AutomationML AS A BASIS FOR OFFLINE - AND REALTIME-SIMULATION -

Planning, Simulation and Diagnosis of Automation Systems

Olaf Graeser, Barath Kumar, Oliver Niggemann, Natalia Moriz and Alexander Maier
inIT-Institut Industrial IT, Hochschule Ostwestfalen-Lippe, Liebigstrasse 87, Lemgo, Germany

Keywords: Automation technology, Modelling, Simulation, HIL test and AutomationML.

Abstract: The growing complexity of production plants leads to a growing complexity of the corresponding automation systems. Developers of such complex automation systems are faced with two significant challenges: (*i*) The control devices have to be tested *before* they are used in the plant. For this, offline- and hardware–in–the loop (HIL) simulations can be used. (*ii*) The diagnosis functions within the automation systems become more and more difficult to implement; this entails the risk of undetected errors. Both challenges may be solved using a system model, i.e. a joint model of the plant and the automation system: (*i*) Offline simulations and HIL tests use such models as an environment model and (*ii*) diagnosis functions use such models to define the normal system behaviour—allowing them to detect discrepancies between normal and observed behavior. System models cannot be modelled by one person in a single development step. Instead, such models must mirror the modularity of modern plants and automation systems. Here, the new standard AutomationML is used as basis for such a modular system model. But a modular system model is only a first step: Both testing and diagnosis require the simulation of such models. Therefore, a corresponding modular simulation system for AutomationML models is presented here; for this, the Functional Mock–Up Unit (FMU) standard is used. A prototypical tool chain and a model factory (MF) is used to show results for this modular testing and diagnosis approach.

1 INTRODUCTION

The virtual planning of factories (“Digital Factory”) has been a major trend over the last years (see e.g. (Brecher et al., 2008; Kühn, 2006)): By using simulated factories in early development phases, system integrators and factory operators want to (*i*) minimise design and construction times, (*ii*) reduce the number of errors in the factory and (*iii*) optimise factory designs and configurations. This could lead to more complex factories, faster processes, and more integrated production chains.

So far, research in virtual factories has mainly focused on the plant and machine factory construction aspect. But more complex factories also lead to more complex automation systems. Here, we will present a concept and first prototypes for a modelling and simulation approach for an automation-centred virtual design and testing process. In other words, unlike with machine-centred approaches, precise models of the automation systems are used while the machines and plants are only modelled with a limited level of details—often timed event sequences are sufficient.

A software tool chain for model based development and simulation of a plant is introduced. This tool chain starts with an editor for plant modelling and ends in an simulation framework for offline simulations, HIL tests and online plant diagnosis. These are the three main use cases for the editor and the simulation framework.

- In the offline simulation, Soft-PLCs (Programmable Logic Controllers), i.e. the control applications, can be tested by simulating Soft-PLCs and the corresponding process.
- HIL tests can be used to test a PLC given as hardware. Unlike an offline simulation, this requires a real–time simulation.
- Online diagnosis can be used to detect changes and anomalies in the behaviour of a plant—mainly by comparing the real plant behavior to the simulated behaviour.

Graeser O., Kumar B., Niggemann O., Moriz N. and Maier A..
AutomationML AS A BASIS FOR OFFLINE - AND REALTIME-SIMULATION - Planning, Simulation and Diagnosis of Automation Systems.
DOI: 10.5220/0003537403590368
In *Proceedings of the 8th International Conference on Informatics in Control, Automation and Robotics* (ICINCO-2011), pages 359-368
ISBN: 978-989-8425-75-1

2 STATE OF THE ART

For the automation technology, the challenge is to model and develop control software, visualisation and diagnosis software. Such software modules should be reusable (for future projects), testable on a standalone PC and platform independent (with respect to the used communication hardware).

In the area of embedded systems software development, several projects and standards addressed the abovementioned challenges. For example safety–related systems (IEC 61508, see (Gall, 2008)), automotive software development (Steiner and Schmidt, 2003), the field of automation (IEC 61499 and IEC 61131), or several research projects like Modale (Szulman et al., 2005), OrVia (Stein et al., 2008) or Medeia (Strasser et al., 2008).

