Hindawi Publishing Corporation
Advances in Software Engineering
Volume 2014, Article ID 470201, 14 pages
http://dx.doi.org/10.1155/2014/470201

Research Article

Model-Driven Development of Automation and Control Applications: Modeling and Simulation of Control Sequences

Timo Vepsäläinen and Seppo Kuikka

Department of Automation Science and Engineering, Tampere University of Technology, P.O. Box 692, Korkeakoulunkatu 3, 33101 Tampere, Finland

Correspondence should be addressed to Timo Vepsäläinen; timo.vepsalainen@tut.fi

Received 20 March 2014; Revised 24 June 2014; Accepted 8 July 2014; Published 7 August 2014

Academic Editor: Henry Muccini

The scope and responsibilities of control applications are increasing due to, for example, the emergence of industrial internet. To meet the challenge, model-driven development techniques have been in active research in the application domain. Simulations that have been traditionally used in the domain, however, have not yet been sufficiently integrated to model-driven control application development. In this paper, a model-driven development process that includes support for design-time simulations is complemented with support for simulating sequential control functions. The approach is implemented with open source tools and demonstrated by creating and simulating a control system model in closed-loop with a large and complex model of a paper industry process.

1. Introduction

Model-driven development (MDD) is a system and software development methodology that emphasizes the use of models during the development work. In MDD, models conform to modeling languages that have formal metamodels, for example, unified modeling language (UML). In addition to manual development work, models can be processed with model transformations that revise existing and create new, refined models. The use of transformations may automate error-prone tasks such as importing information to models from preceding development phases and tools. Design models can be used for generating code or to analyze the developed systems. Automated model checks may reveal problems and inconsistencies in models and between phase products.

The mentioned benefits of MDD are related to development tasks that are repetitive and simple enough to be treated with preprogrammed rules. However, MDD has not been able to, and probably cannot, automate all the complex tasks in system and software development. Demanding design decisions over alternative solutions to achieve (sometimes informal) objectives and product characteristics need to be made by professional developers. However, although genuine design decisions cannot be automated, developers do not always have to rely solely on their experience. For example, simulation is a technique that has been traditionally used in the domain within control algorithm development and control system testing.

Automation and control system development is also an application domain in which the use of MDD techniques has been researched extensively during recent years. However, despite the research activities and the tradition of using simulations, ability to simulate early software design models has not yet been sufficiently addressed in the domain.

In their previous work, the authors have developed a simulator integration [1] to the tool-supported Aukoton MDD process [2] for automation and control applications. The approach is based on UML Automation Profile (UML AP) [3]. It enables modeling and simulation of cyclically executed control functions including feedback and binary control as well as interlocks (interlocks are used in control systems to protect the controlled processes from causing harm to themselves or personnel, e.g., by forcing actuators to safe states based on measured states of the processes).

The simulation support is intended to be usable during both platform independent and platform specific modeling

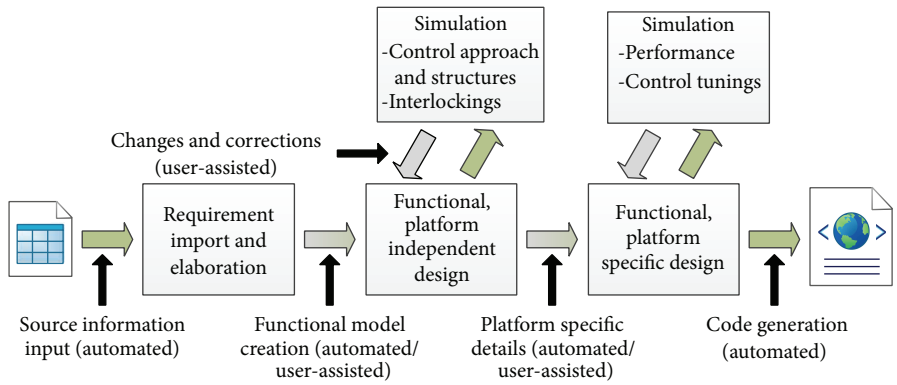

Figure 1: The MDD process with simulation extensions.

phases of the development process; see Figure 1. During the platform independent phase, it is possible to, for example, evaluate alternative control approaches, structures, and interlocks. During the platform specific phase, the approach enables the evaluation of platform specific functions, tunings, and predicted overall performance of the system. Technically the approach utilizes model-in-the-loop simulations so that UML AP control system models are transformed to ModelicaML models. In this paper the approach is complemented by enabling simulation of sequential control activities. The activities are modelled with Automation Sequences of UML AP and visualized with Automation Sequence Diagrams.

The contributions of this paper are as follows. The modeling notation is discussed in comparison to the well-known state machine notation of UML. An approach that enables the simulation of control sequences in a state-machine-like form is presented and implemented as a model transformation. The approach is integrated to the previous simulation integration. The approach is applied to batch control of a paper industry process.

The rest of this paper is organized as follows. Section 2 reviews work related to the use of MDD and simulations in the industrial control domain. In Section 3, the previous work is introduced briefly, which is necessary for understanding how the new work integrates to it. Section 4 discusses the use of control sequences in process industry, presents the UML AP approach to modeling control sequences, and presents the model transformation to create simulation models. In Section 5, before discussion and conclusions, the approach is applied to an illustrative pulp batch processing system.

2. Related Work

The use of model-driven techniques has been researched extensively in the domain of industrial control during recent years. Modeling of requirements, architecture, and details of control applications has been seen as an important part of design processes and as a means to cope with the ever-increasing size and complexity of the applications. Many of the recent approaches have also integrated simulations to the development processes in order to be able to test early and concurrently to the development work.

In addition to industrial control, MDD with simulation features has been applied to control system development for automotive and other embedded applications. The general simulation approaches that can be applied when models are used for generating code include model-in-the-loop (MiL), software-in-the-loop (SiL), processor-in-the-loop (PiL), and hardware-in-the-loop (HiL) simulations [4]. The approaches differ in the control system configurations that are used to control plant models in closed-loop simulations. Examples on use of MiL, SiL, and HiL simulations in the embedded system domain include [5] that describes a general framework for and two examples of use of MiL simulation. A testing environment that uses SiL simulations is presented in [6]. HiL simulation and testing have been utilized, for example, in [7–9].

Another classification of simulation approaches is related to the amount of simulation engines. Simulation of a controlled system, with a model of a process to be controlled and a model of a control system, can be performed within a single simulation engine or as cosimulation. In cosimulation, the models are simulated within different but connected environments. This requires a mechanism to synchronize the simulations including their values and states. Commands and functions, for example, running, replaying, freezing, and loading states (see [10] for a list of basic simulation functions) must be replicated to all used engines.

The management and coupling of cosimulations have been recently addressed with FMI standard [11] and also with model based techniques [12]. However, the area of expertise of control application developers may still not be in simulation techniques. As a consequence, the use of a single simulation engine can be considered more recommendable.

In the industrial control domain, MAGICS approach for MDD of industrial process control software is presented in [13]. As a modeling notation the approach utilizes ProcGraph that has been implemented on the Eclipse platform on top of Eclipse Modeling Framework (EMF). The approach utilizes several diagram types including Entity Diagrams (ED), State Transition Diagrams (STD), and State Dependency Diagrams

(SDD), of which STD is suitable for modeling sequential behavior. The approach enables the generation of executables but does not address simulations.

The FLEXICON project studied the integration of Commercial-Off-The-Shelf tools, including MATLAB/Simu- link and ISaGraf, to support the development of control appli- cations for marine, automotive, and aerospace systems [14]. The approach uses cosimulation, which is enabled by DSS (data delivery service) middleware between the tools.

Vyatkin et al. [15] developed a model-integrated design framework for designing and validating industrial automation systems. It is based on the Intelligent Mechatronic Component (IMC) concept and the use of IEC 61499 architecture. New systems are developed from IMCs that are integrated together and with their models enable formal verification, closed-loop MiL simulation of IEC 61499 models and code deployment.

The approach of the MEDEIA project [16] builds on use of several model types as well as bidirectional model transformations. The process supports the use of closed-loop MiL simulations which are based on use of an IEC 61499 environment. Simulation models of the process parts are in the approach defined with either timed state charts or external behavior descriptions (external simulation tools).

The abovementioned standard, IEC 61499 [17], is a specification and modeling language for industrial control applications. It extends the function block concept of another IEC programming language, IEC 61131-3 [18], with event driven execution and support for distribution of applications. With an appropriate tool support, IEC 61499 models can also be used for simulation purposes.

