

Model-Based Generation of Software Configurations in Mechatronic Systems

Martin Paczona,¹ Heinrich C. Mayr,² Guenter Prochart³

Abstract: An essential part of the mechatronic system is the software, which is responsible to bring functionality into the system consisting of mechanical, electronic and electrical parts. The software must be tailored to the specific hardware to fulfill tasks (e.g. control, monitoring) according to the system requirements. In today's industrial practice, the design is mainly done manually. First the entire architecture is drawn using drawing tools. Based on this software developers derive the low-level specification using their low-level development environments. This is error prone and time-consuming due to the fact, that a large number of hardware parameters have to be taken in account and the informal specification does not allow to derive these parameters. To improve this we present here an approach where the overall architecture of the mechatronic system is described using a Domain-Specific Conceptual Modeling Language (DSML) using the example of Electric Vehicle Testbeds. Based on this model the low-level software configurations are generated rule-based. In this paper we present the concepts of the DSML, explain the transformation rules and show the functionality of the generator by introducing a practical example.

Keywords: Modeling; Metamodel; Software; Generator; Transformation; Configuration

1 Introduction

Tuning the software for a mechatronic system is a tough job. This includes setting up the communication structure, finding the optimal control parameters, configure the IOs and implement the user interface. To develop this software and to setup the parameters cooperation of hardware developers, electrical designers and electronic designers is needed [To07]. Although comprehensive process models have been proposed for this purpose, they are hardly used in daily practice [SP09]. The reasons given include: too complicated, no time for processes, lack of tool support, lack of visible benefit, academic exercise. Furthermore, general-purpose modeling languages such as SysML and UML are only moderately accepted in the mechatronic domain because they do not provide the domain vocabulary and have a level of abstraction that discourages practitioners from using it. Instead of this process model still informal descriptions are used. Systems are described in drawing tools; based on this representation engineers collect the requirements and derive

¹ AVL List GmbH, Electrification Products, Hans-List-Platz 1, 8020 Graz, Austria martin.paczona@avl.com

² Alpen-Adria-Universität Klagenfurt, Application Engineering Research Group, Universitätsstraße 65-67, Austria heinrich.mayr@aau.at

³ AVL List GmbH, Electrification Products, Hans-List-Platz1, 8020 Graz, Austria guenter.prochart@avl.com

the low-level software implementation. Now we will have an look on Electric Vehicle Testbeds (EVTs). EVT's are customized solutions for testing electric vehicles and electric vehicle components like batteries. The design of such testbeds is done by drawing the entire architecture in collaboration with the customer using a tool such as MS Visio. From this "specification", software developers, circuit plan designers and hardware developers derive detailed specifications and designs using their specific low-level design and development environments (e.g. EPLAN[EP19], Automation Studio[Bu19], SolidWorks[So19]). A further problem is that the software of a mechatronic system has a strong hardware connection, which is little considered in software development environments. Therefore, software and hardware are usually developed separately. Resulting inconsistencies often lead to system integration problems when the software is installed on the prototype system.

In addition, there is a high time pressure in the development of mechatronic systems. This applies in particular to EVT's caused by the E-Car boom [Fo18, KSK18]. Delivery times of 3-12 months, however, are common for EVT components [Ve19, Ro17, ec19]. The development therefore begins here with the specification of the hardware and its components, in particular, the long leads. Software design and development begin when the hardware has already been specified and is in production.

In order to accelerate and increase the efficiency of software development, the industry relies on the reuse of customizable software components. Software development for EVT's therefore essentially consists of configuration i.e., the selection of suitable modules, their parameterization and integration. In this paper, we propose an approach to automatically generating software configurations using EVT development as an example. This approach is based on a domain-specific conceptual modeling language (DSML) for the integrated description of hardware and software. A DSML is a language which target is a specific domain, therefore it has fewer more specific concepts rather than many generic concepts compared to a general-purpose language. The first step in the development of a DSML is the definition of the metamodel, which defines the domain concepts and the relation between the concepts. The approach is based on the "single source of truth paradigm" and avoids the inconsistencies previously mentioned as a problem. The conception of our DSML is such that the engineers can easily and intuitively understand it: to achieve this, we have conducted a series of surveys and practice workshops with stakeholders and domain experts. First experiences show a good acceptance, which is also due to the fact that our modeling language allows a very efficient and effective modeling due to its domain orientation. The further structure of the paper is as follows: Section 2 focuses on the conceptual aspects by introducing the metamodel, the DSML and the modeling tool developed. In chapter 3 we sketch the process of generator development followed by a detailed discussion of the rules for generating configurations from a model in chapter 4. Chapter 5 outlines the implementation of our approach using the metamodeling platform ADOxx. We also will show how our solution integrates into the overall development process. As a proof of concept, in chapter 6 (Evaluation) we compare the generator output with a set of manually developed artefacts.

