MULTI-PARADIGM MODELLING FOR MODEL BASED SYSTEMS ENGINEERING: EXTENDING THE FTG+PM

Randy Paredis
Joeri Exelmans
Hans Vangheluwe

University of Antwerp – Flanders Make
Middelheimlaan 1
Antwerp, BELGIUM
{randy.paredis, joeri.exelmans, hans.vangheluwe}@uantwerpen.be

ABSTRACT

Model Based Systems Engineering (MBSE) supports Cyber-Physical Systems (CPS) design, by experts from various domains, through complex *workflows*, manipulating models in different *formalisms*, each with their own methods, techniques and tools. We refer to often-used combinations of workflow and formalism patterns as modelling *paradigms*. The Formalism Transformation Graph (FTG) and Process Model (PM) are key components in Multi-Paradigm Modelling (MPM) based MBSE, which combines multiple modelling paradigms. This paper extends the FTG+PM framework: a Process Trace (PT) records all start/end *events* of engineering activities as well as all *versions* of all artifacts consumed/produced by them. A PT results from the enactment of a PM. The artifacts and activities in the PT are "virtual": though referred to in a technology-agnostic manner, they are realized using various technologies. *Adapters* translate between both, making the FTG+PM *federated*. An Automated Guided Vehicle (AGV) example is used.

Keywords: Formalism Transformation Graph and Process Model (FTG+PM), Model-Based Systems Engineering (MBSE), Multi-Paradigm Modelling (MPM)

1 INTRODUCTION

The technical systems we build, and in particular, Cyber-Physical Systems (CPS), are of an ever increasing complexity. CPS emerge from the networking of multi-physical (mechanical, electrical, hydraulic, biochemical, ...) and computational (control, signal processing, logical inference, planning, ...) processes, often interacting with a highly uncertain environment, including human actors, in a socio-economic context (Carreira, Amaral, and Vangheluwe 2020). It is the heterogeneity in views, components, abstractions and their many inter-relationships, in combination with the many stakeholders from different domains, collaboratively designing such systems, that contribute to their complexity. It is the purpose of the Systems Engineering (SYE) discipline to design, integrate, and manage such complex systems over their entire life cycle (Kassiakoff et al. 2010). Starting from an initial set of goals, SYE carries out a number of *activities* (human and/or automated) to achieve these goals. The combination of these activities is called the *workflow* or *life-cycle*. It can be explicitly modelled in a workflow or *process* model (PM), in an appropriate modelling language such as UML Activity Diagrams (Object Management Group 2017). Russell, Van Der Aalst, and Ter Hofstede (2016) discuss a set of workflow patterns which are useful when designing a PM.

To help realize MPM, Mustafiz et al. (2012) introduced a framework for Model-Based Systems Engineering in which a *Process Model* (PM) is combined with a *Formalism Transformation Graph* (FTG), a "map" of all artifact types (also known as meta-models) and activity types (in the form of contracts) and how they are related.

Despite its demonstrated advantages, we have noticed that the FTG+PM is incomplete to fully support systems engineering. On the one hand, there is a need to record a *trace* of all artifact versions and activity executions. This, to allow for repeatability, replicability and explainability (Plesser 2018). On the other hand, there is a need for a uniform and technology-neutral way to represent and manipulate artifacts and activities.

The remainder of this paper is structured as follows. Section 2 discusses some related work and section 3 introduces a Line-Following Robot system engineering example. Based on this example, section 4 introduces the FTG+PM as well as proposed extensions, Process Trace and Storage/Service/RealWorldArtifact and the mapping between them by means of adapters. Section 5 shows how the proposed extensions allow for advanced queries. Finally, section 6 concludes the paper.

2 RELATED WORK

The Unified Modeling Language (UML) 2.0 defines so-called *Activity Diagrams* (Dumas and Ter Hofstede 2001). They capture the behaviour and process flow, and are similar to *flowcharts* (European Computer Manufacturers Association et al. 1966). The Business Process Model and Notation (BPMN) standard is used to describe workflows within (mostly) a business context (White 2004). These processes can easily be mined and optimized to detect common (anti-)patterns, reduce bottlenecks and increase efficiency (van der Aalst 2016). The Interaction Flow Modeling Language (IFML) is a similar graphical notation to describe user interactions and front-end behaviour (Brambilla and Fraternali 2014). The aforementioned languages are often used to describe and analyze Product Lifecycle Management (PLM) (Grieves 2006). Van Mierlo et al. (2018) compare multiple workflow modelling tools/languages, including Activity Diagrams, BPMN and FTG+PM.

Different models often provide different viewpoints (for specific stakeholders) on a system. These models are often described in appropriate Domain-Specific Languages (DSLs). Karaduman et al. (2021) clarify the use of viewpoints in FTG+PM. The "black box" combination, integration and interaction of heterogeneous models is called *model federation* (Golra et al. 2016, Wagner et al. 2020, OpenMBEE Organisation 2011). Instead of enlarging a single meta-model with new information, *adapters* are used to bridge the gap between the multiple meta-models. Tools such as PAMELA (Guérin et al. 2021) and Syndeia (Bajaj et al. 2016) support model federation.

Many definitions for *megamodel* exist (Hebig, Seibel, and Giese 2012). In essence, it is a collection of related models. Salay et al. (2015) describe how such megamodels can be managed, and Favre and Nguyen (2005) discusses how software (and thus system) evolution can be modelled. Maro (2020) performs a literature study and some optimizations for traceability in software systems. It also discusses some tools and techniques (including versioning) to obtain traceability. Hassane et al. (2019) introduce MAPLE-T, a tool for process enactment with traceability support based on megamodels.

One advantage of an explicitly modelled workflow of system development becomes apparent when one tries to recreate the system or even just to understand what decisions in the past led to the current realized system. In scientific experimentation, the terms *repeatability* (the same team can produce the same results with the exact same experimental setup), *replicability* (a different team can produce the same results with the exact same experimental setup) and *reproducibility* (a different team can produce the same results with a different

experimental setup) are used (Hong 2021). Explicitly modelling and recording workflow clearly supports repeatability and replicability in systems engineering projects.

In its 2035 vision for SYE, the International Council on Systems Engineering (INCOSE 2021) describes the critical role Model-Based Systems Engineering (MBSE) plays in tackling increasing system complexity, mainly when supported by toolchains (Ma et al. 2022). It also states the importance of integrated analysis in a broad set of system domains. We believe that *Multi-Paradigm Modelling* (MPM) can help realize INCOSE's 2035 vision. MPM proposes to *model* every part and aspect of a system *explicitly*, at the most appropriate level(s) of abstraction, using the most appropriate modelling formalism(s), while explicitly modelling workflows (Mosterman and Vangheluwe 2002, Mosterman and Vangheluwe 2004, Amrani et al. 2021).

