Managing Heterogeneity in Model-Based Systems Engineering of Cyber-Physical Systems

Bert Van Acker*, Joachim Denil*, Hans Vangheluwe† and Paul De Meulenaere*
*Constrained Systems Lab (CoSys-Lab), University of Antwerp, Belgium
†Ansymo, University of Antwerp, Belgium
Email: {Bert.Vanacker, Joachim.Denil, Hans.Vangheluwe,Paul.Demeulenaere}@uantwerp.be

***Abstract*—Model-based Systems Engineering plays a pivotal role in the design of distributed embedded systems by enabling early virtual integration of the different parts of the system. Traditionally, the system model is composed of subsystem models at the same level of abstraction and with one particular view. However, in some cases the system model may comprise subsystem models at different levels of abstraction. Integration of these different abstraction level models imposes some important drawbacks which hinder the overall system simulations. These drawbacks need to be addressed to facilitate the simulation of systems composed with multi-level subsystem models.**

In this paper we report on modelling techniques for embedded and distributed systems to deal with this heterogeneity. We describe a methodology to (semi-)automatically generate an executable multi-level system simulation model starting from an abstract system architecture of the system. A platooning system example is used to demonstrate the new modelling techniques.

I. Introduction

Development of cyber-physical systems (CPS) is becoming more expensive because of the rising complexity of such systems. CPSs are systems of synergistically interacting computing and physical elements distinguished from classical embedded systems by their complex, networked character [5]. Engineers deal with the complexity by modelling and simulating the system under design(SUD). However, design has to be done at the correct level(s) of abstraction for each subsystem of the SUD [6].
In addition, a complex system is very hard to understand and to manage from a single point of view [1], therefore the system modeller will represent the SUD with a variety of *domain-specific views* of each subsystem. In addition, each subsystem is also represented by a set of subsystem models at different *levels of abstraction*, induced i.e. by the incremental development of the system. This way, the design space comprises a large set of subsystem models, at a particular level of abstraction and view and somehow these models have to ”fit together” in order to perform a variety of system-level analysis.

Traditionally, a system model will be composed of subsystem models at the same level of abstraction and with one particular view. In some cases, this typical approach is not possible or not sufficient. In these cases, we will integrate subsystem models at different levels of abstraction and/or with different views in one system model. Simulation of those system models, comprising subsystem models at different levels of abstraction, can be reffered to as **multi-level system simulation**.

In this paper, we will discuss the motivation for multi-level simulation and identify some possible pitfalls of the usage. We propose a platooning system as a running example to demonstrate the modelling techniques for embedded and distributed systems, simultaneously dealing with multiple modelling abstraction levels. A platooning system is a group of vehicles showing collective driving behaviour. Platooning system models should at least contain computing, commmunication and control models. The platooning system is particularly interesting for simulation with multiple abstraction levels in case the platoon is heterogeneous, i.e. if it consists out of vehicles with different functionality.

The rest of the paper is organised as follows : Section ”Background and Related Work” presents some essential background and related work on vehicle platooning. Section ”Multi-level simulation” discusses the motivation for multi-level simulation and identifies some important pitfalls. Section ”Case study” implements a case study. In Section ”Discussion”, we discuss our proposed methodology. Finally, Section ”Conclusion and future work” concludes the paper.

II. Background and related work

A. Platooning system

Platooning is the coordinated motion of groups of (heavy-duty) vehicles cooperating with each other to reach the same destination with a common velocity and a constant inter-vehicle spacing[10] or constant time headway[2]. These platooning systems present a significant opportunity to both improve traffic efficiency and improve the efficiency and safety of vehicles within the platoon because the vehicles can travel closer together which improves the aerodynamic performance and steady state traffic flow. This has the potential to lower fuel consumption (reduced carbon emission)[8], improve network capacity and reduce traffic collisions.
A multitude of control strategies for vehicle platooning can be found in the literature since the 1950s. These studies range from theoretical work on vehicle platooning to more experimental research. Based on a brief survey on platooning and inter-vehicle control [4] , we distinct three major kinds

978-1-4673-9473-4/15 $31.00 © 2015 IEEE
DOI 10.1109/3PGCIC.2015.29

of platooning systems, each with a corresponding level of complexity :

- platooning systems **without** communication
- platooning systems with **Inter-Vehicle Communication (IVC)**
- platooning systems with **IVC and Smart Infrastructure (SI)**

Implementation of these different platooning systems imposes different requirements on the sensory and communication equipment of the platooning vehicles(PVs). The most basic platooning system implies that the following vehicles of the platoon have relative and/or absolute location awareness to other following vehicles, through a whole series of sensors, and a smart ECU to control the longitudinal and/or lateral movement of the vehicle. The more sophisticated platooning system requires a communication mechanism between the PVs, or so called inter-vehicle communication and the most advanced system implies a Smart Infrastructure which informs the vehicles with specific road information like traffic density and speed limits.

