A language for modeling constraints of characteristics in complex heterogeneous systems

Bart Cools

University of Antwerp

bart.cools@student.uantwerpen.be

Abstract

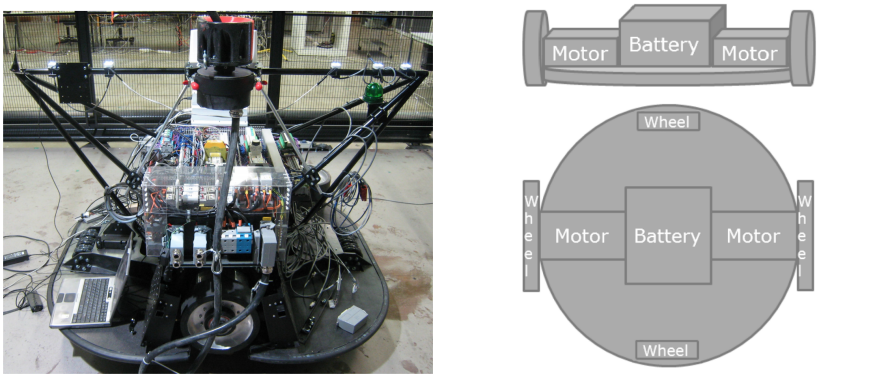
When trying to model everything, obstacles will occur in every step of the process. One large hurdle is the combination of multiple systems, created by different fields, into one large working model. Work has been done on creating these systems, together with tackling inconsistencies in the constraints as soon as possible. One problem with these solutions is that information might get cluttered and hard to see with so much relationships between attributes. In this paper we attempt to tackle this problem by finding a way to resolve this problem by creating an extra step in the process: a textual modeling language to maintain these systems. This reading report is a stepping stone to the actual implementation in future work.

Keywords: `textual modeling, modeling constraints, inconsistency management, xtext`

1. Introduction

In the philosophy of "Model Everything", a lot of hurdles need to be tackled. One approach of a modeling task is to use the top-down MDE approach: you should start by building domain specific structural and behavioral models of the system under development [1]. Following these models, you also have to allow changes on these models, the *Model Transformations*, often referred to as the *heart of MDE*[2]. A new problem occurs when we have to model a larger, complex system. When we want to model a car, we have different kinds of systems that have to be modeled, most of them in their own modeling environment. We are talking here about electrical systems, mechanical systems, geometric systems and so on. All of these systems are designed and maintained by different teams, but eventually one large, complex system will have to be created, incorporating all these different systems into one working model. When we combine all these different systems and describe the transformations working on - and between - those different models, we can talk about *FTG-PM*, which is short for Formalism Transformation Graph - Process Modeling. We model the process of these different systems and combine them together into creating, for example, a working car. One problem when combining different systems is to keep track of the major dependencies in systems, and after combining multiple systems, also the implicit dependencies and relationships that one decision might bring to the other systems.

This paper is structured as follows. Section 2 contains the related work, where we will go over the work already done on this subject. Section 3 will handle why we chose to do this project and which benefits it will bring to the researching world. Section 4 discusses what tools we will use and how we will setup the experiment. Finally, section 5 will recap everything discussed in the paper and handle the next step in this project.



(a) The automated guided vehicle of the case study.

(b) Schematic overviews of the AGV: front and top, respectively.

Figure 1: The automated guided vehicle (AGV)[3]

2. Related Work

One problem we found, which is also addressed in [4][5][6][3], is the inconsistencies in choices made that could influence different systems outside of the scope of that first system in which the decision was made in. When we take the example of building an automated guided vehicle (AGV) used in [3] (as can be seen in figure 1), we can illustrate the inconsistency problem with an example. The total mass of the AGV is defined as to be the mass of the platform used, the mass of the motor and the mass of the battery. All three of these elements are part of the same mechanical model, so we do not have to worry about cross-system inconsistencies, which we will deem outside the scope of this paper. We have multiple constraints on the mass of all these separate parts of the mechanical model, but on top of that, we have a *totalMass* constraint on the AGV itself. One of the problems that [3] tries to solve is to solve constraints in the enactment phase, so at runtime. This can be seen that, when $totalMass <= 150[kg], platformMass = 100[kg]$ and $motorMass = 50[kg]$, no matter what the mass of the battery will be, the constraint of the totalMass will always be violated, since $mass > 0$ is also a constraint, this time a constraint based on the laws of physics. When putting all of this together, we get a model like figure 2, with a closer view on the constraints in figure 2.

