Simulation-Based Development of Safety Related Interlocks

Timo Vepsäläinen and Seppo Kuikka

Tampere University of Technology, Department of Automation Science and Engineering
P.O. Box 692, FIN-33101 Tampere, Finland
{timo.vepsalainen,seppo.kuikka}@tut.fi

Abstract. Dynamic simulations could support in several ways the industrial automation and control systems development, including their interlocking functions, which constitute an important and tedious part of the development. In this paper, we present a tool-supported, automated approach for creating simulation models of controlled systems and their interlocking functions based on UML AP models of control systems and ModelicaML models of the systems to be controlled. The purpose of the approach is to facilitate manual development work related to model-based development of control systems and to enable early testing and comparison of control and interlocking strategies. The tools and the techniques are demonstrated with an example modelling project and the paper also discusses extending the approach to verifiable safety systems including their security aspects.

Keywords: Model-based development, UML AP, Simulation, Industrial control, Interlocks, Safety.

1 Introduction

Model-based development of software applications and systems has recently been the topic of numerous publications in different application domains, including software engineering and industrial control. Due to these interests, there already exist guidelines, languages and tool sets for implementing such approaches. For example, Object Management Group (OMG) has pioneered in standardization of model-based development approaches (Model-Driven Architecture, MDA) and languages for modelling (UML and profiles e.g. SysML), metamodeling (Meta Object Facility, MOF) and transforming (Query/View/Transformation, QVT) purposes. The modelling and transformation languages are already mature and supported by different tool vendors on several platforms, such as the open source Eclipse platform.

The idea of Model-Driven Architecture (MDA) and related approaches, e.g. Model-Driven Development (MDD) and Model-Driven Software Development (MDSD) is to use models (instead of documents) as primary engineering artefacts during the development. In the systems engineering domain, Model-Based Systems Engineering (MBSE) refers to applying models as part of the systems engineering process with the aim to support analysis, specification, design and verification of the systems being developed [5].

N. Pina et al. (Eds.): *Simulation & Modeling Methodologies, Technologies & Appl.*, AISC 197, pp. 165–182.
DOI: 10.1007/978-3-642-34336-0_11

In model-based development processes, models are refined towards executable applications by use of model transformations but also manual development work with the models. Such processes often enable automated processing of bulk design information and are aimed at automatic code generation but can also aid analysis, understanding and documentation of the system.

In addition to analysis of models and automating error-prone development phases, another approach to improve the quality of systems and applications could be to integrate the use of simulations to model-based development. Especially, simulations could be used to facilitate the manual development work of developers by enabling, for example, comparisons of alternative design decisions. In their previous work, the authors of this paper have created and prototyped a preliminary approach to transform functional models conforming to the UML Automation Profile (UML AP, see [12],[6]) to simulation models conforming to ModelicaML [13]. The concept was presented in detail by Vepsäläinen et al. [19] and its purpose is to facilitate control system development by enabling automated creation of simulation models of controlled manufacturing systems.

In the process, the simulation models of controlled systems are composed by creating and integrating a ModelicaML simulation model of the control system to an existing ModelicaML model of the process to be controlled. The focus of the paper was in basic control functionality, e.g. feedback and cascade control structures, and the ability to support simulation of both platform independent and platform specific functions. However, according to, for example, our discussions with professionals of industrial control domain in Finland, an important and tedious part of development of control applications is related to interlocking or constraint control functions.

Interlocks could be characterized as non-safety-critical safety functions. They are often aimed to prevent deviation situations from occurring or the instrumentation from being misused, such as, to prevent pumps from running dry or to be started against closed pipelines. Interlocks do not need to be developed according to safety standards because safety is usually ensured with separate safety systems. However, because actual safety systems are often designed to ensure the safety in a simple manner, e.g. to shut down the whole processes, they should not be activated unless absolutely necessary. Another goal of interlocks can thus be seen in keeping the system in its designed operating state in order to improve the availability and productivity of it. To achieve this goal, interlocks can be more complex than actual safety functions because they do not need to meet the strict requirements of safety standards.

The development of interlocks is, however, difficult. This is because of both the complexity of the functions and because they are specific to applications and thus cannot be re-used similarly as, for example, control functions (e.g. parameterizable function blocks implementing control algorithms) can be. The actual logic, how to keep the system in its designed operating state and protect the devices, is dependent on both the controlled process and the control approach for controlling the plant or process. Another reason for the difficulty is that interlocks may originate from several sources. For example in industrial processes, part of the interlocking needs may originate from process design whereas others originate from hydraulics and electrics design. Because of the separate sources, they may have unpredictable cross-effects to the controlled system. This is another good reason for using simulations.

In this paper, we aim to extend our approach to automatically generate simulations to cover and facilitate the development of interlocking functions. We present a modelling framework supporting the modelling of the functionality of interlocks and how a simulation model of a controlled system can be created using model-based techniques. The paper also discusses the relationship between safety functions and interlocks with the purpose of assessing whether also the development of critical safety functions could be based on modelling and simulations. For defining interlocks, we do not suggest any new modelling notation. Instead, we integrate a commonly used notation to our model-based approach. The novelty of the approach is, thus, not in the way of specifying the interlocks but in the way in which simulations are integrated to model-based interlock development and how the simulation models can be created based on early design models.

This paper is organized as follows. Section 2 reviews work related to use of simulations and model-based development in industrial control and automation domain. Sections 3 and 4 present a more detailed introduction to interlocking functions, our approach to simulation-assisted development of interlocks and the developed tool support, respectively. Section 5 presents an example modelling project in which the approach and tools are utilized. Finally, before concluding the paper, section 6 discusses whether model-based, simulation assisted development techniques could be used in development of actual safety functions and to reveal security-related problems.

2 Related Work

Simulations can facilitate the development of manufacturing processes, machines and plants as well as automation and control systems in several ways. For example, Karhela in [9] mentions the use of simulations to control system testing, operator training, plant operation optimisation, process reliability and safety studies, improving processes, verifying control schemes and strategies, and start-up and shutdown analyses.