After a short discussion of related work in chapter 7, the paper closes with a conclusion and an outlook on further work.

2 Software Configuration Model

Developing a Domain-Specific Modeling Method is a multi-stage and iterative process [MM15] comprising the development of an appropriate metamodel, the definition of one or, when required, more notations (representation languages on the model and data level according to the OMG Meta object Facility MOF [OM02]), the grounding of the metamodel concepts in an ontology, tool development, evaluation etc. [Ma18].

Since the overall metamodel and the notation principles are subject of another papers [PMP19, PM19], we limit ourselves here to the introduction of the concepts for software system and configuration modeling, as sketched in Fig. 1. Clearly, each concept comes with a set of meta-attributes that are not depicted in the figure but exemplarily mentioned in the modeling tool description below (e.g. Fig. 3). The concepts in the metamodel are connected with the generalization-relation.

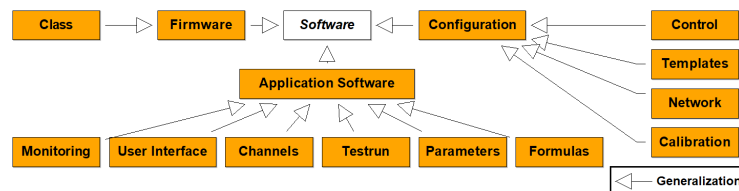


Fig. 1: Software Configuration Metamodel (excerpt).

Each of the concepts in the meta-model needs a proper representation. We refine here the basic notation by more specific elements for *Configuration*, *Firmware* and *Application Software*. To visualize the relationship between configuration concepts and other software-related concepts (rectangle with tab), these elements use a rounded rectangle with a yellow color. Inside the shape a symbol indicates the type of *Configuration*. These symbols correspond to the usual representations in automation and software. To make the meaning clearer as in the base representation, also a textual description is used [Mo09]. Fig. 2 shows the notation elements.

A modeling language without tool support has little chance of widespread use. Moreover, the automatic generation of software configurations needs a lot of attribute specifications which can only be handled efficiently by an appropriate modeling tool. An investigation of the requirements of non-software experts like Electrical Engineers, Managers, and Hardware Developers revealed that they require a clear and navigable overview over an EVT software configuration supporting intuitive understandability (see [PMP19]). Another important requirement of the domain experts is the availability of consistency checks. Based on the software configuration model (SCM), which is defined using the concepts of the metamodel (e.g. Control), the generator performs the transformations depending on the model structure

and model-element-attributes. Fig. 3 and Fig. 4 show excerpts of the attributes of the Control and Transformer elements based on the findings made during generator development (see chapter 3).

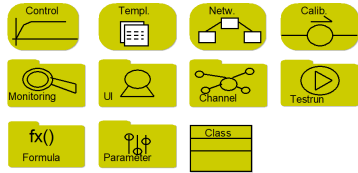



Fig. 2: Notation of the software-related concepts.

Control-167000 (Control)	
MPC_CC_Is_State_Traj:	0,000000
MPC_CC_K_Modul:	0,000000
MPC_CC_Q_I1:	0,000000

Fig. 3: Control Parameter Definition in the Circuit Plan Model.

Grounding a metamodel in an ontology strengthens its semantic soundness. Domain ontologies are conceptualizations of aspects of a given domain shared by the respective user group [Ro11]. In a “sound “metamodeling language every modeling construct has a corresponding concept in the ontology [ES13]. Fig. 5 sketches the structure of an ontology draft which we have compiled for the domain of EVT software configuration.



Transformer-167409 (Transformer)	
Feedforward_ID:	0,000000
Feedforward_IQ:	0,000000
Feedforward_VDC:	0,000000

Fig. 4: Transformer Control Parameter Definition in the Circuit Plan Model.

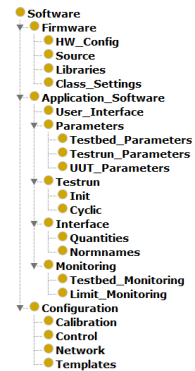


Fig. 5: EVT Software Configuration Ontology.

3 Generator Development

Since DSMLs provide conceptualizations of domain aspects, transformation rules can be directly linked to the concepts. This is an advantage over general-purpose modeling languages, for which common solutions are more difficult to find. In addition, the use of already available components can lead to faster results than building something from scratch [SK03]. If generators already have been implemented in the domain, they should be docked to the existing solution, for example by one generator generating the input for another. Before going into details, we will briefly explain the most important terms of code generation as well as the overall structure of the generator development process. The generator input, called source, is processed using transformation rules to produce the target. The generated

target is the code (or program) executed on the target platform. In our case the source is the EVT-Model (EVTM) including the circuit plan (CPM) and the SCM submodels; the target are the software configurations of the EVT components. Fig. 6 shows this process according to [Ku11]. The generator thus bridges the gap between the platform independent models and the platform model by adding platform specific information.