B. Platooning system setup

As stated before, there is a multitude of conducted and ongoing research on platooning systems, each with their own point of view on the platooning system. Most of the researches focus on a particular part of the platooning system e.g. the inter-vehicle spacing [10],[2],[13] or the Vehicle-to-Vehicle communication (V2V)[3]. For the good understanding, the used terminology of a platooning system needs to be clarified.

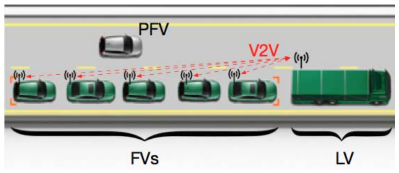


Fig. 1: Platooning system actors [8]

1) Actors: The key actors of a platooning system are the leading vehicle (LV) and one or more following vehicles (FVs). The LV is mostly a heavy-duty truck, equiped with high-end communication technology and is driven by a licensed and trained driver. As stated before, the FVs need to be equiped with more or less advanced sensors and communication technologies, depending on the kind of platoon system. Vehicles who are not (yet) part of a platoon can be categorized in (a) vehicles who can potentially join a platoon (PFVs) and (b) vehicles which are not allowed to join a platoon. In order to achieve autonomous driving, the road infrastructure can also be equipped with suitable sensors and communication devices. This equipped road is called a back office (BO).

2) Communication: The actors of the more complex platooning systems will be equiped with communication mechanisms in order to achieve either solely V2V communication or the combination of V2V and the more advanced Vehicle-to-Infrastructure (V2I) communication. This V2V and V2I communication improves the efficiency and safety of the platoon by providing road and maneuvering information e.g. oncoming traffic jams or lane switching of the LV.

III. Multi-level system simulation

A. Motivation

As stated before, a *multi-level system simulation*[1] is a simulation where the simulated system model comprises subsystem models at different levels of abstraction. There are many reasons why we integrate subsystem models at different levels of abstraction in one system model. One possible cause is the kind of development process. Within some development processes, this heterogeneity is inherent to the type of development process. E.g. within hardware/software co-design (such as embedded system development processes), meaning that the hardware and software are developed simultaneously, it regularly occurs that the individually developed models of the hardware and software don't have the same level of abstraction caused by small deviations to the synchronicity of the hardware/software development process. Another reason occurs when using *product variants*. Adding a new feature to an existing system will likely induce the integration of a low abstraction level subsystem model with higher abstraction level subsystem models.

B. Pitfalls and limitations

When conducting a multi-level system simulation, there are some pitfalls that we need to bear in mind in order to perform proper simulations. Some of the possible pitfalls are identified below.

1) Port mismatch: The first issue concerns the in- and output ports of the models. When we develop subsystem models, we iterativaly add detail to the model which results in models at different levels of abstraction. It can, and likely will, occur that by adding detail, the number of in- and output ports change. This way, when integrating subsystem models at different levels of abstraction it can happen that the number of input ports don't match the number of output ports of the preceding model(s).

2) Unit mismatch: The second issue concerns the units of the connected in- and output ports of different models. It can occur that the units of the signals on the interconnected ports don't match, especially when integrating models at different levels of abstraction.

3) Validity of the system simulation model: Another concern is the *validity of the overall system model*. In a single-level simulation, the validity of a system model is inherent to validity of the connected subsystem models. However, when composing models consisting of submodels from different origins, the global system validity must be verified, based on the

[1]Besides multi-level system simulation, we can also encounter *multi-view system simulations*. This is the case when we integrate subsystem models with a different domain-specific view. This is out of scope of this work.

given validity of the individual submodels. Although (semi-)automatic methods are appropriate to address this problem, the current paper will deal with the verification problem in a manual way.