The simulation approach in [19] is based on mixing real control hardware with simulated one while simulating the plant in another (SIMBA 3D) environment. The benefit of the cosimulation approach is the ability to test early, by executing already implemented parts and simulating the rest.

The simulation approach closest to our work has been recently presented in [20]. In a manner similar to [15, 16] IEC 61499 is used for simulation purposes also in [20]. Model transformations are used for creating IEC 61499 plant models from MATLAB/Simulink plant models to obtain closed-loop behavior within a single (MiL) simulation environment.

Difference from the work to be presented, the referred simulation approaches in the domain ([14–16, 19, 20]) do not address modeling and simulation of sequential control separately from, for example, stabilizing feedback control. In [13] the sequential control aspect is addressed with respect to modeling. However, simulation of the models is not suggested. The use of simulations can thus be assumed to be possible no earlier than after code generation.

On the other hand, the referred development approaches that support simulations rely either on cosimulation ([14, 19]) or use of IEC 61499 as a simulation language ([15, 16, 20]). The use of IEC 61499 to simulations was not a viable alternative in this work to be presented because it is not used in the MDD process [2] that is extended with simulations. On the other hand, UML AP models that are used in the development process are not simulatable as such. The use of cosimulation would thus have either required a transformation to a simulatable form or delayed simulations to simulating plant models with produced executables. These reasons, however, apply to a number of MDD processes with nonsimulatable modeling languages such as UML and UML profiles.

Other works related to sequential control with model-based characteristics include [21] that presents an approach to transform Grafcet [22] models to Mealy machines for testing purposes (Grafcet is a conventional means to specify control sequences). Execution semantics of Sequential Function Charts (SFC) [18] have been addressed in [23]. The SFC notation is part of IEC 61131-3 [18] and based on the earlier version Grafcet.

In the simulation approach to be presented, the target simulation language is Modelica [24] with Modelica Modeling Language (ModelicaML) [25] as an intermediate language. Modelica is an object-oriented, equation based simulation language. The basic concepts of it are simulation classes that contain properties, equations, and connectors. Similarly to classes of object-oriented programming languages, Modelica classes can inherit properties (and equations) of parent classes. Simulatable Modelica models consist of instances of the classes that are connected together with their connectors. ModelicaML [25], on the other hand, is a UML profile for Modelica. It consists of stereotypes and tagged values that correspond to the key words and features of Modelica and enable modeling of Modelica models with UML tools. ModelicaML models are not simulatable as such but can be transformed to simulatable Modelica form with OpenModelica tools [26]. For simulating Modelica models there are both open source (e.g., OpenModelica [26]) and commercial tools (e.g., Dymola [27]) available.

The ModelicaML (profile) implementation uses UML2 plugins on the Eclipse platform which are built on Eclipse Modeling Framework (EMF) implementation of OMG Meta Object Facility (MOF). The ModelicaML implementation is thus technically similar to the UML AP implementation [28] that is used in this work for control system modeling. It has been implemented by extending UML2 and Topcased SysML metamodel with EMF. This similar background of the tools enables implementing the transformation from UML AP to ModelicaML using standardized QVT (Query/View/Transformation) languages [29] and their open source implementations on the Eclipse platform.

3. On Simulating Control Application Models

The objective of integrating simulations to MDD for automation and control applications is to support design-time quality assurance activities. It should be possible to compare alternative control approaches and structures and tunings as well as interlocks. Design flaws should be found and corrected as early as possible and to the extent possible so that they would not affect adversely subsequent design phases. By enabling simulation of design-time models, it could be also possible to obtain at least part of the general benefits of simulations before implementation of the applications. Such general benefits include, for example, improvements

to the design, development, and validation of the control programs, as reported in [10].

Without specific support for sequential control, the approach to create simulation models from UML AP models has been presented in [1]. In UML AP, the modeling concepts for functional modeling are automation functions (AFs) that have been divided to a hierarchy of measurements, actuations, controls, and interlocks. Measurement and actuation AFs are interfaced with sensors and actuators of the controlled processes while performing conversions of signals to and from engineering units. Control AFs perform computation of control signals according to control algorithms. The purpose of interlock AFs is to compute releasing and locking signals for actuators and devices. AFs interchange signals and information with ports.

The transformation for simulating functional UML AP models (that consist of AFs) creates and appends simulation counterparts of the AFs to Modelica plant simulation models. For platform independent AFs, the transformation utilizes a library of predefined simulation counterparts (classes) of them. To support platform and vendor specific AFs, the transformation is capable to utilize external libraries of simulation classes. To support application specific AFs, for example, interlocks that require tailoring for each application, the transformation is capable to create simulation classes based on logic diagram descriptions of AFs [1]. The process described in this paper is an equivalent approach to create new simulation classes but based on Automation Sequence Diagrams instead of logic diagrams.

The decision to use model transformations in this manner was made because UML AP models, as they are used in the tool, are not simulatable as such. Transforming plant models to the control application models would not have enabled closed-loop simulation, for example, in [16, 20] in which (IEC 61499) models were simulatable. In a similar manner, use of cosimulation as in [14, 19] would have required transformation to a simulatable form before applying cosimulation. Additionally, the cosimulation approach would require additional work; see Section 2, related to, for example, coupling simulations skills that not all control application developers can be assumed to have. The approaches to obtain closed-loop simulations by transforming plant models, by transforming control application models, and by using cosimulation have also been recently compared in [30].

An example structure of a plant model before and after executing the transformation that appends the control application specific parts to it is illustrated in Figure 2. Before executing the transformation, the model contains simulation class definitions of the parts of the plant and a description of how the interconnected instances of the classes form the system model. This part of the model, referred to as the original process model, is circled with blue, dashed line. The transformation (1) copies and creates new simulation class definitions based on the control system model, (2) creates instances of the classes according to the control system model, and (3) couples the required instances of the classes to the original model. In the figure, the newly created parts of the model are circled with red, dashed line.

4. Modeling and Simulation of Control Sequences

Control sequences are needed by process industries to perform start-ups of complex processes, for example, power plants or paper machines and to drive them to their designed operating states. In a similar manner, shutting down a process in a controlled and energy efficient manner may require changing set-points of process variables and shutting down devices and sub-systems in a specific order. On the other hand, batch processes constitute a challenging part of industrial processes. In batch processes production of the end products may require, for example, addition of source materials and substances according to time constraints and achievement of defined process states, for example, temperatures and concentrations.

The UML AP approach to modeling sequential control is based on (automation) sequences that have been developed to enable a SFC-conformant modeling notation within UML AP models. Sequences are modelled with a domain specific, new diagram type, Automation Sequence Diagram (ASD). Graphically the ASD notation resembles both the state machine and activity modeling notations of UML.

4.1. Description of the Modeling Notation. Sequences that are described in ASDs consist of Steps that are basic procedural elements in the approach (e.g., upper level batch recipe steps or device level controls). Similar to states of UML state machines, Steps contain Entry, Step, and Exit Activities that are executed when arriving to the Step, during the Step and when exiting the Step, respectively. In addition, Steps may reference other Sequences that can be defined with other ADSs. This is an equivalent characteristic to composite states of UML. Containing activities and referencing a sub-Sequence are exclusive alternatives for a Step. In addition to basic Steps, Sequences may contain Allocations. Allocations are intended for reserving process items and devices for the Sequences that they appear in. When used, Allocations are next to initial Steps in the Sequences.

The execution order of Steps within a Sequence is determined by Transitions that may contain different kinds of conditions that control when a Transition is fired. First, the condition can be a Boolean condition that explicitly specifies a Boolean valued condition based on, for example, values of the variables of the AF that contains the Sequence. Secondly, a condition can be a timeout condition specifying how long the Transition must wait after the execution of the previous Step is finished. Additionally, the Transition can be a one shot Transition which is fired immediately after the previous Step has been executed.

In addition to Steps, Allocations, and Transitions, Sequences contain initial and final as well as fork and join Steps. They can be used in a similar manner that the corresponding pseudostates of UML state machines, that is, to control the execution of Sequences. Use of initial and final Steps is also a necessity in each Sequence because whether a transition may occur from a Step to another is not always dependent (only) on the conditions of the Transitions.