3 EXAMPLE USE-CASE: AUTOMATED GUIDED VEHICLE DIGITAL SHADOW

To illustrate the contributions of this paper, the proposed FTG+PM extensions will be applied to a simple Automated Guided Vehicle (AGV) (*i.e.,* a Line-Following Robot (LFR)) example that was first introduced in (Paredis and Vangheluwe 2021). An AGV needs to be designed and realized whose goal is to follow a line on the ground as closely as possible whilst moving swiftly, economically and safely. In the digital world, a virtual (simulated) copy is created, acting on the same input from the environment as the real system. The real and the simulated robot should follow the exact same trajectory. Through comparison of the two trajectories, anomalies in the operation of the robot can be detected. Such a setup is called a Digital Shadow (Kritzinger et al. 2018). Paredis and Vangheluwe (2021) includes an early version of an FTG+PM for an AGV system, which was later adapted in (Paredis, Gomes, and Vangheluwe 2021).

The AGV is parametrized by its wheel radius r and distance between the wheels d. Its time-varying state is captured by the robot's velocity v and its heading ω. Ideally, the position (and heading) of the real and the virtual AGV should be sufficiently close at all points in time, independent of the line to follow.

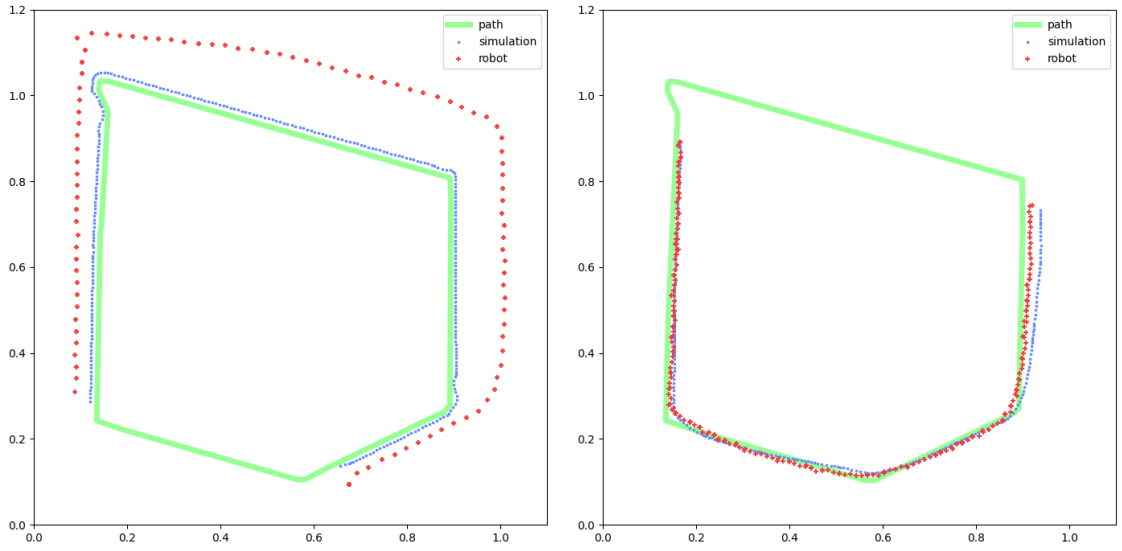
Figures 1a and 1b show traces of this system's behaviour. The full green line represents the path to follow, the blue, dotted line identifies the Digital Shadow simulated position and the red, dotted line depicts the position of the realized robot. In $trace_1$ (Figure 1a), the realized system is unable to accurately follow the line. This is because the model used in the simulation is used outside its validity range (Denil et al. 2017) with respect to the real-world physics. $trace_2$ (Figure 1b) shows a better *version* produced after a few design iterations. Such iterations in workflows are common in industry, and the purpose of this work is to explicitly describe them.

Anecdotal evidence from industrial partners has shown that over 80% of the time spent goes into finding old information (models, data, choices made) in the massive amount of historical information. In many cases, it turns out the needed information was not even recorded. This is exacerbated in cases of collaboration with multiple stakeholders (in different domains). Not explicitly tracking/storing all information about the executed activities and the artifacts they manipulate will often result in a severe waste of time trying to locate/re-create this information.

Despite its simplicity, the AGV use-case presented contains a large number of distinct processes and intermediate results, making it good demonstrator for our research.

4 FORMALISM TRANSFORMATION GRAPH AND PROCESS MODEL (FTG+PM)

The FTG+PM was introduced to guide system developers in their (model based) system engineering efforts (Mustafiz et al. 2012). Figures 5 (FTG) and 2 (PM) show a simplified version of an FTG+PM describing



(a) $trace_1$ (b) $trace_2$

Figure 1: Behaviour (position) traces of the AGV realization (robot) and Digital Shadow (simulation).

the creation of an AGV. For simplicity, the "*Digital Shadow*" part was omitted in this paper and only the system design and realization is shown.

4.1 Process Model

The *Process Model* (PM) is a workflow model, which precisely specifies a combination of activities (represented by rountangles) that are carried out to achieve the system engineering goal. These activities are connected through *control flow* connections (represented by bold, blue arrows), which may split into concurrent activities and may subsequently be joined, denoting synchronization. Both split and join are represented by solid blue bars. Multiple control flow outputs of a single activity indicate choice. For example, the sys_ana: System Analysis activity may result in a new version and loop back for another iteration, or terminate the process (denoted by a solid blue circle with a line around it). Note the underlined instance:Type notation, which will be further explained later. There is also data flow: each activity may use and/or produce *artifacts* (represented by green arrows and rectangles). PM concrete visual syntax was originally based on UML 2.0 activity diagrams, but was modified based on "physics of notation" principles presented in (Moody 2009), for improved clarity and consistency. Activities can be carried out by humans (identified with a grey background) or automatically, for example by a computer (denoted by a gear icon and a yellow background). Activities may be hierarchically de-/composed (represented by a sitemap icon in the activity rountangle).

