A Multi-Paradigm Modeling Foundation for Collaborative Multi-view Model/System Development

István Dávid
Modelling, Simulation and Design Lab (MSDL)
University of Antwerp - Flanders Make
istvan.david@uantwerpen.be

Abstract

The complexity of current engineered systems has increased drastically over the last decades. Due to this complexity, these systems are typically developed in collaborative settings with stakeholders from different domains involved. A pertinent example is engineering of cyber-physical systems (CPS). Such collaborative endeavours are severely hindered by inconsistencies that arise due to semantic overlaps between different models. Additionally, since the involved domain-specific languages of stakeholders may be very disparate, inconsistencies often do not manifest on the linguistic level.

To cope with this problem, we propose an approach that enables better understanding how inconsistencies arise, evolve and how they should be managed. The core of our approach is a rich process modeling formalism that allows modeling multiple aspects of the development workflow in accordance with the guidelines of multi-paradigm modeling (MPM). We support our approach with an open-source prototype tool for designing engineering processes, defining inconsistency patterns and their respective management alternatives, and with the ability to optimize the original process for various optimality criteria, such as consistency and costs.

Keywords inconsistency management, process engineering, model-based design, cyber-physical systems

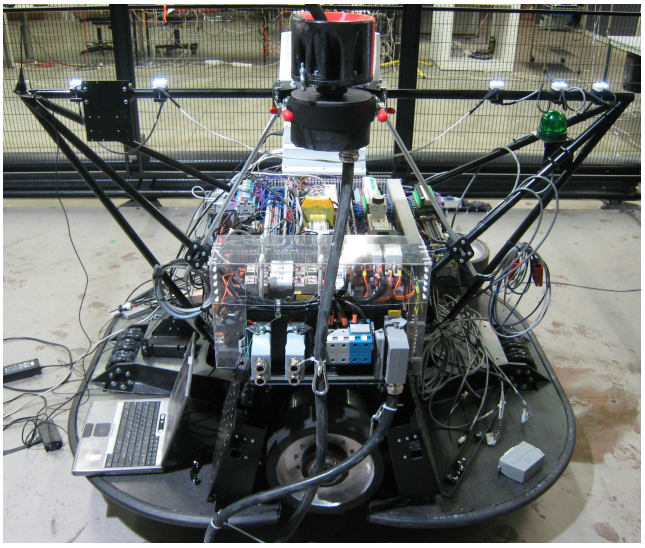

Figure 1: Automated guided vehicle (AGV), a pertinent example of truly heterogeneous, complex systems, result of an interplay between mechanical-, electrical-, control- and software engineering.

1. Problem and motivation

The complexity of current engineered systems has increased drastically over the last decades. A pertinent example are today's mechatronic and cyber-physical systems (CPS). These are characterized by heterogeneity, namely the complex interplay between physical, software, and network components (29).

Due to their complexity, these systems are no longer engineered by a single individual, but rather by the collaboration of experts. Such collaborative endeavors involve stakeholders from different domains, who bring their point of view on the system to be built, resulting in typical settings of multi-view and multi-paradigm modeling (MPM) (26), which proposes to tackle complexity by modeling and relating all aspects of the system – including development processes – explicitly, using the most appropriate formalisms, at the most appropriate levels of abstraction.

Semantic inconsistencies Collaborative modeling scenarios are vulnerable to model inconsistencies. This is a consequence of the multiple views on the same virtual product that give rise to outdated and incorrect data. *Overlaps in the semantic domain* of models have been identified as the primary reason of model inconsistencies by many authors (29; 22; 37). That is, properties of different models often turn out to be logically connected or sometimes even (nearly) the same (16; 20). Such a property can be, for example, the "safety" of the designed system, which in turn can be implied by the property of "stability" of a specific subsystem, meaning that the two properties are connected as they semantically overlap. Involving different engineering domains which typically feature disparate modeling formalisms further aggravates the problem.

Although the problem of inconsistencies is a well-studied area in software engineering (34), state-of-the-art techniques typically focus on syntactic inconsistencies (32; 17; 18). Since retaining semantic consistency is typically linked to simulation and model checking techniques which can often be resource demanding and time consuming, state-of-the-art techniques fail to efficiently address the consistency issues of semantic properties in the broader system engineering domain.

