Towards Inconsistency Management by Process-Oriented Dependency Modeling

István Dávid[1], Joachim Denil[1], Hans Vangheluwe[1,2]

[1] Modeling, Simulation and Design Lab
University of Antwerp
Middelheimlaan 1, 2020 Antwerp, Belgium
{istvan.david, joachim.denil, hans.vangheluwe}@uantwerpen.be
[2] McGill University, Montreál, Canada

Abstract. Multi-paradigm modeling settings inherently and frequently cause inconsistencies between models in different languages involved in the design. The proper management of inconsistencies starts with choosing the appropriate formalisms to characterize inconsistent situations. In this paper, we introduce a formalism, which allows reasoning about inconsistencies in the context of the design process. We support our ideas by examples from a case study on the design of an unmanned aerial vehicle.

Keywords: inconsistency management, process modeling, mechatronic design

1 Introduction

The design and evolution of mechatronic and cyber-physical systems involves multiple stakeholders with different views on the system as a whole. Because of overlapping elements and properties between the various views and models, there is a need to manage the dependencies and possible inconsistencies between these models [1].

The process of inconsistency management (ICM) breaks down into three main activities. First, overlap between different models is *characterized*; based on this information, inconsistencies can be *detected* and consequently *resolved*. Different authors approach this process with various granularity and see the role of characterization (often referred as *inconsistency definition*) different. Spanoudakis et al. do not emphasize the role of proper inconsistency characterization [2], while Van Der Straeten acknowledges characterization as the foundational first step towards a managed approach towards handling inconsistencies [3]. Indeed, the role of characterization is twofold. First, the inconsistency rules defined in this step serve as inputs for the actual detection activity. Second, the formalisms used to characterize inconsistencies, heavily influence the choice on techniques for detection and resolution.

In this paper, we present a novel formalism to characterize dependencies among different models. We argue, that because inconsistencies arise from mod-

els being used in complex design processes, their characterization is to be approached from a process oriented point of view. This approach considers not just static relations among models, but also the dynamics of the process. Although we do not explicitly investigate the subsequent activities of detection and resolution, we give specific directions on how these should be addressed to exploit the full potential of our formalism.

Our work is motivated by the specific nature of a multi-paradigm settings in the mechatronic domain. Although ICM is an adequately studied field in software engineering, mechatronic design introduces challenges that make software focused ICM techniques inefficient in mechatronics. This is due to the involvement of *models of physics*, such as models of finite element methods (FEM) or computational fluid dynamics (CFD). Typically, ICM techniques targeting pure software systems, even if being situated in a multi-model/multi-paradigm setting, focus on *syntactic* (structural) inconsistencies arising from the involvement of detached linguistic meta-models [4].

In a design process with physics involved, however, *semantic* inconsistencies, arising from semantic overlaps of various models, have a significant impact as well [5]. While the semantics of a software model are fully to be defined by the designer, semantics of models of physics are inherited from a foundational belief system, such as laws of physics, mechanics, electronics, etc.

The rest of the paper is organized as follows. In Section 2 we motivate our work using an example scenario from a case study on the design of an unmanned aerial vehicle (UAV), and present state-of-the-art techniques targeting different aspects of the scenario. In Section 3, we introduce the core ideas of our approach. Section 4 discusses the related work. Finally, Section 5 briefly discusses the results and the planned future directions.

2 Process and dependency modeling: the state of the art

In this section, we motivate our work by an example scenario and present the state-of-the-art solutions tackling well-defined parts of it.

2.1 Illustrative example

We illustrate our approach using an example from a case study on the design of an unmanned aerial vehicle (UAV). In the example scenario, the wings of the UAV are designed using models in various formalisms. The aim is to formalize the relations among the properties derived from these models either by syntactic or semantic links (e.g. by simulation).

The design process is sequential and it starts with *modeling the overall geometry* using a CAD tool. After validating the model against the requirements, a *finite element method* is used to analyse the system. Different physical properties are predicted using a geometric and finite element analysis, for example the mass of the UAV. The design team than creates a plant model of the UAV using a lumped parameter model. This plant model is computationally less intensive

and ideal for creating a control model for the UAV. Finally, a control model is designed.