IV. Approach

This section introduces the proposed methodology of our approach. We propose a step-by-step workflow which will facilitate the multi-level system simulation by eliminating some of the pitfalls, more specifically (a) the mismatch of in- and output ports and (b) the mismatch of units.
The result of this workflow is a (semi-)automatically generated system simulation model which can be used for multi-level simulations. Within the generated simulation model, there will be some **adaptations** to compensate for the mismatch of in- and output ports and the mismatch of units. We limited the scope of this work by taking some assumptions:

- All subsystem models provide sufficient information to establish the system simulation model. This means that at least the number of in- and output ports and the corresponding units need to be defined for each subsystem model.
- The generated stubs/monitors and unit conversion blocks are undefined and need to be manually implemented. The engineer must have adequate system knowledge to implement these elements.
- The generated system simulation model is a continuous model and all the subsystem models are developed using one uniform modelling formalism.

A. Workflow

The following steps are involved to establish the executable system simulation model.

1) step 1 : ***Define the abstract system architecture****:* The first step of our workflow is based on the divide-and-conquer principles found in many development processes, namely decomposing the system in interconnected subsystems. With this decomposition, we can easily define the abstract system architecture model. This model contains the definition of all the subsystems and their abstract topology. Figure 2 shows an example of an abstract system architecture of a typical representation of an embedded system. This abstract system architecture model will most likely be developed by a System Architect[9] at the early phase of the development process. In our approach, the abstract system architecture model is a text-based model defined with an extensive mark-up language (XML). This enables the model to be human- and machine readable, which is deemed neseccary to process this model in the subsequent steps of the workflow.

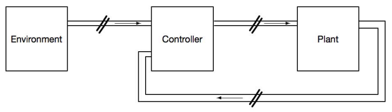
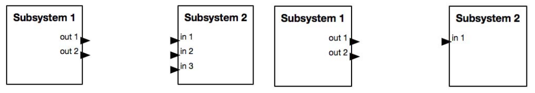


Fig. 2: Abstract system architecture example

2) step 2 : ***Select the proper subsystem models****:* After defining the abstract system architecture, we need to select a proper subsystem model for each subsystem. In practice, this can be done by defining the absolute path to the subsystem model and append it to the corresponding subsystem in the abstract system architecture model.

3) step 3 : ***Check the in- and output ports mismatch****:* Next step in the workflow is checking whether the in- and output ports of the interconnected subsystems match. Matching in- and output ports mean that the number of needed input ports is covered by the number of output ports of the preceeding subsystem model(s). Adaptation is necessary when the needed input ports of a subsystem exceed the number of preceeding input ports or vice versa. This is shown in figure 3 (a) and (b).



(a) Stubbing required (b) Monitoring required

Fig. 3: In- and output port mismatch - adaptations

In figure 3a, the number of input ports of subsystem 2 exceeds the number of output ports of subsystem 1. In this case, we need to add a **stub for one of the input ports of subsystem 2**. This stub will be a mock-up for the input signal of the stubbed port. In figure 3b, the number of input ports of subsystem 2 is smaller than the number of output ports of subsystem 1. In this case, we need to add a **monitor for one of the output ports of subsystem 1**. Determining the needed stubs and/or monitors is done automatically by extracting all the static information of the subsystem models, which is assumed to be present within or jointly with each model. The abstract system architecture model enables us to determine the preceeding model(s) in order to perform the above described check. This abstract system architecture model is then transformed to a **non-reticulated** simulation model which contains the needed stubs and monitors for that particular setup. This is shown in figure 4.

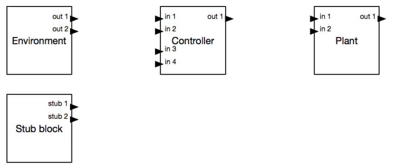


Fig. 4: Non-reticulated simulation model with stubbing

At this step of the workflow, we don't know which input and/or output port needs to be stubbed or monitored.

*4) step 4 : **Define the subsystem model interconnections**:* Step 4 is the completion of this non-reticulated simulation model, meaning that the user needs to define the interconnections between the subsystem models. This is as easy as defining the "links" between the in- and output ports of interconnected subsystem models, as shown in red in figure 5. In addition, we need to define the units for each of the stub and monitor ports, in preparation for the next step of the workflow.

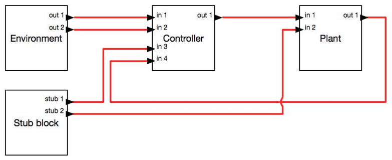


Fig. 5: Defining the subsystem model interconnections

Up till now, this step is performed manually by the user or the simulation engineer. This could be automated when the "links" or "interfaces" between the subsystems are precisely defined in a machine-readable language.