Managing (in)consistencies Finkelstein (15) hints that instead of just removing inconsistencies, one should *manage* them. This entails reasoning about the causes and sources of inconsistencies, their evolution, interaction and impact on the overall design. We argue that this can be best achieved by investigating inconsistencies in the context of (i) the *design process* of the virtual product, (ii) the *modeling languages* and transformations used in the process, and (iii) the ontological and linguistic *properties* of the virtual product that are manipulated during the design.

Consequently, the goal of inconsistency management is to transform inconsistent design processes into consistency preserving processes, and that, preferably by introducing (semi) automated consistency management tasks, instead of manual ones.

Tolerating inconsistencies Even though incremental techniques (37; 13; 19) offer better scalability compared to batch techniques in the case of syntactic inconsistencies, applying them on semantic cases results in frequent re-computations of properties in order to inspect the consistency of them. Inconsistencies are stateful entities that might occur, evolve and later potentially disappear as the natural consequence of the design workflow. This gives room for temporarily *tolerating* them (3), i.e. allowing inconsistencies to exist for a period of time, which promises lower resolution costs by (i) postponing resolution to a more appropriate phase of the design process; or in some cases even (ii) completely avoid resolution as specific inconsistencies get resolved on their own.

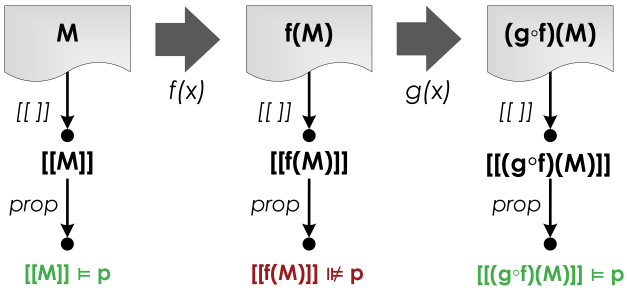

Figure 2: Lifecycle of an inconsistency. Model M is mapped to its semantic domain ($[\![\mathrm{M}]\!]$), where the satisfaction of property p can be checked ($[\![M]\!] \models p$). After the model evolves (*f(M)*), the property may be not satisfied, but after another evolution ($g \circ f(M)$), the property is again satisfied. Cases like this raise the need for temporal tolerance of inconsistencies.

Motivation The motivation of our work is the lack of a comprehensive, process-oriented approach to managing semantic inconsistencies with the ability to be flexible, i.e. tolerate inconsistencies in certain (temporal) cases. In this paper we present the foundations of such an approach, link it to the state-of-the-art and present future development directions.

The rest of the paper is structured as follows. Section 2 gives an overview on the background of this research and the related work. Section 3 briefly presents our approach. In Section 4 we discuss the current results of the approach. Finally, we conclude the paper by discussing our contributions in Section 5.

2. Background and related work

Inconsistency management Inconsistency management is a well-studied topic in the domains of software engineering, mechatronic design and cyber-physical systems, due to the typically multi-view and often multi-paradigm approach to system design. Persson et al (29) identify consistency between the various views of cyber-physical system design as one of the main challenges in design of such complex systems. This is due to relations between views, with respect to their semantic relations, process and operations which often overlap. Our technique embraces these ideas and addresses the problem of inconsistencies by explicitly modeling semantic properties and relating them to engineering processes.

Other approaches also acknowledge the role of semantic techniques in inconsistency management, and try to relate semantic concepts to the linguistic concepts of modeling. Hehenberger et al (21) organize structural design elements and their relations into a domain ontology to identify inconsistencies. A limited set of semantic properties are expressed with linguistic concepts which enables reasoning over semantic overlaps to a sufficient extent. Similarly, Chechik et al (6) introduce the notion of approximate properties: linguistic properties expressed as graph patterns which are accurate enough to appropriately approximate a semantic property. Approximate properties suitable to implement smart locking mechanisms in collaborative model-based design as they introduce a trade-off between the computational resources to obtain or check a property, and the accuracy of the results. As opposed to these, our approach makes semantic properties first-class artifacts and relates them to processes, instead of linguistic model elements, which enables management of a richer class of inconsistencies.