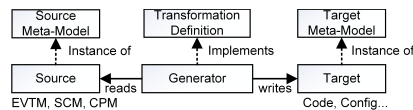


Fig. 6: Terminology of Code Generation (modified) [Ku11].

A relationship between the software and hardware elements is called cross reference [Ga10]. In Detail, a system element satisfies a requirement, if this requirement is implemented by a function which is realized by this system element. In chapter 4 we will present how this is implemented for the different EVTm transformation cases. The implementation of this cross-references and the code generation based on software and hardware models is one of the main challenges. The goal of the generation process is to increase the productivity and to increase the quality of the EVT-solution.

The generator development process can be divided into the following 7 steps:

1. *Domain Analysis*: Identifying the target (configuration files) to be generated and the information to be provided for these files (see Tab. 1).
2. *Structure Information*: dividing the information into “recurring“ information that may be compiled in templates (see step 5), and information requiring user parameterization.[SK03] propose the “piecemeal generator approach“. This means that initially most information is fixed in a template and then stepwise inserted into the models. Clearly, the number of possible generator outputs rises with the number of parameters and must represent valid configurations [Vo13].
3. *Check completeness*: Identifying information that has no correspondence in the models (source). This can be done by mapping each information unit to the corresponding model component. If an information cannot be mapped the model has to be extended accordingly (step 4).
4. *Update models*: Completing the models to cover the previously detected gaps, to ensure that complete configurations may be generated. In the EVT case, the decision, which model to complement, is basically driven by the user’s skills. For instance, if a new parameter is added to the CPM than circuit plan designers should know them.
5. *Templates*: Platform information which need not to be changed by the user is transferred into template files, the template files have to be created.
6. *Transformation rules*: The generator development starts by defining the source to target mapping (see Chapter 4).

7. *Evaluation*: In the last step the generator functionality is checked using a set of test-cases e.g. comparing with manual generated artefacts (see Chapter 6).

<i>Description</i>	<i>Type</i>	<i>Format</i>
Control Param., Calibration Data, Cabinet and Network Settings	Text	CSV
Version Information (Build, Package...)	Text	custom
PLC Hardware Configuration	Text	XML
Automation System Files	Script	PUMA/LYNX
Source Code Class-Definitions, Header-Files, Error Codes	Source	C++
Variable Definitions	Source	C++, ST

Tab. 1: Generator Output Overview (PUMA/LYNX are Test Autom. Systems from AVL List GmbH).

4 Transformation Rules

Transformation rules typically apply to a certain subset of model components [SK03]. In our case, we start from the EVT, the CPM and the SCM which are defined by the domain-experts. Note that these “application models“ [Ga10] together form the EVT-Model. The EVT defines the overall structure of the EVT, the CPM the electrical part and the SCM specifies the software. The relevant components for which targets have to be produced then are *Source-Code*, *Error-Codes*, *Network-Settings*, *Control Parameters*, *Automation System Files*, and *Calibration Parameters*. Consequently, we define the transformation rules for these aspects including their scope and limitations. Fig. 7 gives an overview of the proposed transformation process. Based on the EVT (and its submodels) a Raw Software Configuration is generated (RSC) using again the SCM notation. I.e., the RSC is an intermediate model, which the software developer may complement and refine, for example by individual code. After that the Software Configuration Generator (SCG) produces the final software configurations.

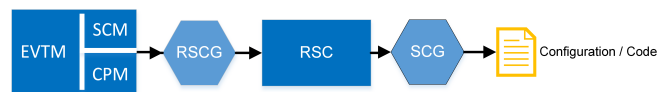


Fig. 7: Generator Process Structure Overview.

In the following we introduce the transformations for the different target types. For each type a short overview including the transformation rule and an excerpt of the generated output is given. The rules have been figured out by analysing the target artefacts in detail. A generic process for defining the rules, a formal definition of the rules and the development of generic rules for the mechatronic domain is part of future work. In the Rule definition *Output* (O) represents the result of the transformation, *Source* (S) the input of the transformation, $f()$ a function and $+$ connects two information sources. The rule definition is shown below.