Starting from the AGV requirements, the system is decomposed into three parts: a control algorithm ctrl_alg: Algorithm, plant equations plant_eqs: Equations and a design sketch agv_dsg: Sketch. These three artifacts are developed somewhat concurrently to result in the AGV model (agv_model: CBD) and the AGV physical realization (agv: AGV). The deploy_sim: Deployment and Simulation hierarchical activity will result in a deployed system and a behaviour trace, on which some analysis may be done, potentially yielding insights to be used as input in the development of a new version. Note that this is a representative workflow for this system, but may evolve over time. Additionally, the developed framework/architecture may change as well. Any implementation of the FTG+PM must support this kind of *evolution* (Meyers and Vangheluwe 2011).

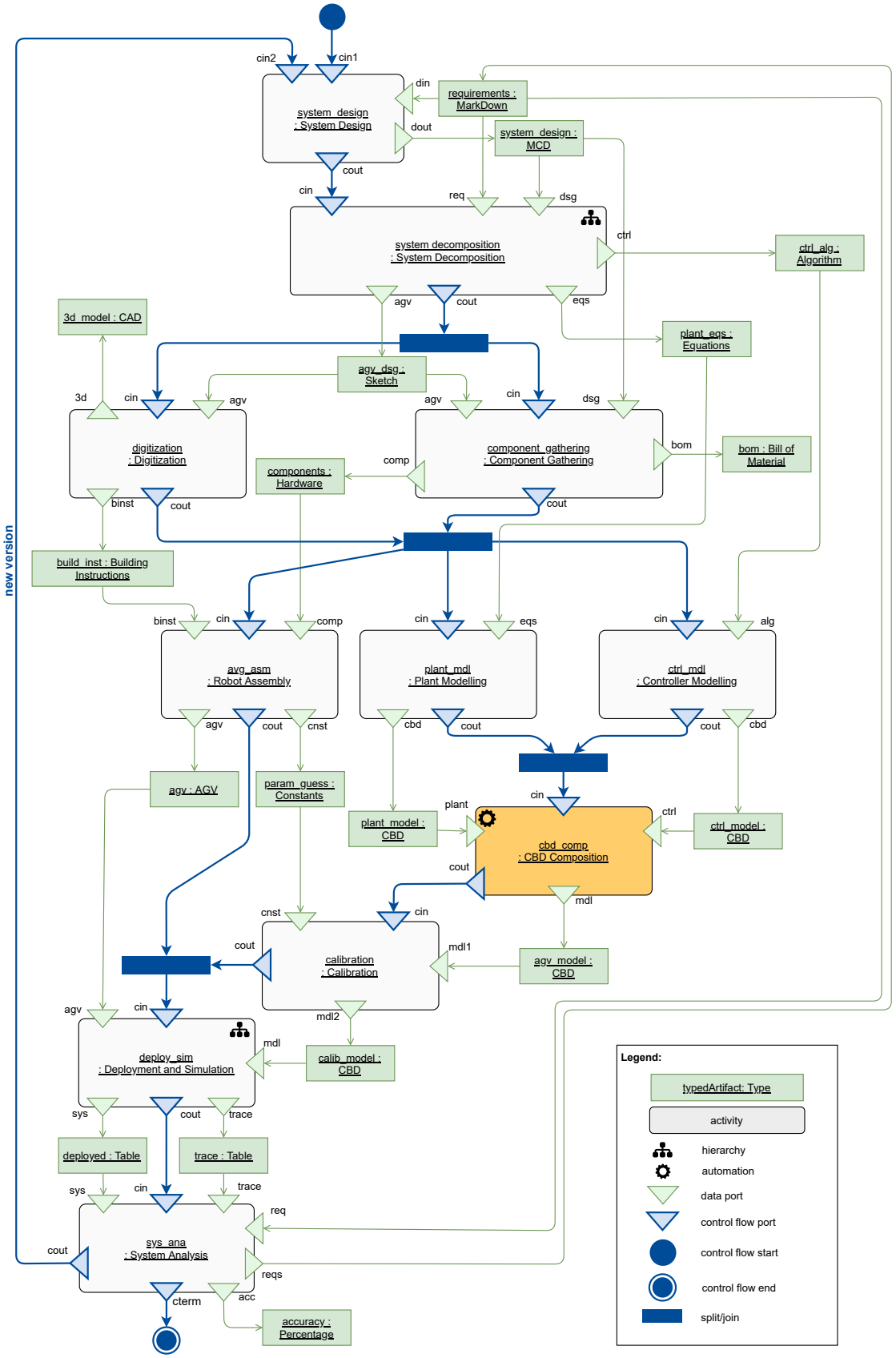
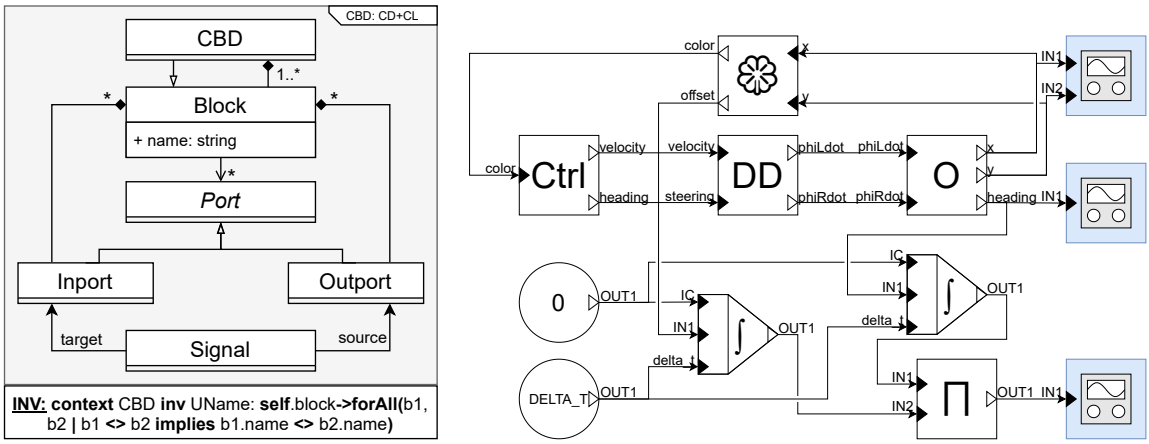


Figure 2: PM of the full AGV system.

Note that each artifact and activity is identified with a name and a type (*i.e.,* given by a Linguistic Type Model, also known as a meta-model). By using introspection on this meta-model, the artifact's structure can be identified and navigated. Figure 3 shows a simplified Causal Block Diagram (CBD) (Gomes, Denil, and Vangheluwe 2020) meta-model on the left and an example CBD instance model on the right. This instance is artifact agv_model: CBD. Note the (OCL) constraints in the Class Diagram meta-model specifying that all Block instances must have unique names. The CBD model instance *conforms* to the description given in the meta-model. Such instances are also referred to as artifacts, for example in the PM.