As opposed to the above techniques, inconsistency management in collaborative modeling is more frequently addressed on the linguistic level. Qamar et al (30) approach inconsistency management by making inter- and intra-model dependencies explicit. Dependencies are direct results of semantic overlaps and are used to notify stakeholders about possible inconsistencies when dependent properties change. Our approach introduces an indirection between models and properties by relating them to specific activities that during working over models also access properties with specific intents. Blanc et al (5) approach the detection of inconsistencies from a model operation based point of view, where models are stored as sequences of change events and inconsistencies are expressed in terms of CRUD operations. Our approach generalizes this approach by introducing intents that are analogous with model operations, but they express change operations in terms of activities and properties. Egyed et al (13) investigates the impact of single inconsistencies on the whole system by introducing the notion of change impact based scopes. Scopes are used to carry out resolution steps on the required regions of the models and thus enhancing the efficiency of the inconsistency management framework. We carry out a similar scope detection and management on the property model of our approach. Specific technical challenges of collaborative modeling have been addressed by state-of-the-art techniques, such as (9) for comparing and merging models and EMFStore (11) for model persistence. These techniques can serve as an implementational basis for improving our tool.

Process engineering Modeling, analyzing and optimizing processes has been a topic of interest in project management. The resource-constrained project scheduling problem (RCPSP) (2) consists of finding a schedule of minimal duration by assigning a start time to each activity such that the precedence relations and the resource availabilities are respected. More formally, the RCPSP is a combinatorial optimization problem with potentially multiple dimensions of optimality (e.g. optimizing for material costs and duration as well). The problem has been well-researched and multiple solution techniques exist, but this construction lacks the notion of formalism and corresponding models being manipulated during the process.

BPMN2.0 (27) has been widely used for modeling and executing business processes. It enables high-level modeling to support stakeholders from the business domain. Its syntax is, therefore, simple to be used by a non-expert, especially when compared to the formalism presented in this paper.

Inconsistency tolerance Balzer et al (3) introduces the notion of temporal tolerance by deconstructing inconsistency rules to two derived rules, the appearance and disappearance rule which span a temporal interval of the model(s) being in an inconsistent state, hence making inconsistencies stateful entities. By allowing further engineering activities to be executed during the inconsistent interval, the better parallelization of the design workflow can be

achieved and ultimately, these may lead to the inconsistencies to be resolved without interrupting the design process for further reconciliation. As a limitation, the technique only deals with the most simplistic version of temporal consistency relations, in which a pair subsequent operations form an identity transformation. In practice, more complex structures of operations have to be supported. Easterbrook et al (10) propose a framework for temporal inconsistency tolerance in the context of multi-view modeling. Tolerating inconsistencies decouples the viewpoints and introduces flexibility in the design process as deciding upon *when to resolve inconsistencies* is the responsibility of the owner of the view. The authors provide a formal approach for guiding the decision in form of pairs of pre- and post-conditions. The technique is, however, not explicit about the metric used for evaluating the divergence of views (and viewpoints) and consequently, it does not scale well for larger problems. The lack of a distance metric also makes it hard to assess the impact of unresolved inconsistencies and reason over their accumulation and evolution.

3. A process-oriented approach for inconsistency management

In this section, we give an overview on the foundations of our process-oriented inconsistency management approach.

3.1 Overview of the approach

Potential sources of inconsistencies are identified by considering characteristics of the process model. Management of inconsistencies is achieved by selecting the appropriate techniques from a catalogue of management patterns and applying them on the unmanaged process to achieve a managed one. Typical patterns include re-ordering activities of a process, ensuring property checks around inconsistency-prone regions and using design contracts. Since the same type of inconsistency may be managed via different management patterns, the selection of the most appropriate one should happen through quantified cost measures. The selection method is translated to a constraint solving and optimization problem which finds the best process alternative while managing every potential source of inconsistencies. The concept is shown in Figure 3.

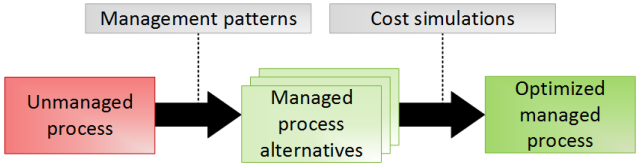

Figure 3: Conceptual overview of the approach.

3.2 A formalism for modeling processes

To model engineering processes with sufficient semantics for managing inconsistencies, we propose a formalism that augments the process with the syntactic and semantic *properties* that depict specificities of the engineered system.

We build our formalism on the FTG+PM (23) formalism, which enables the usage of process models ("PM") in conjunction with the model of languages and transformations (the formalism-transformation graph - "FTG") used throughout the process. As shown in Figure 4, languages and transformations serve as a type system to the processes: objects of the process are instances of languages of the FTG; and activities of the process realize transformations.