Source: Source Model Source Component

(Rule Notation)

Output: SCM Attribute= Source.Attribute + Platform Properties

4.1 Source-Code

The transformation rule for source-code generation assumes that the *Hardware_Component* including the *Hardware_Connections* contains most of the information needed for source code generation. During generator development it was found out, that only *Hardware_Components* need to be considered for the source-code generation which are directly connected to *IOs* of automation systems. This approach is inspired by the work presented in [Ba08] where each modeling element has a template file assigned. However, we do not use a template for each element but instead slightly different transformation rules for each modeling element. A pseudo-code version of the general rule is given below. This rule is refined for the particular elements. As an example consider a voltage sensor, which has the function to measure a voltage, and which is connected to an input channel of a control unit. The information produced by such a sensor differs from other types of elements. Since we use domain-specific models, the transformation rules can be tailored to such elements. This is a considerable advantage over the use of general-purpose modeling languages, where functions are defined based on requirements, and hardware elements based on these functions [Ga10]. In our approach, this relationship is exploited the other way round (see chapter 3). In case a sensor is connected to an analog input we can assume that the class of this sensor has a function *get_AI()*. Since the circuit model also defines the connection between the sensor and the IO we can derive the function *get_AI_+SensorName*. This makes the generated code easier to understand. We exploit this hardware relation also for generating the class and the variables. The abstract definition of the transformation shows R1 and R2. In both rules the *source (S)* is the *HW_Component* of the CPM and the Target is the generated code.

S: CPM HW_Comp (R1)

O: Class.Name= HW_Comp.Name + Platform_Info

S: CPM HW_Comp (R2)

O: Class.Prop= prefix + HW_Comp.Prop + connected HW_Comp.Prop,
Platform_Info

4.2 Error-Codes

The transformation rule for error generation is based on the hypothesis that the CPM already contains most of the information needed for generating the error texts of a mechatronic system. To allow for customizing the error information of each *Hardware_Component* a string value was added to the component. Fig. 8 shows the example of the *PLC*. Errors

related to software are specified in the SCM element properties. An example is shown in chapter 6. The string values are then used to generate the Error-Codes for each *System*.

S: CPM HW_Comp (R3)
O: Errortxt=Error-Type + HW_Comp inside System + HW_Comp.Prop

4.3 Network Settings

The network setting of a testbed depends on the structure of the EVT elements, by parsing the testbed structure directly, the IDs can be calculated (R4). To keep the approach flexible the SCM Network element also provides the possibility to set the ID manually. An example is given in chapter 6.

S: EVT Network (R4)
O: ID= f(Distribution_Box_cnt,Switch_Box_cnt) + System.Name + Platform_Info

4.4 Control Parameters

The control parameter can be divided into parameters depending on the hardware (hardware parameters) and into parameters depending on the user input (user parameters). The hardware depending parameters are calculated based on the hardware properties and hardware structure (*Inside, Assembly_Connection*) see (R5). The user dependent parameters are mapped from the user input in the SCM (R6).

S: EVT HW_Comp Assembly_Connection (R5)
O: Para=f(HW_Comp.Prop, f(System elements Inside EVT, System element Assembly_Connection)) + Platform_Info
S: SCM Control (R6)
O: Para=Control.Para+ Platform_Info

4.5 Automation System Files

The automation system controls all the systems of the EVT. The generator input are the *Application_Software* elements: *Monitoring, Parameters, Testrun, User_Interface, Channels* and *Formulas*. Each of these elements provides the possibility to set properties and to define customized scripts. Fig. 9 shows a *channel* definition example. The raw software configuration generator (RSCG) accesses the EVT to generate the RSC, this is then enhanced by the user. The SCG generates the target configuration out of this by performing a horizontal transformation. (R7) shows this relation channel definitions are generated based on the *Supply* properties and platform info.

S: SCM Channels, EVTm Supply

(R7)

O: Channels.Prop= f(Supply.Prop) + Platform_Info

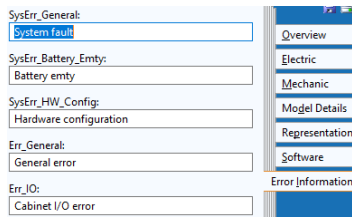


Fig. 8: PLC Component error settings.

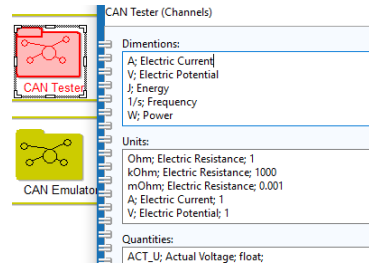


Fig. 9: Automation System, Channels.

4.6 Calibration Parameters

Each sensor of a physical system needs calibration data. This calibration data must be defined during a calibration process. Therefore, the modeling language provides a corresponding concept. The calibration data cannot be calculated, as it directly relies on the performed measurement during calibration. The main part of the generator is here a horizontal transformation which adds the platform specific content of the target-domain. The calibration data is linked to a hardware (sensor) on the CPM (using a model INTERREF). This connection is used to perform constrain checks on the calibration data and to extract further information e.g. gain values for the sensor.