(a) A simplified CBD meta-model.

(b) A CBD instance model.

Figure 3: A CBD linguistic type model and instance model.

Similarly, activities also conform to a type model, which takes the form of an *activity contract*. This is demonstrated in Figure 4. The green rectangles identify the types of artifacts that are expected to be consumed/produced. Note that similarly, the control flow inputs/outputs may also define specific activities that must precede/follow this activity.

4.2 Formalism Transformation Graph and Meta-Models

The *Formalism Transformation Graph* (FTG) is a hypergraph which summarizes the *relationships* between the formalisms/languages used during the creation of the system and the types of activities used to realize these relationships. Formalisms are denoted by rectangles, and activity contracts/transformations by rountangles. Figure 5 shows the full FTG for the AGV use-case. As can be seen, there is a clear relationship between the PM type names and the FTG transformations. Similarly, the artifact type names correspond to the formalisms. Note that in current FTGs, as the name indicates, the relationships are actually directed *transformations*. If general (bi-directional) relationships are included, the term Formalism Relationship Graph (FRG) would be more appropriate.

The FTG focuses on the formalism/language/linguistic type aspect of MPM. Note that the name in a node of the FTG refers to a meta-model (linguistic type model) MM. As an addition to our earlier FTG definition, we make explicit the need for these meta-models such that the syntax of the formalisms used is unambiguously defined. These meta-models also determine how to Create/Read/Update/Delete instances/artifacts. Note that navigation through instances is a special case of Read.

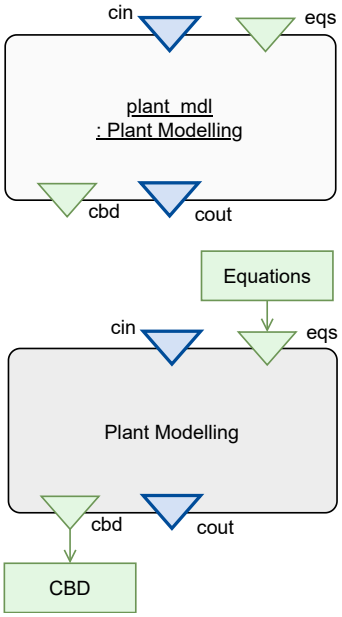


Figure 4: An activity (top) and its contract (bottom)

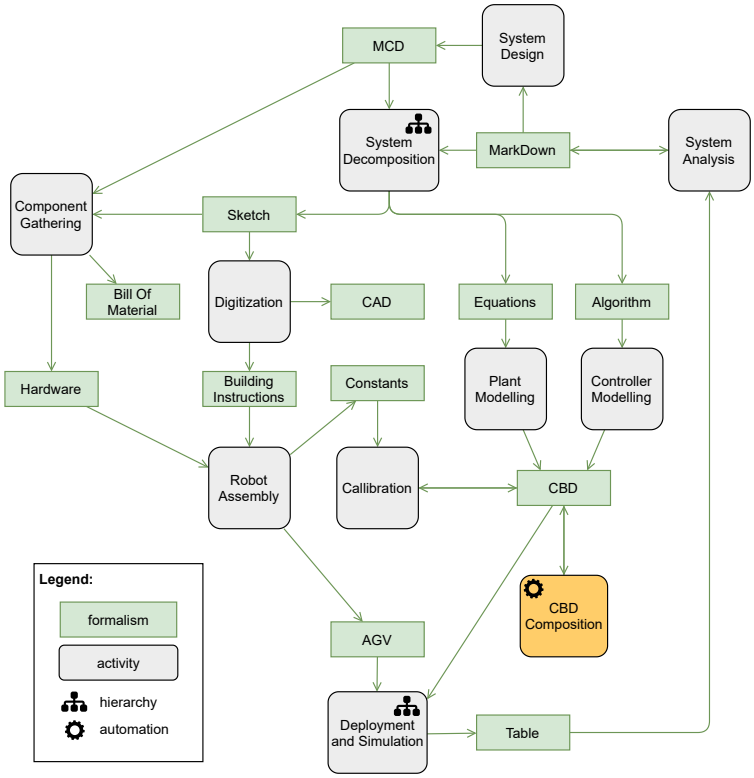


Figure 5: FTG of the full AGV system design.

4.3 Process Trace

Traceability, the ability to explore the provenance of artifacts, *i.e.,* which artifacts were used in their construction and through which activities, is crucial in SYE (Maro 2020). To support this, the FTG+PM is extended with the ability to capture an execution of a process model, in the form of a *Process Trace* (PT). A PT allows *horizontal traceability* (between MM, FTG, PM and PT) and *vertical traceability* (a slice from a historical log of executed activities and (versions of) generated artifacts). Horizontal traceability or typing/conformance is shown by using the same names and types in the MM, FTG, PM and PT. For didactic purposes, this may be marked explicitly using grey, dotted arrows (not shown in this paper). Vertical traceability appears when adding a PT to the FTG+PM. This PT contains *all* information about past executed activities and (versions of) generated artifacts. A PT can be traced back to the PM it is an enactment of. Furthermore, a PT model is append-only: added elements become immutable and as such provide an archival record that can be analyzed/mined and will always give the same analysis results, even if the the PM evolves (also in an append-only fashion). This satisfies our replicability/repeatability requirement. Part of the PT for the AGV example is shown on the right in Figure 6.

The rountangles identify the start and end (events) of specific activities. Timestamps denote when these occur. Upon starting a new activity, artifacts are used and when finishing it, new ones are produced. These are *versioned artifacts* (double bordered rectangle), and they are globally unique in the PT. When a workflow iteration performs the same activity multiple times, each of these activities is marked with a unique timestamp and generates a new, versioned artifact (that provide traceability via *ancestor* links (yellow, striped arrows in Figure 6)). In order to correctly reason about the full system process, all past artifact versions must remain accessible and be immutable.