We extend the above formalism with the following aspects.

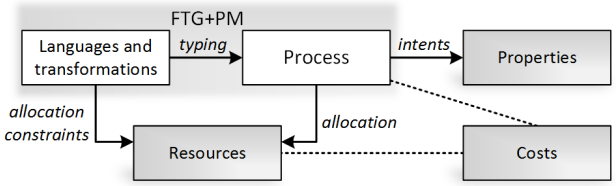

Figure 4: The main parts of the process formalism.

Properties These serve as the foundational basis for reasoning about semantic inconsistencies. The property model, therefore is used to *guarantee a managed process*, but not for optimizing it. We assume activities of an engineering process have a meaningful purpose of enhancing the system. This purpose is expressed as the *intent* of an activity with respect to a property or a set of properties.

Resources As opposed to properties, explicit modeling of resources serves for reasoning over how the process can be *restructured for optimality*. Activities are *allocated* to a certain set of resources, but the availability of the resources is typically limited. We also distinguish between automated and manual resources to further improve optimality by favoring automated resources as much as possible.

Costs Finally, to actually *quantify optimality*, at least one cost model is required. Our approach, however, enables using multiple cost models. Typical cost models include process execution time, queueing time, material costs. The cost model itself may be as simple as assigning a usage cost to each resource; but may be more complex by additionally assigning non-resource induced costs directly to activities.

Example. As an example, consider the engineering process of the AGV in Figure 1. Initially, components of the system, such as the battery, are selected based on approximations and domain expertise. The mass of the initially selected battery is considered during the Mechanical design phase to identify the mass constraints on other parts of the system. After the mechanical design phase, the electrical model is designed in details. This includes identifying the required capacity of the battery by Simulating the electrical model, in order to fulfill the autonomy requirement.

Inconsistencies may arise when the Battery capacity property is changed, because the Battery mass property depends on it: batteries with bigger capacity are typically heavier. As the capacity is changed, the mass becomes inconsistent with the capacity. Should the inconsistency get unnoticed, the engineered system will fail to meet the requirements.

Our tool provides a graphical modeling environment for this purpose, implemented using the Sirius framework (12). Figure 5 presents an excerpt from the process model of the example.

3.3 Process optimization

The process optimization problem is NP-hard in the strong sense. This can be shown by reducing the RCPSP to our problem, and the former one is a known NP-hard problem in the strong sense (2). The optimization, therefore cannot be approached with exhaustive techniques. We solve the problem by model transformation based multi-objective design space exploration (DSE) (1) as shown in Figure 6.

The exploration mechanism takes the original *unmanaged process* as an input and produces an *optimal managed process* as a series of *model transformations* applied on the original process. (The property and resource models are left intact as it reflects domain

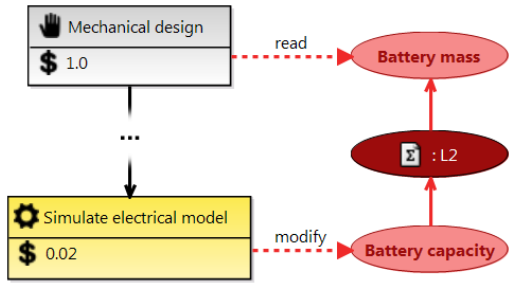

Figure 5: Excerpt from the process model of the case study.

Figure 6: Detailed overview of the optimization approach.

knowledge and as such, typically should not be changed because of a single process.) The exploration process is guided by hard *constraints* and optimality soft *objectives*.

The purpose of using model transformations is twofold. We use them to (i) augment the process with inconsistency management techniques and for (ii) optimizing it. An example for the latter one is parallelizing as many activities as possible. Of course, this will affect the applicable inconsistency management patterns, and therefore, the execution and evaluation of these transformations must be achieved in a coupled way. Transformation rules aiming to augment the process with inconsistency management techniques, are derived from the inconsistency patterns and management patterns. These transformations have an inconsistency patterns as left-hand side precondition, a management pattern as a rewrite rule and are triggered when the appropriate inconsistency pattern is detected.

Hard constraints and soft objectives are used to guide the exploration process and evaluate the solution candidates. We constrain the set of solutions to processes that are well-formed, have no unmanaged inconsistencies and a feasible allocation to the resources exists. As the objective function, the cost functions are used. Since the cost of non-linear processes (i.e. the ones featuring directed cyclic graphs) is not deterministic, simulations of various kinds can be used to obtain the cost, such as event queueing networks or discrete event simulations.