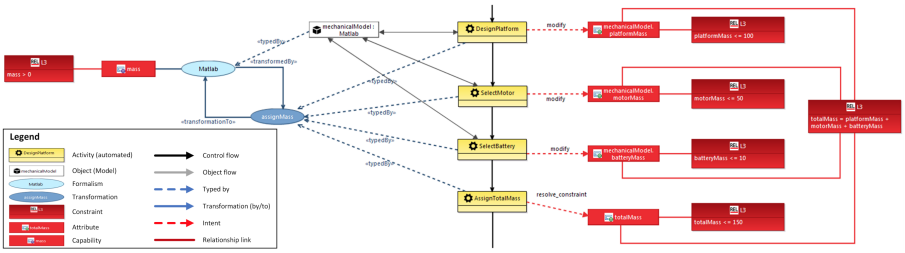


Figure 2: Overview of the FTG+PM with capabilities (left), attributes and constraints (right) [3]

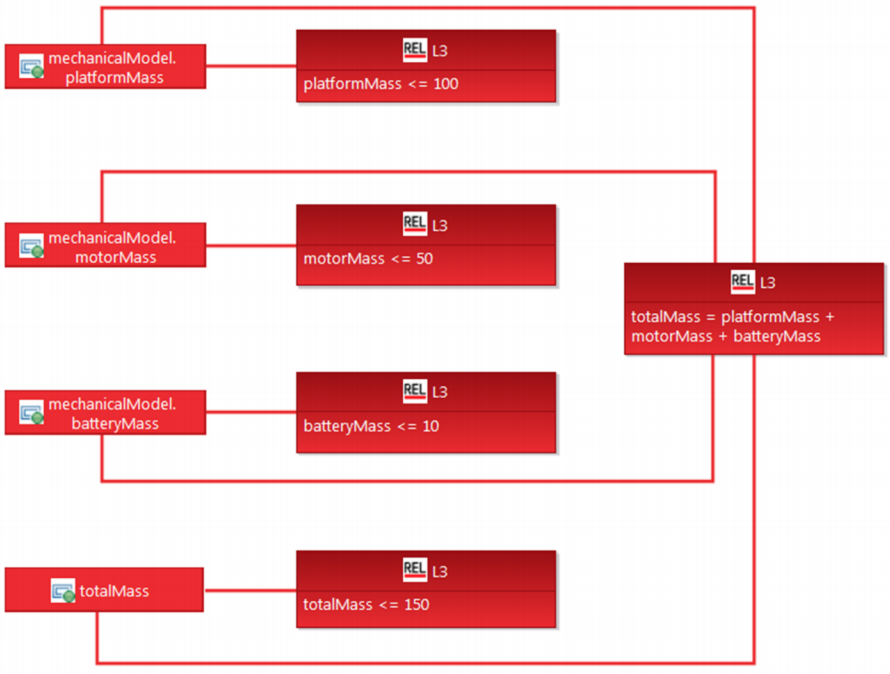


Figure 3: Constraints imposed regarding the mass of the system[3]

When we talk about constraints, an extra difficulty we have to address are the different levels of precision constraints that are available. In [3], only $\mathbb{L}3$ constraints, which are constraints that are mathematically expressible, are considered.

3. Motivation

On figure 2, we can see the capabilities on the left and attributes (and constraints) on the right, separated by the activities or processes in the middle. When we take into account that this is just the model of a small but complex system, it is easy to see that these models can become extremely cluttered for larger, more complex systems. This will hinder usability, which can be seen in figure 3, since it's harder to make sense of all the connections and elements, but maybe even more so will make the maintainability and adaptability a lot harder. For example, adding a constraint might mean finding some attribute from a certain system and associate it with some other attribute. One of the approaches to tackle this problem is to use combine the visual syntax with a modeling language to extend and maintain the models. This way we can write the attributes, rules and constraints in a very concise and textual manner, but after transforming we can still see the visual syntax to see the big overview. We will test this out and see the potential benefits from this new approach.

4. Experimental Setup

[7] proposes different formalisms to use textual modeling. TEF (Textual Editing Framework) [8], EMFText [9] and MPS (Meta Programming Systems)[10] are some of the formalisms described. For the purpose of this paper, we decided to use Xtext[11], mainly because it's a major part of the Eclipse TMF (Textual Modeling Framework). Since most of our experience in modeling comes from the Eclipse framework and the paper on which we base this report also uses Eclipse as their main tool for their models, we decided Xtext would be the most appropriate formalism to use in this case. We can't say that it's overall the