The introduction of explicit systems, software and plant models is an important trend in embedded software development (Niggemann and Stroop, 2008; Niggemann and Otterbach, 2008) and also in the field of automation technology (Szulman et al., 2005; Strasser et al., 2008; Streitferdt et al., 2008). The modelling approach in this work is based on control and plant component models. For this purpose, standards like AutomationML (Drath et al., 2010) and Modelica (Modelica Association, 2011) are used and improved.

Furthermore, the concept of *separation of concerns* is used. A complete plant model consists of various aspects, like logical software models, hardware topologies, network descriptions and a system model. For a better reusability of the single models, the models are modelled and stored separately. The combination of the single models is done in a later step. The approach used in this work is inspired by the Object Managements Group's (OMG's) Model–Driven Architecture (MDA) (Allen, 2002; Mellor et al., 2004). In the first step, a logical structure with software modules and plant models is created. This correlates to the MDA's Platform Independent Model (PIM). Later, this structure is mapped onto specific hardware descriptions. Components of the PIM, which describe a class or a role of a device in the model, are thereby linked to a real hardware. The result correlates then to the MDA's Platform Specific Model (PSM).

3 MODELLING

3.1 Concept

Factories and the corresponding automation systems comprise more and more heterogeneous and interconnected components. Therefore the modelling formalism and the simulation framework must support such systems. The new standard AutomationML is able to integrate heterogeneous description formalisms and models automation systems, plants, sensor/actuators, and especially their inter–connection.

Our modelling approach is in general inspired by OMG's MDA paradigm (see (Mellor et al., 2004) for details): In a first step, a logical architecture comprising only software modules and plant models is created—MDA's PIM. Only later on, this logical architecture is mapped onto a specific hardware topology, creating the system architecture—MDA's PSM. This can also be seen in figure 1.

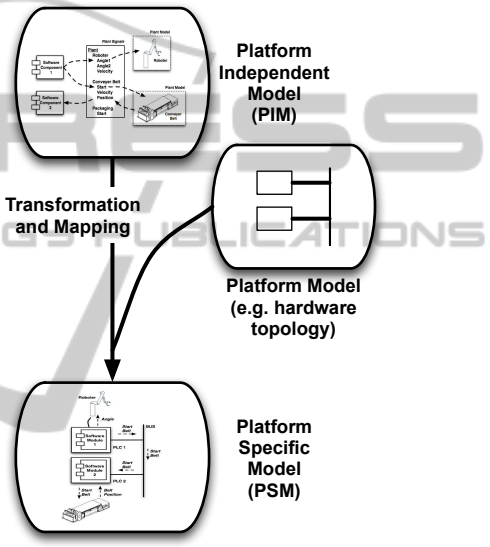

Figure 1: Model Driven Architecture Approach.

3.1.1 Platform Independent Model

As can also be seen in figure 2, PIMs comprise software components (i.e. the software architecture) and plant models. Software components are control algorithms, diagnosis software algorithms, or monitoring software modules. These software components do not communicate directly with each other or with I/O drivers but via plant signals. Plant signals are variables whose semantics are defined by the plant model, e.g. in figure 2 software component 1 sets the robot's "angle1" signal or start the conveyer belt by a "start" signal.

Please note, that plant signals usually correspond to (physical) states of the real plant, i.e. unlike software interfaces, they exist in reality and hence need not be changed when the software architecture is modified. This increases the level of software reuse since software modules do not have to refer anymore to software–specific interfaces. Since we only con-

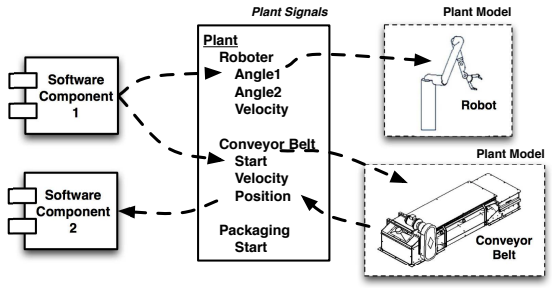

Figure 2: Platform Independent Model.

sider automation systems, in most cases it is sufficient to restrict to the plant signals.