In [3] the author compares the I/O simulation approach to the traditional approach of performing system testing only on-site with the actual processes. According to the paper, the use of simulations may result in shorter start-up times as well as less waste of end products during the start-ups. In addition, simulations enable better operator training, ability to test control programs in smaller modules, and the ability to thorough testing of emergency and dangerous situations. [3]

A more recent survey on use of simulations in industrial control domain was made by Carrasco and Dormido in 2006 [2]. According to the paper, the benefits of using control systems in simulators before installation include improvements to 1) design, development and validation of the control programs and strategies, 2) design, development and validation of the HMI (human-machine Interface) and 3) adjustments of control loops and programs. [2] It is thus evident that simulations may facilitate both the development and commissioning of control systems. Simulation solutions are nowadays also provided by major control system vendors as listed in [2].

The goal of our approach is to enable automated utilization of design-time models of control systems and applications so that, for example, early simulated testing of a control or interlocking approach would not need the actual control system hardware

or tools and fully setting the system parameters. Later in development, the same techniques could enable testing and validating larger entities. Development of simulation models could be less tedious and they could be utilized also by companies performing out-sourced development phases. In our approach, we assume that a simulation model of the process to be controlled is already available. In creation of a simulation model of the controlled system including both the parts of the control system and the controlled process, we utilize model transformations that are commonly used in model-based development approaches, such as MDA of OMG.

Model-Driven Architecture (MDA) is an initiative of OMG that encourages the use of models in software development as well as re-use of solutions and best practices. MDA identifies three types of models which are Computation Independent Model (CIM), Platform Independent Model (PIM) and Platform Specific Model (PSM). [10]

In MDA, the development starts from CIM models and proceeds to PIM models and finally to PSM models which are the most detailed ones and often source models for code generation. In our approach, the focus is in PIM and PSM models with the goal of being capable of utilizing both PIM and PSM models in creation of simulation models. Thus, for example, a preliminary (early) simulation model could be created based on PIM and used for evaluating control strategies. Later, after selection of the control system vendor, the model could be refined to PSM level and simulated in conjunction with vendor specific functions in order to obtain more precise results.

In addition to our approach (see [20] and [6]), the use of model-based techniques in the automation domain has been recently proposed by several projects and papers. However, not all of these approaches identify and highlight simulation as an essential and beneficial part of development. The approach of the MEDEIA project, as discussed by Strasser et al. [15] and Ferrarini et al. [4], is based on Automation Components - composable combinations of embedded hardware and software including integrated simulation, verification and diagnostics services. In their approach, the simulation of models will be based on their interfaces, behaviour and timing specifications using IEC 61499 as a basic simulation model language [14].

Another application of model based techniques to development of industrial control applications has been presented by Tranoris and Thramboulidis [16]. In their approach, the design and deployment of applications is addressed by means of the function block (FB) construct of IEC 61499. Model transformations are used to create function block models. In the paper, they dont address simulations but similarly to the MEDEIA approach, FB models could possibly be used with simulations of the process to be controlled.

In both the approach of MEDEIA and that of Tranoris and Thramboulidis, simulations could be supported with the implementation technology (IEC 61499) of produced applications. The essential difference to our approach is that we aim to support simulation with a simulation language so that, for example, basic simulation functions of simulation tools could be fully exploited. These functions are listed in [2] and include saving and loading current and initial states, freeze, run and replay simulation, working in slow and fast mode and support for malfunction situations.

Furthermore, we identify simulation as a beneficial and important activity also in case of model-based development. We claim that also model-based development

requires manual work and genuine design decisions made by developers because it may not be possible to express all the relevant aspects in models and all the relevant knowledge about decision making in model transformations. To facilitate the manual design work, we foresee that simulation techniques could provide a feasible solution and that model-based techniques could facilitate the creation of the required simulation models.

Similarities between interlocks of basic control system and safety functions of safety systems are remarkable. The main difference is that actual safety functions need to be developed according to safety standards, such as IEC 61508 [7], which may require a sophisticated development process, use of techniques recommended by the standards and a detailed documentation about the system and the development activities used. In their recommendations, standards are always conservative which may be one reason why the use of model-based techniques in safety system development has been unusual in the past. However, according to the present (second) edition of IEC 61508, automatic software generation could aid the completeness and correctness of architecture design as well as freedom from intrinsic design faults. Hence, the use of model-based techniques in development of also safety-critical applications may be increasing in near future. The question of how to develop safety-critical systems with model-based techniques is thus both important and current but not addressed by many researchers, so far.

However, Biehl et al. [1] have attempted to integrate safety analysis to model-based software development in automotive industry in order to automate performing of safety-analysis on refined models with minimal effort. In [21] the authors have extracted the key safety-related concepts of RTCA DO-178B standard into a UML profile in order to use them to facilitate the communication between different stakeholders in software development.

3 Simulation of Interlocking Designs

The focus of this paper is in interlocking (or constraint control) functions of basic control systems, which are an important and challenging part of control system development. Interlocks are control functions, the purpose of which is to either guarantee the safety of the process or to keep the system in its designed operating state and protect the devices and actuators from being misused by the control system. Quite often, safety is achieved with a separate safety system so that the purpose of the interlocks is the latter one.

Interlockings are typically designed during the basic design phase of the control systems [20]. The amount of program code, related to interlockings is often smaller than that of code related to basic control functionality. However, their development is still time-consuming and prone to errors because interlocks cannot be reused similarly as, for example, controllers can be. This is due to the fact that the actual interlocking needs, logics and delays are always specific to the application. Solutions to re-occurring needs in controlled processes can be librarized but even they need careful examinations and potential modifications before re-use.