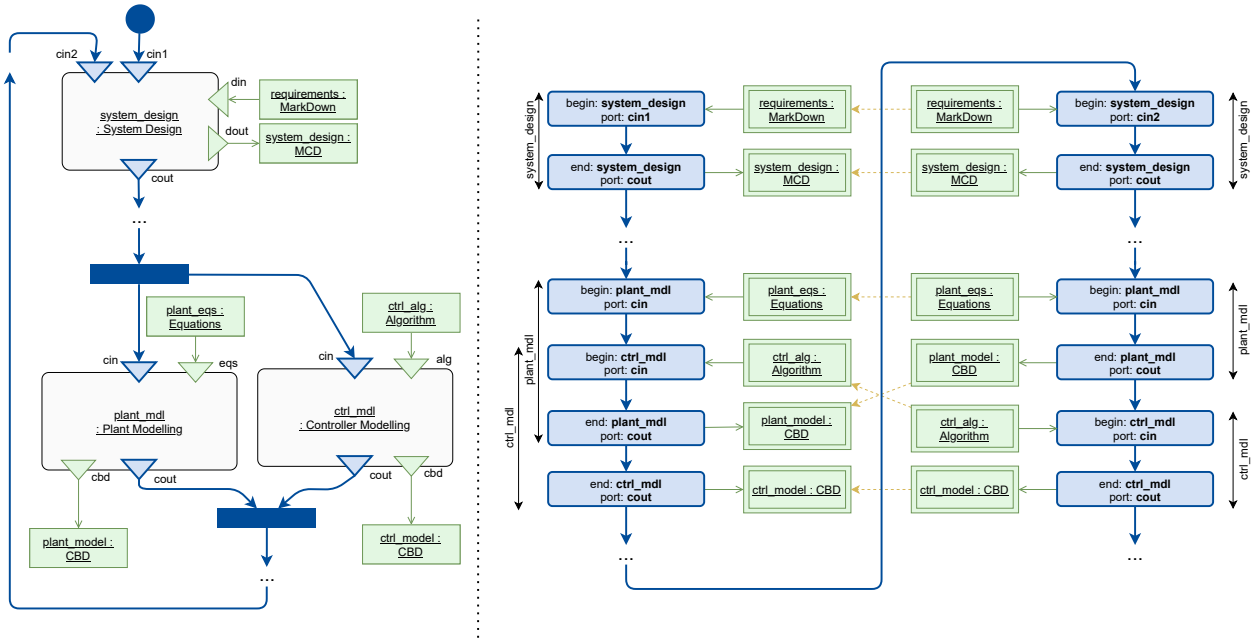


Figure 6: Part of the AGV system design PM (left) and its corresponding PT (right).

The left of Figure 6 shows the PM for the system_design: System Design activity followed by the concurrent plant_mdl: Plant Modelling and ctrl_mdl: Controller Modelling activities, with their input and output artifacts. It also contains the "*new version*" iteration (see also Figure 2 from which this is an excerpt). On the right, the PT for these activities is shown. For readability, the relationship between start and end of individual activities is denoted by vertical arrows. In the first iteration of the system design, the concurrent activities are interleaved (begin followed by begin followed by end followed by end events), but, as can be seen in a later iteration, they may also be sequential (begin followed by end followed by begin followed by end events). This is a consequence of resource allocation and planning. All artifacts in the later iteration have ancestors in the earlier iterations. This ancestry relationship is denoted by the horizontal dashed yellow arrows pointing left.

4.4 Storage / Services / Real-World Artifacts

Another new extension to the FTG+PM framework is the ability to point to stored files for meta-models, versioned artifacts, service versions for executed activities, and physical locations for real-world artifacts (such as a built robot). This will be referred to as (physical) Storage/Service/Real-World Artifacts (S/S/RWA). Explicitly storing this information next to the PT allows a clear, permanent introspection of all required components, such that the decisions made can be revisited and recalled later. These are also necessary in order to fully support traceability, repeatability and replicability.

Figure 7 visualizes the PT and S/S/RWA. An orange cylinder denotes a concrete realized entity. The icon inside specifies its kind (respectively: database/gears/cube icon for storage/service/real-world artifact). The text inside the container specifies the object/service that is stored/provided at an external location. The location is given along the arrow label.

Note that it actually does not matter how data is stored if one or more bi-directional *adapters* exist between activity and artifact types in the PM on the one hand and service, storage and real-world artifact types in the S/S/RWA on the other hand. For instance, a CBD model may be stored in a SIMULINK™ file. It

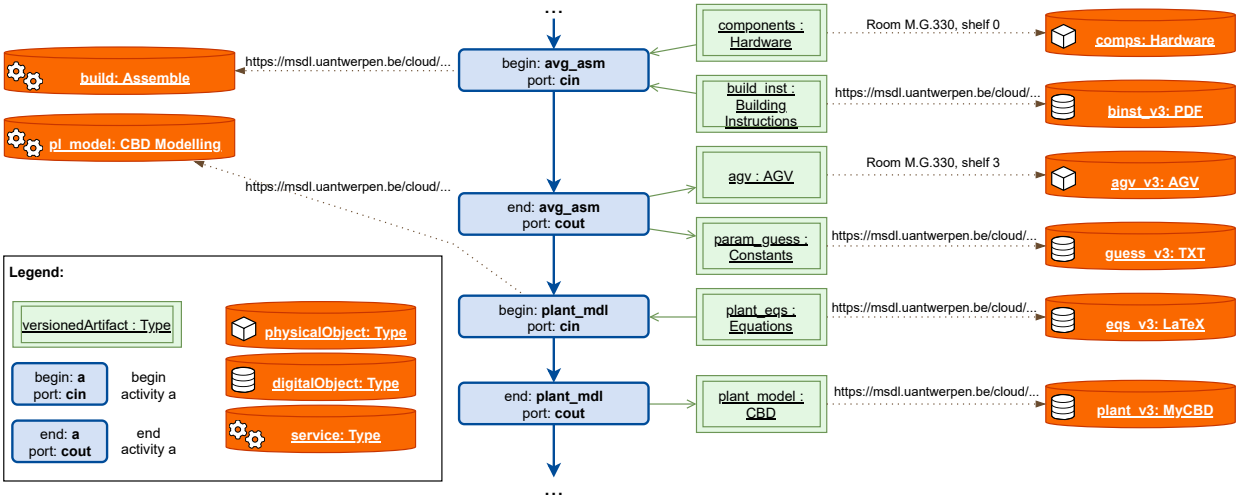


Figure 7: PT with storage annotation.

does not matter *how* it is stored, as an adapter will transform, when needed, from/to an artifact in the PT, properly typed by a MM. An ODBA framework such as *Ontop* (Calvanese et al. 2015) hides the physical structure of data sources such as federated (relational) databases and exposes (and adapts) the data using an *ontology* (through SPARQL queries (World Wide Web Consortium 2013)). Figure 8 shows how adapter *contracts* can be denoted. This way, it is also possible to have multiple adapters for a single versioned artifact. The rountangles define the adapter specification (*e.g.,* "*Table || CSV*" denotes that the adapter can change CSV objects into instances of the Table meta-model as well as the other way around). The orange cylinder notation is used to identify the kind of storage. This must match the actual storage annotation as was given in Figure 7.