*5) step 5 : **Check the in- and output ports unit mismatch**:* The penultimate step of the workflow is checking whether the units of the connected in- and output ports match and adding an **unit conversion block** when needed. The content of the unit conversion blocks can either be automatically generated, using the theory of Ontology of Units of Measure(OM) [7], or can be manually implemented. Manual implementation of the unit conversion blocks needs to be performed after the last step of the workflow, when the executable simulation model is generated.

*6) step 6 : **Generate the executable simulation model**:* Last step of the workflow is transforming the simulation model, with the necessary stubs/monitors and unit conversion blocks into an **executable simulation model**. The formalism used for this executable simulation model depends on the formalisms used for the integrated subsystem models e.g. the Causal Block Diagram (CBD) formalism in Simulink®. Figure 6 shows an example of an executable simulation model in Simulink®. Notice that the stubs and the unit convertion blocks are already implemented in figure 6. This is done by hand within the generated executable simulation model.

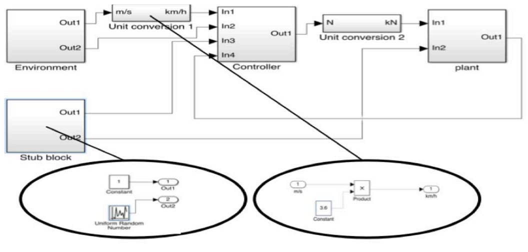


Fig. 6: Executable simulation model in Simulink®

Bear in mind that all the models used in the workflow are text-based models despite the fact that they are visually represented in the paper. To enable explicit modelling these models, a *Domain-Specific Language (DSL)* needs to be created for each of these models. Development of these DSLs can be done via AToMPM, an open-source framework for designing DSML environments [11].

V. Case study

In this section,we use the platooning system application to illustrate the proposed workflow of the previous section. More specifically, we demonstrate how our proposed methodology allows the integration of multi-level subsystem models in one multi-level system simulation model.

A. Platooning system setup

The experiment is a heterogeneous platooning setup where the FVs of the platoon differ, meaning that these vehicles have a different level of abstraction. This is particularly interesting to analyse the behaviour of a platoon composed of vehicles with different functionality. In this setup, the first two FVs support V2V communication and have a special emergency brake mode, the third FV has only a basic adaptive cruise control to establish the platoon functionality.

The first step is defining the decomposition of the platooning system in different subsystems and their abstract interconnection. In this step, we don't take the different abstraction levels of the models into account. We just define the subsystems and their connections in between. The decomposition for this platooning system is shown in Figure 7.

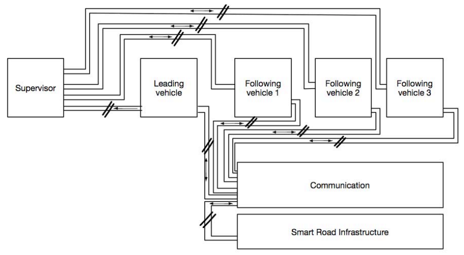


Fig. 7: Abstract system architecture for the platooning system

Note that the interconnections between the subsystems are either uni- or bidirectional, this for simplification of the model. Each subsystem also gets an unique ID in the abstract system architecture model, indicated by the CID as depicted in Listing 1.

Next step in the process is selecting the proper model for each of the subsystems. This is done by defining the absolute path[2] of the selected model within the text-based abstract system architecture model. Besides the definition of the path, we also define the view, level of abstraction,

[2]In Listing 1, the absolute path is abbreviated for readability reasons.

formalism, tool and tool version for each selected model. This is shown for the *communication* subsystem in Listing 1.

```
<Component>
  <P Name="ComponentName">Communication</P>
  <P Name="CID">66</P>
   <SelectedModel>
     <P Name="Name">Communication</P>
     <P Name="View">Conceptual</P>
     <P Name="Abstraction">0</P>
     <P Name="Formalism">CBD</P>
     <P Name="Tool">Matlab/Simulink</P>
     <P Name="Tool version">MATLAB_R2014b</P>
     <P Name="AbsolutePath">C:../Communication/COM_0.slx</P>
   </SelectedModel>
</Component>
```

Listing 1: Selecting the proper subsystem models

After selecting the proper subsystem models, the text-based model is transformed into a non-reticulated simulation model. Before transforming the model, the in- and outport mismatch is checked automatically and the needed stubs/monitors are generated. These stub and/or monitor blocks are added to the text-based model together with the definition of the ports for each of the subsystems. The added stub and monitor blocks are shown in yellow in Figure 8.