For industrial systems, interlocks are often specified with vendor neutral logic diagrams - or vendor specific logic and function block diagrams if the control system

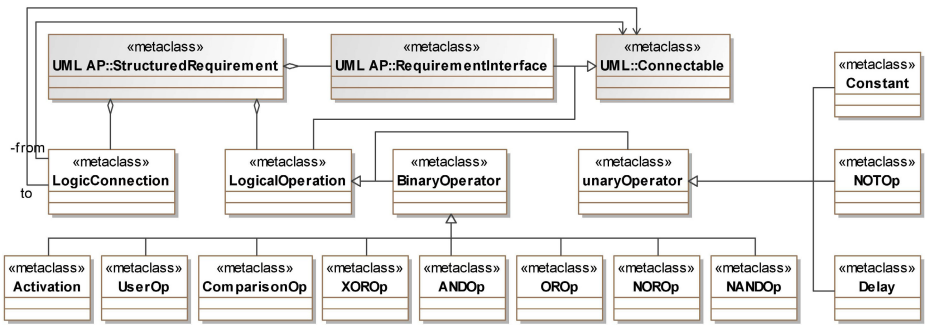

Fig. 1. The essential additions to UML AP metamodel to support the definition of interlocks

vendor has been selected. In the process, the diagrams are used for depicting the activating and disabling conditions of the functions, and possibly overriding control values for locked actuators or devices. Logic diagrams suit well to this purpose because they are familiar to developers and unambiguous. Logic diagrams, as a semi-formal method, are also highly recommended by IEC 61508 to detailed design of safety-critical software [7]. Logic diagram based approach for defining the interlocks is thus both sound and already familiar to developers of the domain.

The purpose of UML AP is to cover both the specification of requirements and functionality of automation and control applications. Logic diagrams may aid in supporting both of these features but used from separate points of view. Especially, in the development of safety-related applications, requirements must be defined clearly and in an unambiguous manner. On the other hand, formal or semi-formal specification of functionality is a necessity in enabling simulation of design or in automating generation of code. In our approach, the logic diagram concepts were added to be used with both the Requirements Modelling sub-profile and functional Automation Concepts sub-profile of UML AP and the UML AP tool (see [17]). The concepts and some related existing modelling concepts of the profile are presented in figure 1. Existing UML AP and UML metamodel elements are highlighted with grey colour.

In UML AP, requirements are structured concepts that can be connected to other requirements with port-like requirement interfaces in order to model dependencies between required functions. The purpose of the logical operations and logic connections, on the other hand, is to enable the modelling of required activations of interlocks and algorithms to compute control values inside requirements. Required interchange of computed signals and values can then be modelled with the requirement interfaces that extend the same UML::Connectable concept than logical operations and can be thus connected together with logic connections. The operations include familiar operations, such as AND and OR, but also delay, constant, Activation gate (that lets its input flow to output when control input is activated), comparison operator and a UserOperation with which the developer can specify the logic to output from inputs with a textual equation. Examples of use of part of the concepts will be provided in section 5.

The functional modelling concepts of UML AP, Automation Functions, constitute a hierarchy of function-block-like concepts. The hierarchy is based on their purpose,

such as to execute control algorithms, compute interlocking signals or to interface with sensors or actuators of the system. The hierarchy including its justification is presented in detail in [6]. Automation Functions (AFs) exchange signals between them with ports that extend the UML::Connectable concept (see figure 1). The logic operators and connections, on the other hand, can be used inside the AFs to define the functionality of them. In the profile and tool implementation this was enabled by adding an aggregation association from the Automation Function base metaclass to logical operation and logic connection metaclasses similarly to the structure presented in figure 1. Consequently, the technical challenges of our approach to simulate the models are in transforming the specifications conforming to UML AP to simulation models. The solution to transform the models to ModelicaML models and finally to simulateable Modelica models will be discussed in next section.

4 Technical Background and Implementation of the Approach

It is first necessary to present some basic information about Modelica and ModelicaML that are used in our approach as target simulation languages. Modelica is an object oriented simulation language for large, complex and heterogeneous physical systems and a registered trademark of Modelica Association. Modelica models are mathematically described by differential, algebraic and discrete equations. Modelica includes also a graphical notation and user models are usually described by schematics that are also called object diagrams. A schematic consists of components, such as pumps, motors and resistors, which are connected together using connectors (ports) and connections. A component, on the other hand, can be defined by another schematic or, on the lowest level, as a textual equation based definition.

Modelica Modeling Language (ModelicaML) has been created to enable an efficient way to create, read, understand and maintain Modelica models with UML tools [13]. ModelicaML is a UML profile and defines stereotypes and tagged values of stereotypes that correspond to the keywords and concepts of the textual Modelica language. For example, a Modelica block with a set of equations can be modelled by creating a UML class, applying a <<block>> stereotype to it and defining the equations to the equations tagged value of to the stereotype.

ModelicaML models are not simulateable as they are (at least with current tool support) but can be transformed to simulateable Modelica models. Tool support for generating textual Modelica models, as well as the profile, is made publicly available by the OpenModelica project. [11] The profile is based on UML2 implementation of the UML metamodel on the Eclipse platform. UML2 is itself based on Eclipse Modeling Framework (EMF) which is an implementation of OMG Meta Object Facility (MOF) specification.

EMF is also utilized in our UML AP metamodel implementation that is the basis of the UML AP Tool [17]. UML AP and ModelicaML models are thus instances of metamodels defined with EMF implementation of MOF and because of this similar background, the shifting between UML AP and ModelicaML can be realized with use of standardized QVT languages. QVT languages are intended for defining model transformations between models conforming to MOF based metamodels and they are also

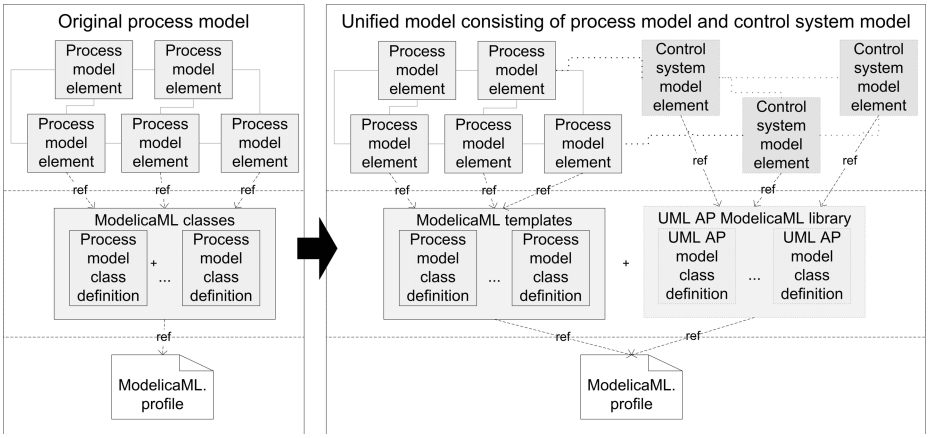

Fig. 2. The purpose of the transformation is to add the control system specific parts to an existing model of the physical process

specified by OMG. The possibility to use standardized transformation languages with existing open source tool support and the open source background of Modelica and ModelicaML are good reasons for selecting Modelica as the target simulation language in our approach.

