a development process in a satisfactory way is not trivial, for it involves diverse elements and viewpoints. Different definitions can be found in the literature, focusing on specific aspects or adopting a specific viewpoint. Some major aspects that are typically found in all definitions are *(i) the structural aspect*, focusing on the key elements a development process is made of; *(ii) the organizational aspect*, referred in a broader sense to all the "surrounding" elements—people and people's roles, deliverables, milestones, timing, and schedules, up to (possibly) monetary and marketing issues; and *(iii) the technological aspect*, referring to all the support tools, guidelines, software infrastructures that the development process relies on.

The structural aspect is at the core of the definitions proposed in (Sommerville, 2007), according to which *"A software process is a set of activities that leads to the production of a software product (...). Four general activities can be identified, that are common to all software processes: specification, design and implementation, validation, and evolution."* In (Fuggetta, 2000), the author focuses on organizational and technological aspects: *"A software (development) process can be seen as the coherent set of policies, organizational structures, technologies, procedures, and deliverables needed to conceive, develop, deploy, and maintain a software product."*

Since no software process can be termed as ideal and successfully applied to any application scenarios (Brooks, 1987), different sorts of systems call for different software processes: their activities need to be *"organized in different ways and described at different levels of detail for different types of software"* (Sommerville, 2007). This consideration is at the base both of the identification of *process models* families, and of the definition of the so-called *Situational Method Engineering* (SME) (Brinkkemper, Saeki, & Harmsen, 1999) (Cossentino, Gaglio, Garro, & Seidita, 2007).

Situational Method Engineering (SME) (Brinkkemper, Saeki, & Harmsen, 1999) means to provide the basis for *"the composition of new, ad-hoc development processes for each specific need".* The term "situation" here concerns the specific problem to be faced, the process requirements, the development context, the legacy environment, and other aspects. SME suggests the adoption of adequate building tools based on the reuse of *method fragments* (Cossentino, Gaglio, Garro, & Seidita, 2007) – that is, portions of (both existing and created ex novo) development process, taken from a specific repository.

Despite their strict relationship, software engineering methodologies and development processes are not the same: as sketched in (Cernuzzi, Cossentino, & Zambonelli, 2005), "*Methodologies focus more explicitly on how an activity or task should be performed in specific stages of the process, [while] processes may also cover more general management aspects concerning who, when, how much, etc*.". Although both aspects are obviously an engineer's concern, development processes focus on *phases*, *relationships among phases*, timing, roles, etc., while methodologies cope more specifically with the *specific techniques* adopted and *work products*. In this sense, development processes and methodologies can be seen as complementary—the development process being provided with suitable methodological guidelines that specify the tools to be used, the techniques to be adopted, the definition of the work products to be produced, best practices, etc.

Meta-models

Multi-Paradigm Modeling (MPM) means to provide software designers with *"the most appropriate modeling abstractions for the particular problem domain, (automatically) transforming the resulting models into solution abstractions of the selected implementation platform"* (Vangheluwe, 2002). In this context, *meta-models* aim to formalize the modeling abstractions, their admissible relationships, and the mapping between abstractions belonging to different conceptual levels. Meta-models capture the rules that connect (i) the modeling abstractions, possibly from different conceptual layers, to each other, and (ii) the activities and roles that compose a development process or methodology.

Several definitions exist in the literature, e.g. (Bernon, Cossentino, Gleizes, Turci, & Zambonelli, 2004) (Gonzalez-Perez, McBride, & Henderson-Sellers, 2005): as the term suggests, they all share the idea that a meta-model is "*a model of a model*" (MDA Home Page, 2001). As highlighted in (Henderson-Sellers & Gonzalez-Perez, 2005), formalizing a methodology with a meta-model is useful for several reasons – consistency check, planning extensions, etc. So, in order to be fruitful, meta-models should deal with all the multi-facet aspects of a methodology – process model, life cycle, activities, techniques, guidelines, up to the organization in the development team. In short, meta-models are *an essential tool from a 360-degree perspective* to study the completeness and expressiveness of a methodology, compare different methodologies (Henderson-Sellers & Gonzalez-Perez, 2005) and integrate methodologies (Cossentino, Gaglio, Garro, & Seidita, 2007).

Meta-modeling Languages: UML & MOF

UML is "*a general-purpose modeling language for describing artefacts in some domain of interest"* (Atkinson & Kuhne, 2001), purposely designed to be useable at multiple levels in a multi-layer architecture—in particular to describe its own meta-model, providing for compactness, conceptual economy, easier support for CASE (Computer-Aided Software Engineering) tools.

According to (Atkinson & Kuhne, 2001), four levels – normally referred to as *M0*, *M1*, *M2*, and *M3*, bottom-up – are adequate and enough for tool interoperability. The top layer, M3, provides the constructs for creating meta-models—that is, the meta-meta-models: a suitable example is MOF (Meta-Object Facility) (MOF Home Page, 2002). M2 is an instance of M3, which defines the modeling language to be used in M1: this is where the UML meta-model (an instance of MOF) is actually defined. M1 is an instance of M2 which defines the languages for describing the application domains: every user model is an instance of the UML meta-model. Finally, M0 expresses the actual run-time entities—instances of model elements defined in M1.

A crucial aspect in such architecture is the one-to-one relationships among UML and MOF: *"every model element of UML is an instance of exactly one model element in MOF"* (MOF Home Page, 2002). So, MOF defines how UML models can be exchanged between tools: this is due to the reuse of the core modeling concepts (belonging to the Core Package) between UML, MOF and other OMG meta-models. Such "meta-models' concepts sharing" is at the base of *Model Driven Architecture* (MDA).

Finally, *profiles* are UML's way to customize UML itself. The profile mechanism provides for adapting an existing meta-model (typically the UML meta-model) to a particular domain, thus enabling the definition of Domain-Specific Modeling Languages (DSML).

Model-Driven Engineering

Model-Driven Engineering (MDE) promotes the development of software systems by focusing on the creation of basic models and then transforming such models, across multiple levels of abstraction down to code generation (Rhazali, El Hachimi, Chana, Lahmer, & Rhattoy, 2019).

Each MDE process defines (i) the models to be developed, (ii) in which order they should be developed, (iii) how a higher-level model is to be transformed into a lower-level one. The system under development is then first described by a model at the highest abstraction level (*Computation Independent Model* – CIM), and then iteratively transformed into a platform-specific model via subsequent refinements through different abstraction levels (Rhazali, Yassine, Hadi, & Mouloudi, 2016). Accordingly, the CIM level means to represent only the business process reality, while the next level – *Platform Independent Model* (PIM) – describes models in a way that is effective for analysts and designers. The lowest abstraction level is the *Platform Specific Model* – PSM – made basically of code models (Rhazali, El Hachimi, Chana, Lahmer, & Rhattoy, 2019).

Model Driven Architecture (MDA) (MDA Home Page) is OMG's vision of MDE, which relies on UML and MOF to define the structure, semantics, and notation of models.

The major goal of MDA is to develop sustainable models (both CIM and PIM models) to enable the automatic generation of all the application code, achieving a significant productivity gain (Rhazali, Yassine, Hadi, & Mouloudi, 2016). Despite being rooted on model transformations, MDA per se does not propose any methodological transformation process (Rhazali, Yassine, Hadi, & Mouloudi, 2016). Research efforts typically focus on the transformation from PIM to PSM, which are closely linked: instead, transforming the CIM level to the PIM level is rarely discussed in the literature, possibly because of their dissimilarity. A notable work about this crucial aspect is (Rhazali, Yassine, Hadi, & Mouloudi, 2016), where authors propose a methodology to master transformation from service-oriented CIM level to web-based PIM level. More recently (Rhazali, El Hachimi, Chana, Lahmer, & Rhattoy, 2019), a methodology has been proposed for the semi-automatic model transformation from CIM to PIM up to PSM.