By explicit modeling of the design process, valuable information can be gained on data- and control flows along the elementary design activities. Additionally, by explicitly modeling dependencies, inconsistencies among models in the process can be identified. The state-of-the-art addresses these two challenges separately. We argue, that combining these techniques yield significant advantages in managing model inconsistencies.

2.2 FTG+PM: a formalism for process modeling

The *Formalism Transformation Graph and Process Model* (FTG+PM) formalism [6] is a framework for modeling processes in the presence of a strong type system: the FTG, encompassing languages and transformations which processes are built upon.

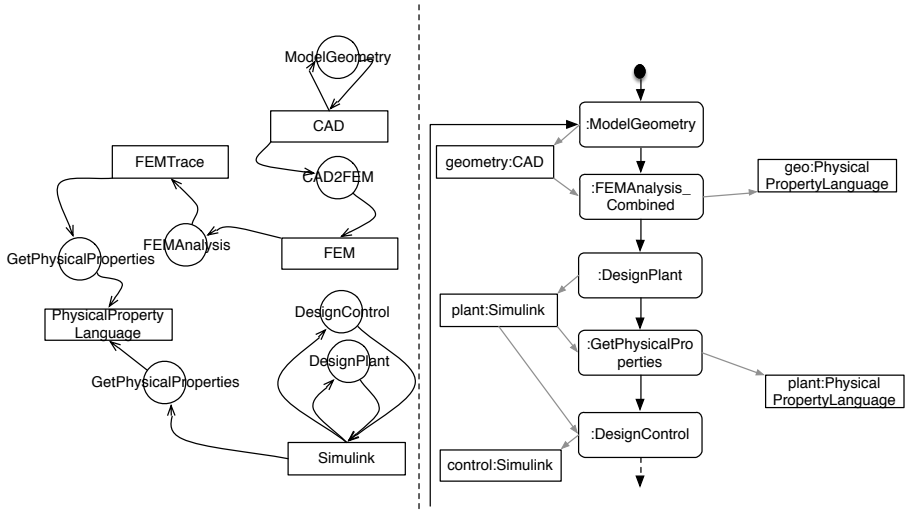

Fig. 1: FTG+PM of the example.

Figure 1 shows the FTG+PM model of the example. The FTG on the left side of the figure presents the formalisms involved in the scenario and the transformations between those. Formalisms type artefacts in the PM (on the right side of the figure). For example, the *CAD* formalism of the FTG types the *geometry* artefact in the PM.

An advantage of the formalism is the explicit type system attached to the process model, which enables reasoning about model properties and qualities

on an additional meta-level as compared to process modeling frameworks like BPMN [7].

2.3 Modeling dependencies among models

Qamar et al. investigate inconsistency management in the mechatronic domain and argue that model-based techniques are an appropriate means for inconsistency characterization [1].

The work interprets semantic overlaps as *dependencies* among elements of different models and captures these in a correspondence model, called the *dependency model* (DM). Figure 2 shows such a DM. Synthesis properties (SP) and analysis properties (AP) are distinguished to capture wether properties represent a system alternative at design time, or a result of an analysis, respectively. Similarly, synthesis dependencies (SD) represent a choice made by the designer, resulting in an SP; while analysis dependencies (AD) represent predictive models to calculate an AP.

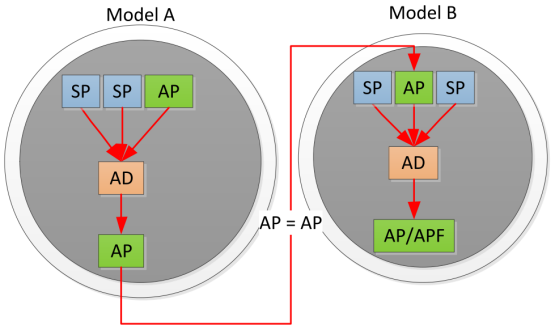

Fig. 2: A dependency model as presented in [1].

In this paper, we do not distinguish between synthesis and analysis properties and dependencies, as the formalism presented later is applicable for each of these.