S: SCM Calibration, CPM HW_Comp

(R8)

O: Calibration=f(Calibration.Prop + HW_Comp.Prop) + Platform_Info

5 Tool

The complete modeling solution including metamodel, modeling tool and transformations has been developed using the metamodeling framework ADOxx. We opted for this platform because it has been intensively tested in many different research projects in science and industry [FK13]. Since the metamodel development, notation and modeling tool development are published elsewhere [PMP19, PM19] our focus in this paper is on the generation process of the software configurations for EVTs. ADOxx includes the scripting language AdoScript to perform model transformation and model-queries (transformation language support [SK03]). Consequently, there is no need to develop the model parsers manually. The transformation rules presented in chapter 4 were implemented in AdoScript like the example shown below which drives the transformation of a *SW_Assembly* based on the *HW_Assembly*. The EVTm is publicly available on the OMiLAB repository. OMiLAB is

a dedicated research and experimentation space for modeling method engineering [OM19]. We decided to make the EVTm tool there publicly available to collect further user feedback. Since the EVTm tool is developed in close cooperation with a company, not all functions are released in the public version to protect company rights. For integrating our approach into the development process of the company, we created a set of guidelines and linked the domain-specific modeling process with the existing solution so that the users can understand the difference. A starting point for this was a comparison as shown in Tab. 2.

Transformation definition in *AdoScript*:

```
SETL sClSource:("HW_Assembly")
SETL sClTarget:("SW_Assembly")
CC"Core"GET_CLASS_ID classname:(sClSource)
CC"Core"GET_ALL_NB_ATTRS classid:(classid)
CC"Core"GET_ATTR_VAL objid:(VAL sObject) attrname:"Name" SETL sNa:(val)
CC"Core"GET_CLASS_ID classname:(sClTarget) SETL idClTarg:(classid)
CC"Core"CREATE_OBJ modelid: (idTargetM) classid: (idClTarg) objname:(sNa)
CC"Modeling"SET_OBJ_POS objid: (objid) x: 5cm y: 8cm
```

<i>Traditional</i>	<i>Domain-Specific Model</i>	<i>Example</i>
Source-Code (Eclipse)	CPM, SCM	Class Definition
Document (MS Visio), Network Settings (Files)	EVTm	IDs Netw. Nodes
Controller Parameter (Text File / MS Excel)	EVTm, CPM, SCM	Controller gain
Error Codes (Code, MS Excel)	CPM	Error enum

Tab. 2: Comparison Table (EVT Domain Example)

6 Evaluation

To validate the correct functioning of our generator and the plausibility of the overall approach, we have taken the usual route of comparing generator outputs with manually developed artefacts from previous EVT projects. The evaluation covered the outputs: *Source-Code*, *Error-Codes*, *Network Settings* and *Control Parameters*. These outputs have been compared regarding the criteria: Completeness, Correctness, Productivity (time to produce the solution), Understandability (to the users), and Re-Usability.

Evaluation Example for a particular testbed project:

The batteries under test may have 48-1100V, max. 1700A, 550kW. The E-Motors under test may have 200-1000V, max. 900A, 250kW. The company norm requires additional safety features included in the testbed system which are: voltage display, a door contact on the supply, and a customer specific IO that becomes active when the output voltage reaches a certain level. This level should be configurable via the *Automation_System*. Further, the company norm requires to have an US safety monitor system integrated into the supply

which replaces the default device. The current rise (t_{90} time) should be within 1ms and the Power Distribution Unit (PDU) capacitance is increased to 1mF. In order to allow stable operation. The systems are connected to the industrial 690V/60 Hz grid. The artefacts created for this example evaluation are shown below. Fig. 10 shows the EVTm. Each of the *System* elements (blue rectangle) has a corresponding CPM, Fig. 11 shows the CPM for Supply1. For each *SW_Assembly* in the EVTm a SCM model is generated Fig. 12 shows *SW_Assembly1* and Fig. 13 shows *SW_Assembly2*. Artefact 1-4 shows the generated output based on the EVTm, CPM and SCM models shown before. The graphical models are transformed into textual representations by using the transformation rules shown in chapter 4.

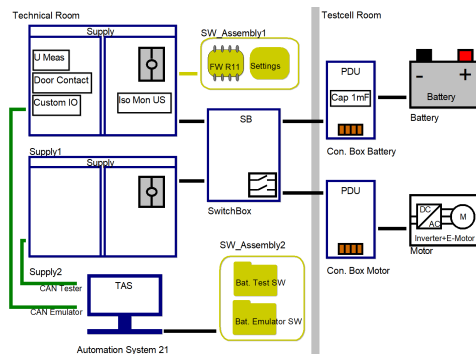


Fig. 10: EVTm Example.

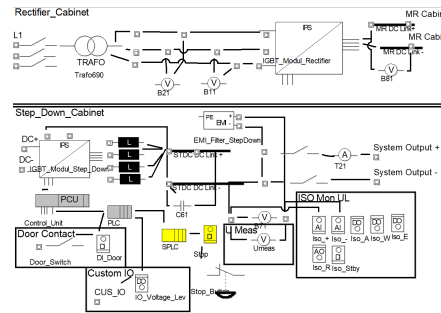


Fig. 11: Supply1 CPM.