Patterns of inconsistency management To provide inconsistency management alternatives for transforming the unmanaged processes into managed one, a catalogue of such management patterns is used. We support our approach with four management patterns by default. This catalogue of patterns is, however, extensible in the prototype tooling.

Reordering and sequencing Reordering and sequencing aim to modify the control flow in order to avoid inconsistencies.

Given a sequential case of activities a_1, a_2, the *reordering* strategy would swap a_1 and a_2, to utilize that the appropriate order of read-modify intents does not lead to inconsistencies, as shown in Figure 5. In parallel cases, the *sequencing* strategy would try every possible order of the activities and eventually select the one that leads to the most optimal process.

Reordering and sequencing are easy-to-apply and inexpensive patterns as they do not require introducing additional management activities. Both patterns work well in simple cases; in more complex processes, however, both patterns tend to introduce other inconsistencies.

Property check Property checking is used to ensure no inconsistencies are introduced on specific sections of the process. A special activity a_{check} is added to the process that accesses the unmanaged properties with a $check$ intent. If the result of the check is satisfactory, the process continues with the subsequent activities; in the case of a failed check, however, the process would fall back to the latest point where the inconsistency is not yet present and facilitate a re-iteration loop.

The property check pattern is a typically expensive management pattern as it introduces directed loops in the design processes and therefore, makes processes inherently non-deterministic.

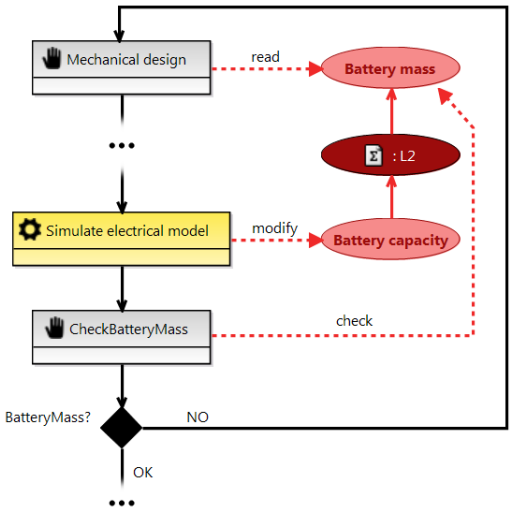

Figure 7: Managed alternative of the process in Figure 5, where an additional activity is introduced to check the consistency of properties.

Contracts In a contract-based approach (36), the stakeholders would agree on acceptance criteria of specific properties *before* executing specific design activities. A special activity $a_{contract}$ is added to the process to represent the contract negotiation phase. The activity accesses the unmanaged properties with a $contract$ intent. The contract is respected during the activities, thus providing means to avoid inconsistencies.

Assumptions A less rigorous approach to contracts is also possible by making an educated guess about the shared properties. In the parallel case, one of the parallel activities makes assumptions about the properties that will be modified by the other activity. However, these assumptions need to be checked once the process rejoins both branches. The benefit of the pattern is that only one of the branches has to be re-executed if the assumption proves to be invalid, i.e. an inconsistency may occur.

3.4 Tool interoperability

After rewriting the process into a managed and optimized one, its enactment and deployment can be supported by automatically generated artifacts, such as executable code snippets or configuration to various workflow engines (25; 31). By that, the interoperability of the engineering tools used throughout the process can be guaranteed as well. Since our formalism enables modeling how single engineering activities use various domain-specific formalisms and tools (see Figure 4), smart bridges and connectors to those tools can be generated. OSLC (33) has been widely used to enable standardized interfacing between various tools. As a consequence, our approach can extend the OSCL collaboration model with the explicit notion of the process and therefore, enable higher level of *orchestration*.

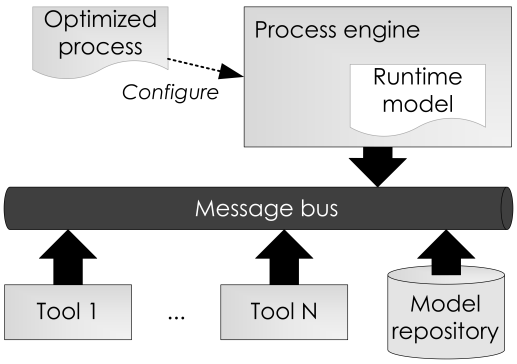

Figure 8: Tool interoperability supported by our approach. The *Process engine* takes the optimized process and orchestrates the execution of engineering activities.