In Modelica (and in ModelicaML) simulation classes are defined separately from their use context, similarly to classes in object oriented programming languages. In ModelicaML models, the model elements also need to reference the ModelicaML profile in order to use the stereotypes and tagged values of it. This results in a structure sketched in the left side of figure 2. ModelicaML models consist of Modelica class definitions and instances of the classes. Classes may contain ports with which they can be connected and both the definitions and instances of the classes need to use the stereotypes of the ModelicaML profile in order to map the concepts to Modelica keywords. In our approach, we assume that ModelicaML models of processes to be controlled are available and conform to this structure.

The purpose of the transformation is to create and add the control system specific parts to the existing model of the process to be controlled and to connect the created parts to the existing model so that the controlled system can be simulated. In this process, Modelica class definitions corresponding to platform independent (PIM) and platform specific (PSM) UML AP elements are copied to the model of the process to be controlled (process model) so that they can be referenced from the process model. Instances of the templates are instantiated to the process model and connected together according to the control system model in the UML AP tool. Instances corresponding to measurement and actuation AutomationFunctions are connected to the elements of the process model that are used to model sensors and actuators.In more detail, this process and the characteristics of different kind of AFs are discussed in detail in [19].

However, because interlocks are specific to applications they cannot be librarized, as explained earlier. Instead, the definitions of interlock classes need to be created by the transformation based on the logic diagrams. This process is rather simple and illustrated

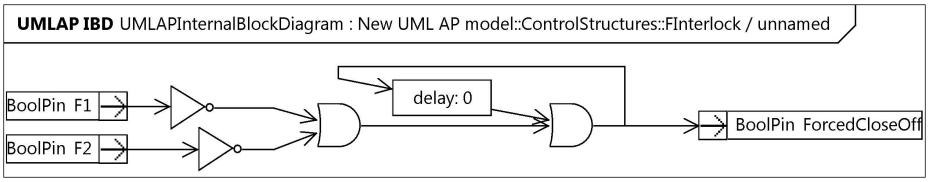

Fig. 3. Simple example of an interlocking function

with an example shown in figure 3. Ports contained by classes, such as interlocks, are special kind of classes in Modelica and finally typed by type definitions in ModelicaML profile. When creating ModelicaML classes based on UML AP classes, instances of such special port classes can be created based on ports used in the UML AP model so that their naming will be maintained and the suitable type chosen based on the type of the port. This applies to both input and output ports.

Figure 3 contains only three kinds of logical operations of the 11 presented in figure 1: two NOT and two OR operations and one delay. The transformation processes logical operations by creating a property (variable) for each operation instance. In case of Boolean operations (NOT, AND, NAND, OR, NOR, XOR), the type of the property is always Boolean. In case of other operations, the type needs to be defined in the UML AP model so that the corresponding ModelicaML type can be chosen by the transformation. The equations determining the values of the properties are created based on the kind of the operation (for example NOT or AND) and the connections coming into the operation which can be followed to another operation or port, for which there will also be a property (variable) with the same name. In case of the example interlock presented in figure 3, the value of the first OR operation (from left in the figure) can be defined to be equal to the logical OR of the values of the NOT operations and the second OR operation to equal to the logical OR of the first OR operation and the delay operation.

The transformation, thus, tries to define the values of properties with equations. However, if a model contains loops, this may not be possible. For example, figure 3 contains a loop the purpose of which is to keep the interlock activated if it once activates so that the output of the second OR operation (from left) is true. Certain kinds of loops may produce errors due to discontinuation of the variables, at least with the OpenModelica tool that we use for simulating. The problem was solved by using algorithms in which operations are applied in an order (instead of equations that apply all the time). This is also one of the interactive features of our transformation. If the transformation detects a loop within an interlock or other kind of AF, it creates algorithmic statements instead of equations based on the model, shows the statements to the user of the tool and lets the user arrange the order in which they will be executed in the model.

Another interactive feature of the transformation is related to connecting parts of the simulation model created based on the UML AP control system model to the existing parts of the process to be controlled. These connections are necessary for, for example, connecting measurement and actuating functions of control systems to sensors and actuators of the process models, respectively. By default, the transformation uses

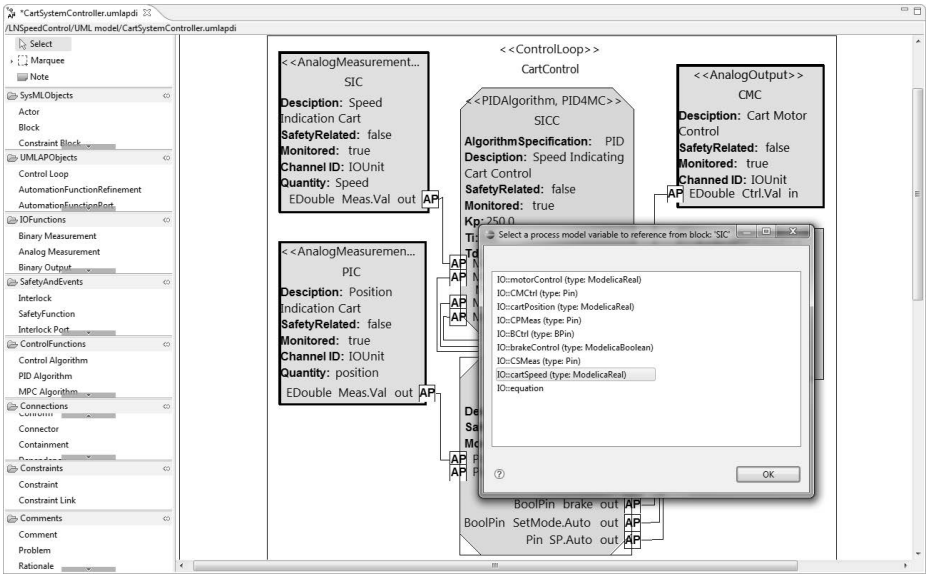

Fig. 4. The graphical user interface of the UML AP Tool for selecting the correct property to reference from the model of the process to be controlled

properties of the process model with specific names or the names of properties that have been specified with a specific <<VariableMapping>> stereotype. However, if suitable properties cannot be found by the transformation, it provides the user of the tool with a list of properties available in the model class in question and lets the user choose the correct property as in figure 4.