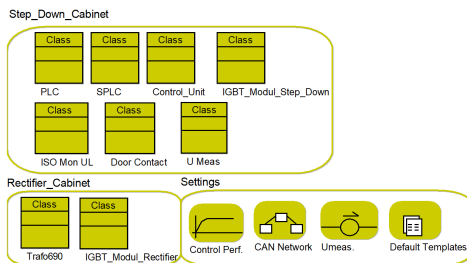


Fig. 12: SW_Assembly1 SCM.

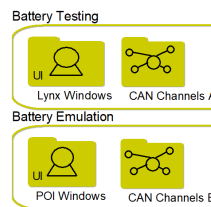


Fig. 13: SW_Assembly2 SCM.

```
WARNING_DI_DOOR, //IO warning
WARNING_TRAFO690, //Transformer temperature warning
ERROR_IGBT_MODUL_RECTIFIER, //IGBT Modul error
```

Artefact 1 Error-Codes.

```
0x11;Supply1 0x12;Supply2 0x0;SwitchBox
0x31;Con.Box Battery 0x32;Con.Box Motor
```

Artefact 2 Network Settings.

```

class Door_Contact{
public: bool get_IO_Door_Switch();
private: iDio* m_Door_Switch;};
class Custom_IO{
public: bool set_IO_Voltage_Lev();
private: iDio* m_Voltage_Lev;}
class Iso_Mon_UL{
public: Iso_Mon_UL(int16 *isoValue,
bool *enable);
virtual ~Iso_Monitor_US();
uint16 getValue();
void setEnable(bool en);
bool isWarning();
bool isAlarm();
private: iDio* m_Enabl;
iDio* m_Iso_A;
iDio* m_Iso_W;
enum{
MAXVAL = 3000, MINVAL = 50};};
59; 10 ; BE High Dynamic
76; 2.0E+00; REF_U_DELAY
77; 1.0E+03; SCALE_PWR
125; 2; PLANT_STRUCTURE
114; 0.00E+00; PLANT_RC1
117; 1.00E-03; PLANT_C2 (PDU)
50; 0.65; FEEDFORWARD IQ
8; 400; PWR_GRID_MX
7; -400; PWR_GRID_MN
36; 720; LIM_U_RMS_MX
94; 700 ; IT_MX

```

Artefact 3 Class definition.

Artefact 4 Control parameters.

The comparison with manually developed artefacts led to the following results:

- The network settings of the generator output are easier to read because it features the names of the systems/components as used in the EVT-Model.
- The generator output is more exact regarding control parameters, as in the manual version these were not adapted to the testbed structure (e.g. cable length).
- The generator produced its output faster as only a small set of parameters had to be set manually in the model, other parameters are derived from the CPM and while other parameters were extracted from the template files. As we do not have the recorded timing data from the manual development process, we can only assume the time for defining redundant information can be saved, resulting in increased productivity.
- The generated errors are more meaningful compared to the manual developed definition as (1) they contain additional error descriptions derived from the CPM and (2) the generated names are consistent to the names used in the hardware domain.
- The generated class definition of the header file provides variable, function and enumeration definitions. Again, the difference to the manual code is that the naming is consistent and the generated artefact ensures that each hardware component has a related definition in the code.

- The connection between the model and the generated artefacts is not straightforward.

To sum up, our evaluation experiments proved that the model-based generator approach increases productivity by reusing already defined information of the hardware domain. Regarding understandability we found out that the generator has a more consistent naming and adds additional comments to make the generated parts even better understandable. A further improvement in terms of understandability would be adding information in the generated artefacts how they are linked to the model elements.

7 Related Work

Related work addresses a variety of aspects regarding software design and generation for mechatronic systems. [SP09] analyzed “successful“ industrial mechatronic system development processes and observed quite a muddle: different notations are used, developers do not have an overview over the complete system functionality, activities with the same target are done several times using different procedures and tools. In several publications the collaboration among specialists and stakeholders with different backgrounds is addressed, by presenting a model-based approach [Ga10, Le08, Ba15, HRZ14]. In the following we will give an overview about domain-specific approaches. MechatronicUML [Ho16] focuses on Requirements Engineering and Model-Driven System Development for the control software of mechatronic systems. Compared to the EVT approach the MechatronicUML is complex, as it addresses the whole domain of mechatronic systems, where the paper focus on a special part of the mechatronic domain (EVTs). The EAST-ADL Architecture Description Language for automotive embedded systems focus is on in-vehicle systems, as it needs to be compatible with the AUTOSAR metamodel; it is not a “lightweight“ DSML compared to our solution [EA13]. Easy lab [Ba08] is a model-based programming tool which enables both the modeling of software and hardware functionality. The generator uses templates for each primitive element to produce code. Now we will have a look on the code generation literature, which is mainly driven by the software engineering community. [Ga10] addresses the following purposes of code generation: (1) separation of application logic from platform details (2) improved productivity, (3) improvement in the quality of the application, and (4) increased performance of the application by generating efficient code. Well known transformation approaches are “Direct Model manipulation“, “Intermediate representation“ and “Transformation language support“. For EVT systems we are using the “Intermediate representation“ technique where the intermediate artefact is represented by the RSC. [BSM14] implements the intermediate artefact by generating an XMI file. [Fe09] defines a code generation approach for railway systems and in [CGL17] a tool to uncover model transformation problems is presented. Mechatronic systems and especially EVT systems need application software and firmware. The software development of EVT systems is affected by the PLC manufacturers as they came up with powerful IDEs [Bu19, SI19]. Well known automation systems for EVT systems are, as example, PUMA, LYNX