3.1.2 Platform Specific Model

PIMs have the advantage that they can be reused for a large variety of hardware topologies (i.e. platforms). For this, the PIMs are mapped in a second step onto a specific hardware topology, creating a PSM. Many features of the PSM can be derived automatically, e.g. the schedule for real time communication networks such as I/O driver configurations or ProfiNet configuration (see (PNO, 2007) for ProfiNet details), the scheduling algorithm is outlined in (Graeser and Niggemann, 2009). This automatism allows developers to focus on the most difficult part of the development process: The development of the applications.

Such a PSM for the PIM from above can be seen in figure 3: Software component 1 has been mapped onto PLC 1, component 2 onto PLC 2. Now, unlike in figure 2, the signal to start the conveyor belt must be transmitted via the bus while the robot control signal can be transmitted directly to the robot.

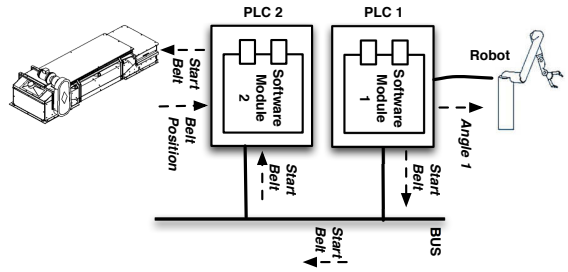

Figure 3: Platform Specific Model.

Please note, that the same PIM can be mapped onto a different hardware topology, i.e. this approach leads to higher level of model reuse.

3.2 AutomationML System Model

As mentioned before, the key challenge lies in the modelling and simulation of heterogeneous and modular systems; systems that comprise both the automation devices and the plants. As shown on the left–hand side of figure 4, AutomationML is well suited for this task, since the goal of AutomationML is to describe a production plant completely, including all components in it and to support each phase and discipline of plant engineering. AutomationML is based on XML and makes use of well established, open and free XML based standards of the involved engineering disciplines. For example, the object model is given by CAEX (IEC 62424, 2008), geometries and kinematics are described by COLLADA (Khronos Group, 2011) and the behaviour of the plant is described with PLCopen XML (PLCopen, 2011). Using these standards ensures a slim specification of AutomationML (Drath et al., 2010).

In AutomationML, a system is defined as a hierarchy of inter–connected objects (bottom part of figure). Each object may be the instance of a specific class from a library—where a “RoleClass” defines an abstract type such as a PLC and a “SystemUnitClass” a specific product. From another point of view, a “RoleClass” can be seen as requirements for an object, and a “SystemUnitClass” is an implementation of these requirements. Furthermore, objects communicate via well–defined interfaces.

Within the concept outlined here, interfaces were defined for plant modules, sensor/actuators, communication networks, and PLCs. These interfaces mainly defined incoming and outgoing signals and timing information.

AutomationML is able to include arbitrary XML files. Here, an XML–based description of FMUs is used to link objects in AutomationML to simulation models. FMUs are a new standard from the Modelisar project (Modelisar project, 2011) to provide a uniform API to arbitrary simulations models such as TheMathwork’s Simulink or Dassault’s Dymola. In this project, a mapping between our interfaces in AutomationML and FMU interfaces has been developed (right–hand side of figure 4). This allows for mapping between AutomationML objects and FMU–based simulation models, i.e. users are able to provide heterogeneous simulation models for all our AutomationML objects, hence supporting the required level of support for heterogeneous and modular systems.