Figure 8: Two adapter contracts for FTG types.

4.5 Properties of Interest

Qamar and Paredis (2012) define *properties* as "descriptors of an artifact". They are attributes that concern the artifact and are either described logically (*e.g.,* the AGV's wheels are nonholonomic) or numerically (*e.g.,* the AGV uses 6 batteries). These properties may be *computed* (*i.e.,* derived from other activities/artifacts) or *specified* (*i.e.,* defined by a user). The *properties of interest* are specific system properties that define the concerns of a specific stakeholder. In Figure 5, the sys_ana: System Analysis activity outputs accuracy: Percentage, which is a computed property within the context of the AGV use-case. Based on this value, the activity can decide whether to start the creation of a new version of the system, or to end the product life-cycle. A final new extension to the FTG+PM framework is therefore the possibility to explicitly describe properties of interest in a SYE workflow.

5 QUERYING

The previous sections set the stage for the main contribution of this paper. The use of a full MM+FTG+PM+PT+S/S/RWA (or FTG+PM++ for short) enables *querying* over all the information contained in it. Thanks to traceability between all the elements of the FTG+PM++, it is possible to find complex, useful patterns, which can be navigated using queries. Six different kinds of traceability can be identified, and are discussed below.

Traceability by linking experiments and system. Allows the identification of which experimentation activities led to artifacts such as simulator output. Conversely, which artifacts were produced by a given experiment. The complex workflows and architectures needed to fully describe experiments also need to be modelled explicitly (Denil et al. 2017).
For instance, in order to select all activities that have a trace: Table as a result (and are therefore experiments due to the contract/type of the experiment activity), the following query can be used: `SELECT * FROM ACTIVITIES AS ac WHERE trace:Table IN ac.results`.

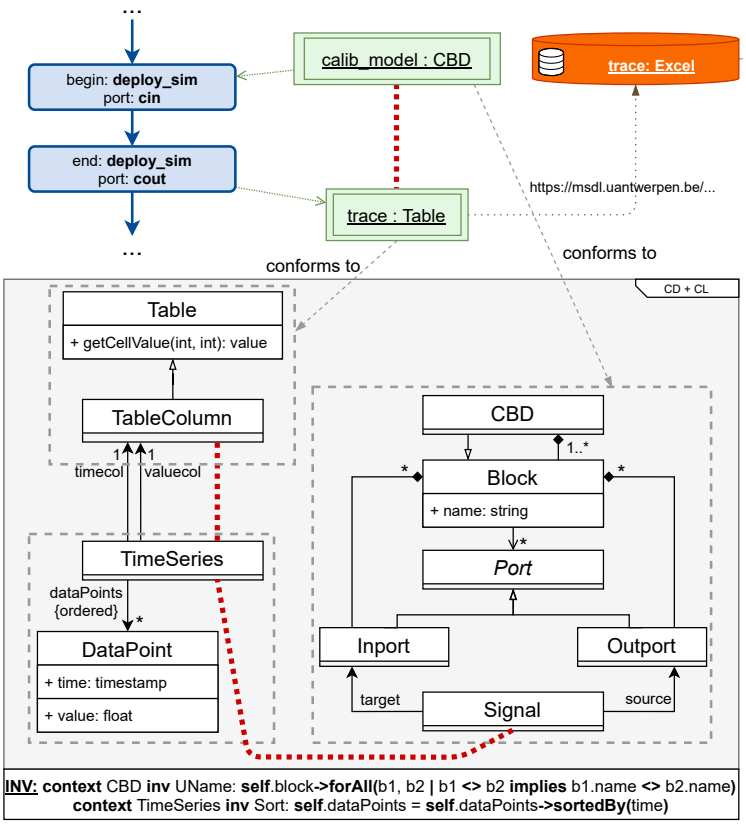
Traceability between instances and their (linguistic) types. Each (versioned) artifact has an explicit type and therefore a precisely defined meta-model in the FTG+PM++. It is hence possible navigate a model, based on its structure. For instance, agv_model is of type CBD, thus all Blocks in the CBD can be accessed. From these Blocks, it is possible to access their ports, etc.
To obtain this metamodel, the following query might be used: `SELECT m.metamodel FROM ARTIFACT agv_model AS m`.

Traceability across artifact versions, given context through a workflow model. For iterative workflows, the PT keeps growing with each iteration. By explicitly ensuring all versions of each artifact are retained, it is possible to navigate to past versions and ask questions about them. For instance, obtaining past controller models of the AGV in order to identify the flaws/strengths of each controller version.
A past version of the ctrl_model might be obtained via querying `SELECT m.versions FROM ARTIFACT ctrl_model AS m`.

Fine-grained traceability between artifact elements. Artifacts may have useful relationships between them, which can only be identified at a low, artifact element, level. Keeping track of these relationships ensures that the artifact element semantics can be meaningfully interpreted when asking questions. Figure 9 illustrates this through the relationship between an Excel column and the corresponding CBD signal from which the column data was generated.
The logic in the figure can be written as `SELECT * FROM ARTIFACTS AS art WHERE art.metamodel CONFORMS TO trace:"time"`.

Traceability based on properties of interest. As discussed in section 4.5, certain properties may be computed/derived from other properties in a SYE workflow. By making these relationships explicit, it is possible to identify the origin/consequences of properties.
Assuming all rigid bodies used are marked as having the "*rigid*" property, they can be obtained using `SELECT * FROM ARTIFACTS AS art WHERE "rigid" IN art.properties`.

Traceability between artifacts on different levels of detail. To evaluate a given property of interest, multiple models may be used. Some may have less detail (possibly in a different formalism) than others, but, when the property of interest is evaluated, the same result is obtained. If this is the case, the less detailed model is called an *abstraction* of the more detailed one. Keeping track of properties of interest, systems, and the different models that satisfy these properties, and under which conditions if useful for model re-use. It also enables reasoning about accuracy/performance trade-offs.