3 Process-oriented modeling of dependencies

Our approach combines the ideas presented in Section 2.2 and Section 2.3. First, we generalize and extend the dependency modeling technique and redefine it in the context of processes modelled using the FTG+PM formalism.

3.1 Levels of precision

The nature of model dependencies in design processes is often well-known for participating stakeholders. In fact, this knowledge often goes beyond the level of simple influence-like relationships: dependencies encompassing *sensitivity* and *exact mathematical* relationships are also common in practice because of domain knowledge and experience.

Explicit modeling of this information enables more complex formal reasoning about model dependencies as compared to [1]. Therefore, we extend the dependency modeling formalism by introducing explicit *levels of precision* to dependency properties. We distinguish between three potential levels a set of model elements can influence another set of model elements.

- $\mathbb{L}1$**: influence graphs.** The first level is equivalent to the information depicted in [1]. The only information available is the structure of the dependency model, i.e. the vertices and edges. It enables reasoning over model elements being connected by semantic relationships.
- $\mathbb{L}2$**: sensitivity.** In engineering practice, it is a common scenario that not only the semantic relationship is known among various model elements, but also the sensitivity. By simulation and analysis, further knowledge can be gained on sensitivity information [8]. From the example in Section 2.1, such an information can be the influence of the dimensions of the controllable surface of the wings on the parameters of the control model.
- $\mathbb{L}3$**: exact mathematical relationship.** The highest level of precision is a mathematical relationship of the depending model elements. We foresee algebraic relationships being the typical examples here. For example, in Section 2.1, the mass of the UAV can be derived as the mass of all the model elements featuring a mass property, i.e. the airframe, the wings and other hardware. Other types of mathematical relationships, such as differential equations can also be used as an $\mathbb{L}3$ relationship.

3.2 A metamodel for modeling dependencies with levels of precision

To enable modeling with levels of precision, we propose a metamodel shown in Figure 3. The metamodel builds on [1], but (i) places it into the context of a design *Process*, and (ii) introduces the notion of levels.

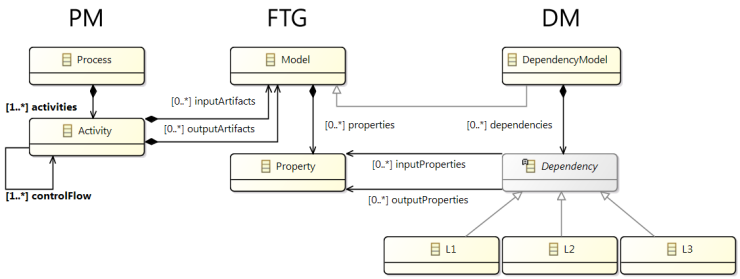

Fig. 3: Metamodel of dependency models with explicit levels of precision ($\mathbb{L}1$–$\mathbb{L}3$)

*Process*es comprise *Activities*, which take *Models* as input and output artefacts. In engineering design processes these models are the ones describing various parts of the virtual product. *Models* feature syntactic and semantic *Properties*. The former one refers to model elements derived from a linguistic metamodel, while the latter ones are typically obtained by simulation or analysis. The *Dependency Model* is a special kind of model, which features *Dependencies*, which, in line with the definition in Section 2.3, connect a set of input with a set of output *Properties*. Additionally, Dependencies are refined into $\mathbb{L}1$, $\mathbb{L}2$, $\mathbb{L}3$ dependencies, based on the level of precision.

3.3 Combining FTG+PMs with DMs

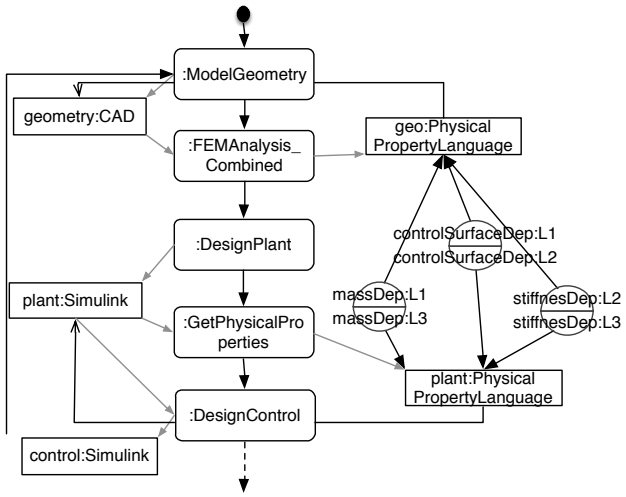

Fig. 4: PM of the example extended with DM information.