The interface to map FMUs to AutomationML objects is similar to all other external data mappings like Collada and PCLopen XML. Both inherit from AutomationML’s interface class “ExternalDataConnector”. Additionally to the “ColladaInterface” and the “PLCOpenXMLInterface”, a “FMUInterface” is defined, which refers to an external FMU file. The in–

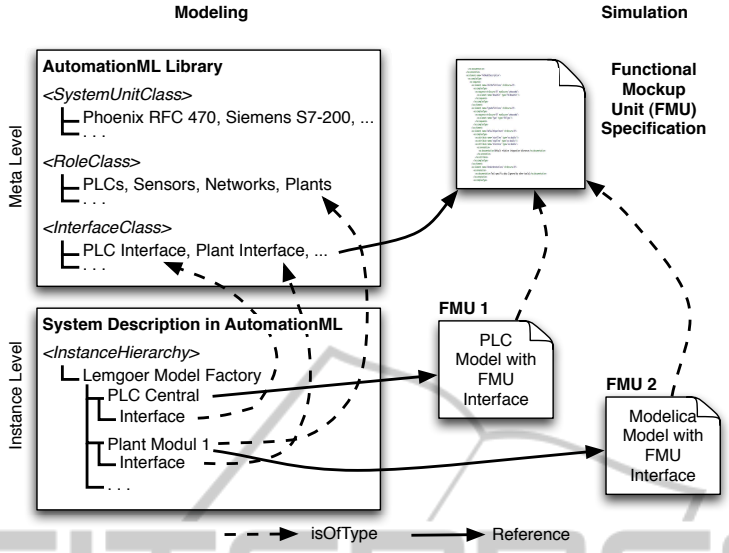

Figure 4: Modelling and simulating heterogeneous modular systems with AutomationML.

and output variables used in the FMU are described in the "modelDescription.xml" inside the FMU file. For now, the mapping to the variables in the AutomationML format is done by name equality.

In this work, a drag and drop editor for a PIM as described above was developed. The editor is based on the Visual Studio 2010 Visualization and Modeling SDK (MSDN, 2011). The graphical user interface can be seen in figure 6. Models created with this editor abstract from the communication hardware used in the plant. As described above, software components (e.g. PLC applications) are connected to sensors and actuators via named signals.

The model of the plant is stored in AutomationML format. Each component of the plant model is mapped to an element of the AutomationML role class lib. As mentioned before, the idea behind the role class lib is, that each component in a plant has to fulfil a function and has to play one or more roles. Role classes can be understood, as a requirement definition, which a component has to satisfy.

Using a graphical editor (Figure 6) has the advantage over an XML editor, that new components (like role classes) can be added to the model per drag–and–drop and the relationships of the components and the flow directions of the communication signals are easy to identify.

Using AutomationML increases the reusability of the models, because it can be expected, that many software tools in the field of industrial automation will at least support AutomationML im- and export.

Currently the plant is modelled only as PIM. Future versions of the editor will also include the PIM to PSM transformation. This can be done by assigning a real hardware description to each AutomationML role class lib based component of the PIM.

3.3 Modelica Behaviour Models

The behavioural model of the whole plant consists of several sub–models, and is modular in nature. As mentioned before in section (3), the plant signal information (i.e. incoming, outgoing signals and timing information) are often sufficient enough to provide the necessary abstraction needed for modelling. One possibility is to use Finite State Machine or Automata (FSM) for constructing these behavioural models. However, FSM do not support modeling complex timed hybrid systems. Hence, our formalism 'Probabilistic regression automata' (PRA) described in (Kumar et al., 2010a), which is an extension of hybrid timed automata (Alur et al., 1995), (Sproston, 2000) is used for this purpose. In PRA, an event can be triggered, when the value of one or several signals cross the specified threshold value(s) within a certain time window reflecting the actual signal timing of the real plant. In addition to the above, probabilistic information is used to handle non-deterministic scenarios (i.e. an automata can show different behaviours for the same set of inputs). A typical case of non-determinism is occurrence of errors which is inevitable (especially, in industrial automation system) - however the occurrence of these errors is less probable then the occurrence of the correct scenarios. Moreover, our formalism supports continuous signals and allows the arbitrary (not only linear) signal value changes within a state.