MULTI-PARADIGM MODELING: OVERVIEW

To get a sense of Multi-Paradigm Modeling (MPM), let us overview the most relevant works, focusing on the development process, meta-model definitions, and the availability of adequate documentation and supporting tools. The same criteria are reused in Section **Comparison and Discussion** for comparative analysis and in-depth discussion.

The basics

MPM as a stand-alone discipline dates back more than a decade. In the first workshop (Giese & Levendovszky, 2006), many committed with the definition proposed by Vangheluwe et al. since 2002 (Vangheluwe, 2002), which defined MPM as "*the integration of model abstraction, multi-formalism modeling and meta-modeling*".

Moving from such definition, (Mosterman & Vangheluwe, 2002), (Mosterman & Vangheluwe, 2004) and (Denil, Vangheluwe, De Meulenaere, & Demeyer, 2012) introduce MPM approaches that explore and combine three aspects:

- *model abstraction*, concerned with *"the relationship between different models at different levels of abstraction";*
- *multi-formalism modeling*, concerned with *"the coupling of and transformation between models, described in different formalisms"*;
- *meta-modeling*, concerned with *"the description (models of models) of classes of models—that is, the specification of formalisms"*.

MPM investigates their blending, by (i) combining, transforming, and relating formalisms one to each other; (ii) generating domain/problem-specific formalisms, methods, and tools; (iii) cross-checking the coherence among different views.

The main reason beyond this idea is that problem-specific formalisms and tools are desirable, yet their development is difficult and time-consuming; moreover, adopting a multi-formalism approach calls for "*interconnecting a plethora of different tools, each designed for a particular formalism*" (Lara & Vangheluwe, 2002). This is precisely where meta-modeling comes to help: while the introduction of a meta-layer enables DSML definition, the meta-layer information enables the generation of specialized tools for supporting newly-defined DSML. Together, they can cut the development cost of a customized tool, making the difference for actual DSML adoption. Moreover, since the generated tools represent the models through a common data structure, transformations between DSML become transformations between these structures.

Indeed, recent work by (Ciccozzi, Vangheluwe, & Weyns, 2019) investigate *blended modeling*, that is, *"the activity of interacting seamlessly with a single model (i.e., abstract syntax) through multiple notations (i.e., concrete syntaxes), allowing a certain degree of temporary inconsistencies."* In short, blended modeling aims to provide blended editing and multiple visualizing notations to interact with a set of concepts, usable for the different aspects of design, development and stakeholder communication in an MDE process.

Multi-Paradigm Modeling with AToM3

To prove the effectiveness of the above ideas, Vangheluwe et al. developed the AToM3 tool: its input is an Entity-Relationship meta-specification extended with suitable constraints, whereas its output is a new, model-specific tool customized for processing the models described in a given DSML. The meta-model drives the automatic generation both of the DSML and of its relative supporting tool(s). AToM3 uses Abstract Syntax Graphs (Oliveira & Loh, 2013) to express models and graph grammar models (Ehrig, Engels, Kreowski, & Rozenberg, 1999) to express

transformations, which take the form of graph rewriting. The meta-model of the DSML is in its turn represented as a *meta-metamodel*, like in the UML architecture: accordingly, new concepts are actually introduced at the meta-meta-level.

Although no specific description is provided about the process to be followed when developing an MPM application, it is apparently left as understood that engineers should first specify the meta-models of the different formalisms to be used: AToM3 takes then care of generating the proper tools for each formalism. Once the tools are available, engineers can model the different pieces of the system adopting the most suitable formalism for each piece. The models (possibly represented in different formalisms) are then connected via *generic links* – AToM3 way of enabling the connection between the different parts. As a final step, AToM3 can automatically generate the source code according to user-defined transformations.

A DSM-based MPM Approach for Simulation

Domain-Specific Modeling (DSM) is one of the basic MDE techniques. (Li, Lei, Wang, Wang, & Zhu, 2013) presents a DSM-based, MPM approach which exploits MDE for integrating Models & Simulations (M&S) paradigms (Balci, 2012): the goal is to build a simulation framework combining multiple M&S languages to describe the different domain behaviors.

This approach exploits several modeling methods—namely, (i) MDE models for the work products, with transformations and code generation for the development process; (ii) DSM to provide domain-specific solutions; and (iii) M&S formalisms to express the domain behaviors.

The simulation model development is structured in four top-down steps, which correspond to the four M&S layers:

1) the *DSML layer*, addressing the conceptual modeling;
2) the *M&S formalism layer*, devoted to model architecting and integration;
3) the *model framework & specification layer*, concerned with the model design and formal analysis, based on the M&S formalism;
4) the *executable model layer*, addressing the model implementation, based on SMP2/C++ transformations.

Authors (Li, Lei, Wang, Wang, & Zhu, 2013) move from the idea of decomposing a (complex) system into "a series of problem domains, defining domain-specific conceptual models for each domain". At the highest (DSML) layer, each sub-system is modeled via a DSML, whose meta-model (i.e., grammar) is designed by the joint effort of domain experts, language engineers and M&S experts. The next (M&S formalism) layer exploits model specification and simulator algorithms to model the behavior and provide the denotational semantics for each DSML. Since a single formalism supports DSM for multiple domains, formal analysis methods can be applied to perform an overall analysis. The third layer (model framework & specification) is the architecture of the simulation system, aimed at providing "a structural foundation for the integration of simulation models of sub-systems across different domains".

Aiming at being technology-independent, DSM is described in terms of high-level concepts, abstracting from the general structure of the specific system. The model framework obtained embodies the architecture from the simulation viewpoint: it is a first-class artifact providing the foundations – intended as the structural elements and their mutual relationships – for each domain. The model framework is then further combined with the behavior modeling, depicting the behavior patterns of each domain. The resulting model framework is finally mapped onto Simulation Model Portability 2 (SMP2), a standard MDA model specification, to provide a

platform-independent model.

The bottom (executable model) layer couples the structural aspects described in SMP2 with the behavioral code generated from the domain-specific models: such parts are finally integrated in the SMP2 framework for the generation of executable components (technically, this step is based on an SMP2/C++ mapping, performed via a custom C++ code generator in Eclipse).

MPM in Model Transformation

(Syriani, 2011) proposes an MPM technique for the engineering of a *model transformation language* – the language expressing model transformations – in order to automate the model transformation itself. The approach, rooted in (Mosterman & Vangheluwe, 2004), adopts the MPM principles, in particular, *multi-abstraction* – which is *not* the same as model abstraction. Quoting the author, *"Model abstraction is a view of a system exhibiting some of its properties and hiding others, [while] multi-abstraction is the ability to express models at different abstraction levels".*

The model transformation language is defined at the syntax level, while the semantics is modeled through meta-models and model transformations—the necessary enabling technologies. The aim is twofold: on the one side, to raise the abstraction level; on the other, to improve the mapping between model transformation languages and their corresponding domains, minimizing accidental complexity. The effectiveness of the resulting framework is illustrated by Syriani in the design and implementation of a model transformation language. However, emphasis is on expressiveness, rather than on the development process—in fact, no tools are actually provided.

MPM in FTG+PM

In (Mustafiz, Denil, Lucio, & Vangheluwe, 2012) an MPM framework is proposed, called *Formalism Transformation Graph + Process Model* (FTG+PM), which is also based on MDE and adopting the MPM definition in (Mosterman & Vangheluwe, 2002). As its name suggests, the first component is a graph, whose nodes are languages and edges are transformations, while the second component models the software lifecycle and its transformations activities via UML 2.0 activity diagrams. Different modeling languages can be used at different abstraction levels: authors adopt UML modeling languages, domain-specific modeling languages, natural languages, and general-purpose languages, mainly meta-modeled via UML class diagrams.