Figure 4 shows the process model of the example scenario, extended by dependency information. Three dependency properties between the language of geometrical properties and the language of physical properties are depicted using directed edges of $\mathbb{L}1$–$\mathbb{L}3$ levels of precision. (Every directed dependency edge is typed in the FTG, which is not presented hereby.) For example, there is an $\mathbb{L}3$ relationship between the *mass* of the wing and the mass considered in the control model, in this direction. This dependency link allows propagating changes from the geometrical model through an exact mathematical relationship to the control model. In the other direction, however there is only an $\mathbb{L}1$ relationship, meaning if the mass in the control model is changed, the changes cannot be propagated directly to the geometry model, but there is an indication that they should be.

The generality of this approach is shown using Figure 5. The figure shows the relation between properties of submodels. By using slicing techniques, a relation is established between a part of the model and a certain property. For example, in our example we could be interested in only the mass of the wing of the UAV. By slicing the model to only contain the wing, we can use the geometrical and material information to establish a re-

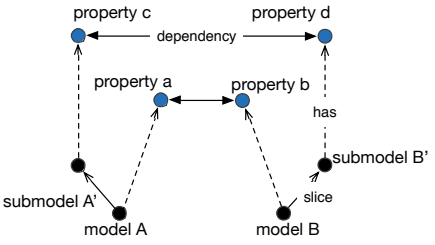

Fig. 5: Properties related to submodels by using slicing.

lation with only the wing segment of the CAD model. The slicing can also occur during the analysis step. This leaves the relation between a property and the submodel implicit. For example, because the analysis is black-box. Leaving the relation between the submodel and the property implicit could have an influence on how resolution strategies can cope with an inconsistency.

3.4 Typical uses of the technique

We show two typical uses of our technique to motivate its usefulness in complex design processes.

Inconsistency characterization, detection and resolution Our approach is situated in a conceptual inconsistency management framework as an important formalism to build inconsistency management processes upon. Since the dependency model of our approach constitutes a graph (more specifically: a hypergraph), characterizing inconsistencies as *graph patterns* over the graph of dependencies, seems to be a natural fit. Additionally, graph patterns can be evaluated at run-time very efficiently and therefore, it gives a well-performing technique to detect inconsistencies [9]. The way these graph patterns can be employed in a specific ICM technique, depends on the level of precision dependencies are captured on.

Qamar et al. use $\mathbb{L}1$ dependencies to identify *potentially* inconsistent states [1]. If an input element to a dependency is changed, the output elements become potentially inconsistent and manual inspection of these is required to maintain a consistent state. In the background, the influence graph is captured by graph patterns; a change in the match set of a graph pattern is interpreted as a potential inconsistency.

$\mathbb{L}2$ dependencies extend this approach by giving additional hints on the magnitude of inconsistency in the output property. Therefore, $\mathbb{L}2$ dependencies enable reasoning about the quality of the process topology and optimize it for consistency. (Section 3.4)

Since $\mathbb{L}3$ dependencies extend the former two by defining an exact mathematical relationship among model elements, they enable a semi-automated detection of inconsistencies by continuously evaluating these relationships.

Process optimization Dependency information can help restructure processes into more efficient alternative topologies.

A typical example of such an optimization can be making sequential design processes *parallel* in order to speed up processes, by identifying activities having no data- or control dependency, but featuring explicitly modelled semantic dependencies. Apart from the former two constraints, the semantic dependency model ensures inconsistencies will be managed over parallel branches as well. Techniques based on change propagation, incremental synchronization typically focus on sequential sub-process topologies and therefore, they fall short to tackle inconsistencies in parallel settings. $\mathbb{L}1$–$\mathbb{L}3$ dependencies all support ICM over

parallel branches, although the efficiency of ICM is determined by the level. Figure 6 shows the parallel equivalent of the PM in Figure 4.