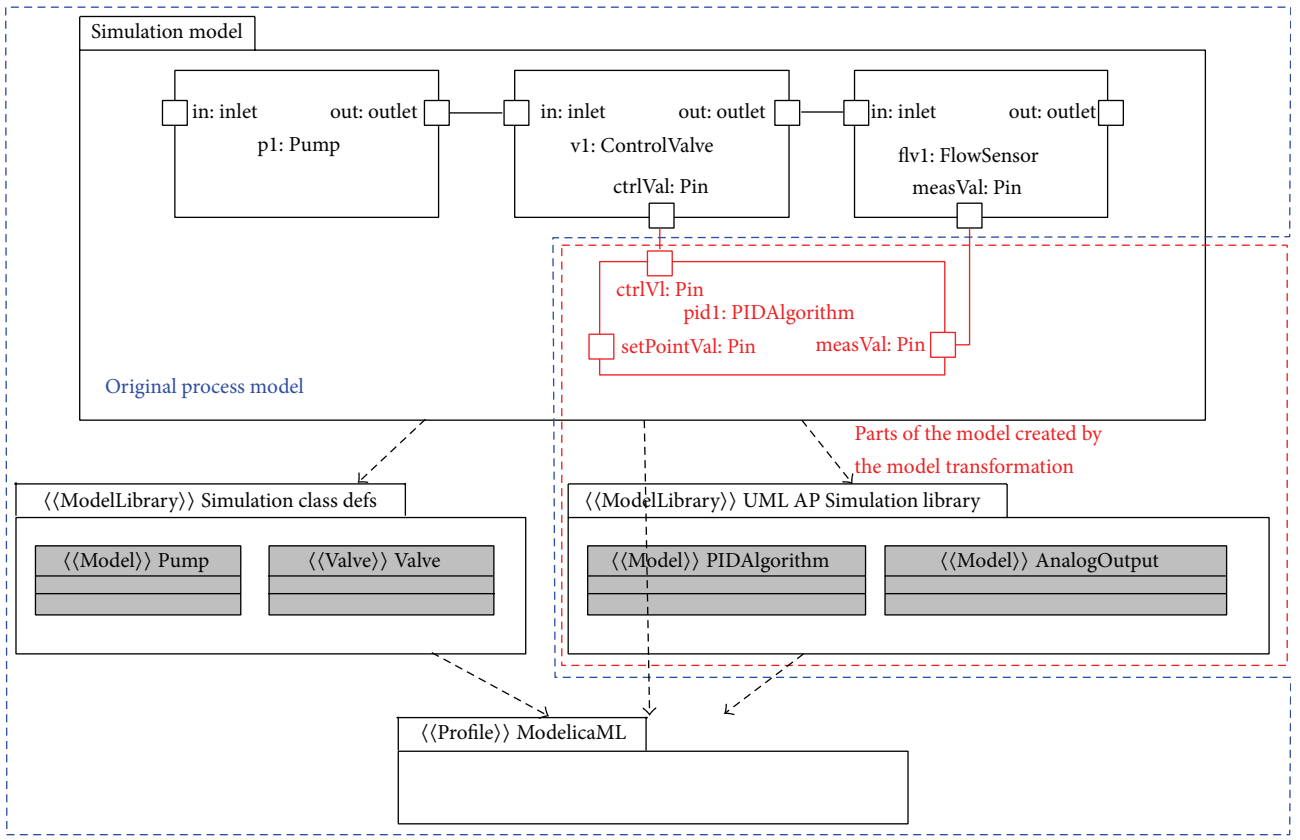

Figure 2: The transformation adds the control system specific parts to an existing model of the physical process.

Consider, for example, the two example diagrams in Figure 3. The figure also illustrates the graphical representation of initial and final Steps, (basic) Steps, Steps that reference sub-Sequences, forks, joins, and Allocations. In the Sequence at the left-hand side, all the Steps reference sub-Sequences that consist of Steps and possibly other sub-Sequences. For example, the WLF (White Liquor Fill) Sequence in the right-hand side diagram is referenced from the third Step in the left-hand side diagram.

Because the Transitions in the Sequence at the left-hand side are one shot Transitions, it is obvious that whether a Transition can fire is also dependent on the completion of the referenced Sequences. Referenced Sequences need to have performed their control activities in a similar manner as in SFCs [18], which is a domain specific notation based on which the ASD notation has been developed. For a Transition to be fired from a Step referencing a sub-Sequence, the referenced Sequence must have reached its final Step. This is a clear semantic difference of the notation in comparison to UML state machines.

Some other obvious differences to UML state machines are also visible in Figure 3. The first Step in the Sequence on the right-hand side is an Allocation. In the example, the allocated process parts are (tanks) T100, T300, and T400 as well as (pump) P100. (In UML state machine diagrams, there are no similar concepts.) After the first of the fork Steps, the transition condition on the right-hand side is of type timeout with value "1" indicating that the Transitions must wait 1 (sec) after the execution reaches the fork. In UML state machines the semantics of the timeouts is slightly different since the waiting time of UML AP Transitions starts from the completion time of the Step preceding the transition.

Lastly, as can be seen in the example Sequence at the right-hand side in Figure 3, Sequences may have several branches executing at the same time. In UML state machines, an analogous feature would be the possibility for a system to be in two (or more) states at the same time. This requires using composite states, each within a region of its own.

Another modeling notation of UML that the ASD notation resembles (both graphically and semantically) is the activity diagram notation that would enable concurrent Sequences of activities and explicit constraints on flows but no timing constraints. However, UML activities cannot be broken up to concepts corresponding to Entry, Step, and Exit Activities of Steps. Activity diagrams may additionally contain decision nodes for which there are no corresponding concepts in ASDs. Lastly, activity diagrams usually describe workflows of entire systems, whereas in UML AP Sequences are used to describe sequential behavior of individual AFs.

Because of the mentioned conceptual differences to the modeling notations of UML that have similar appearance, it was not possible to use directly research work that has been previously done to enable their simulation.

4.2. Model Transformation for Simulating Sequences. In general, Modelica is an equation based language so that the values