Like any MDE process, FTG+PM includes several activities – from requirements to code synthesis – modeled at different abstraction levels, which depend on the different MDE phases: higher-level models (requirements models, domain-specific models) are transformed step-by-step into source code. Each abstraction layer is associated to the detailed tasks to be performed: quoting the authors, they are "first declared as transformation definitions in FTG, then instantiated as PM activities". Transformations represent the development glue: input models are turned onto one output model, all conforming to the meta-model.

The FTG+PM approach is supported by AToMPM (*A Tool for Multi-Paradigm Modeling*), which covers both the definition of meta-models and transformations, and the execution of the transformation chain.

An Orientation Framework for Multi-Paradigm Modeling

Moving from the idea that each methodology is built upon some characteristic assumptions – its *paradigm* –, (Lorenz & Jost, 2006) observe that an effective modeling approach calls for best match of three key aspects:

- *purpose*, defined as "the motivation of the intended modeling effort"

- *object*, defined as “the real-world context under investigation”
- *methodology*, defined as “a comprehensive, integrated series of techniques or methods creating a general systems theory of how a class of thought intensive work ought to be performed”.

Since it is often the case that no single set of modeling paradigm and methodology fits satisfactorily a given context and purpose, the intriguing idea is to try to combine different methodologies, in an MPM perspective, taking the “best-fit” methods from different methodologies. The assumption is that such an approach should match the reality more closely: at the same time, effectively combining different paradigms is an issue. A possible way to do so is to select the “best-fit” paradigm for each sub-problem and then build the multi-paradigm model in terms of interacting modules: however, choosing the best-fit paradigm/method for each part of the system is critical. While authors present some hints to face this point, little is actually said about the development process, the supporting tools, and the meta-model techniques.

MPM in software architectures

In (Balasubramanian, Levendovszky, Dubey, & Karsai, 2014) authors discuss the multi-paradigm issues encountered during the development of a domain-specific architecture description language for distributed embedded systems.

Their software infrastructure, DREMS (Distributed Real-time Embedded Managed Systems) is aimed at “*designing, implementing, configuring, deploying, and managing distributed read-time embedded systems*”. The two major subsystems respectively deal with (a) modeling, analyzing, synthesizing, implementing, debugging, testing, and maintaining the software application, and (b) deploying and managing the application on a network.

Three main challenges are faced: (i) how to integrate the textual code inside the graphical modeling language; (ii) how to transform the high-level scheduling properties into the target platform schedule; and (iii) how to integrate different design-time analyses.

The first challenge derives from the use of a graphical UI on the user side and of a textual language in the underlying platform, which adopts the Component-Based Software Engineering (CBSE) approach: there, applications are made of reusable software components whose interfaces are specified via the Interface Definition Language (IDL)—an OMG standard for this purpose; code generators process IDL to generate stubs, which are then merged with the user logic to build the final component. In DREMS, the integration of the textual IDL language is performed by creating an add-on to the modeling language that parses the textual language for setting attributes onto the graphical elements.

The second challenge comes from the use of the DREMS temporal partition scheduler, which transforms the high-level scheduling constraints and artefacts (the process-partition assignment graph, and other details) into a detailed schedule usable by the underlying operating system. A schedule calculator then binds the individual temporal partitions to the DREMS schedule.

The third challenge originates from the need to integrate three automated analyses—namely, the security of communications, the network quality of service (QoS), and the software component schedulability. While security is granted via a multi-level security policy, the analysis of QoS requirements is based on the support for QoS profiles in the modeling language, which is capable to describe the evolution of the network parameters and the QoS requirements. Schedulability analysis involves the verification of system properties, and is performed by generating a suitable Colored Petri Net (Jensen & Kristensen, 2009).

The proposed solution goes beyond their specific case, and can be applied to similar

situations, especially if aimed at providing an analogous level of tool automation. Because of the focus on domain-specific languages, however, other aspects – the development process, the supporting tools, the meta-model techniques – are left aside.

Multi-Paradigm Design with Feature Modeling

(Vranic, 2005) introduces *Multi-Paradigm Design with Feature Modeling* (MPDfm), aimed at enabling explicit reasoning about "paradigms viewed as solution domain concepts", and at evaluating their "appropriateness for application domain concepts". A solution domain represents the language for describing a solution—typically, a programming language.

Application and solution domain are both modeled via *feature modeling* (Vranic, 2005), a modeling technique for multi-paradigm design. Application domain and feature models are subject to transformational analysis to define a suitable mapping between these domains. The mapping is expressed in the form of yet-another feature model, with the information about the application domain concepts and features (Vranic, 2005); overall, these determine the basic code structure. So, the development process is structured in four main steps: (i) application domain feature modeling; (ii) solution domain feature modeling; (iii) transformational analysis; and (iv) code skeleton design. The first two can proceed in parallel: then, the detailed design and implementation can follow the MPDfm approach.

Authors report that the method is verified, with good results, adopting feature modeling as the application domain and the AspectJ language as the solution domain; yet, no specific supporting tools seem to be available.

Model-driven System Engineering for Virtual Product Design

(Dalibor, Jansen, Rumpe, Wachtmeister, & Wortmann, 2019) present a small Model-Driven System Engineering (MDSE) methodology to integrate the SysML paradigm with the CAD paradigm, that is, a method for integrating abstract system descriptions in the OMG Systems Modeling Language – SysML (SysML Open Source Project Home Page, 2003) with Computer-Aided Design (CAD) models. SysML is an UML subset extended with a graphical modeling language for systems engineering, while the Computer-Aided x (CAx) paradigm embraces various Computer-Aided methods such as CAD, Computer-Aided Engineering (CAE) and Computer-Aided Manufacturing (CAM).

The context is virtual product development, i.e. "a practice to support all phases of the product development process using a digital environment". CAD modeling methods are applied in all the development-relevant phases of the product life-cycle, to simulate, verify, validate, and manufacture the product, while minimizing the creation of physical prototypes. Their combination for virtual product development leads to the so-called "CAx process chain".

Authors introduce the idea of connecting such a chain with the SysML models (created with the Object-Oriented Systems Engineering Method-OOSEM). Process systems engineers would then be able to describe the system in SysML (via an ad-hoc profile), and integrate the model into the CAx process chain. The process starts with a SysML modeling activity, where the systems engineer creates an abstract system model specifying the elements of the physical parts of the system. The next step is to forward the parameters that are relevant for CAD modeling. Using the resulting CAD model in combination with additional information from the SysML diagram, engineers can perform additional CAE analyses and CAM modeling.

A suitable SysML Profile enables engineers to mark blocks of a SysML Block Definition Diagram as «CAD Element», and value properties that should serve as parametric design parameters as «CAD Parameter». Moreover, model information exchange in the SysML-CAD

process chain is automated, via a special-purpose plug-in for the MagicDraw modeling environment (MagicDraw Home Page, 2019) that interchanges model information in a format suitable for Autodesk Inventor: so, the dimensions of concrete physical parts can be designed automatically.

Despite being very special-purpose, the approach seems quite appealing: however, it is strictly tied to the example proposed in the paper, so its application to other product seems not so trivial, especially due to the work necessary for adapting the automatic the exchange of model information.

A Multi-paradigm Modeling Framework for Modeling and Simulating Problem Situations

(Lynch, Padilla, Diallo, Sokolowski, & Banks, 2014) propose a Multi-Paradigm Modeling Framework (MPMF) for modeling and simulating problem situations. The framework adopts three different levels of granularity (*macro*, *meso*, and *micro*) to analyze what is known and assumed about the problem situation. Such levels are then independently mapped to different modeling paradigms, which are combined to build up the comprehensive model and the corresponding simulation.

MPMF is based on Modeling and Simulation-System Development Framework (MS-SDF) (Diallo, Andreas Tolk, Gore, & Padilla, 2005), which defines a high-level approach on how to derive a model from a problem situation, and a simulation from the model. MS-SDF provides an effective approach to capture the information about the problem situation and represent it in a manner which is suitable for simulation, as well as to identify contradictions and inconsistencies.