	A	B	C	D
1	time	x	y	heading
596	118.8	0.215843	-0.29999	0.677788
597	119	0.216126	-0.29976	1.302247
598	119.2	0.216222	-0.29941	1.599066
599	119.4	0.216212	-0.29904	1.46084
600	119.6	0.216252	-0.29868	1.124072
601	119.8	0.216409	-0.29836	0.877111
602	120	0.216641	-0.29808	0.865045
603	120.2	0.216877	-0.2978	1.027622
604	120.4	0.217065	-0.29749	1.198563
605	120.6	0.217197	-0.29715	1.255207
606	120.8	0.21731	-0.2968	1.190118
607	121	0.217445	-0.29647	1.084936
608	121.2	0.217614	-0.29615	1.025846
609	121.4	0.217803	-0.29583	1.041682
610	121.6	0.217986	-0.29552	1.099589
611	121.8	0.218151	-0.2952	1.146234
612	122	0.218301	-0.29487	1.151624
613	122.2	0.218449	-0.29453	1.123763
614	122.4	0.218606	-0.29421	1.092062
615	122.6	0.218773	-0.29388	1.080216
616	122.8	0.218944	-0.29356	1.090973

Figure 9: Fine-grained traceability between an Excel file column and a CBD model signal.

6 CONCLUSION AND FUTURE WORK

This paper has shown how the FTG+PM can be used to aid Systems Engineering (SYE). We extended the original framework with meta-models (MM), tracing information (PT) and Service/Storage/Real-World Artifact (S/S/RWA). By keeping track of all intermediate artifacts, including all past (historical) versions thereof, the full process can be queried in order to extract useful information about the system. Furthermore, the explicit nature of the FTG+PM++ automatically enables replicability of a system. The creation of the full MM+FTG+PM+PT+S/S/RWA is therefore required in the context of SYE and its future, as outlined in (INCOSE 2021). Note that the FTG+PM++ provides a framework in which architecture and view de-/composition, as well as the collaboration between multiple stakeholders can be described. This explicit description is future work.

We have built a prototype implementation of our framework using https://diagrams.net (formerly known as https://draw.io) as a visual editing front-end and our Modelverse (Van Tendeloo and Vangheluwe 2017) as a model management back-end. All images in this paper were produced using this prototype.

The FTG+PM++ can be used for a plethora of additional research, including (but not limited to) process mining, safety analysis, system security, proof-of-validatity ... Future work includes looking into these methods and positioning the FTG+PM++ into these practices.

Some aspects such as activity resource (human or computer) allocation and planning (how activities will be scheduled) are still missing from the FTG+PM++ framework. Furthermore, we plan to validate the extension to the FTG framework presented in this paper by applying it to multiple use-cases. We expect certain re-usable patterns to appear, possibly after automated process mining (van der Aalst 2016).

The FTG+PM++ formalism will be used to prescribe and trace all interactions with an evolving virtual knowledge graph (currently, the *ModelVerse*), to eventually obtain a fully self-describing environment for multi-paradigm modelling.

ACKNOWLEDGMENTS

This research was partially supported by Flanders Make, the strategic research center for the manufacturing industry. The authors thank the anonymous reviewers for their constructive comments.

REFERENCES

Amrani, M., D. Blouin, R. Heinrich, A. Rensink, H. Vangheluwe, and A. Wortmann. 2021. "Multi-Paradigm Modelling For Cyber–Physical Systems: A Descriptive Framework". *Software and Systems Modeling* vol. 20 (3), pp. 611–639.

Bajaj, M., D. Zwemer, R. Yntema, A. Phung, A. Kumar, A. Dwivedi, and M. Waikar. 2016. "MBSE++ – Foundations for Extended Model-Based Systems Engineering Across System Lifecycle". In *INCOSE International Symposium*, Volume 26, pp. 2429–2445. Edinburgh, UK, Wiley Online Library.

Brambilla, M., and P. Fraternali. 2014. *Interaction Flow Modeling Language: Model-Driven UI Engineering Of Web And Mobile Apps With IFML*. Morgan Kaufmann.

Calvanese, D., B. Cogrel, E. Güzel Kalaycı, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Muro, and G. Xiao. 2015, 01. "OBDA With The Ontop Framework". In *Proceedings of the 23rd. Italian Symposium on Database Systems*. Gaeta, Italy.

Carreira, P., V. Amaral, and H. Vangheluwe. (Eds.) 2020. *Foundations of Multi-Paradigm Modelling for Cyber-Physical Systems*. Springer International Publishing.

Denil, J., S. Klikovits, P. J. Mosterman, A. Vallecillo, and H. Vangheluwe. 2017, April. "The Experiment Model and Validity Frame in M&S". In *Proceedings of the Symposium on Theory of Modeling and Simulation (TMS/DEVS)*, TMS/DEVS '17, part of the Spring Simulation Multi-Conference, pp. 1085 – 1096. Virginia Beach, VA, USA, SCS.

Dumas, M., and A. H. M. Ter Hofstede. 2001. "UML Activity Diagrams As A Workflow Specification Language". In *International conference on the unified modeling language*, pp. 76–90. Springer.

European Computer Manufacturers Association et al. 1966. "Standard ECMA-4: Flow Charts". *European Computer Manufacturers Association*.

Favre, J.-M., and T. Nguyen. 2005. "Towards A Megamodel To Model Software Evolution Through Transformations". *Electronic Notes in Theoretical Computer Science* vol. 127 (3), pp. 59–74.

Golra, F. R., A. Beugnard, F. Dagnat, S. Guerin, and C. Guychard. 2016. "Addressing Modularity For Heterogeneous Multi-Model Systems Using Model Federation". In *Companion Proceedings of the 15th International Conference on Modularity*, pp. 206–211. Malaga, Spain.

Gomes, C., J. Denil, and H. Vangheluwe. 2020. *Causal-Block Diagrams: A Family of Languages for Causal Modelling of Cyber-Physical Systems*, Chapter 4, pp. pp. 97–125. Springer International Publishing.

Grieves, M. 2006. "Product Lifecycle Management". *Nova Iorque, McGraw-Hill*.

Guérin, S., G. Polet, C. Silva, J. Champeau, J.-C. Bach, S. Martínez, F. Dagnat, and A. Beugnard. 2021. "PAMELA: An Annotation-Based Java Modeling Framework". *Science of Computer Programming* vol. 210, pp. 102668.

Hassane, O., S. Mustafiz, F. Khendek, and M. Toeroe. 2019. "MAPLE-T: A Tool for Process Enactment With Traceability Support". In *Proceedings of the 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS ' 19)*, pp. 759–763. Munich, Germany, IEEE.