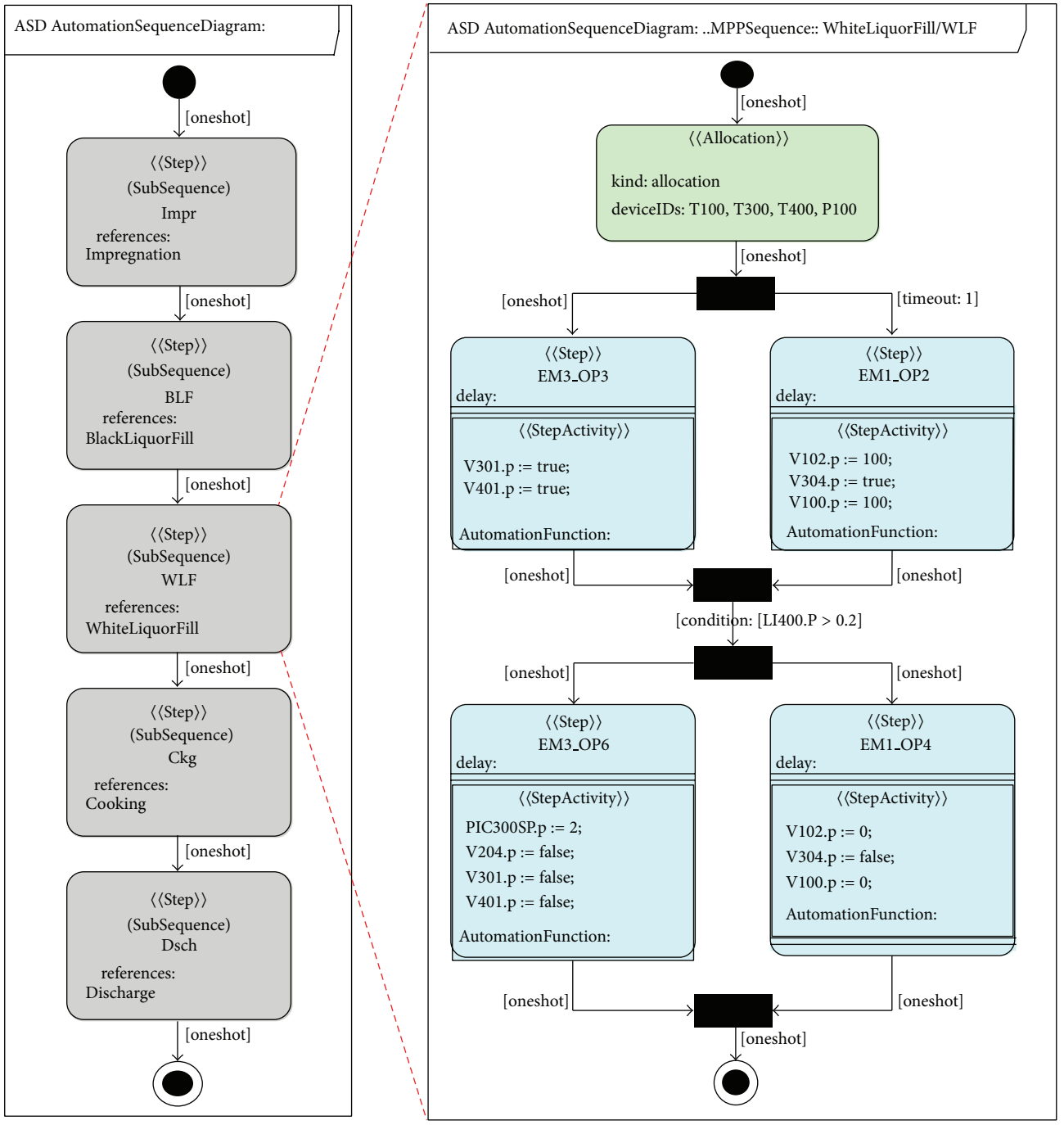

Figure 3: Automation sequence diagrams illustrating a sequence and a referenced subsequence of it.

of variables of Modelica models are determined by equations. However, in addition to the equations that apply all the time, the language includes an algorithmic concept for calculations in which statements are applied in an order. Algorithms are also the constructs of the language that the transformation uses for simulating the Sequences.

The simplified (hiding unnecessary details) metamodel of the ASD diagram type is presented in Figure 4. In the metamodel, Sequence is extended from the UML state machine. The Step and Allocation concepts are extensions of UML state. Entry, Step, and Exit Activities are extended from the UML activity concept and contained by Steps with metamodel properties of UML State (that are hidden from the figure). In addition to the concepts that are shown in the figure, ASDs may contain instances of the mentioned pseudostates of UML, namely, initial, join, and fork states

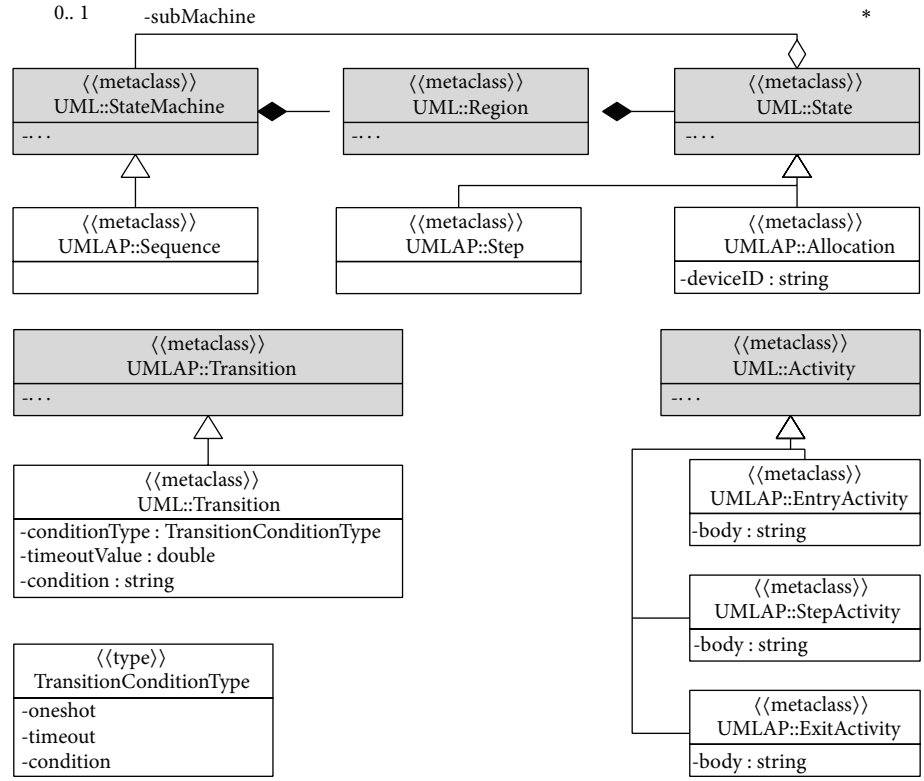

Figure 4: Simplified metamodel of the ASD diagram type with relations to the UML metamodel.

(Steps) as well as final states. Transitions between Steps, Allocations, and pseudo Steps are modelled with a Transition concept that has been extended from UML Transition.

To simulate the behavior of Sequences, they are used as a basis for creating variables and algorithmic code. The systematically named variables are used to keep track of the execution, whereas the algorithmic code changes the values of the variables. The (Entry, Step, and Exit) Activity code of Steps is also included in the algorithms. The variables, which are created to be owned by a Modelica class that corresponds to the AF that owns the Sequence, are created as follows. For the sequence that an ASD represents, and, for each sub-Sequence that is referenced from the Steps of the Sequence, a Boolean variable with the same name than the name of the Sequence is created. These variables are used to indicate the execution of the Sequence being in the Sequence or sub-Sequence in question. In addition, exactly one UML OpaqueBehavior for each highest level Sequence is created to contain the algorithmic code to be generated.

For each Step in a Sequence, the transformation creates two variables. First is a Boolean variable with a name consisting of the name of the Sequence and the name of the Step. The second variable is an Integer variable with a name consisting of the name of the sequence, the name of the Step, and "Phase" literal. The Boolean variables are used to indicate the execution of the Sequence being in the Step in question, whereas the Integer variables keep track of which Activities (Entry, Step, or Exit) have been executed in a Step.

For (exactly one) initial Step in a Sequence, the transformation creates a Boolean variable with a name consisting of the name of the Sequence and "Initialized" literal. For a final Step in a Sequence, the transformation creates a Boolean variable with a name consisting of the name of the Sequence and the name of the final Step. These variables indicate whether or not the execution has reached the initial and final steps in question.

For each fork-to-join region the transformation creates a Boolean variable with a name that consists of the name of the fork, the name of the join, and "Region" literal. In addition, a Boolean variable is created for each branch going out from the fork and coming into the (exactly one) join. The names of these variables consist of the name of the fork (Step) and the number of the branch. The variables corresponding to the branches are used in guard conditions for exiting the join (Step), whereas the other variables are used to indicate the execution of the Sequence being in the fork-to-join region.

For Transitions, the transformation creates variables only if their transition condition is of type timeout. In this case, the name of the real valued variable consists of the name of the Sequence, the name of the Step from which the transition starts, and "Time" literal. These time variables keep track of completion times of the Steps that the Transitions exit from.

Table 1: Mappings between UML AP and UML (ModelicaML) metamodel elements.

Source model (UML AP)	Target model (UML with ModelicaML)		
Element	Model element	Element name	Element type
Sequence	Property	Seq. name	Boolean
	Opaque Behavior	Seq. name + "Algorithm"	—
(UML) Initial (pseudostate)	Property	Seq. name + "Initialized"	Boolean
Step	Property	Seq. name + Step name	Boolean
	Property	Seq. name + Step name + "Phase"	Integer
(UML) FinalState	Property	Seq. name + FinalState name	Boolean
(UML) Fork (pseudostate)	Property	Seq. name + Fork name + "Branch" + #	Boolean
(UML) Join (pseudostate)	Property	Fork name + Join name + "Region"	Boolean
Transition	Property	Seq. name + Step name + "Time"	Double
Allocation	Property	Seq. name + allocation name	Boolean
	Class	"Allocations"	—
	Property	Device ID	Integer

Lastly, for the Allocations, the transformation generates a record (class) and a property for each individual device ID that becomes reserved in the Sequences owned by the AF. The mappings between UML AP and UML metamodel elements are also presented in Table 1.

Some of the algorithmic constructs that are created based on the ASDs are illustrated in an example in Figure 5. First, a when-construct is created that is executed only once at the 7 start of the simulation ("when initial() then"). It sets all the Boolean phase variables (Steps, pseudo Step, Allocations, and sub-Sequences) to false. The Integer variables related to Steps are set to 0 to indicate that no activities have been performed. The Integer variables related to Allocations are also set to 0, to indicate that no allocations are active. The initialization code is created only for each highest level Sequence, not for referenced sub-Sequences.