In MS-DSF, three high-level constructs are introduced – *reference modeling, conceptual modeling,* and *simulation building* – that MPMF expands to support simulation *implementation*. The good level of *traceability* – another key MS-SDF feature – is also maintained in MPMF, thanks to the recursive layout of the framework in the modeling process.

In a multi-paradigm environment, the number of modeling paradigms to be handled represents a critical aspect to keep complexity manageable: accordingly, the number of modeling paradigms used to implement the simulation should be kept to the minimum.

In our knowledge, no tool or documentation are currently available for this framework.

Multi-Paradigm Modeling approach to live modeling

A different, more recent perspective to MPM, based on the above-discussed FTG+PM (Mustafiz, Denil, Lucio, & Vangheluwe, 2012), can be found in (Van Tendeloo, Van Mierlo, & Vangheluwe, 2019), aimed at adding *liveness* to modeling languages in a way that is reusable across multiple formalisms: the reason is that the support for live modeling has been identified as a key feature to advance the usability of model-driven techniques.

The basic idea is to transpose the essence of live programming to the modeling domain, in a generic way. Live programming concepts and techniques should be transposed to domain-specific executable modeling languages, clearly distinguishing between generic and language-specific concepts. However, domain-specific modeling means to handle (many) different domain-specific formalisms, each possibly with a handful of users, making it difficult to justify the investment for implementing live modeling techniques in an ad hoc way. So, to effectively support live modeling in the context of domain-specific formalisms, authors “deconstruct and reconstruct” the traditional live programming process. All the activities related to liveness are distilled into a single operation (“sanitization”): so, to make a formalism live, only sanitization needs to be updated, while the other aspects of live modeling can be reused.

Author considered three types of executable modeling formalisms: Finite State Automata (FSAs), Discrete Time Causal Block Diagrams (DTCBDs), and Continuous Time Causal Block Diagrams (CTCBDs). They are all supported, together with their live implementation, in the Modelverse tool (Van Tendeloo, Van Mierlo, & Vangheluwe, 2018), which supports all the necessary phases – language engineering, model transformations, process enactment – as well as the use of multiple interfaces. There, users first start the live modeling process which pertains to the desired formalisms, possibly providing an initial model. It is worth highlighting that only the design models are stored: simulation, instead, is always started anew. The enactment closely resembles the usual modeling interface, except for the presence of an extra simulation window – merely, an external program that visualizes the simulation results. In order to emphasize the uncoupling between model and formalisms, authors discuss three examples based on the exact same (parameterized) FTG+PM model, showing that only the sanitization operation is to be redefined for each formalism individually: the visual interfaces are untouched, as everything is based on process enactment. So, because of the independence between model and domain-specific formalisms, and thanks to the availability of an effective tool based on FTG+PM, this approach seems potentially applicable to a wide variety of modeling formalisms.

At the same time, Modelverse is a new tool, mainly based on the writing of Python code to create models via meta-model instantiation, (Van Tendeloo, Van Mierlo, & Vangheluwe, 2018), currently featuring no visual diagram support—a drawback which could limit actual usability, especially in more complex scenarios.

COMPARISON AND DISCUSSION

This section aims to discuss comparatively the approaches introduced in Section 3 according to three main criteria:

(1) the development process

(2) the possible adoption of meta-models

(3) the availability of suitable supporting tools

The very reason beyond the first criterion is well synthesized in (Sommerville, 2007): "*Different types of systems need different development processes (...) [so] the use of an inappropriate software process may reduce the quality or the usefulness of the software product to be developed and/or increased*". This is particularly relevant for multi-paradigm systems, where the development process needs to adapt to diverse paradigms and application domains, in order to model the different parts of the system with a variety of abstractions.

The second criterion derives from meta-models being the conceptual and practical tool to bridge between (abstractions from) different paradigms: in fact, MPM approaches exploit meta-modeling techniques to define Domain-Specific Modeling Languages (DSMLs), which are then used for modeling specific sub-systems; in turn, these are further processed via (possibly automatic) transformations to map a DSML onto another.

The third criterion comes from the finding that the actual applicability of the multi-paradigm approach depends on the availability of effective supporting tools—both because of the inherent complexity and multi-facet nature of the development process, and specifically for the definition of the DSML and the related processing. At the same time, the actual usability of such tools is often critical also for experienced users, which makes it a crucial factor to be accounted for.

Development Process Comparison

The analysis of the development process needs to start from the analysis of the process *model*.

Appropriate comparison criteria for this aspect can be derived from Sommerville's definition:

(i) the kind (and structure) of process model
(ii) whether the process is adequately documented (other than, indirectly, reasonably easy to learn and apply).

As far as the first issue is concerned, one clear result is that, whenever a process model can be identified, it is a *transformational* process. This is not surprising, since a number of such approaches share the same root (Vangheluwe, 2002).

The second, often underestimated, issue is particularly critical in the MPM context since the development of adequate documentation does not follow the physiological evolution of the process itself, which is inherently complex because of the inter-twining of different paradigms. Unfortunately, documentation and tutorials are often outdated and do not reflect the new process requirements or design techniques. As a further consequence, the lack of suitable documentation negatively impacts on the learning process.

Table 1 summarizes the results of the comparison among the above-discussed approaches. Since the works by Vangheluwe, on which many approaches are based, refer to MDE principles and focus onto the automatic transformations among modeling domains/languages, the adoption of supporting tools like AToM3, AToMPM and Modelverse is an important factor. In many cases the presence of CASE tools *influences* the development process itself—indeed, the adoption of a given tool nearly imposes the process to be followed in the system development (Denil, Vangheluwe, De Meulenaere, & Demeyer, 2012), (Mosterman & Vangheluwe, 2002) (Mosterman & Vangheluwe, 2004) (Lara & Vangheluwe, 2002) (Mustafiz, Denil, Lucio, & Vangheluwe, 2012) (Van Tendeloo, Van Mierlo, & Vangheluwe, 2019). Alternatively, works inspired to the Features-Oriented software development, like (Vranic, 2005), adopt a transformational process model but blended with Features-Oriented ingredients.

The inadequate availability of up-to-date and complete documentation makes it often difficult not only to compare different development processes, but also to choose the most suited for the specific designer's situation. Of course, this problem is not peculiar only to the MPM community, but affects other communities, like e.g. the Agent-Oriented Software Engineering (AOSE) community, which also needs a standard way of documenting the development processes. This is why the IEEE-FIPA standardization organism supported in the last years the development of a standard Documentation Template (Cossentino, Seidita, Hilaire, & Molesini, 2014), based on a variant of SPEM 2.0 (Software & Systems Process Engineering Metamodel Specification. Version 2.0, 2008), which seems general enough to be potentially adopted also in domains other than AOSE—like, for instance, MPM.

As shown in Table 1, the only proposal adopting a formal approach – namely, SPEM – to deal with the documentation issue is (Denil, Vangheluwe, De Meulenaere, & Demeyer, 2012): however, it adopts the (older) SPEM 1.0, and only for modeling a single fragment of the development process. Other works opt for informal process documentation or examples to document their approaches: (Li, Lei, Wang, Wang, & Zhu, 2013) and (Mustafiz, Denil, Lucio, & Vangheluwe, 2012) are perhaps the ones providing the most complete documentation—with a positive impact on the learning process. Other approaches rely mostly on the supporting tools and related documentation for illustrating the development process, possibly suffering from the obsolescence problem of such documentation. In most cases, tools do not seem to be particularly suited for an easy learning, especially by non-expert designers.

Table 1. Comparison of the development processes.