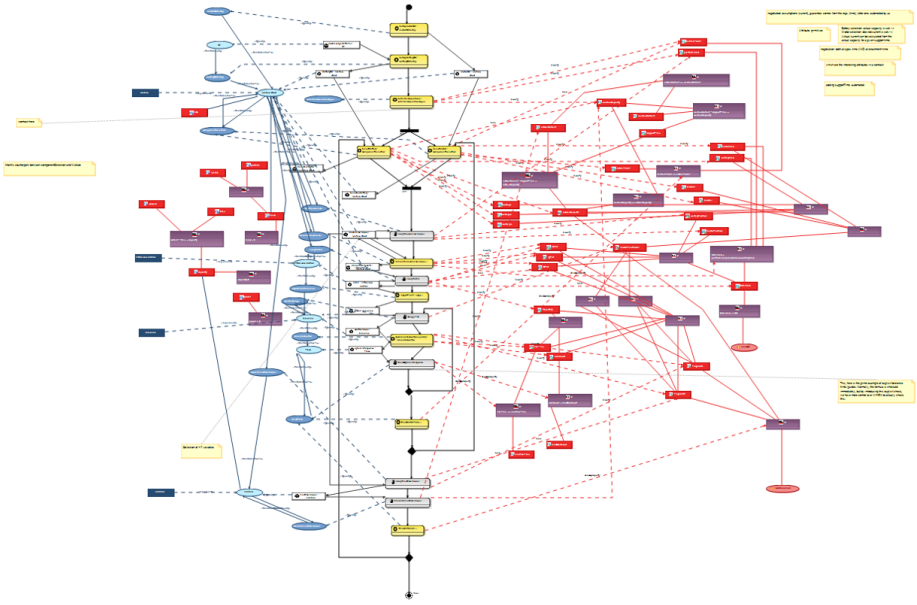


Figure 4: Overview of FTG+PM for a small complex system

most appropriate formalism, since this would require deeper research into the other formalisms, which is outside of the scope of this project. We will use the *Eclipse* framework to create a modeling language using *Xtext*.

5. Conclusion and Future Work

Most of the conclusions will be drawn after the second step of the project, which will address the actual implementation of the textual modeling language. The next steps will consist of using an iterative and incremental approach to write a grammar in Backus-Naur Form[12]. After we figured out a valid and working grammar and syntax, we will use the example used in this paper, coming from [3] to test our findings and figure out the usability and maintainability of our approach. We will at this time also determine which levels of constraints we will support. $\mathbb{L}3$ will most certainly be supported. Since this can be decoupled from the actual language, we will most likely make the precision levels of constraints generalizable, meaning that the language can specify the precision, but the run

time behavior is of course solved by the system itself.

References

[1] L. Lúcio, S. Mustafiz, J. Denil, H. Vangheluwe, M. Jukss, FTG+PM: An Integrated Framework for Investigating Model Transformation Chains, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 182–202. doi:10.1007/978-3-642-38911-5_11.
URL https://doi.org/10.1007/978-3-642-38911-5_11

[2] S. Sendall, W. Kozaczynski, Model transformation: The heart and soul of model-driven software development, IEEE software 20 (5) (2003) 42–45.

[3] I. Dávid, B. Meyers, K. Vanherpen, Y. V. Tendeloo, K. Berx, H. Vangheluwe, Modeling and enactment support for early detection of inconsistencies in engineering processes.

[4] I. Dávid, J. Denil, H. Vangheluwe, Towards inconsistency management by process-oriented dependency modeling., in: GEMOC+ MPM@ MoDELS, 2015, pp. 32–41.

[5] I. Dávid, J. Denil, K. Gadeyne, H. Vangheluwe, Engineering process transformation to manage (in) consistency., in: COMMitMDE@ MoDELS, 2016, pp. 7–16.

[6] I. Dávid, E. Syriani, C. Verbrugge, D. Buchs, D. Blouin, A. Cicchetti, K. Vanherpen, Towards inconsistency tolerance by quantification of semantic inconsistencies., in: COMMitMDE@ MoDELS, 2016, pp. 35–44.

[7] B. Merkle, Textual modeling tools: Overview and comparison of language workbenches, in: Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion, OOPSLA '10, ACM, New York, NY, USA, 2010, pp. 139–148. doi:10.1145/1869542.1869564.
URL http://doi.acm.org/10.1145/1869542.1869564

[8] Textual editing framework, https://www2.informatik.hu-berlin.de/sam/meta-tools/tef/tool.html, accessed: 2017-12-14.

[9] Emftext, http://www.emftext.org/, accessed: 2017-12-14.

[10] Jetbrains' meta programming systems, https://www.jetbrains.com/mps/, accessed: 2017-12-14.

[11] Eclipse's xtext, https://www.eclipse.org/Xtext/, accessed: 2017-12-14.

[12] Backus-naur form, https://en.wikipedia.org/wiki/Backus%E2%80%93Naur_form, accessed: 2017-12-14.