In this work, FSMs are implemented with Modelica. Modelica is a free standardised equation–based object–oriented modelling language. Modelica is suitable for modular modelling of complex heterogeneous systems. By using Modelica, it is possible to model signals with arbitrary complex behaviour, since Modelica supports the hybrid DAEs (Differential Algebraic Equations) systems. Modelica also supports component–oriented modelling. This concept is conform with the concept of *separation of concerns*. This also increases the reusability of single components of the whole system. Moreover, single components can be refined as precisely as necessary.

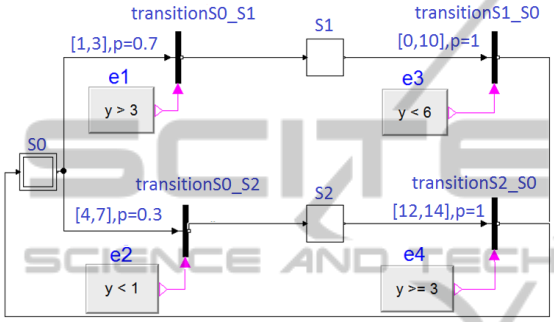

Figure 5: Finite state machine modelled with Dymola.

The automata used in this work were implemented in Dymola (DASSAULT SYSTEMES, 2011). Dymola is suitable for the requirements of modelling and simulating production plants, because this tool supports hierarchical model composition, many libraries of reusable components from several engineering domains, faster simulation and open interfaces to other programs (FMIs). Dymola enables users to develop their own components. In our tool chain, the basis for the PRA prototype is developed with the *StateGraph* and *StateGraph2* Modelica–standard libraries. This basis is extended as following: (*i*) the interface is added (the names of the variable are the same as in the PLC program), (*ii*) the signal behaviour is integrated, (*iii*) the timing and probability information is added. Currently our own library for PRA is under development. An PRA prototype is represented in figure 5.

Created behavioural models are stored as FMU. FMUs are executables, which implemented the Functional Mock–Up Interface (FMI). The FMI functions are called by a simulator (here the ”FMU Simulation Component”) to create and execute one or more instances of the FMU. These FMUs can be referenced in the abovementioned PIM-editor.

4 THE SIMULATION FRAMEWORK

4.1 Simulation Framework

As mentioned before, the simulation framework must support heterogeneous and inter–connected components—mirroring AutomatioML's capability to model such systems. Here, a corresponding simulation framework is described which uses the new standard FMU. This standard encapsulates heterogeneous behaviour models such as Modelica or the Mathwork's Simulink behind a standardised API.

The use cases of the project are offline simulation for testing the control algorithms, HIL tests of PLCs (and plant components), and anomaly detection in the operating plant. For these use cases, a flexible Simulation Framework was developed, in which Simulation Components can be easily removed, replaced or added. The Simulation Framework consists mainly of the Simulation Manager, which is responsible for the instantiation and scheduling of Simulation Components, and the Process Data Manager, which manages the process data of the plant model and distributes them to the Simulation Components (Figure 6).

The Simulation Framework makes use of the Managed Extensibility Framework (MEF) (Microsoft, 2011) to connect all available Simulation Components to the Simulation Framework. MEF is part of Microsoft's .NET frameworks and supports the dynamic plug and play concept of software components. Which Simulation Components are needed, and in which quantity, is defined in the AutomationML plant description file coming from the Modelling Tool (Section 3). The Simulation Components are identified by theirs class name (e.g. ”FMU Simulation Component”). If more than one Simulation Component of the same class is needed, additionally a specific instance name is used.

The Process Data Manager is part of the Simulation Manager. If a Simulation Component is instantiated by the Simulation Manager, a process data input list and an output list has to be registered at the process data manager. The Process Data Manager then compares the process data in the lists to the already registered process data by name. New process data will be added, already existing process data are linked together (Figure 8).