Proposal *Authors and Bibliography reference*	**Process Model** *Type and Influence*	**Documentation**	
		Formality	*Examples*
(Denil, Vangheluwe, De Meulenaere, & Demeyer, 2012)	Type: transformational Influence: AToM3	*Formal (via SPEM 1.0)*	*Not provided*
(Mosterman & Vangheluwe, 2002)	Type: transformational Influence: CASE tool	*Not provided*	*Not provided*
(Mosterman & Vangheluwe, 2004)	Type: transformational Influence: CASE tool	*Not provided*	✓
(Vangheluwe, 2002)	Type: transformational Influence: –	*Informal*	*Not provided*
(Lara & Vangheluwe, 2002)	Type: transformational Influence: AtoM3	*Not provided*	✓
(Li, Lei, Wang, Wang, & Zhu, 2013)	Type: transformational Influence: N/A	*Not provided*	✓
(Syriani, 2011)	Type: not specified Influence: –	*Not provided*	✓
(Mustafiz, Denil, Lucio, & Vangheluwe, 2012)	Type: transformational Influence: AToMPM	*Informal*	✓
(Lorenz & Jost, 2006)	Type: not specified Influence: –	*Not provided*	✓
(Balasubramanian, Levendovszky, Dubey, & Karsai, 2014)	Type: not specified Influence: –	*Informal*	*Not provided*
(Vranic, 2005)	Type: transformational Influence: features design	*Informal*	✓
(Dalibor, Jansen, Rumpe, Wachtmeister, & Wortmann, 2019)	Type: transformational Influence: MDE	*Not provided*	✓
(Lynch, Padilla, Diallo, Sokolowski, & Banks, 2014)	Type: not specified Influence: MS-SDF	*Not provided*	✓
(Van Tendeloo, Van Mierlo, & Vangheluwe, 2019)	Type: iterative Influence: FTG+PM and Modelverse	*Informal*	✓

Meta-model Comparison

As meta-models are one of the most powerful conceptual tools to bridge between different paradigms, one obvious comparison criterion is whether they are adopted in a given MPM approach: if so, two further aspects concern which language(s) is used for their utterance and for

which purpose(s) they are used. While most MPM approaches are actually rooted in (Mosterman & Vangheluwe, 2002), where meta-models are at the core, other approaches, like (Lorenz & Jost, 2006) and (Vranic, 2005), are based on different techniques—e.g., *features*.

Meta-models can then be seen as a sort of grammar specifying the syntax of a set of models, while model transformations specify the corresponding semantics: their combined use provides for the automatic generation of the modeling environment. So, meta-models both bridge between paradigms and set the practical foundation for automating the building of Domain Specific Modeling environments.

Table 2 reports the result of the comparison of the above approaches based on the meta-modeling lens. When meta-modeling techniques are used, the most common representation language appears to be UML, possibly combined with Entity-Relationship (E-R): this is not surprising, since UML is the most used meta-modeling language. In other cases, however, either no assumption is made on the adopted language, or different languages are adopted.

In UML-based proposals, meta-models are typically used first at meta-level for creating the specific DSML, and then at a lower level for modeling specific sub-systems: automatic model transformations, based on the relationships defined at the meta-level, finally map a DSML onto another, transforming an input model onto a new model in a different (lower level) language.

Tool Comparison

Tools must support not only the development process in general, but the specific abstractions and techniques that are peculiar to MPM: so, their design and implementation require a considerable development effort. The complexity and learning curve of the tools also needs to be evaluated, as it is often an obstacle also for experienced users.

In the approaches derived from (Vangheluwe, 2002), based on meta-models, tools are asked to support the definition of the desired DSML and the related transformation processing; other approaches have their own, different requirements. While most of the above-presented approaches are not supported by a specific tool, three of them do have one—namely, AtoM^3 (Lara & Vangheluwe, 2002), AToMPM (Syriani & Vangheluwe, 2012), and Modelverse (Van Tendeloo, Van Mierlo, & Vangheluwe, 2019). Their evaluation criteria should therefore include the tool power *vs.* complexity, the user-friendliness of the installation procedure, the learning curve, and the state of the documentation: results are summarized in Table 3.

All tools require their own (i.e., typically not the last) version of Python—namely, Python 2.3 for AtoM^3 and Python 2.7 for AToMPM and Modelverse, none of which is compatible with Python 3.x. Moreover, AToMPM adopts a client-server architecture, which requires that the server is properly configured to include a specific Python graphic package; the client instead just run inside Google Chrome, exploiting its JavaScript support. Modelverse, indeed, requires the SCCD compiler and runtime to be executed. They also require a proportionally long time to be learned and fruitfully exploited.

AtoM^3 may likely result too complicated for users with little experience in meta-model creation: in addition, users have to write by themselves the Python code for expressing pieces of models. On the other hand, it is well-documented— although the documentation and examples refer to an older version, with different GUI and functionalities, which makes it uneasy to follow the provided examples. AToMPM, instead, comes with a nice video and introductory tutorial

Table 2. Comparing meta-models.

Proposal	**Meta-models Adoption**	**Meta-modeling Language**	**Used for**	
			DSML	*Transformation*
(Denil, Vangheluwe, De Meulenaere, & Demeyer, 2012)	✓	E-R + UML	✓	✓
(Mosterman & Vangheluwe, 2002)	✓	different languages	✓	✓
(Mosterman & Vangheluwe, 2004)	✓	UML	✓	✓
(Vangheluwe, 2002)	✘	(not applicable)	✘	
(Lara & Vangheluwe, 2002)	✓	E-R + UML	✓	✓
(Li, Lei, Wang, Wang, & Zhu, 2013)	✓	no assumption	✓	✓
(Syriani, 2011)	✓	UML	✓	✓
(Mustafiz, Denil, Lucio, & Vangheluwe, 2012)	✓	UML	✓	✓
(Lorenz & Jost, 2006)	✘	(not applicable)	✘	
(Balasubramanian, Levendovszky, Dubey, & Karsai, 2014)	✓	UML	✓	✘
(Vranic, 2005)	✘	(not applicable)	✘	
(Dalibor, Jansen, Rumpe, Wachtmeister, & Wortmann, 2019)	✓	UML	✓	✘
(Lynch, Padilla, Diallo, Sokolowski, & Banks, 2014)	✘	(not applicable)	✘	
(Van Tendeloo, Van Mierlo, & Vangheluwe, 2019)	✘	(not applicable)	✘	

Table 3. Comparison of the supporting tools.

Tool	**Installation requirements**	**Learning time**	**Documentation**
AtoM3	Python 2.3	Considerable	documentation + examples (referred to an old version)
AToMPM	Python 2.7 + graphic package + JavaScript package	considerable	video + tutorial slides
Modelverse	SCCD compiler and runtime + Python 2.7	considerable	documentation + technical report

slides: still, the lack of a complete guide might force users to navigate menus and try out some modeling libraries to actually unleash its potential. Modelverse, however, requires users to write Python code in order to create their models.

Another key point concerns *scalability*. Most of the above MPM approaches adopt the transformational process model and the MDE techniques, which means that the system under development is – roughly speaking – decomposed in multiple, independently-modeled sub-systems, and the code is finally generated via model transformations. However, most of the accompanying examples refer to rather simple situations, like the car windows system in (Mosterman & Vangheluwe, 2004), whose decomposition is equally simple and the involved paradigms are just two or three. So, more complex case studies, with multiple involved paradigms, would be required for an effective assessment of the applicability of MPM approaches to intricate scenarios such as pervasive computing (Satyanarayanan, 2001), pervasive intelligent systems (Mariani & Omicini, 2013), self-organizing systems (Di Marzo Serugendo, Gleizes, & Karageorgos, 2006).