Steps, Allocations, sub-Sequences, and pseudo Steps are handled with conditional (if-else if) code blocks that can be all entered only once. This is necessary because Modelica models are executed cyclically. In a cycle the execution must continue from the phase to which the execution ended in the previous cycle. For example, arriving to the allocation phase in the example is enabled in the initialization phase and disabled in the allocation phase, which in turn enables the next phase.

Entry, Step, and Exit Activities are executed only once so that when arriving to a Step, the Entry Activity is executed first in addition to changing the phase value to 1. Next, Step Activity is executed and the phase value set to 2. The execution of the Exit Activity and setting the phase variable to 3 waits until the transition condition (if any) to next Step in the Sequence is satisfied so that the transition can occur immediately after performing the Exit Activities. If the Step in question does not contain Entry, Step, or Exit Activities, the corresponding algorithmic code only changes the value of the phase variable.

Allocations are assumed to be next to initial Steps in Sequences. They are intended to model allocations of devices that have IDs corresponding to the ID variables of Allocations. For Allocations, the algorithmic code increases (by one) the variables of the record that correspond to the allocated IDs. At the end of Sequences, allocations are relieved by decreasing the values of the variables by one. In the simulations, the Allocations thus do not force execution to wait but only warn about double allocations, which are indicated by the values of the variables becoming greater than 1. Such problems can then be inspected by developers.

Fork-to-join regions are in the approach handled by creating variables for each branch in the region. The branches may execute independently of each other but for a transition to exit a join Step, all branches must have reached the join. This condition is used as an exit guard for the join, in addition to possible transition conditions related to the transition exiting it.

In the approach, the Modelica code structures resemble the structures in [31] that are used for Modelica simulating state machines. The most notable differences are as follows. Steps or sub-Sequences that are next to another sub-Sequence are not enabled until the sub-Sequence reaches its final Step. This prevents a transition in a higher-level sequence to fire before the final Step is reached. The phases of Steps, that is, whether the Entry, Step, and Exit Activities have been executed, are recorded with Integer variables. The transition conditions to exit Steps are used inside the Steps as guards for shifting to the Exit Activities and enabling the next Step/sub-Sequence. In Allocations that do not have activities the transition conditions are similarly used as conditions to enable the next Steps or sub-Sequences. In referenced sub-Sequences, the next Steps or sub-Sequences are enabled in the final Steps. Lastly, in case of a transition containing a timeout condition, a real valued time variable is created for the previous Step the value of which is set to equal the completion time of the previous Step, pseudo Step, or sub-Sequence. The time variables can be used in the transition conditions as illustrated in Figure 5.

4.3. Constraints and Assumptions. In development of the modeling and simulation approach, a decision was made that Sequences must always be owned by AFs. In this way, the variables and algorithmic code corresponding to a Sequence can be created for a Modelica class corresponding to the AF that owns the Sequence. In the approach a Sequence thus describes the sequential behavior of the AF that owns

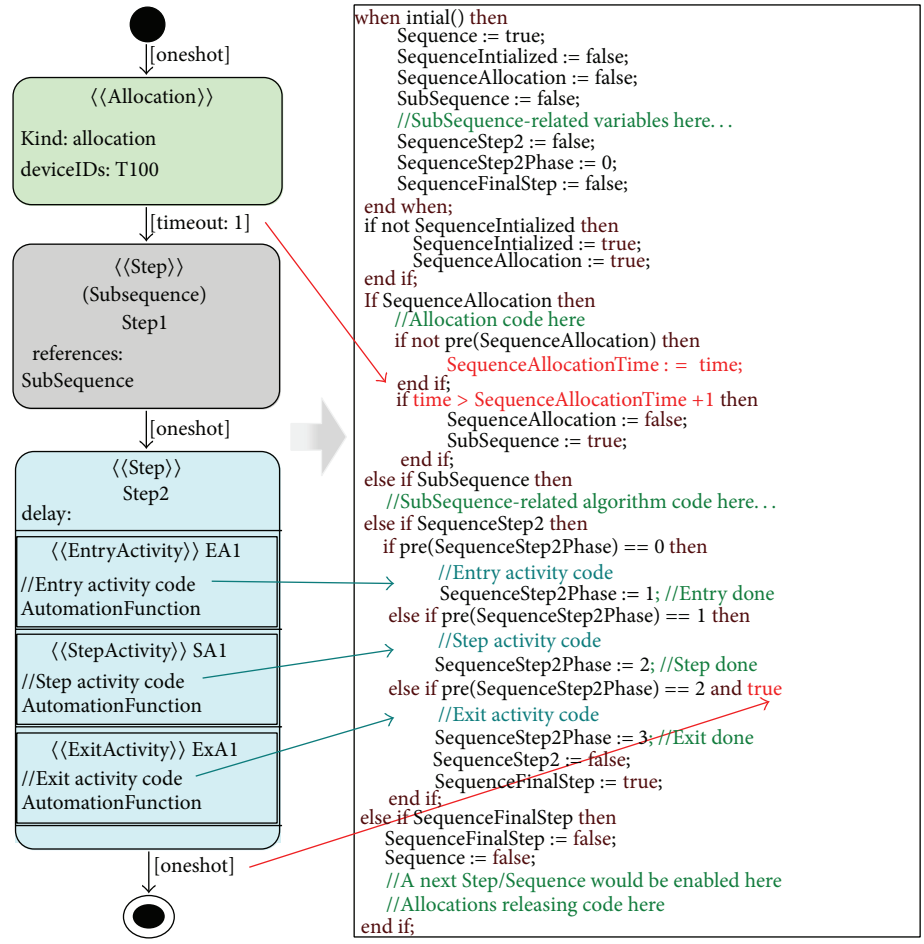

Figure 5: An automation sequence diagram with a corresponding (Modelica) algorithm section.

the Sequence. A Step being executed in a Sequence is a Step of the AF. For control or other signals to be forwarded to other AFs, the AF must be connected to them with use of ports in the interfaces of the AFs. The execution of a single Sequence is thus centralized in an AF. However, an AF may contain several Sequences. On the other hand, a control application model may contain several AFs that define Sequences so that at runtime there would be several Sequences executing concurrently and independently of each other.

The properties that are created for implementing the sequential behavior (see previous section) become the properties of the (ModelicaML) class that is created to correspond to the AF. The properties are necessary for implementing the dynamic behavior, by controlling the execution of algorithmic statements. During simulation, however, they also indicate the execution of the Sequence. For example, the Boolean valued properties created to correspond to Sequences (and its possible sub-Sequences) have value true only when the execution is in the sub-Sequence in question. This feature has been used, for example, in Figure 9 in which the upper plot presents the sub-Sequences of a pulp batch processing Sequence.

There are also restrictions related to the use of Sequences in the approach. Currently, for the simulation transformation to work properly, fork-to-join regions must be balanced so that branches exiting a fork meet each other in one join. On the other hand, the transformation does not support loops within Sequences so that a Step could be entered more than once in a Sequence. It is also assumed that a Sequence always contains an initial Step and at least one final Step. Whether the restrictions related to initial and final Steps hold is checked before performing the transformation.