4.2 Scheduling

One important task of the Simulation Manager is to schedule all Simulation Components. For this pur-

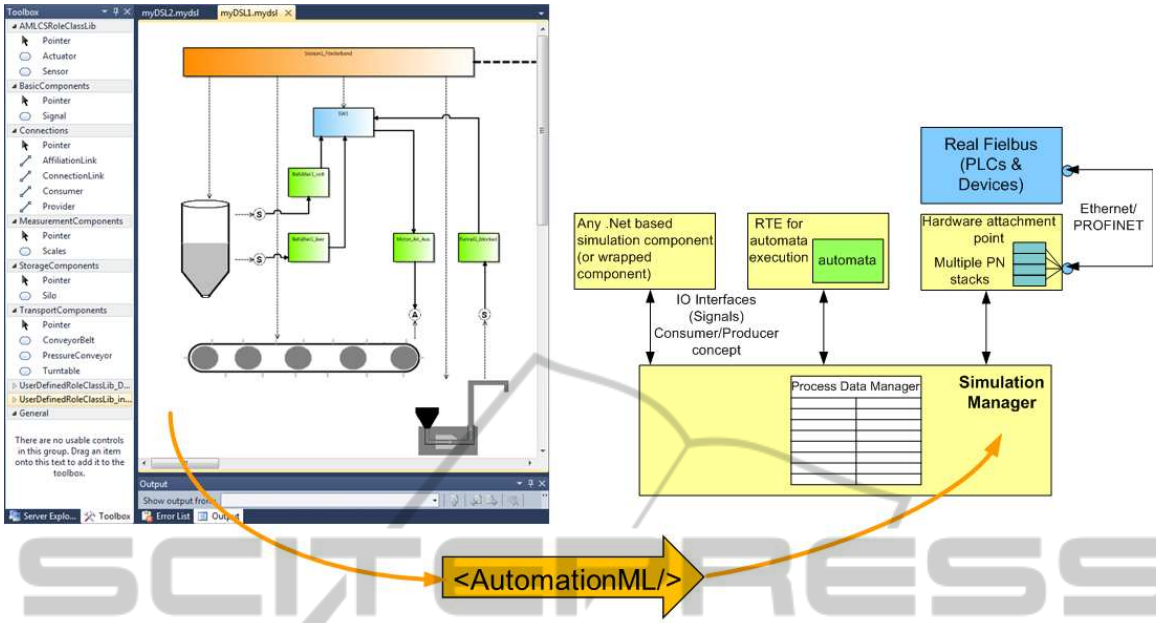

Figure 6: The PIM editor stores the plant model as AutomationML file. This file is then loaded into the Simulation Framework to instantiate all needed Simulation Components.

pose a simple static scheduling algorithm is used. Every Simulation Component has to declare, with what time step the component needs to be scheduled. For all these time steps, the least common multiple (LCM) is calculated. Every Simulation Component is scheduled at the beginning (t_0). Furthermore, each Simulation Component is scheduled again with it's specific time step, as long as the time step sum is smaller than the LCM. For example, in figure 7 three Simulation Components (SC_1, SC_2, SC_3) are scheduled. SC_1 with a time step of 10ms, SC_2 with a time step of 20ms and SC_3 with a time step of 40ms. The LCM is therefore 40ms. The resulting schedule is cyclic, this means, as soon as the end of the schedule is reached, the schedule starts all over again.

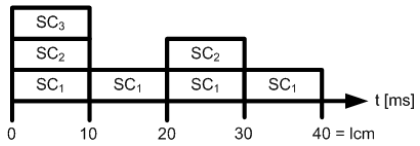

Figure 7: Simulation Manager example schedule.

Currently, the Simulation Framework is not capable of parallel processing of Simulation Components. Hence, the components are processed in sequence. It is the responsibility of the Process Data Manager to maintain two states of each process data (Figure 8). The first state contains the input process data of the components. After the processing of a Simulation Component, the resulting process data are written into the second state.