ENGINEERING MULTI-PARADIGM SYSTEMS BEYOND MPM

The Multi-Agent Systems paradigm

Due to its different perspective, the Multi-Agent System paradigm is mostly ignored in MPM field, despite its widespread adoption as a "*general-purpose paradigm for software development*" (Zambonelli & Parunak, 2003) – possibly because the MPM community traditionally focuses on application domains rooted in the M&S field (Balci, 2012). A notable exception is (Lorenz & Jost, 2006), where agent-based modeling is considered in the simulation context. However, the agent paradigm is widely acknowledged to be well-suited to model complex systems, thanks to features such as agent autonomy, sociality, and more generally to foundational notions of goals, actors, environment, etc. (Molesini, 2008) which make it possible to design complex, interactive systems at the "adequate" abstraction level—usually, higher than other paradigms. Moreover, application scenarios such as complex socio-technical systems (Bryl, Giorgini, & Mylopoulos, 2009) and pervasive intelligent systems (Mariani & Omicini, 2013) could likely benefit from a multi-paradigm approach that includes some key aspects of the MAS paradigm—for instance, to coordinate and govern the interaction among the many autonomous entities.

Yet, most MPM approaches could hardly be applied to the agent paradigm "as they are", since this would require that agent entities are transformed into suitable coordination "active entities" and vice-versa— would call for the agent paradigm to be integrated with other relevant paradigms, such as coordination (Papadopoulos, Stavrou, & Papapetrou, 2006) and event-based (Omicini, 2015) paradigms. In its turn, this would amount at (i) integrating the agent meta-model with the coordination meta-model, and (ii) creating a new development process by assembling fragments of agent-oriented methodologies with fragments of coordination methodologies. Analogously, integrating the agent paradigm with the event-based paradigm, as suggested in (Omicini, 2015), would require the suitable modeling of interactions between autonomous entities and the environment where they are immersed.

Situational Process Engineering

Situational Process Engineering (SPE) techniques (Cossentino, Gaglio, Garro, & Seidita, 2007) (Cossentino, Seidita, Hilaire, & Molesini, 2014) – that is, the evolution of the SME initially developed in the object-oriented field (Brinkkemper, Saeki, & Harmsen, 1999) – provide the

conceptual and practical foundations for defining ad-hoc development processes, by suitably *re-combining* pieces (reusable *fragments*) of existing processes, so as to create a methodology that is specific for a given purpose.

Adopting SPE in a multi-paradigm context opens several challenges. First, the SPE approach cannot be used *as is*: integrating process fragments rooted on the same paradigm (and therefore sharing similar abstraction) is one thing, but doing the same with process fragments coming from methodologies based on completely different paradigms is all another story.

The availability of such fragments is another issue: while process fragments from AOSE methodologies are available for integration, fragments from other paradigms would need to be created from scratch. This is all but trivial, since fragment extraction requires a deep knowledge of the specific methodology; alternatively, the methodology should be very well documented in a standard way (Cossentino, Seidita, Hilaire, & Molesini, 2014)—another critical issue *per se*. Suitably documenting such process fragments is one third, critical aspect. While the Documentation Template under development in the AOSE community could be of help for methodologies, the development of a standardized fragment documentation template is still a work in progress.

The intriguing case of Systems of Systems

From an opposite perspective, paradigms could be used as *isolated worlds*, if each sub-system is modeled with a separate paradigm and sub-systems have little inter-dependencies on each other. The development of each sub-system could then follow its own development process, tailored to the specific paradigm.

An appealing approach could be to define a sort of "process orchestrator" to (i) manage the different sub-processes and their mutual inter-relationships in terms of time scheduling and interleaved activities, and (ii) analyze and design the interactions among sub-systems.

An example is represented by the so-called *System of Systems* (SoS) (Nielsen, Fitzgerald, Woodcock, & Peleska, 2015)—basically, a set of ad-hoc systems that put their resources and skills together, so that the resulting system provides more features and services than the simple sum of its parts. SoS move from the idea that "*the growing interdependency of systems contributes to an ever-emerging complexity*" (Ross, Ulieru, & Gorod, 2014): the use of multi-paradigm approach in that field is under investigation. A key challenge in this context is "*to identify the boundaries of the overall SoS and of its independent constituent systems"* (Nielsen, Fitzgerald, Woodcock, & Peleska, 2015), where "boundaries" refers both to technical (interfaces definition, integration, testing) and organizational aspects (governance, stakeholders).

Moreover, an open research aspect is how to possibly exploit the presence of a "process orchestrator" to structure the design at two levels—a (lower) level for each sub-system, and the orchestrator in charge of the global development process, obtained by integrating and scheduling the development process of each sub-system, at a higher level.

Multi-Agent Systems for Systems of Systems

(Nielsen, Fitzgerald, Woodcock, & Peleska, 2015) argue that *"the space of SoS might be described in terms of eight dimensions*"—autonomy, independence, distribution, evolution, dynamic reconfiguration, emergence of behavior, interdependence, interoperability. Since these aspects fall well inside the MAS features, it is conceivable that the MAS paradigm can be exploited to address at least some of the SoS challenges: again, this is an open research issue.

CONCLUSIONS

Summing up, MPM approaches typically adopt a transformational process model, with UML being the most common representation language for meta-modeling techniques. However, documentation is often unsatisfactory and inadequate to the learning curve, especially in case of non-expert designers. Tools are powerful, but also suffer from the inadequacy of documentation, tutorials, and effective examples.

Current MPM approaches seem not to consider the MAS paradigm, which could be of help to cope with complex scenarios such as pervasive intelligent systems, ubiquitous systems, and self-organizing systems. The perspective of a blend of MAS within MPM approaches is appealing, both for the intrinsic agent features and because of the chance, provided by SPE, to build ad-hoc development processes by combining process fragments rooted into different paradigms: this is an open research area. The emergence of the SoS paradigm (Nielsen, Fitzgerald, Woodcock, & Peleska, 2015), where multi-paradigm approaches are also investigated (Ross, Ulieru, & Gorod, 2014), opens another promising conceptual and technical framework for agent-oriented abstractions, technologies, and methodologies.

Interestingly enough, the term "paradigm" is left in the background in most of the above works in the "multi-paradigm" modeling field: typically, it is merely mentioned to highlight the simultaneous presence/application of different approaches, set of rules, methods, etc. "—yet, without pointing out what a "paradigm" is supposed to be, nor highlighting the possible paradigm shifts that, according to (Kuhn, 2012), inherently define the paradigm concept per se. This is not peculiar to the MPM field: looking wider, the same seems to hold for multi-paradigm programming (MPM), despite programming languages date back to the sixties.

Overall, with the growing complexity of software systems in all fields, it can be expected that the practice of combining ideas, methods, approaches, processes, methodologies – *paradigms*, in the broadest sense – from diverse fields, possibly far from each other, becomes increasingly common for conceptual, economical and practical reasons. At the same time, being "multi-paradigm" is much more than just "adopting" two paradigms in some way: suitable models, guidelines, and new ad-hoc research are needed to make this scenario actually fruitful.

REFERENCES

Atkinson, C., & Kuhne, T. (2001). The essence of multilevel metamodling. In G. M, & K. C., *UML 2001 - The Unified Modeling Language. Modeling Languages, Concepts and Tools* (Lecture Notes in Computer Science ed., Vol. 2185, pp. 19-33). Berlin Heidelberg: Springer.