The third interactive feature is related to non-connected input ports. When an unconnected input port is detected by the transformation, the user of the tool is asked for a constant value for the port. In this case, the user of the tool may define a constant value for it in order to be able to simulate the model or, if the port has been left unconnected unintentionally, leave the port unconnected and fix the problem before executing the transformation again.

The transformation definition was written with QVT operational mappings language and it specifies how to process a target ModelicaML model based on a source UML AP model. Executable Java-transformation code implementing the definition and to be used in the Eclipse environment was generated with SmartQVT tooling. In order to be able to launch and control the transformation from the UML AP Tool, the Java class was packaged to a plugin defining an extension to one of the extension points of the tool. The structure of the plugin was similar to the plugin structure presented in [18]; the referred paper also presents in detail the extension points of the tool.

In the plugin structure, the generated transformation class is extended with an own (hand-written) transformation class that can be used to implement also required black box operations. In QVT vocabulary, black boxes are operations written with common programming languages (in this case Java) in order to implement operations that are

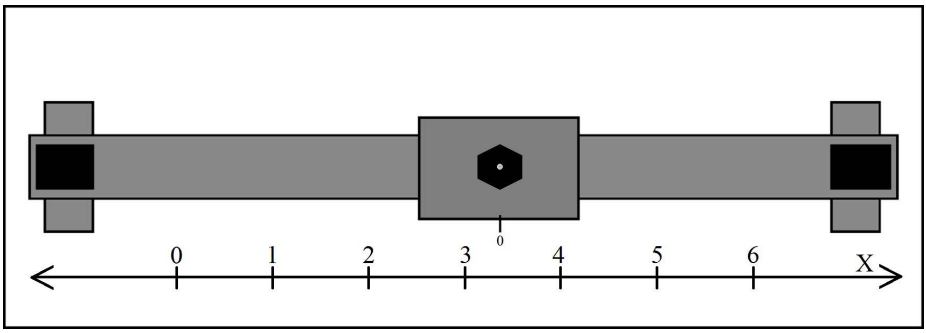

Fig. 5. Simple example system to be controlled includes a cart that can be moved along a rail

hard to express with QVT concepts. In this case, for example handling of stereotypes and tagged values related to them as well as interactive features of the transformation were implemented as black boxes.

5 Example System and Utilization of the Tools

The purpose of this section is to provide a simple example in which the modelling concepts and tools are used in creation of a simulation model of a controlled process to evaluate two alternative interlocking approaches.

An illustration of the (partial) system to be controlled is shown in figure 5. The system consists of a cart and a rail along which the cart can be moved with an electric motor. The cart can be stopped with a brake, if necessary. The purpose of the cart is left unspecified and not illustrated in the figure. It could be assumed, for example, to operate a boom or a gripping device nearby the rail. The control needs to be addressed in the example are related to controlling and interlocking the velocity and location of the cart. The operator of the system controls the system by giving speed requests (setpoints) with a joystick that is connected to the control system. The control system, on the other hand, is required to control the velocity of the cart with feedback control and to protect the cart from colliding to stoppers at the end of the rail. The location of the cart must be kept between 0.0 and 6.0.

In industrial installations the need for a similar stopping interlocks could be caused by, for example, forbidden areas in factories and sites or needs to restrict operation ranges of booms and devices of mobile platforms to ensure their stability in varying terrains. Consequently, despite the simplicity of the example, it can be considered as a generalization of a common functionality in industrial systems.

There are several ways in which the functionality could be implemented; however, in this paper we will sketch and simulate only two simple versions of them. Firstly, the control system could observe the location and direction of the cart and stop it with the brake, if the cart violates the limits. For example, when reaching coordinate 0.0, the control system could activate the brake and keep it activated until the speed request would be towards back to the allowed area. Secondly, the control system could

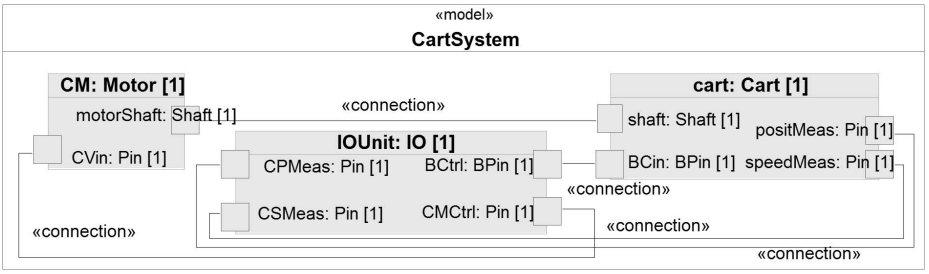

Fig. 6. Model of the system to be controlled as a ModelicaML model, composite structure diagram

be designed to constrain the speed setpoint near the limits so that the setpoint would be zero at the limit coordinates and it would be reduced already before reaching the limits. These approaches will be simulated next based on a ModelicaML model of the process to be controlled and UML AP models of the two control approaches.