Subsequentially, we need to define the specific connections between the subsystems. This is done by specifying the abstract interconnections as shown in Listing 2. To specify the interconnections, we need the definitions of the ports of each subsystem and the subsystem ID. The following syntax is used:

[SubSystemID]_[ModelVariableIndex]

The SubSystemID and the ModelVariableIndex respectively define the source/destination subsystem and port.

```
<Relation>
    <P Name="DE_ID">R4</P>
    <Subrelation>
        <P Name="SDE_ID">R1_1</P>
        <P Name="Src">CID:68_3</P>
        <P Name="Dst">CID:65_1</P>
    </Subrelation>
</Relation>
```

Listing 2: Specification of the subsystem interconnections

After specifying the connections, we establish a reticulated simulation model. This model serves as input for the second transformation. This transformation will add unit conversion blocks when needed. This can be automatically performed by examining the *unit tag*, specified with each signal or ModelVariable, of the interconnected ports. There are two unit conversion blocks needed in our simulation setup, shown in green in Figure 8. The last step in our workflow is generating the executable simulation model. In this step, we use the text-based model of the previous step and transform it to an executable model. In this case study example, this executable model will be a Matlab/Simulink® model because all the subsystem models are developed using Matlab/Simulink®. The automatically generated executable simulation model is shown in Figure 8.

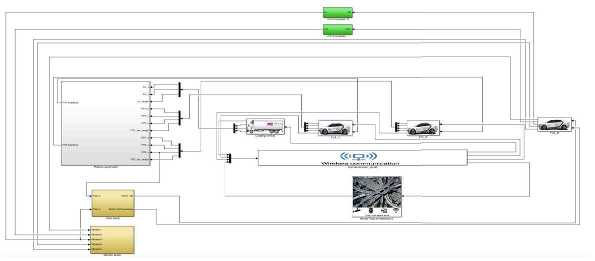


Fig. 8: Executable system simulation model as a Matlab/Simulink® model

With this generated executable simulation model, we can analyse the behaviour of a heterogeneous platoon. In our setup, we analyse the headway between the different PVs when performing an *emergency brake (EB)*. In this case, the first two FVs receive a notification/message of the EB from the LV, which triggers a special EB mode in the FVs. The third FV doesn't support communication and doesn't have an EB mode, so it will try to establish a constant headway using the adaptive cruise control logic. This third FV could cause safety critical situations for the whole platoon under specific EB conditions, as shown in figure 9.

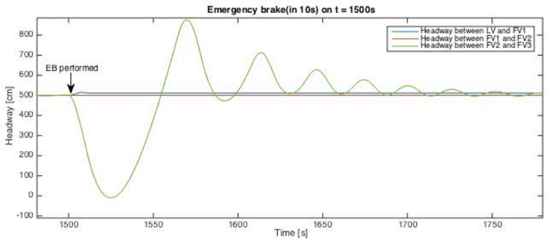


Fig. 9: Simulation results of the multi-level system simulation of the heterogeneous platoon

In Figure 9, we see the headways of the different FVs. The first two FVs react almost immediatly to the EB, the third only reacts when the headway decreases but when the EB time is very short, it will not be able to perform a stable control. This way, the headway decreases, and drops below zero. This results in safety critical situations for the whole platoon because the third vehicle bumps into the second FV. This effect can even be worse when the desired spacing between the PVs decreases or the EB time decreases.

VI. Discussion

Our approach relies on the defined information or meta-knowledge for each subsystem model. This information about the model is deemed necessary to implement the flow of our approach. In our case, we defined a *modelDescription*, based on the FMI[3] modelDescription, for each of the models. We extended this modelDescription to fit our needs. Further enhancement of this meta-knowledge, e.g. defining the modelling assumptions, signal ranges, etc., will extend our capabilities e.g. validity analysis.

As stated in Section IV, all the models used in the workflow are text-based because defining a simple domain-specific language for these models would not add value to the described process. This doesn't mean that we ignore the importance of a graphical modelling language, which needs to be scalable for large systems, to define these models.

Finally, in this paper we proved that the *platooning system* is a nice case study for further reseach on these multi-level/multi-view system simulations. In this case, we focused on the multi-level system simulations, we identified possible pitfalls and provided a way to facilitate the usage. This could also be conducted for multi-view simulations and ultimately, combined multi-level and multi-view system simulations. A nice case for the multi-view system simulation of the platoon is if we change the following vehicle models to an aerodynamic drag model to analyse the drag in respect to the platoon headway and/or the fuel consumption of the different platooning vehicles.