[AV19] and KS Tornado [KS19]. These systems have been analyzed, as basis for designing the software configuration metamodel.

8 Conclusion

The analysis of the literature on mechatronic system design showed that one of the greatest challenges is to deal with the complexity of the collaboration between the engineering disciplines, including management [To07]. The common way to attack this challenge is a model-based approach supporting the requirement management process which we also apply here. The collaboration is enhanced by having an overview model (EVTM) that can be understood across domains and this is further refined in the engineering disciplines. With this solution domain experts can see the big-picture of the EVT and are not lost in details. Another aspect is to ensure a smooth integration of software and hardware design by providing proper tool support. Only few publications, however, address the issue of software generation for mechatronic systems. Such systems consists of hardware components, which interact with IOs that are connected to microcontrollers and PLCs. To program these devices object-oriented programming languages like C++ and C# gain importance, in addition C and other IEC languages are still used. Manufacturers of such hardware have a high impact on the development, as they often deliver the corresponding development environments. The seamless integration into these tools is one of the cornerstones that a generator solution is accepted.

The approach presented here builds on the work cited in chapter 7, especially the transformation approaches, the cross-references [Ga10] which define the software to hardware relation and the state-of-the-art automation systems. We showed, how to generate software configurations in the EVT domain out of hardware and software models that are represented in a DSML. The EVT domain served as an example, and we are convinced that it is also applicable in other mechatronic engineering disciplines. The work showed that it is really worth to investigate in DSML development and generator development because it results in increased productivity and quality. Compared to traditional approaches which use complex languages our approach is lightweight, which all the resulting benefits. To gain further feedback from the modeling community a version of the modeling tool has been made public available on the OMiLAB [OM19] repository. But there is still some work to do. As an example, the modeling language needs to be extended by concepts for formulating constraints over attribute values in order to allow for more pinpoint transformations. In addition the transformation rules have to be defined formal, including a definition of the general process to define such rules for the mechatronic domain. Also, we plan to enhance the tool to automatically generate circuit plans based on the developed CPM. In the end, the modeling environment should cover all the software and hardware aspects of a mechatronic system for a particular domain. Another interesting application would be to use the domain specific models of the mechatronic system as digital twin to train engineers and to monitor the state of the system.

References

- [AV19] avl.com, <https://www.avl.com/>, accessed: 20/09/2019.
- [Ba08] Barner, S. et al.: EasyLab: Model-Based Development of Software for Mechatronic Systems. In: 2008 IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications. IEEE, Beijing, China, pp. 540 – 545, 11 2008.
- [Ba15] Barbieri, G. et al.: Modelling and Simulation for the Integrated Design of Mechatronic Systems. IFAC-PapersOnLine, 48(10):75–80, 2015.
- [BSM14] Bardaro, G.; Sempredon, A.; Matteucci, M.: AADL for robotics: a general approach for system architecture modeling and code generation. JOSER, pp. 32–44, 2014.
- [Bu19] Automation Studio, <https://www.br-automation.com/en/products/software/automation-studio/>, accessed: 20/09/2019.
- [CGL17] Cuadrado, J. S.; Guerra, E.; Lara, J.de: Static Analysis of Model Transformations. IEEE Transactions on Software Engineering, 43(9):868–897, sep 2017.
- [EA13] EAST-ADL Domain Model Specification, https://www.east-adl.info/Specification/V2.1.12/EAST-ADL-Specification_V2.1.12.pdf, accessed: 20/09/2019.
- [ec19] Component Lead Times, <https://www.ecianow.org/>, accessed: 20/09/2019.
- [EP19] EPLAN Electric P8: Power für Elektroprojektierung und Engineering, <https://www.eplan.at/at-de/loesungen/elektrotechnik/eplan-electric-p8/>, accessed: 21/09/2019.
- [ES13] Eessaar, E.; Sgirka, R.: An Ontological Analysis of Metamodeling Languages. In: Information Systems Development. Springer, New York, pp. 381–392, 2013.
- [Fe09] Ferrari, A. et al.: Modeling Guidelines for Code Generation in the Railway Signaling Context. In: Proceedings of the First NASA Formal Methods Symposium. Moffett Field, California, 2009.
- [FK13] Fill, H.; Karagiannis, D.: On the Conceptualisation of Modelling Methods Using the ADOxx Meta Modelling Platform. Journal Enterprise Modelling and Information Systems Architectures, 8:4–25, 2013.
- [Fo18] 2-Prozent-Marke geknackt: E-Autos boomen - aber nur mit Zwang und Förderung, [focus.de](https://www.focus.de), accessed: 20/09/2019.
- [Ga10] Gausemeier, J. et al.: Computer-Aided Cross-Domain Modeling of Mechatronic Systems. In: INTERNATIONAL DESIGN CONFERENCE. volume 2, Dubrovnik, 5 2010.
- [Ho16] Holtmann, J. et al.: The MechatronicUML Requirements Engineering Method: Process and Language. Technical report, Software Engineering Department, Fraunhofer IEM & Software Engineering Group, Heinz Nixdorf Institute, Paderborn University, 12 2016.
- [HRZ14] Hackenberg, G.; Richter, C.; Zäh, M.: A Multi-disciplinary Modeling Technique for Requirements Management in Mechatronic Systems Engineering. Procedia Technology, 15:5–16, 12 2014.
- [KS19] Tornado Test Bed Prüfstandsautomatisierung., <https://www.ksengineers.at/de/Automotive-Testing/Management-und-Automation-Tools/Tornado-Test-Bed>, accessed: 20/09/2019.