To be able to utilize the tools and techniques presented in this paper, the system to be controlled need to be available as a ModelicaML model. The UML composite diagram presenting the simplified model of the system is presented in figure 6. The model was specified with open source Papyrus UML tool, with OpenModelica extensions, and it consists of 3 ModelicaML components that are instances of ModelicaML classes. The cart is operated with a motor (CM) that takes its control signal from the IOUnit that collects all measurement and control signals between the process and control system. The total weight of the cart and motor is assumed to be 20kg (m_{total}) and the radius of the drive wheel 0.1m (r_{dw}). The torque (T) and acceleration (a) equations of the motor and cart based on drive voltage (V_d) are presented in equations 1, 2 and 3. The numerical values of the constants of the motor are: $R_m = 0.5$, $L_m = 0.0015$, $K_{emf} = 0.05$ and $K_t = 0.01$. The brake is assumed to be able to decelerate the cart with force of 200N (Fb). The equations are, thus, simple but sufficient for demonstration purposes.

$$V_d - \omega * K_{emf} = L_m * dI/dt + R_m * I \tag{1}$$

$$T = K_t * I \tag{2}$$

$$T/r_{dw} + F_b = m_{total} * a \tag{3}$$

The UML AP control structure diagram presenting the similar parts of the two control solutions for the system is depicted in figure 7. The control solution consists of analogue measurements of cart position and speed, an interlock, a PID controller and an analogue and a binary output for controlling the motor and the brake, respectively. The two interlocking approaches discussed earlier influence mainly the contents of the interlock. In the first approach, the speed request (setpoint) is not constrained. However, in order to enable that to be implemented in the second approach, the speed request is relayed through the interlock AF.

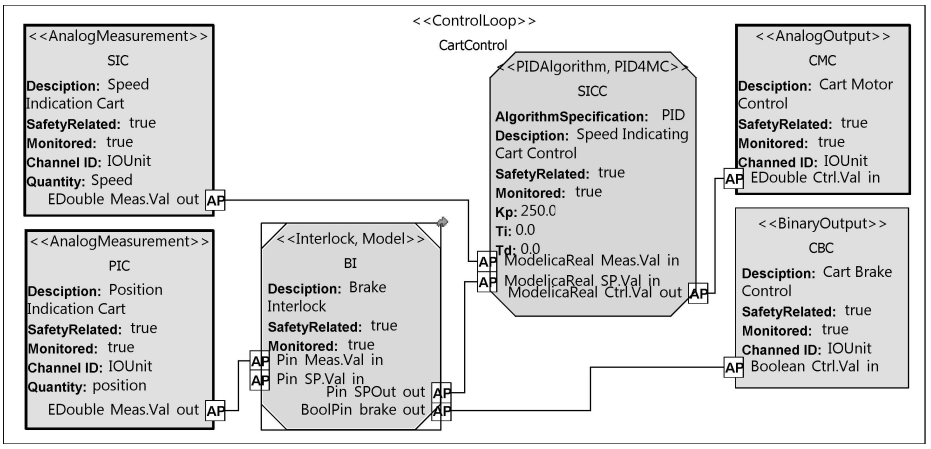

Fig. 7. UML AP control structure diagram of a control solution for controlling the process

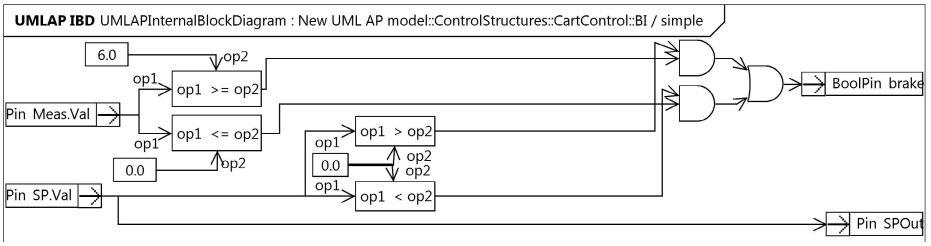

Fig. 8. An illustration of the first interlocking solution

The detailed logic of the first interlocking solution is presented in figure 8. The solution is designed to activate the brake outside the intended working area if the speed request is driving the cart away from the working area. In order to be able to revert back to the working area, the brake is not activated if the speed request is towards the allowed working area. In order to work well, the system should probably also wait for the cart to stop before deactivating the brake; however, because of simplicity this feature was not modelled.

After specification of the detailed control solution, the transformation, discussed in section 4, was used to transform the UML AP control solution to ModelicaML and to append it to the existing model of the physical process (see figure 6). In order to simulate the model, the ModelicaML model was further transformed to Modelica code with OpenModelica tooling. The shifting from UML AP model of the control solution to simulateable model of the system including both the process and the control system was, thus, automated with two model transformations.

The simulation result related to the first interlocking approach is presented in figure 10a. At the beginning, both the position and velocity of the cart are 0. The speed request (from the operator) is ramped from 0 to 1 and kept at 1 for 7 seconds in order to drive

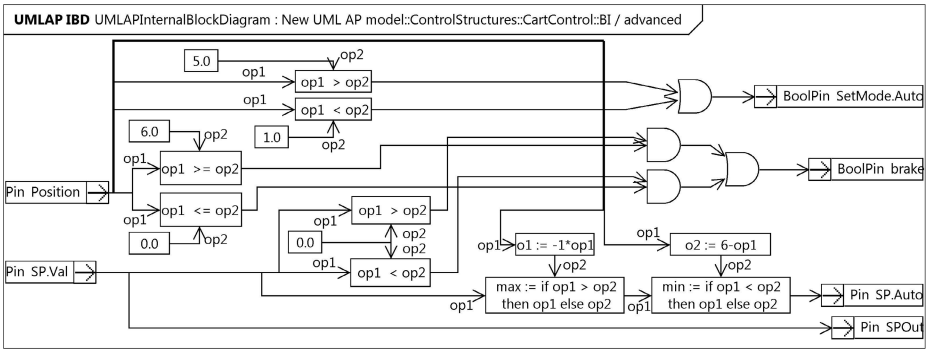

Fig. 9. The second interlocking solution

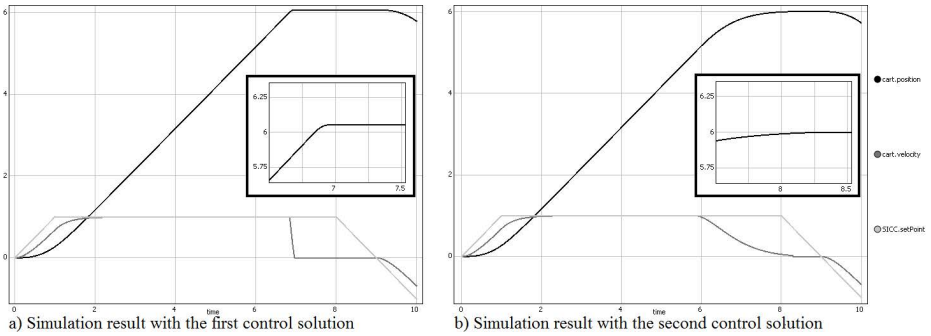

Fig. 10. Simulation results of the first (a) and second (b) control solutions plotting cart position, velocity and speed request from the operator (SICC.setPoint)