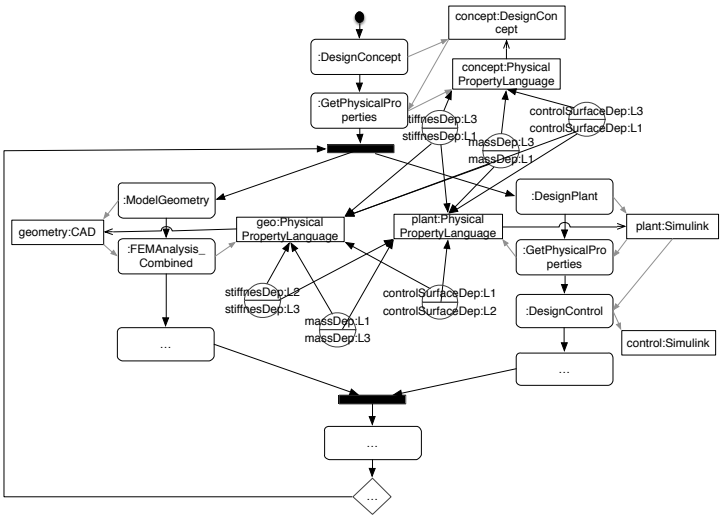

Fig. 6: Parallel equivalent of the sequential PM of the example.

$\mathbb{L}2$ relationships can help further optimizing processes for decreasing the amount of inconsistencies encountered, e.g. by moving *high-impact activities* in the earlier phases of the process and therefore, potentially reducing the amount of required re-iterations. The lifting of design choices to a higher level of abstraction to allow for parallel design also hints towards the use of *viewpoint contracts* in the design process, similar to Toerngren et al [10].

Finally, organizing processes in a topology where *higher level dependencies point forward* in the process increases the efficiency of ICM techniques based on change propagation.

4 Related work

Characterization of model inconsistencies has been approached using various techniques, such as description logic based [3], rule-based [11] and model-based.

Triple graph grammars (TGG) have been widely used as a formal underpinning to model-based ICM techniques [12]. Adourian and Vangheluwe present a technique for characterizing and detecting semantic consistencies between geometric and dynamic views of mechanical systems based on TGGs [5]. Furthermore bidirectional synchronization techniques are often used to resolve inconsistencies [13]. The structure of TGG formalism, however, inherently prevents expressing multi-level connections among model elements, as opposed to the FTG+PM formalism used in our work. As an implementation of the principles

of megamodeling [14], the FTG+PM formalism enables filling semantic gaps, as highlighted in Section 3.3.

Design structure matrices (DSM) [15] are used to model structures of complex processes or systems. Clustering techniques for DSMs improve the design and manufacturing processes [16]. By introducing levels of precision, our technique enables reasoning about quality of process topologies in a more detailed way.

Herzig et al. provides support for automating the task of inconsistency characterization by exploring semantic relationships using similarity metrics [17].

Qamar et al define five levels of *detail* of modeling dependencies [18]. As opposed to levels of *precision*, which capture how precise the applied dependency modeling formalisms are, levels of *detail* capture the extent of the dependency modeling being applied in a design process. More specifically, our work is situated on *Level 2* of detail, where "dependencies are formally captured through a model".

5 Conclusions

In this paper, we introduced a novel formalism for characterizing and detecting inconsistencies among design models. Our work was motivated by the specific nature of multi-model settings in the mechatronic domain. The approach extends the dependency modeling technique of [1] and places it into the context of design processes. As a process modeling formalism, we employed FTG+PM [6]. As shown in Section 3.3, an important benefit of this formalism is that its FTG part fills a logical gap between the metalevels of processes and our dependency modeling language.

The main advantage of our work is that inconsistencies can be characterized using a graph-like structure, which enables using state-of-the-art graph querying tools to evaluate (or enforce) model consistency.

As a future work, we plan to further evolve the formalism by including authorization and ownership models of design processes and therefore, enable reasoning about the limits of automation of inconsistency resolution techniques and including explicitly modelled manual steps in the resolution process.