4.4. Implementation of the Approach. The transformation for simulating Sequences was implemented by extending the previous version of the transformation [1]. In addition to Control Structure and logic diagrams that were supported by the previous version, the transformation processes Sequences contained by AFs to properties and algorithm sections of Modelica classes. The core of the transformation was written with the QVT (Query/View/Transformation) operational mappings language [29] and the executable Java code generated with the SmartQVT tool. The generated Java transformation class was complemented by extending it with

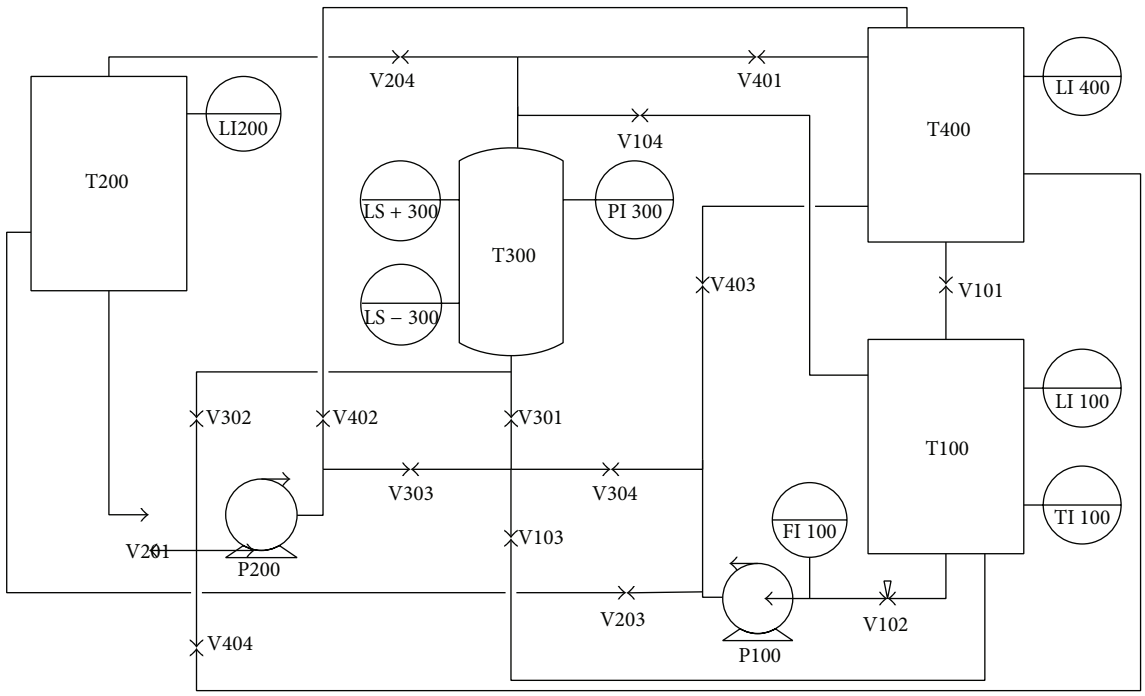

Figure 6: P&I diagram of the pulp production process.

a manually written class. It takes care of, for example, the creation of tagged values of stereotypes and other tasks that are hard to express with QVT languages. To enable launching the transformation with the graphical user interface of the supporting UML AP tool [28], the transformation was packaged to an Eclipse plugin. The plugin architecture and integration to the tool was implemented as outlined in [32].

5. Illustrative Example

The example system to be used in this paper is a laboratory scale pulp processing plant the piping and instrumentation (P&I) diagram of which is in Figure 6. The plant includes 3 storage tanks, a boiler, 2 pumps, 2 control valves, 13 solenoid valves, and piping that enable pumping fluid from any tank to the boiler and via boiler back to any of the tanks. The tanks contain instrumentation to measure liquid levels in them, temperature in tank T100, and pressure in boiler T300. The process is used to simulate batch processing of pulp which is located in the boiler and processed with process substances (impregnation liquor, black liquor, and white liquor) according to timing, pressure, and temperature constraints. In specific phases of the processing sequence, feedback control is required to control the temperature of the white liquor (in tank T100) and pressure in the boiler (T300).

To enable the simulation of the process with a modelled control solution, the process was modeled with ModelicaML. This included defining simulation classes for the physical parts of the process including tanks, boiler, pumps, solenoid valves, control valves, pipes, and pipe crossings with 3 and 4 inlets. Tanks keep record on liquid levels and temperatures inside them. For temperature equations, ideal mixing of fluids is assumed. The liquid flows in pipes and in control valves that are proportional to constants measured from the process and to square roots of the pressure differences between the ends of the pipes/valves. Pumps increase the pressure in their output sides and solenoid valves stop the liquid flows regardless of the pressure differences.

The simulatable ModelicaML model was then defined by creating instances of the classes and connecting them together according to the connections in the physical process. This was done with a structured class diagram. A small part of the diagram, related to the surroundings of the tank T400, is presented in Figure 7.

The control solution for the batch process is illustrated with Figures 8 and 3. Figure 8 presents a (UML AP) Control Structure Diagram of the control solution. It contains binary and analogue valued input and output AFs for interfacing with the sensors and actuators of the process. The Sequence is implemented within the MPPSequence AF that controls some actuators directly and uses controllers for controlling T300 pressure (by throttling valve V104) and T100 temperature with heater E100. To illustrate how logic diagrams and ASDs are used to define behavior of AFs, the figure has been complemented with the MPP Sequence and a logic diagram definition of the temperature controller.

The other illustrating figure (Figure 3) was used as an example earlier and illustrates the MPPSequence and one of the sub-Sequences of it, WhiteLiquorFill. MPPSequence consists of 5 main phases: Impregnation, BlackLiquorFill, WhiteLiquorFill, Cooking, and Discharge. During the phases, the boiler is filled with impregnation liquor and pressurized, filled with black liquor that replaces the impregnation liquor (BlackLiquorFill), filled with white liquor that replaces

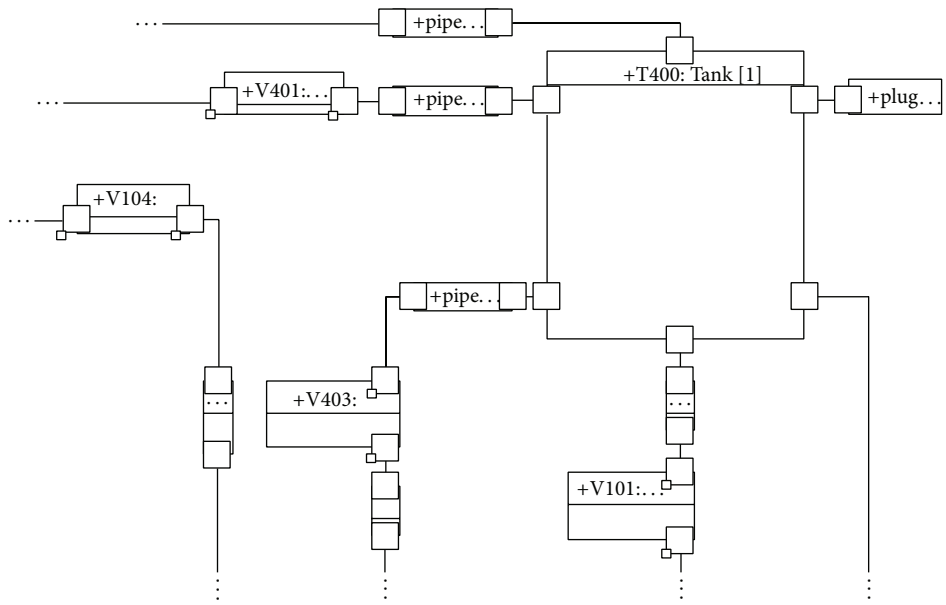

FIGURE 7: A part of the ModelicaML plant model.