Balasubramanian, D., Levendovszky, T., Dubey, A., & Karsai, G. (2014). Taming Multi-Paradigm Integration in a Software Architecture Description Language. In D. Balasubramanian, C. Jacquet, P. Van Gorp, S. Kokaly, & T. Meszaros (Ed.), *8th Workshop on Multi-Paradigm Modeling (MPM@MODELS 2014). 1237*, pp. 67-76. RWTH Aachen University: Sun SITE Central Europe. Retrieved from http://ceur-ws.org/Vol-1237/paper7.pdf

Balci, O. (2012, jul). A Life Cycle for Modeling and Simulation. *Simulation, 88*(7), 870-883. doi:10.1177/0037549712438469

Bernon, C., Cossentino, M., Gleizes, M.-P., Turci, P., & Zambonelli, F. (2004). In J. Odell, P. Giorgini, & J. P. Muller, *Agent Oriented Software Engineering V* (Lecture Notes in Computer Science ed., Vol. 3382, pp. 62-77). Berlin Heidelberg: Springer. doi:10.1007/978-3-540-30578-1_5

Boehm, B. W. (1988, may). A Spiral Model of Software Development and Enhancement. *IEEE Computer, 21*(5), 61-72. doi:10.1109/2.59

Brinkkemper, S., Saeki, M., & Harmsen, F. (1999). Meta-Modelling Based Assembly Techniques for Situational Method Engineering. *Information Systems, 24*(3), 209-228. doi:10.1016/S0306-4379(99)00016-2

Brooks, F. P. (1987, apr). No Silver Bullet Essence and Accidents of Software Engineering. *IEEE Computer, 20*(4), 10-19. doi:10.1109/MC.1987.1663532

Bryl, V., Giorgini, P., & Mylopoulos, J. (2009, feb). Designing Socio-Technical Systems: From Stakeholder Goals to Social Networks. *Requirement Engineering, 14*(1), 47-70. doi:10.1007/s00766-008-0073-5

Cernuzzi, L., Cossentino, M., & Zambonelli, F. (2005, mar). rocess Models for Agent-Based Development. *Engineering Applications of Artificial Intelligence, 18*(2), 205-222. doi:10.1016/j.engappai.2004.11.015

Ciccozzi, F. a., Vangheluwe, H., & Weyns, D. (2019). Blended Modelling – What, why and how. *First International Workshop on Multi-Paradigm Modelling for Cyber-Physical Systems.* Munich. Retrieved from https://msdl.uantwerpen.be/conferences/MPM4CPS/2019/wp-content/uploads/2019/09/mpm4cps2019_Blended.pdf

Cossentino, M., Gaglio, S., Garro, A., & Seidita, V. (2007). Method fragments for agent design methodologies: from standardisation to research. *International Journal of Agent Oriented Software Engineering (IJAOSE), 1*(1), 91-121. doi:10.1504/IJAOSE.2007.013266

Cossentino, M., Seidita, V., Hilaire, V., & Molesini, A. (2014). *FIPA Design Process Documentation and Fragmentation Working Group*. Retrieved from FIPA Design Process Documentation and Fragmentation Working Group: http://www.pa.icar.cnr.it/cossentino/fipa-dpdf-wg/

Dalibor, M., Jansen, N., Rumpe, B., Wachtmeister, L., & Wortmann, A. (2019). Model-Driven Systems Engineering for Virtual Product Design. *MPM4CPS 2019 : First International Workshop on Multi-Paradigm Modelling for Cyber-Physical Systems.* Retrieved from https://msdl.uantwerpen.be/conferences/MPM4CPS/2019/wp-content/uploads/2019/09/Paper_MDSE4VirtualProductDesign.pdf

Denil, J., Vangheluwe, H., De Meulenaere, P., & Demeyer, S. (2012). Calibration of Deployment Simulation Models: A Multi-Paradigm Modelling Approach. *2012 Symposium on Theory of Modeling and Simulation -- DEVS Integrative M&S Symposium (TMS/DEVS '12).* Orlando, Florida: Society for Computer Simulation International. Retrieved from http://dl.acm.org/citation.cfm?id=2346629

Di Marzo Serugendo, G., Gleizes, M.-P., & Karageorgos, A. (2006, jan). Self-Organization in Multi-Agent Systems. *The Knowledge Engineering Review, 20*(2), 165-189. doi:10.1017/S0269888905000494

Diallo, S., Andreas Tolk, A., Gore, R., & Padilla, J. (2005). Modeling and simulation framework for systems engineering. In D. Gianni, A. D'Ambrogio, & A. Tolk, *Modeling and Simulation-Based Systems Engineering Handbook* (First Edition ed., p. 377-401). Boca Raton: CRC Press. doi:10.1201/b17902

Ehrig, H., Engels, G., Kreowski, H.-J., & Rozenberg, G. (1999). *Handbook of Graph Grammars and Computing by Graph Transformation.* River Edge, NJ, USA: World Scientific. doi:10.1142/4180

Fondement, F., & Silaghi, R. (2004). Defining Model Driven Engineering Processes. In M.

Gogolla, P. Sammut, & J. Whittle (Ed.), *3rd UML Workshop in Software Model Engineering (WiSME 2004)* (pp. 1-11). Lisbon, Portugal: Universidade Nova de Lisboa. Retrieved from http://ctp.di.fct.unl.pt/UML2004/workshop.html#ws5

Fuggetta, A. (2000). Software Process: A Roadmap. *ICSE '00: Proceedings of the Conference on The Future of Software Engineering* (p. 25-34). Limerick, Ireland: ACM Press. doi:10.1145/336512.336521

Giese, H., & Levendovszky, T. (Eds.). (2006). Proceedings of the Workshop on Multi-Paradigm Modeling: Concepts and Tools 2006. *Proceedings of the Workshop on Multi-Paradigm Modeling: Concepts and Tools. 2006/1*. BME-DAAI Technical Report Series. Retrieved from http://avalon.aut.bme.hu/\~mesztam/conferences/mpm06/mpm06_proc.pdf

Gonzalez-Perez, C., McBride, T., & Henderson-Sellers, B. (2005). A Metamodel for Assessable Software Development Methodologies. *Software Quality Journal, 13*(2), 195-214. doi:10.1007/s11219-005-6217-7

Group, O. (2008). *SPEM 2.0* . Tratto da Software & Systems Process Engineering Metamodel Specification. Version 2.0: http://www.omg.org/spec/SPEM/2.0/

Hardebolle, C., & Boulanger, F. (2009, nov). Exploring Multi-Paradigm Modeling Techniques. *Simulation, 85*(11-12), 688-708. doi:10.1177/0037549709105240

Henderson-Sellers, B. (2002). Process Metamodelling and Process Construction: Examples Using the OPEN Process Framework OPF. *Annals of Software Engineering, 14*(1), 341-362. doi:10.1023/A:1020570027891

Henderson-Sellers, B., & Gonzalez-Perez, C. (2005, jan). A Comparison of Four Process Metamodels and the Creation of a New Generic Standard. *Information & Software Technology, 47*(1), 49-65. doi:10.1016/j.infsof.2004.06.001

Jensen, K., & Kristensen, L. M. (2009). *Coloured Petri Nets. Modelling and Validation of Concurrent Systems.* Berlin Heidelberg: Springer. doi:10.1007/b95112

Kuhn, T. S. (2012). *The Structure of Scientific Revolutions* (50th Anniversary ed.). Chicago : University of Chicago Press.