Interactions between tools are typically automated activities of the process, but in some cases semi-automated activities requiring a human-in-the-loop may be more appropriate. Modeling interaction patterns will be supported by using statecharts (28) in conjunction with the process model.

3.5 Inconsistency tolerance

By *temporal inconsistency tolerance* (10), we mean postponing the resolution of an inconsistency to a later point in the process as the inconsistency may be resolved at that point or even disappear as the natural consequence of the process. Tolerating inconsistencies, even for a (temporal) period of time, can be seen as a compromise between the quality and the cost of the process.

In our previous work (8), we presented a formal underpinning for reasoning about inconsistency tolerance by explicitly quantifying how diverging single viewpoints(10) of the system are. The process modeling formalism described in this paper allows the explicit modeling of cost factors in conjunction with inconsistency patterns and thus, makes the quantification approach (i) more precise by relating it to the actual process, and (ii) provides the quantification algorithms with sound information regarding the semantic domains of the models.

4. Results

We support our approach with a prototype tool that allows (i) modeling processes and (ii) augmenting processes with inconsistency management patterns, while identifying the optimal managed process. The tool is built on top of the Eclipse platform and is available under the EPL licence from `https://github.com/david-istvan/icm`.

The tool offers an extensible catalogue of inconsistency patterns and their management alternatives. The extensible nature of the catalogues allows the framework to be tailored to the domain and the problem at hand. Inconsistency patterns are captured by a declarative graph query language (35) and the respective management patterns are defined by model transformation rules (4).

Since the primary target audience of our approach are multidisciplinary engineering teams, we support inter-domain communication by providing domain-specific views on the process, such as design structure matrices (DSM) (14) for mechanical engineers or Gantt charts for project managers.

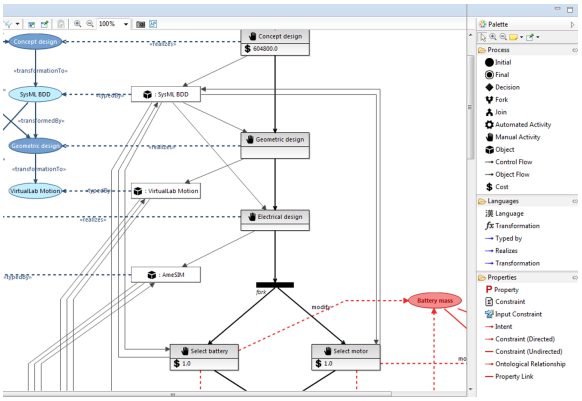

Figure 9: Process modeling in the prototype tooling.

We validated our approach on a case study of an autonomous guided vehicle (AGV). In our experiments we used two types of inconsistencies characteristic to the engineering process which develops the AGV; and four types of inconsistency management techniques were used. After modeling the original process, the goal was to come up with an optimized fully managed process, i.e. one without inconsistencies and with minimal costs. To evaluate the optimality, we used the event queueing network (EQN) formalism of the SimEvents (24) framework. The real challenge of applying inconsistency management patterns in an orchestrated way, so that their application does not give rise to new unmanaged inconsistencies, is tackled by using a heuristic or exhaustive search through the state space. Applying our approach to the whole process of the case study resulted in a *fully managed process* with reasonable increase in costs. In our simulations, we measured up to 10% cost reduction while fully managing the process with two types of inconsistencies.

5. Discussion

In this paper we outlined an ongoing research on managing inconsistencies in the context of engineering complex heterogeneous systems. This research develops the foundations of multi-paradigm modeling (MPM) with a focus on the collaborative, multi-view aspects of model/system development and the resulting consistency issues; and that in the context of complex engineered systems such as mechatronic and cyber-physical systems. The main contributions of this work are the following.

A A formalism that enables modeling the process in conjunction with (i) linguistic and semantic properties, (ii) the formalisms used within the project, (iii) resources the process is executed upon and (iv) cost factors. This rich semantics allows reasoning about trade-offs between the various aspects, most notably compromising quality for costs, i.e. tolerating inconsistencies if

the process costs are more acceptable without managing them. This also entails temporal tolerance of inconsistencies.