VII. Conclusion and future work

This paper presents a methodology to facilitate the use of multi-level system simulations(by eleminating some of the pitfalls). Our proposed process starts with a high abstract model of the architecture of the system. By gradually adding information to this abstract model, we eventually can (semi-) automatically generate an executable simulation model for this system. We applied this process to carry out a multi-level system simulation of the described platooning system.

In the future, we plan to support heterogeneous multi-level system simulations, meaning that the subsystem models don't need to be developed using the same formalism. To enable this, we need a heterogeneous simulation environment. One possible enabler for this is the Functional Mock-up Interface (FMI) standard. Previous conducted research[12] on this topic will facilitate the needed adaptation to support this heterogeneity.

We also aim to incorporate the validity analysis within the process. This way, it will be possible to analyse the validity of the multi-level simulation model and to critically assess the simulation results. This will be tightly coupled with the needed model information or meta-knowledge[9], defined for each subsystem model.

[3]https://www.fmi-standard.org/

Acknowledgment

We would like to acknowledge Busschop Jeroen, Van den Berghe Tom and Verreycken Erik from the University of Antwerp for their conducted research on *platooning systems*. These master students gave a nice overview of the carried out studies on this topic and the examined systems.

References

[1] M. Barbero, F. Jouault, and J. Bezivin, "Model Driven Management of Complex Systems: Implementing the Macroscope's Vision," in *Engineering of Computer Based Systems, 2008. ECBS 2008. 15th Annual IEEE International Conference and Workshop on the*, March 2008, pp. 277–286.

[2] C. Chien and P. Ioannou, "Automatic vehicle-following," in *American Control Conference, 1992*. IEEE, 1992, pp. 1748–1752.

[3] L. Glielmo, "Vehicle-to-Vehicle/Vehicle-to-Infrastructure Control," *The Impact of Control Technology*, 2011.

[4] P. Kavathekar and Y. Chen, "Vehicle platooning: A brief survey and categorization," in *ASME 2011 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. American Society of Mechanical Engineers, 2011, pp. 829–845.

[5] E. Lee *et al.*, "Cyber physical systems: Design challenges," in *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*. IEEE, 2008, pp. 363–369.

[6] P. J. Mosterman and H. Vangheluwe, ""Computer automated multi-paradigm modeling: An introduction"," *Simulation*, vol. 80, no. 9, p. 433, Sep. 2004.

[7] H. Rijgersberg, M. van Assem, and J. L. Top, "Ontology of units of measure and related concepts." *Semantic Web*, vol. 4, no. 1, pp. 3–13, 2013.

[8] T. Robinson, E. Chan, and E. Coelingh, "Operating platoons on public motorways: An introduction to the sartre platooning programme," in *17th world congress on intelligent transport systems*, vol. 1, 2010, p. 12.

[9] G. Sirin, C. J. Paredis, B. Yannou, E. Coatanea, and E. Landel, "A Model Identity Card to Support Simulation Model Development Process in a Collaborative Multidisciplinary Design Environment," 2014.

[10] D. Swaroop, J. Hedrick, C. Chien, and P. Ioannou, "A comparision of spacing and headway control laws for automatically controlled vehicles," *Vehicle System Dynamics*, vol. 23, no. 1, pp. 597–625, 1994.

[11] E. Syriani, H. Vangheluwe, R. Mannadiar, C. Hansen, S. Van Mierlo, and H. Ergin, "AToMPM: A Web-based Modeling Environment." in *Joint Proceedings of MODELS'13 Invited Talks, Demonstration Session, Poster Session, and ACM Student Research Competition co-located with the 16th International Conference on Model Driven Engineering Languages and Systems (MODELS 2013)*, September 29 - October 4, 2013, pp. 21–25.

[12] B. Van Acker, J. Denil, H. Vangheluwe, and P. De Meulenaere, "Generation of an Optimised Master Algorithm for FMI Co-simulation," in *Proceedings of the 2015 Symposium on Theory of Modeling and Simulation - DEVS*, ser. TMS/DEVS '15, part of the Spring Simulation Multi-Conference. Society for Computer Simulation International, Apr. 2015, pp. 946 – 953.

[13] D. Yanakiev and I. Kanellakopoulos, "Nonlinear spacing policies for automated heavy-duty vehicles," *Vehicular Technology, IEEE Transactions on*, vol. 47, no. 4, pp. 1365–1377, 1998.