Hebig, R., A. Seibel, and H. Giese. 2012. "On The Unification Of Megamodels". *Electronic Communications of the EASST* vol. 42.

Hong, N. P. C. 2021. "Reproducibility Badging And Definitions: A Recommended Practice Of The National Information Standards Organization".

INCOSE 2021. "Systems Engineering Vision 2035". Technical report, INCOSE.

Karaduman, B., S. Mustafiz, and M. Challenger. 2021. "FTG+PM For The Model-Driven Development Of Wireless Sensor Network Based IoT Systems". In *24th ACM/IEEE International Conference On Model-Driven Engineering Languages And Systems Companion (MODELS-C 2021)*, pp. 308–318. Fukuoka, Japan, IEEE; Assoc Comp Machinery; ACM SIGSOFT; IEEE Comp Soc; IEEE Tech Council Software Engn.

Kassiakoff, A., W. N. Sweet, S. J. Seymour, and S. M. Biemer. 2010. *Systems Engineering Principles And Practice*. second ed. John Wiley & Sons, Inc.

Kritzinger, W., M. Karner, G. Traar, J. Henjes, and W. Sihn. 2018. "Digital Twin in Manufacturing: A Categorical Literature Review and Classification". *IFAC-PapersOnLine* vol. 51 (11), pp. 1016–1022.

Ma, J., G. Wang, J. Lu, H. Vangheluwe, D. Kiritsis, and Y. Yan. 2022. "Systematic Literature Review Of MBSE Tool-Chains". *Applied Sciences* vol. 12 (7), pp. 3431/1 – 21.

Maro, S. 2020. *Improving Software Traceability Tools And Processes*. Ph. D. thesis, University of Gothenburg.

Meyers, B., and H. Vangheluwe. 2011. "A Framework For Evolution Of Modelling Languages". *Science of Computer Programming* vol. 76 (12), pp. 1223–1246.

Moody, D. 2009. "The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering". *IEEE transactions on software engineering* vol. 35 (6), pp. 756 – 779.

Mosterman, P. J., and H. Vangheluwe. 2002. "Computer Automated Multi-Paradigm Modeling". *ACM Transactions on Modeling and Computer Simulation (TOMACS)* vol. 12 (4), pp. 1–7. Special Issue Guest Editorial.

Mosterman, P. J., and H. Vangheluwe. 2004, September. "Computer Automated Multi-Paradigm Modeling: An Introduction". *Simulation* vol. 80 (9), pp. 433–450.

Mustafiz, S., J. Denil, L. Lúcio, and H. Vangheluwe. 2012. "The FTG+PM Framework For Multi-Paradigm Modelling: An Automotive Case Study". In *Proceedings of the 6th International Workshop on Multi-Paradigm Modeling*, pp. 13–18. Munich, Germany, ACM.

Object Management Group 2017. "OMG® Unified Modeling Language® (OMG UML®) 2.5.1".

OpenMBEE Organisation 2011. "OpenMBEE Homepage". Online: https://www.openmbee.org/. Accessed: 28th of June 2022.

Paredis, R., C. Gomes, and H. Vangheluwe. 2021. "Towards A Family Of Digital Model/Shadow/Twin Workflows And Architectures". In *Proceedings of the 2nd International Conference on Innovative Intelligent Industrial Production and Logistics (IN4PL 2021)*, pp. 174–182. online, SCITEPRESS – Science and Technology Publications, Lda.

Paredis, R., and H. Vangheluwe. 2021. "Exploring A Digital Shadow Design Workflow By Means Of A Line Following Robot Use-Case". In *Proceedings of the 2021 Annual Modeling and Simulation Conference (ANNSIM)*. Fairfax, VA, USA.

Plesser, H. E. 2018. "Reproducibility Vs. Replicability: A Brief History Of A Confused Terminology". *Frontiers in neuroinformatics* vol. 11, pp. 76.

Qamar, A., and C. Paredis. 2012, 08. "Dependency Modeling And Model Management In Mechatronic Design". In *Proceedings of the ASME Design Engineering Technical Conference*, Volume 2. Chicago, IL, USA.

Russell, N., W. M. Van Der Aalst, and A. H. M. Ter Hofstede. 2016. *Workflow Patterns: The Definitive Guide*. MIT Press.

Salay, R., S. Kokaly, A. Di Sandro, and M. Chechik. 2015. "Enriching Megamodel Management With Collection-Based Operators". In *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pp. 236–245. Ottowa, Ontario, Canada, IEEE.

van der Aalst, W. 2016. *Process Mining: Data Science In Action*. Second ed. Springer.

Van Mierlo, S., Y. Van Tendeloo, I. Dávid, B. Meyers, A. Gebremichael, and H. Vangheluwe. 2018, April. "A Multi-Paradigm Approach For Modelling Service Interactions In Model-Driven Engineering Processes". In *Proceedings of the Spring Simulation Multiconference – International Symposium on Model-Driven Approaches for Simulation Engineering (Mod4Sim)*, edited by A. D'Ambrogio and U. Durak, pp. 565–576. Alexandria, VA, USA.

Van Tendeloo, Y., and H. Vangheluwe. 2017, December. "The Modelverse: A Tool For Multi-Paradigm Modelling And Simulation". In *Proceedings of the Winter Simulation Conference (WSC)*, WSC 2017, pp. 944 – 955. Las Vegas, NV, USA, IEEE.

Wagner, D., S. Kim, A. Jimenez, M. Elaasar, N. Rouquette, and S. Jenkins. 2020. "CAESAR Model-Based Approach To Harness Design". In *Proceedings of IEEE Aerospace Conference*. Big Sky, MT, USA.

White, S. A. 2004. "Introduction To BPMN". *IBM Cooperation* vol. 2.

World Wide Web Consortium 2013. "SPARQL 1.1 Overview".

AUTHOR BIOGRAPHIES

RANDY PAREDIS is a PhD student in the Modelling, Simulation and Design Lab (MSDL) at the University of Antwerp. He explores a generic architecture and framework for model-based design of Digital Twins. His email address is randy.paredis@uantwerpen.be.

JOERI EXELMANS is a Ph.D. student in the MSDL. His research interests are the engineering of hybrid languages, model versioning, and inconsistency management and traceability in complex engineering workflows. His email address is joeri.exelmans@uantwerpen.be.

HANS VANGHELUWE is a Professor and head of the MSDL. He develops modelling and simulation theory, methods, techniques and tools to increase system builders' productivity. His email address is hans.vangheluwe@uantwerpen.be.