Lara, J. d., & Vangheluwe, H. (2002). Computer Aided Multi-paradigm Modelling to Process Petri-Nets and Statecharts. In A. Corradini, H. Ehrig, H. -J. Kreowski, & G. Rozenberg, *Graph Transformation* (Lecture Notes in Computer Science ed., Vol. 2505, p. 239-253). Berlin Heidelberg: Springer. doi:10.1007/3-540-45832-8

Larman, C., & Basili, V. R. (2003, jun). Iterative and Incremental Development: A Brief History. *IEEE Computer , 36*(6), 47-56. doi:10.1109/MC.2003.1204375

Li, X., Lei, Y., Wang, W., Wang, W., & Zhu, Y. (2013). A DSM-based Multi-Paradigm Simulation Modeling Approach for Complex Systems. *2013 Winter Simulation Conference: Simulation: Making Decisions in a Complex World (WSC '13)* (pp. 1179-1190). Piscataway, NJ, USA: IEEE Press. Retrieved from http://dl.acm.org/citation.cfm?id=2675983.2676133

Lorenz, T., & Jost, A. (2006). Towards an Orientation Framework in Multi-Paradigm Modeling: Aligning Purpose, Object and Methodology in System Dynamics, AGent-Based Modeling and DIscrete-EVent-Simulation. In A. Grosler, E. A. Rouwette, R. S. Langer, J. I. Rowe, & J. M. Yanni (Ed.), *24th International Conference of the System Dynamics Society* (pp. 2134-2151). Albany, NY, USA: System Dynamics Society. Retrieved from http://www.systemdynamics.org/conferences/2006/proceed/papers/LOREN178.pdf

Lynch, C., Padilla, J., Diallo, S., Sokolowski, J., & Banks, C. (2014). A multi-paradigm modeling framework for modeling and simulating problem situations. *Proceedings of the*

Winter Simulation Conference 2014 (p. 1688-1699). Savanah, GA, USA: IEEE. doi:10.1109/WSC.2014.7020019

Mariani, S., & Omicini, A. (2013). Molecules of Knowledge: Self-Organisation in Knowledge-Intensive Environments. In G. Fortino, C. Badica, M. Malgeri, & R. Unland (A cura di), *Intelligent Distributed Computing VI. 446*, p. 17-22. Berlin Heidelberg: Springer. doi:10.1007/978-3-642-32524-3_4

Molesini, A. (2008). *Meta-Models, Environment and Layers: Agent-Oriented Engineering of Complex Systems.* Ph.D Thesis, Alma Mater Studiorum - Università di Bologna , Dipartimento di Elettronica, Informatica e Sistemistica, Bologna.

Mosterman, P. J., & Vangheluwe, H. (2002, oct). Guest Editorial: Special Issue on Computer Automated Multi-Paradigm Modeling. *ACM Transactions on Modeling and Computer Simulation (TOMACS), 12*(4), 249-255. doi:10.1145/643120.643121

Mosterman, P. J., & Vangheluwe, H. (2004). Computer Automated Multi-Paradigm Modeling: An Introduction. *Simulation, 80*(9), 433-450. doi:10.1177/0037549704050532

Mustafiz, S., Denil, J., Lucio, L., & Vangheluwe, H. (2012). The FTG+PM Framework for Multi-paradigm Modelling: An Automotive Case Study. *6th International Workshop on Multi-Paradigm Modeling (MPM '12)* (pp. 13-18). New York, NY, USA: ACM. doi:10.1145/2508443.2508446

Nielsen, C. B., Fitzgerald, J., Woodcock, J., & Peleska, J. (2015). Systems of Systems Engineering: Basic Concepts, Model-Based Techniques, and Research Directions. *ACM Computer Surveys, 48*(2), 1-41. doi:10.1145/2794381

NoMagic. (2019). *MagicDraw Home Page*. Retrieved 11 20, 2019, from NoMagic: https://www.nomagic.com/products/magicdraw

Oliveira, B. C., & Loh, A. (2013). Abstract Syntax Graphs for Domain Specific Languages. *ACM SIGPLAN 2013 Workshop on Partial Evaluation and Program Manipulation (PEPM '13)* (p. 87-96). New York, NY, USA: ACM. doi:10.1145/2426890.2426909

OMG. (1997). *UML Home Page*. Retrieved from UML: http://www.uml.org

OMG. (2001). *MDA Home Page*. Retrieved from MDA: http://www.omg.org/mda/

OMG. (2002). *MOF Home Page*. Retrieved from MOF: http://www.omg.org/mof/

OMG. (2008). *Software & Systems Process Engineering Metamodel Specification. Version 2.0*. Retrieved from SPEM: http://www.omg.org/spec/SPEM/2.0/

Omicini, A. (2015). Event-Based vs.\ Multi-Agent Systems: Towards a Unified Conceptual Framework. In G. Fortino, W. Shen, J.-P. Barthès, J. Luo, W. Li, S. Ochoa, . . . M. Ramos (A cura di), *2015 19th IEEE International Conference on Computer Supported Cooperative Work in Design (CSCWD2015)* (p. 1-6). Los Alamitos, CA, USA: IEEE Computer Society. doi:10.1109/CSCWD.2015.7230924

Organization, S. (2003). *SysML Open Source Project Home Page*. Retrieved from SysML Open Source Project: https://sysml.org/

Papadopoulos, G. A., Stavrou, A., & Papapetrou, O. (2006, mar). An Implementation Framework for {S}oftware {A}rchitectures Based on the Coordination Paradigm. *Science of Computer Programming, 60*(1), 27--67. doi:10.1016/j.scico.2005.06.002

Rhazali, Y., El Hachimi, A., Chana, I., Lahmer, M., & Rhattoy, A. (2019). Automate Model Transformation From CIM to PIM up to PSM in Model-Driven Architecture. In B. B. Gupta, *Modern Principles, Practices, and Algorithms for Cloud Security* (p. 262-283). doi:10.4018/978-1-7998-1082-7.ch013

Rhazali, Yassine, Hadi, Y., & Mouloudi, A. (2016). CIM to PIM Transformation in MDA: from

Service-Oriented Business Models to Web-Based Design Models. *International Journal of Software Engineering and Its Applications, 10*(4), 125-142. doi:http://dx.doi.org/10.14257/ijseia.2016.10.4.13

Ross, W., Ulieru, M., & Gorod, A. (2014). A Multi-Paradigm Modelling & Simulation Approach for System of Systems Engineering: A Case Study. *9th International Conference on System of Systems Engineering (SoSE 2014)* (p. 183-188). Piscataway, NJ, USA: IEEE. doi:10.1109/SYSOSE.2014.6892485

Satyanarayanan, M. (2001). Pervasive Computing: Vision and Challenges. *IEEE Personal Communications, 8*(4), 10--17. doi:10.1109/98.943998

Schmidt, D. C. (2006, feb). Guest Editor's Introduction: Model-Driven Engineering. *IEEE Computer, 39*(2), 25-31. doi:10.1109/MC.2006.58

Sommerville, I. (2007). *Software Engineering* (8th ed.). Edinburgh, UK: Addison-Wesley.

Syriani, E. (2011). *A Multi-Paradigm Foundation for Model Transformation Language Engineering.* McGill University, School of Computer Science. Montreal, QC, Canada: McGill University.

Syriani, E., & Vangheluwe, H. (2012). *AToMPM Home Page*. Retrieved from AToMPM: http://www-ens.iro.umontreal.ca/$%5Csim$syriani/atompm/atompm.htm

Van Tendeloo, Y., Van Mierlo, S., & Vangheluwe, H. (2018). *Modelverse Home Page*. Retrieved from Modelverse: https://msdl.uantwerpen.be/git/yentl/modelverse

Van Tendeloo, Y., Van Mierlo, S., & Vangheluwe, H. (2019, oct). A Multi-Paradigm Modelling approach to live modelling. *Software & Systems Modeling, 18*(5), 2821--2842. doi:10.1007/s10270-018-0700-7

Vangheluwe, H. (2002). An Introduction to Multiparadigm Modelling and Simulation. *AI, Simulation & Planning in High Autonomy Systems (AIS 2002)* (p. 9-20). Lisbon, Portugal: Society for Modeling & Simulation International (SCS).

Vangheluwe, H. (2002). An Introduction to Multiparadigm Modelling and Simulation. *AI, Simulation & Planning in High Autonomy Systems (AIS 2002)* (p. 9-20). Lisbon, Portugal: Society for Modeling & Simulation International (SCS).

Vranic, V. (2005, jun). Multi-Paradigm Design with Feature Modeling. *Computer Science and Information Systems, 2*(1), 79-102. doi:10.2298/CSIS0501079V

Zambonelli, F., & Parunak, H. V. (2003). Towards a Paradigm Change in Computer Science and Software Engineering: A Synthesis. *The Knowledge Engineering Review*(4), 329--342. doi:10.1017/S0269888904000104