B Our approach enables expressing tacit domain knowledge explicitly and thus making it reusable across different processes (projects), at least partially, which is a typical concern in companies on CMMI levels 3 and above. (7) In order to enhance the reusing of domain knowledge, techniques of ontological reasoning will be investigated.

C The prototype tooling enables modeling, analyzing and optimizing processes. The prototype is built on top of the Eclipse platform. An extensible catalogue of inconsistency patterns and management patterns allows customizing the optimization process. The approach has been evaluated over a case study of a real mechatronic system using our prototype tool, using simulations. In the near future, we also plan to evaluate the approach in real engineering settings.

Acknowledgements

This work has been partially carried out within the MBSE4Mechatronics project (grant nr. 130013) of the Flanders Innovation & Entrepreneurship agency (VLAIO). This research was partially supported by Flanders Make vzw.

The authors wish to thank the following colleagues their help and valuable insights: Hans Vangheluwe, Joachim Denil, Klaas Gadeyne, Kristof Berx, Ken Vanherpen, András Szabolcs Nagy, Eugene Syriani, Antonio Cicchetti and Dominique Blouin.

References

[1] H. Abdeen, D. Varró, H. Sahraoui, A. S. Nagy, C. Debreceni, Á. Hegedüs, and Á. Horváth. Multi-objective optimization in rule-based design space exploration. In *Proceedings of the 29th ACM/IEEE Int. Conf. on Automated software engineering*, pages 289–300. ACM, 2014.

[2] C. Artigues, S. Demassey, and E. Neron. *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications*. ISTE, 2007.

[3] R. Balzer. Tolerating inconsistency. In *13th International Conference on Software Engineering*, pages 158–165, 1991.

[4] G. Bergmann, I. Dávid, Á. Hegedüs, Á. Horváth, I. Ráth, Z. Ujhelyi, and D. Varró. VIATRA 3: A Reactive Model Transformation Platform. In *Theory and Practice of Model Transformations*, pages 101–110. Springer, 2015.

[5] X. Blanc, I. Mounier, A. Mougenot, and T. Mens. Detecting model inconsistency through operation-based model construction. In *Software Engineering, 2008. ICSE '08. ACM/IEEE 30th Int. Conf. on*, pages 511–520, May 2008.

[6] M. Chechik, F. Dalpiaz, C. Debreceni, J. Horkoff, I. Ráth, R. Salay, and D. Varró. Property-Based Methods for Collaborative Model Development. In *Proc. of 3rd Int. Workshop on The Globalization of Modeling Languages (GEMOC 2015)*, 2015.

[7] CMMI Product Team. CMMI for Development, Version 1.3, Tech. Rep. CMU/SEI-2010-TR-033, 2010.

[8] I. Dávid, E. Syriani, C. Verbrugge, D. Buchs, D. Blouin, A. Cicchetti, and K. Vanherpen. Towards inconsistency tolerance by quantification of semantic inconsistencies. *1st International Workshop on Collaborative Modelling in MDE*, 2016.

[9] C. Debreceni, I. Ráth, D. Varró, X. De Carlos, X. Mendialdua, and S. Trujillo. Automated Model Merge by Design Space Exploration. In *19th Int. Conf. on Fundamental Approaches to Software Engineering*, 2016.

[10] S. Easterbrook, A. Finkelstein, J. Kramer, and B. Nuseibeh. Coordinating distributed viewpoints: the anatomy of a consistency check. *Concurrent Engineering*, 2(3):209–222, 1994.

[11] Eclipse Foundation. EMFStore Website. `http://eclipse.org/emfstore/`. Acc: 2016-07-19.

[12] Eclipse Foundation. Sirius Website. `https://eclipse.org/sirius/`. Accessed: 2016-07-19.

[13] A. Egyed. Automatically detecting and tracking inconsistencies in software design models. *Software Engineering, IEEE Trans. on*, 37(2):188–204, 2011.

[14] S. D. Eppinger and T. R. Browning. *Design structure matrix methods and applications*. MIT press, 2012.

[15] A. Finkelstein. A foolish consistency: Technical challenges in consistency management. In *Database and Expert Systems Applications*, volume 1873 of *LNCS*, pages 1–5. Springer, 2000.

[16] A. C. W. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. Inconsistency handling in multiperspective specifications. *IEEE Transactions on Software Engineering*, 20(8):569–578, Aug 1994.

[17] R. France and B. Rumpe. Model-driven development of complex software: A research roadmap. In L. Briand and A. L. Wolf, editors, *Future of Software Engineering*, pages 37–54, Minneapolis, may 2007. IEEE Computer Society.