- [KSK18] Kuhnert, F.; Sturmer, C.; Koster, A.: Five trends transforming the Automotive Industry. techreport, pwc.at, 2018.
- [Ku11] Kuester, J.: Model-Driven Software Engineering Code Generation. IBM Research, 2011.
- [Le08] Lennon, Tony: Model-based design for mechatronic systems. ELECTRONICS WORLD, 114:23–26, 05 2008.
- [Ma18] Mayr, H. C. et al.: A Model Centered Perspective on Software-Intensive Systems. In: Proc. 9th Int. Workshop on Enterprise Modeling and Information Systems Architectures. volume 2097, CEUR-WS.org, Rostock Germany, pp. 58–64, 2018.
- [MM15] Michael, J.; Mayr, H. C.: Creating a Domain Specific Modelling Method for Ambient Assistance. In: Proc. Int. Conf. on Advances in ICT for Emerging Regions ICTer2015. Colombo, 2015.
- [Mo09] Moody, D. L.: The “Physics“ of Notation: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering”. IEEE Transactions on Software Engineering, 35:756–779, 2009.
- [OM02] OMG: MetaObjectFacility (MOF) Specification. OMG, 2002.
- [OM19] Open Models Laboratory, <http://austria.omilab.org>, accessed: 20/09/2019.
- [PM19] Paczona, M.; Mayr, H. C.: Model-Driven Mechatronic System Development. In: 2019 IEEE 15th International Conference on Automation Science and Engineering (CASE). IEEE, Vancouver, Canada, pp. 1730–1736, aug 2019.
- [PMP19] Paczona, M.; Mayr, H. C.; Prochart, G.: Model-based Testbed Design for Electric Vehicles. In: Proc. EMISA 2019, Lecture Notes in Informatics, GI. Tutzing, 2019.
- [Ro11] Roussey, C. et al.: An Introduction to Ontologies and Ontology Engineering. In: Ontologies in Urban Development Projects. Springer-Verlag, London, pp. 9–38, 2011.
- [Ro17] Roos, G.: Power Semiconductor: Lead Times Continue to Stretch. EPSNews, 2017.
- [SI19] PLC Programming with SIMATIC STEP7, <https://new.siemens.com/global/en/products/automation/industry-software/automation-software/tia-portal/software/step7-tia-portal.html>, accessed: 20/09/2019.
- [SK03] Sendall, S.; Kozaczynski, W.: Model Transformation: The Heart and Soul of Model-Driven Software Development. Software, IEEE, 20(5):42 – 45, 10 2003.
- [So19] SOLIDWORKS 3D CAD, <https://www.solidworks.com/product/solidworks-3d-cad>, accessed: 21/09/2019.
- [SP09] Stetter, R.; Pum, U.: PROBLEMS AND CHALLENGES IN INDUSTRIAL MECHATRONIC PRODUCT DEVELOPMENT. In: International Conference on Engineering Design (ICED 09). Palo Alto, California, 2009.
- [To07] Tomiyama, T. et al.: Complexity of Multi-Disciplinary Design. CIRP Annals, 56(1):185–188, 2007.
- [Ve19] Vennes, J.: Coopert Transformer Lead Times. techreport, boarderstates.com, 2019.
- [Vo13] Voelter, M.: DSL Engineering Designing, Implementing and Using Domain-Specific Languages. CreateSpace Independent Publishing Platform, 2013.