Acknowledgement This work has been carried out within the framework of the MBSE4Mechatronics project (grant nr. 130013) of the agency for Innovation by Science and Technology in Flanders (IWT-Vlaanderen).

The authors wish to thank the insightful comments of Johan Vanhuyse, Tuur Benoit, Klaas Gadeyne and Maarten Witters.

References

1. Qamar, A., Wikander, J., During, C.: Managing dependencies in mechatronic design: a case study on dependency management between mechanical design and system design. Engineering with Computers (07/2014 2014) 1–16
2. Spanoudakis, G., Zisman, A.: Inconsistency management in software engineering: Survey and open research issues. In: in Handbook of Software Engineering and Knowledge Engineering, World Scientific (2001) 329–380

3. Van Der Straeten, R.: Inconsistency management in model-driven engineering. An Approach Using Description Logics (Ph. D. thesis), Vrije Universiteit Brussel, Brussels, Belgium (2005)
4. Lucas, F.J., Molina, F., Toval, A.: A Systematic Review of UML Model Consistency Management. Inf. Softw. Technol. **51**(12) (December 2009) 1631–1645
5. Adourian, C., Vangheluwe, H.: Consistency between geometric and dynamic views of a mechanical system. In: Proceedings of the 2007 Summer Computer Simulation Conference. SCSC '07, San Diego, CA, USA, Society for Computer Simulation International (2007) 31:1–31:6
6. Lúcio, L., Mustafiz, S., Denil, J., Vangheluwe, H., Jukss, M.: FTG+PM: An Integrated Framework for Investigating Model Transformation Chains. In Khendek, F., Toeroe, M., Gherbi, A., Reed, R., eds.: SDL 2013: Model-Driven Dependability Engineering. Volume 7916 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2013) 182–202
7. Object Management Group (OMG): Business Process Model and Notation (BPMN) – Version 2.0 (2011)
8. Forrester, J.W.: Principles of Systems. Productivity Press (1968)
9. Bergmann, G., Horváth, Á., Ráth, I., Varró, D., Balogh, A., Balogh, Z., Ökrös, A.: Incremental evaluation of model queries over emf models. In Petriu, D., Rouquette, N., Haugen, Ø., eds.: Model Driven Engineering Languages and Systems. Volume 6394 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2010) 76–90
10. Törngren, M., Qamar, A., Biehl, M., Loiret, F., El-khoury, J.: Integrating viewpoints in the development of mechatronic products. Mechatronics **24**(7) (2014) 745 – 762 1. Model-Based Mechatronic System Design 2. Model Based Engineering.
11. Egyed, A.: Automatically detecting and tracking inconsistencies in software design models. Software Engineering, IEEE Transactions on **37**(2) (March 2011) 188–204
12. Schürr, A.: Specification of graph translators with triple graph grammars. In: in Proc. of the 20th Int. Workshop on Graph-Theoretic Concepts in Computer Science (WG '94), Herrsching (D, Springer (1995)
13. Giese, H., Hildebrandt, S.: Incremental model synchronization for multiple updates. In: Proceedings of the Third International Workshop on Graph and Model Transformations. GRaMoT '08, New York, NY, USA, ACM (2008) 1–8
14. Bézivin, J., Jouault, F., Valduriez, P.: On the need for megamodels. In: Proceedings of the OOPSLA/GPCE: Best Practices for Model-Driven Software Development workshop, 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications. (2004)
15. Eppinger, S.D., Browning, T.R.: Design structure matrix methods and applications. MIT press (2012)
16. Li, M., Li, D.: Modular decomposition method based on design structure matrix and application. TELKOMNIKA Indonesian Journal of Electrical Engineering **10**(8) (2012) 2169–2175
17. Herzig, S.J., Qamar, A., Paredis, C.J.: An approach to identifying inconsistencies in model-based systems engineering. Procedia Computer Science **28**(0) (2014) 354 – 362 2014 Conference on Systems Engineering Research.
18. Qamar, A., Paredis, C.J., Wikander, J., During, C.: Dependency modeling and model management in mechatronic design. Journal of Computing and Information Science in Engineering **12**(4) (2012) 041009