[18] J. Gausemeier, W. Schäfer, J. Greenyer, S. Kahl, S. Pook, and J. Rieke. Management of cross-domain model consistency during the development of advanced mechatronic systems. In *DS 58-6: Proceedings of ICED 09, the 17th International Conference on Engineering Design, Vol. 6, Design Methods and Tools (pt. 2), Palo Alto, CA, USA, 24.-27.08.2009*, pages 1–12, 2009.

[19] H. Giese and S. Hildebrandt. Incremental model synchronization for multiple updates. In *Proceedings of the Third International Workshop on Graph and Model Transformations*, GRaMoT '08, pages 1–8, New York, NY, USA, 2008. ACM.

[20] J. Grundy, J. Hosking, and W. B. Mugridge. Inconsistency management for multiple-view software development environments. *IEEE Transactions on Software Engineering*, 24(11):960–981, Nov 1998.

[21] P. Hehenberger, A. Egyed, and K. Zeman. Consistency Checking of Mechatronic Design Models. In *30th Computers and Information in Engineering Conf.*, volume 3, pages 1141–1148. ASME, 2010.

[22] S. J. Herzig, A. Qamar, and C. J. Paredis. An approach to identifying inconsistencies in model-based systems engineering. *Procedia Computer Science*, 28:354–362, 2014.

[23] L. Lúcio, S. Mustafiz, J. Denil, H. Vangheluwe, and M. Jukss. FTG+PM: An Integrated Framework for Investigating Model Transformation Chains. In *SDL 2013: Model-Driven Dependability Engineering*, volume 7916 of *LNCS*, pages 182–202. Springer, 2013.

[24] MathWorks Inc. SimEvents Website. mathworks.com/ products/simevents. Acc.: 2016-08-02.

[25] Metaversant Group, Inc. Activiti Website. http://activiti.org/. Acc.: 2016-08-02.

[26] P. J. Mosterman and H. Vangheluwe. Computer automated multiparadigm modeling: An introduction. *Simulation*, 80(9):433–450, 2004.

[27] Object Management Group. BPMN2.0 Specification. http://www.omg.org/spec/BPMN/2.0/. Acc.: 2016-08-02.

[28] J. J. Odell, H. V. D. Parunak, and B. Bauer. Representing agent interaction protocols in uml. In *Agent-oriented software engineering*, pages 121–140. Springer, 2001.

[29] M. Persson, M. Törngren, A. Qamar, J. Westman, M. Biehl, S. Tripakis, H. Vangheluwe, and J. Denil. A characterization of integrated multi-view modeling in the context of embedded and cyber-physical systems. In *Proceedings of the International Conference on Embedded Software*, 2013.

[30] A. Qamar, C. J. Paredis, J. Wikander, and C. During. Dependency modeling and model management in mechatronic design. *Journal of Computing and Information Science in Engineering*, 12(4), 2012.

[31] Red Hat Software. jBPM Website. http://www.jbpm.org/. Acc.: 2016-08-02.

[32] J. Reineke and S. Tripakis. Basic problems in multi-view modeling. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 8413 of *LNCS*, pages 217–232. Springer, 2014.

[33] M. Saadatmand and A. Bucaioni. OSLC Tool Integration and Systems Engineering – The Relationship between the Two Worlds. *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 93–101, Aug. 2014.

[34] G. Spanoudakis and A. Zisman. Inconsistency management in software engineering: Survey and open research issues. In *in Handbook of Software Engineering and Knowledge Engineering*, pages 329–380. World Scientific, 2001.

[35] Z. Ujhelyi, G. Bergmann, Á. Hegedüs, Á. Horváth, B. Izsó, I. Ráth, Z. Szatmári, and D. Varró. EMF-IncQuery: An integrated development environment for live model queries. *Science of Computer Programming*, 98, Part 1:80 – 99, 2015.

[36] K. Vanherpen, J. Denil, I. Dávid, P. De Meulenaere, P. Mosterman, M. Törngren, A. Qamar, and H. Vangheluwe. Ontological Reasoning for Consistency in the Design of Cyber-Physical Systems. In *CPSWeek workshop proceedings*, 2016.

[37] R. Wagner, H. Giese, and U. Nickel. A plug-in for flexible and incremental consistency management. In *Proc. of the International Conference on the Unified Modeling Language*, page 93, 2003.