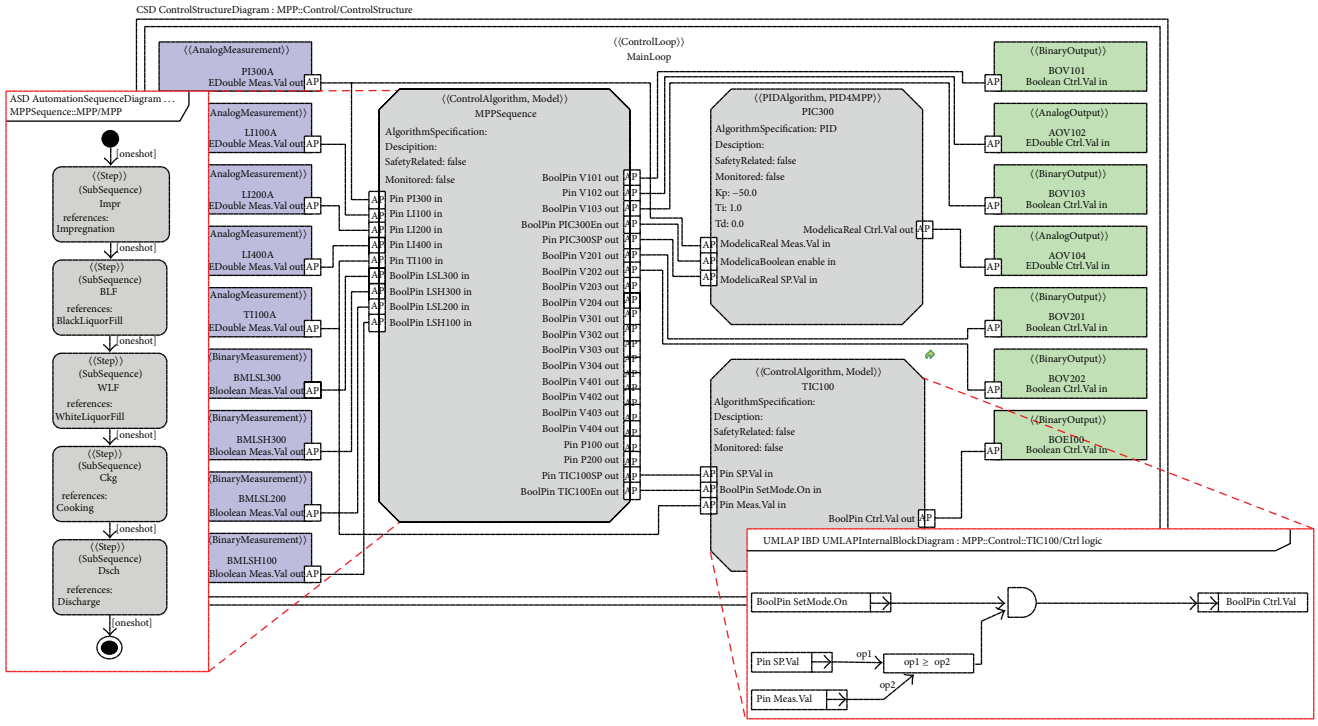

FIGURE 8: Modeling of automation sequences integrates to previous work with control structure and logic diagrams.

the black liquor (WhiteLiquorFill), heated to cooking temperature and pressurized (Cooking), and finally drained back to white liquor tank T400 (Discharge). WhiteLiquorFill, on the other hand, opens valves V301, V401, V102, and V304, and pumps liquor until the level in tank T400 exceeds 0.2 (m).

In order to obtain simulation results of a closed-loop system, the developed transformation was used to transform and connect the control system model to the plant model. Practically this included selecting the simulator export functionality of the tool and the (target) ModelicaML plant model file. After performing the transformation, the model was simulated with OpenModelica [26] tools. The ModelicaML model was first transformed to Modelica code and then loaded to the simulator environment. Initial values for

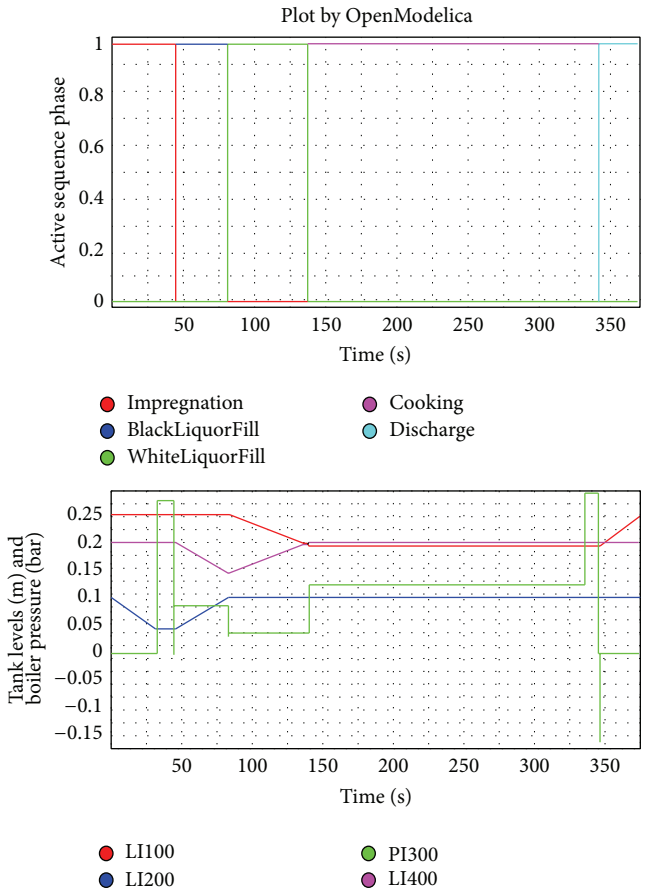

FIGURE 9: Simulation results plotting active phases of the sequence, levels in tank T100 (LI100), T200 (LI200), and T400 (LI400) as well as pressure in boiler T300 (PI300).

the plant, for example, levels and temperatures in the tanks, were defined in the process model. For different simulations they could have been changed at this point too.

A plot illustrating the results from simulating the Sequence is shown in Figure 9. The main phases are plotted in the upper part of the figure, a value being one indicating execution of the phase in question. The lower part of the figure plots the levels of liquor in tanks T100 (LI100), T200 (LI200), and T400 (LI400) and pressure in tank T300 (PI300). According to the results, the control solution including the Sequence works as intended. Processing liquors are used in the correct order and the boiler pressurize, during black liquor fill and cooking phases.

The values shown in the figure were selected for plotting after performing the simulation. The simulator keeps record on all variables related to a simulation. Any other set of variables related to an aspect in the process or in the control solution, for example, functioning of a controller, could have been selected for plotting as well.

6. Discussion

This paper has addressed the issue of simulating sequential control activities within MDD of control applications. The approach integrates to the previous work of the authors and enables the use of Automation Sequence Diagrams (ADSs) of UML AP to define sequential behavior of Automation Functions for simulation purposes. The transformation to simulatable ModelicaML form was implemented with open source modeling and model transformation (QVT) tools on the Eclipse platform. The ASD diagram type that is in the approach used for modeling sequential control has been extended from UML state machine diagrams. However, because of significant differences in execution semantics of state machines, it was not possible to rely on existing work [31] related to simulating them in Modelica form.

The benefits from using Modelica (ML) as the (target) simulation language of the approach included the ability to use standard model transformation techniques. Modelica is also an object-oriented simulation language, which was taken the advantage of mainly in development of the plant simulation model. From the point of view of simulating the control application, however, object-oriented features were not used. As a consequence, it is expected that the presented approach could be used also with other simulation languages that can be accessed with model transformations, for example, Simulink. An approach to execute Sequences without equation based, acausal execution semantics of Modelica could also be similar to the one presented in this paper. Algorithmic constructs were used also in case of Modelica instead of equations that apply all the time.

The novelty of the simulation approach is in the ability to simulate control application models at design time, before IEC language [17, 18] implementations of the applications. Closed-loop MiL simulations are created with model transformations so that a genuine simulation language (Modelica) is used for simulating both plant and control application models. Other MDD approaches in the domain (in which simulations have been supported) have utilized IEC 61499 as a simulation (in addition to implementation) language [15, 16, 20] or relied on the use of cosimulation [14, 19]. On the other hand, sequential control as a special aspect of control systems has been addressed only in [13] but not with respect to simulations. With the work presented in this paper and [1], the simulation approach covers all the common aspects of basic control systems including binary and feedback control, sequential control, and interlocks.

An issue that is not yet addressed in the approach [1] is delays in control systems hardware, for example, networks in distributed control systems. However, the objective of the approach is to enable simulations early, already before, for example, finishing control system hardware design. On the other hand, effects of delays and, for example, random noise in instrumentation can be included in the models, in simulation classes of sensors and actuators of the process. It is also assumed that, for example, delays in typical control system hardware are less significant than those in instrumentation.

Support for model-based control software development is also part of some commercial products. For example, B&R (Automation Studio) [33] and Beckhoff (TwinCAT 3) [34] support the development of control applications in MATLAB/Simulink environment and generating executable (PLC) code based on the models. As a difference to such

products, the work presented in this paper intends to support simulations in an MDD approach in which all models are not simulatable. Instead, models are developed gradually from requirements towards executable applications using model transformations for shifting between models and, for example, importing source information to models. In addition, the models cover special needs such as traceability between requirements and design artifacts that are becoming more and more important in the domain.

To illustrate the simulation approach, it was applied to simulation of a controlled pulp batch production process. For the case study, the pulp production process was modelled with Modelica. Flow, pressure, and temperature equations for all the plant components in the model led to the total number of equations for the closed-loop system to be approximately 1400. As such, the closed-loop system was the largest that has been utilized in the simulation experiments of the approach so far. It also demonstrates the scalability of the approach for practical, nontrivial simulation needs.