the cart to the forbidden area in the positive end of the rail. After this, the speed request is ramped to -1 in order to revert the cart away from the border of the forbidden area. According to the simulation, the control solution works as it was intended; however, because it takes time to stop the cart, the location of the cart reaches approximately 6.05 before stopping. Clearly, the control solution could be improved by decelerating the cart already before reaching the limits. In case of strict limits, violating the forbidden area (overshoot) could also be avoided by activating the brake before the strict limits; however, this would reduce the size of the actual, allowed working area.

The second control solution is illustrated in figure 9. In this solution, the braking is implemented similarly to the first solution and the speed request from the user is relayed similarly to the controller. However, when the measured location of the cart is between 0 and 1 or between 5 and 6, an automatic mode is activated and another speed setpoint is calculated by the interlock function. The setpoint is constrained so that between 0 and 1 and between 5 and 6, the maximum allowable speed setpoint towards the forbidden area is equal to the distance left to the forbidden area. For example, if the location of the cart is 5.5, the maximum allowed speed setpoint to the positive direction is 0.5. In order to relay the second, calculated setpoint signal and the mode activation signal, the interlock

AF has been added two new ports. Similar ports were added also to the controller block (see figure 7) and its equations.

The simulation result related to the second, improved interlocking AF is presented in figure 10b. The speed request obtained from the operator is similar to that of the first simulation. In this case, the cart is smoothly decelerated already before reaching the limit and the overshoot is much smaller than that in the first simulation. Clearly, this alternative provides a better control performance as even in case of strict safety limits, the cart would not need to be stopped with the brake before the actual limit.

6 Towards Development of Certifiable Safety Functions

The use of model-based techniques in development of safety-critical applications has not been recommended by safety standards, such as IEC 61508, until recently. However, due to the new version of the standard, they could be used, for example, during architecture design to aid the completeness and correctness of design as well as freedom from intrinsic design faults.

Perhaps the most essential difference between the development of safety systems and basic control systems is that safety systems require extensive documentation about all the development activities and their results for verification, validation and certification purposes. The development of safety systems should also be risk-driven so that the requirements and design artefacts could be always traced to the perceived safety needs which originate from risk and hazard analysis. When assessing the quality of design, design artefacts could be compared to the original safety needs and when in doubt, developers could always refer to the risks and hazards.

We are currently striving to extend the scope of UML AP to cover also the development and design of safety systems. The work is targeted to the requirement concepts of the profile (see [6]) but also to documenting the results of risk and hazard analysis. Including the risk and hazard information in a compact form in the same models with the design would not only enable the use of explicit traces between them but also aid the discovery of the information when the developers need it. Moreover, compared to use of separate documents, a unified (one) model could be easier to maintain and keep up-to-date if and when something needs to be changed in design.

An admitted difficulty in development of both safety critical and basic control systems is related to the specification of requirements. In development of safety-critical applications, the functional requirements (what the system must do) originate from hazard analysis and the non-functional requirements (how well it must be done) from risk analysis. However, unambiguous and complete specification of the functional requirements is still difficult. Perhaps this task could be facilitated with a semi-formal, domain specific modelling approach. Another working direction is the ability to simulate designs and specifications.

In [8] the author has analysed the quality of produced software in about 12500 projects from year 1984 to 2008 and the defects delivered (and removed) during the projects. The results may not be directly generalizable to safety-critical applications as such. However, according to the survey, also in the best-in-class-quality, a significant portion of defects delivered were related to defects in requirements specifications, partly because defects in requirements are difficult to discover.

If the design could be simulated earlier, for example with the techniques presented in this paper, simulations could also be used to assess whether the required functionality is able to detect and handle the hazardous situations. The feedback loop between design and requirements could thus be shortened. This could further facilitate the development of both basic control and safety-systems.

Testing or simulation-aided testing of design and development specifications cannot be used to prove the correctness of them. However, simulations can be used to test the reactions of control or safety systems to events in the system that could not be tested with the actual system without compromising safety. Moreover, simulations may aid in comparing alternative solutions in terms of, for example, availability of the controlled system that may be important from the point of view of the developer organization or the end user but not that of safety standards. Extensive testing is also required by safety standards. The problem with conventional testing is that the system should be already implemented in order to be tested. If both the simulation model and the executable application would be produced by trusted, automatic transformations based on the same model, testing could as well utilize the simulation model of the application. Consequently, with our approach, an improvement would also be the ability to test earlier in the development process based on platform independent models.

Another issue that must be increasingly considered in industrial control systems in future is security, as demonstrated by the recent Stuxnet worm. It can be easily justified that compromising security may lead to compromising safety, for example if a safety-critical, measured value is lost or modified. However, security is hardly mentioned in safety standards such as IEC 61508. We are expecting a change in this in near future. Security issues could also be taken into account in simulations. For example, security-related test simulations could be supported by making it possible to mark vulnerable information channels for the simulation engine. The engine could then add a constant or time-varying gain for the values transferred with use of the channel to test the reactions of the control system to an unusual situation. Losing a connection totally would also be a meaningful simulation case that could reveal serious vulnerabilities in systems.

7 Conclusions

This paper has presented a tool-supported approach to transform functional UML AP models and their interlocking specifications to ModelicaML models and finally to simulateable Modelica models. The aim of the approach and transformation implementation is to enable automated and less tedious creation of simulation models and thus to support model-driven development of control systems, including their interlocking and constraint control functions. Compared to present development practices of control systems, this could enable the testing of the solutions earlier during the development process. The approach also offers the other, listed benefits of simulations.

The example system and the control approaches presented in this paper were simple but still adequate for demonstrating the techniques in creation of two simulation models. Simulations could be used to compare the two designed interlocking approaches within a feedback control system. This is also how simulations are currently typically used if their development is considered worthwhile.

Simulations can facilitate the analysis – not directly the synthesis – of control systems. Nevertheless, simulations can help developers in making judicious design decisions. The purpose of model-based techniques is often to automate simple development tasks. However, also within model-based development, real design decisions need to be made by developers and this work can be eased with simulations. In our approach, we use model-based techniques also for developing the simulation models. We thus aim to enhance model-based development of control systems by widening the scope of model-based techniques.

A future working direction of our approach is to shift towards safety functions which share several similarities with interlocks. It is clear that also development of safety functions could benefit from simulations; possibly also from the security point of view. However, the development of safety related systems requires extensive documentation of design and traceability between design artefacts. This is why we are currently working with the requirement sub-profile of UML AP. With this work, we not only support the detailed definition of requirements but also documentation of information originating from risk and hazard analysis. The rationale is that the requirements of safety functions are based on these analyses but the information is not always visible for, for example, the software developers, which makes it difficult to judge the correctness and completeness of design.

References

1. Biehl, M., DeJiu, C., Törngren, M.: Integrating safety analysis into the model-based development toolchain of automotive embedded systems. In: LCTES 2010, pp. 125–132. ACM, New York (2010)
2. Carrasco, J., Dormido, S.: Analysis of the use of industrial control systems in simulators: State of the art and basic guidelines. ISA Transactions 45(2), 295–312 (2006)
3. Dougall, J.: Applications and benefits of real-time I/0 simulation for PLC and PC control systems. ISA Transactions 36(4), 305–311 (1998)
4. Ferrarini, L., Dede, A., Salaun, P., Dang, T., Fogliazza, G.: Domain specific views in model-driven embedded systems design in industrial automation. In: INDIN 2009 the 7th IEEE International Conference on Industrial Informatics, Cardiff, UK, June 23-26 (2009)
5. Friedenthal, S., Moore, A., Steiner, R.: A practical guide to SysML. Morgan Kaufmann OMG Press, San Francisco (2008)
6. Hästbacka, D., Vepsäläinen, T., Kuikka, S.: Model-driven Development of Industrial Process Control Applications. The Journal of Systems and Software 84(7), 1100–1113 (2011), doi:10.1016/j.jss.2011.01.063
7. IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems. parts 1-7 (2010)
8. Jones, C.: Software quality in 2008: A survey of the state of the art. Software Productivity Research LLC, 59 p. (2008), `http://www.jasst.jp/archives/jasst08e/pdf/A1.pdf` (achieved February 13, 2011)
9. Karhela, T.: A software architecture for configuration and usage of process simulation models: Software component technology and XML-based approach. PhD Thesis, VTT Technical Research Centre, Finland (2002)
10. Object Management Group. Technical Guide to Model Driven Architecture: The MDA Guide. Version 1.0.1 (2003)

11. OpenModelica project website (2011), http://www.ida.liu.se/pelab/modelica/OpenModelica.html
12. Ritala, T., Kuikka, S.: UML Automation Profile: Enhancing the Efficiency of Software Development in the Automation Industry. In: The Proceedings of the 5th IEEE International Conference on Industrial Informatics (INDIN 2007), Vienna, Austria, July 23-27, pp. 885–890 (2007)
13. Schamai, W.: Modelica Modeling Language (ModelicaML) a UML Profile for Modelica, Technical Report 2009:5, EADS IW, Germany, Linköping University, Institute of Technology
14. Strasser, T., Rooker, M., Ebenhofer, G.: MEDEIA - Model-Driven Embedded Systems Design Environment for the Industrial Automation Sector. 1st Version of the MEDEIA open source modelling prototype, documentation (2009), http://www.medeia.eu/26.0.html
15. Strasser, T., Rooker, M., Hegny, I., Wenger, M., Zoitl, A., Ferrarini, L., Dede, A., Colla, M.: A research roadmap for model-driven design of embedded systems for automation components. In: INDIN 2009 the 7th IEEE International Conference on Industrial Informatics, Cardiff, UK, June 23-26 (2009)
16. Tranoris, C., Thramboulidis, C.: A tool supported engineering process for developing control applications. Computers in Industry 57, 462–472 (2006)
17. Vepsäläinen, T., Hästbacka, D., Kuikka, S.: Tool Support for the UML Automation Profile - for Domain-Specific Software Development in Manufacturing. In: The Proceedings of the 3rd International Conference on Software Engineering Advances, Sliema, Malta, October 26-31, pp. 43–50 (2008)
18. Vepsäläinen, T., Hästbacka, D., Kuikka, S.: A Model-driven Tool Environment for Automation and Control Application Development - Transformation Assisted, Extendable Approach. In: Proceedings of the 7th Nordic Workshop on Model Driven Software Engineering, Tampere, Finland, August 26-28 (2009)
19. Vepsäläinen, T., Hästbacka, D., Kuikka, S.: Simulation Assisted Model-Based Control Development - Unifying UML AP and Modelica ML. In: 11th International Middle Eastern Simulation Multi Conference, Alexandria, Egypt, December 1-3 (2010)
20. Vepsäläinen, T., Sierla, S., Peltola, J., Kuikka, S.: Assessing the Industrial Applicability and Adoption Potential of the AUKOTON Model Driven Control Application Engineering Approach. In: Proceedings of International Conference on Industrial Informatics, Osaka, Japan, July 13-16 (2010)
21. Zoughbi, G., Briand, L., Labiche, Y.: A UML Profile for Developing Airworthiness-Compliant (RTCA DO-178B), Safety-Critical Software. In: Engels, G., Opdyke, B., Schmidt, D.C., Weil, F. (eds.) MODELS 2007. LNCS, vol. 4735, pp. 574–588. Springer, Heidelberg (2007)