7. Conclusions

MDD techniques are under active research in the application domain of industrial control systems. However, despite the research activities, and the tradition of using simulations, simulations have not yet been sufficiently integrated to MDD in the domain.

In MDD, it is possible to utilize model transformations for obtaining simulation models already before programmed implementations of the applications. This possibility should be taken advantage of. Control applications models should be evaluated in a timely manner and in closed-loops with the models of the processes to be controlled. In order to relieve control application developers from the task of coupling simulation engines, the simulations should follow the model-in-the-loop approach using a single simulation engine.

The presented approach complements the simulation approach of the authors with the possibility to simulate sequential control activities in conjunction to feedback and binary control as well as interlocks. The new work has been targeted for the sequences of process and batch industry. However, control sequences can be beneficial also in simulations of other kinds of processes. For example, in a previous simulation experiment [1], the set-point trajectories to evaluate a control system in different conditions needed to be defined manually. With the work presented, the set-point trajectories can be included in Sequences of the models.

According to our experiences, the simulation approach is useful in revealing defects in control algorithms, structures, and tunings. The simulations can be performed already at design time and so that decisions made in a development phase can be evaluated before they affect decisions in later phases. By creating simulation models with automated model transformations, simulations can be used as a continuous, design-time quality assurance method. This can be done without causing excessive additional workload to developers.

It is also expected that the task of developing models of the processes to be controlled with Modelica becomes easier and more attractive for industry in near future. This is due to improvements in libraries of simulation classes, the Modelica standard library, from which it is possible to compose plant models. It is also a clear benefit of Modelica that it includes support for standard and user/company specific libraries. Modelica is already supported by both commercial and open source tools that can be used by both industry and academy.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] T. Vepsäläinen and S. Kuikka, "Simulation-based development of safety related interlocks," in *Simulation and Modeling Methodologies, Technologies and Applications*, pp. 165–182, Springer, 2013.

[2] D. Hästbacka, T. Vepsäläinen, and S. Kuikka, "Model-driven development of industrial process control applications," *Journal of Systems and Software*, vol. 84, pp. 1100–1113, 2011.

[3] T. Ritala and S. Kuikka, "UML automation profile: enhancing the efficiency of software development in the automation industry," in *Proceedings of the 5th IEEE International Conference on Industrial Informatics (INDIN '07)*, pp. 885–890, June 2007.

[4] H. Shokry and M. Hinchey, "Model-based verification of embedded software," *Computer*, vol. 42, no. 4, pp. 53–59, 2009.

[5] A. Plummer, "Model-in-the-loop testing," *Proceedings of the Institution of Mechanical Engineers I*, vol. 220, pp. 183–199, 2006.

[6] H. Chae, X. Jin, S. Lee, and J. Cho, "TEST: testing environment for embedded systems based on TTCN-3 in SILS," *Communications in Computer and Information Science*, vol. 59, pp. 204–212, 2009.

[7] M. Short and M. J. Pont, "Assessment of high-integrity embedded automotive control systems using hardware in the loop simulation," *Journal of Systems and Software*, vol. 81, no. 7, pp. 1163–1183, 2008.

[8] M. Schlager, R. Obermaisser, and W. Elmenreich, "A framework for hardware-in-the-loop testing of an integrated architecture," in *Software Technologies for Embedded and Ubiquitous Systems*, pp. 159–170, Springer, 2007.

[9] G. Stoeppler, T. Menzel, and S. Douglas, "Hardware-in-the-loop simulation of machine tools and manufacturing systems," *IEE Computing and Control Engineering*, vol. 16, no. 1, pp. 10–15, 2005.

[10] J. A. Carrasco and S. Dormido, "Analysis of the use of industrial control systems in simulators: state of the art and basic guidelines," *ISA Transactions*, vol. 45, no. 2, pp. 295–312, 2006.

[11] MODELISAR Consortium, "Functional Mock-up Interface for Co-simulation," Version 1.0., 2010.

[12] G. Hemingway, H. Neema, H. Nine, J. Sztipanovits, and G. Karsai, "Rapid synthesis of high-level architecture-based heterogeneous simulation: a model-based integration approach," *Simulation*, vol. 88, no. 2, pp. 217–232, 2012.

[13] T. Lukman, G. Godena, J. Gray, M. Heričko, and S. Strmčnik, "Model-driven engineering of process control software-beyond device-centric abstractions," *Control Engineering Practice*, vol. 21, no. 8, pp. 1078–1096, 2013.

[14] H. Thompson, D. Ramos-Hernandez, J. Fu, L. Jiang, J. Nu, and D. Dobinson, "The FLEXICON co-simulation tools applied to a marine application," *Proceedings of the Institution of Mechanical Engineers M: Journal of Engineering for the Maritime Environment*, vol. 222, pp. 81–94, 2008.

[15] V. Vyatkin, H. Hanisch, C. Pang, and C. Yang, "Closed-loop modeling in future automation system engineering and validation," *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, vol. 39, no. 1, pp. 17–28, 2009.

[16] I. Hegny, M. Wenger, and A. Zoitl, "IEC 61499 based simulation framework for model-driven production systems development," in *Proceedings of the 15th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA '10)*, September 2010.

[17] International Electrotechnical Commission, *IEC 61499-1: Function Blocks-Part 1: Architecture*, International Standard, Geneva, Switzerland, 1st edition, 2012.

[18] International Electrotechnical Commission, *IEC 61131-3: Programmable Controllers part 3, Programming Languages*, IEC Publication, 2013.

[19] L. Ferrarini and A. Dedè, "A model-based approach for mixed Hardware in the Loop simulation of manufacturing systems," in *Proceedings of the 10th IFAC Workshop on Intelligent Manufacturing Systems (IMS '10)*, pp. 36–41, July 2010.

[20] C. Yang and V. Vyatkin, "Transformation of Simulink models to IEC 61499 Function Blocks for verification of distributed control systems," *Control Engineering Practice*, vol. 20, no. 12, pp. 1259–1269, 2012.

[21] J. Provost, J. Roussel, and J. Faure, "Translating Grafcet specifications into Mealy machines for conformance test purposes," *Control Engineering Practice*, vol. 19, no. 9, pp. 947–957, 2011.

[22] R. David, "Grafcet: a powerful tool for specification of logic controllers," *IEEE Transactions on Control Systems Technology*, vol. 3, no. 3, pp. 253–268, 1995.

[23] A. Hellgren, M. Fabian, and B. Lennartson, "On the execution of sequential function charts," *Control Engineering Practice*, vol. 13, no. 10, pp. 1283–1293, 2005.

[24] P. Fritzson and V. Engelson, "Modelica—a unified object-oriented language for system modeling and simulation," in *Proceedings of the Object-Oriented Programming Conference (ECOOP '98)*, pp. 67–90, Springer, 1998.

[25] W. Schamai, *Modelica Modeling Language (ModelicaML): A UML Profile for Modelica*, Linköping University Electronic Press, 2009.

[26] OpenModelica, https://www.openmodelica.org/.

[27] Dassault Systemes, Dymola, http://www.3ds.com/products-services/catia/capabilities/systems-engineering/modelica-systems-simulation/dymola.

[28] T. Vepsäläinen, D. Hästbacka, and S. Kuikka,, "Tool support for the UML automation profile—for domain-specific software development in manufacturing," in *Proceedings of the 3rd International Conference on Software Engineering Advances (ICSEA '08)*, pp. 43–50, 2008.

[29] OMG, "Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification (QVT), Version 1.0," Object Management Group, 2008.

[30] T. Vepsäläinen and S. Kuikka, "Benefit from simulating early in MDE of industrial control," in *Proceedings of the IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA '13)*, pp. 1–8, 2013.

[31] W. Schamai, U. Pohlmann, P. Fritzson, C. J. Paredis, P. Helle, and C. Strobel, "Execution of UML State machines using modelica," in *Proceedings of the 3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT '10)*, pp. 1–10, 2010.

[32] T. Vepsäläinen, D. Hästbacka, and S. Kuikka, "A model-driven tool environment for automation and control application development-transformation assisted, extendable approach," in *Proceedings of 11th Symposium on Programming Languages and Software Tools and 7th Nordic Workshop on Model Driven Software Engineering*, pp. 315–329, 2009.

[33] B&R Automation Studio, http://www.br-automation.com/en/products/software/automation-studio/.

[34] Beckhoff TwinCAT 3, http://www.beckhoff.fi/english.asp?twincat/default.htm.