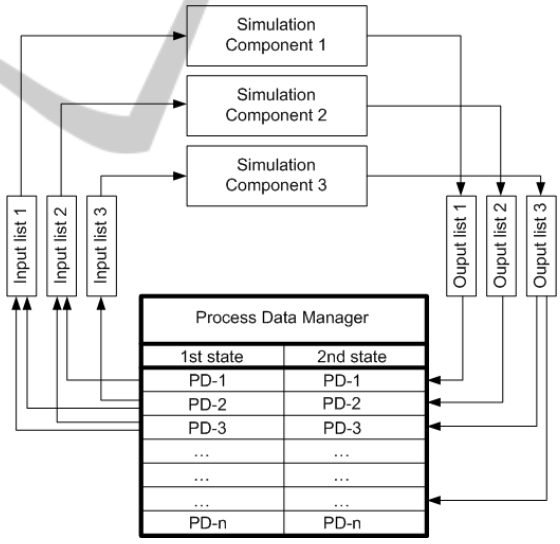

Figure 8: The Process Data Manager.

After all parallel Simulation Components had been processed, the resulting process data (second state) are written into the first state and are now available as input values for the next simulation step. Using this double buffer concept makes sure, that two or more Simulation Components, which are scheduled at the same time, work on the same set of process data.

4.3 Offline Simulation

For the offline simulation the AutomationML plant

description file is loaded and all plant behaviour models, represented by FMUs, are loaded and instantiated as FMU Simulation Components. Because the Simulation Component "FMU Simulation Component" is used more than once, the instances have to be identified by theirs instance names, which can be given by the file names of the loaded FMUs. After the creation of a schedule for all necessary simulation components, the components are executed. The actual process data in each time step are managed by the Process Data Manager and can be logged by a specialised simulation component.

4.4 HIL Simulation

Hardware-In-the-Loop (HIL) is an approach to test hardware devices such as PLCs or plant modules—these devices are then called unit under test (UUT). HIL differs from real–time or offline simulation by considering the 'actual' device in the testing loop. In a HIL test, a simulator (e.g. a special PC) simulates a model of the environment for the attached UUT (i.e. all other communicating partners of the overall system, except the UUT); this can be seen in figure 9. The purpose of HIL test is to make the UUT believe, that it is in fact connected to the real system. By using a HIL test, UUTs can be tested with realistic plant signals providing the necessary sensor/actuator triggers to fully exercise the UUT — without requiring the real system.

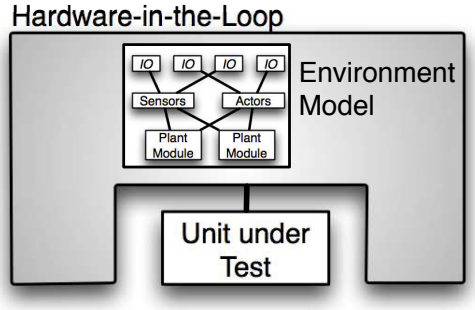

Figure 9: Hardware-in-the-Loop (HIL) Test Setup.

In automation, several parts of the overall system may become the UUT; e.g. when several PLCs are tested. Modular models as described in this paper, allow for a fast creation of the environment model by removing those parts from the overall model that correspond to the UUT.

Figure 10 shows a typical HIL case in the domain of industrial automation: A real PLC is tested; for this, the PLC model is removed from the overall model to create the environment model. The PLC is then tested for correct responses, i.e. in terms of (*i*) setting its actuator variables and (*ii*) performing its duties meeting the real–time requirements of the overall system. The appropriate sensor inputs needed by the PLC are sent at the right time by simulating the environment model exploiting the input sample space of the PLC under test.

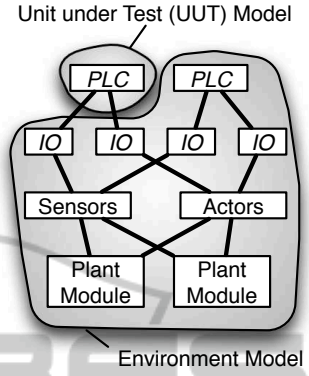

Figure 10: Typical HIL case for automation.

Some typical errors in the automation domain for which a PLC has to be tested are: (*i*) Functional input space errors, i.e. does the PLC respond correctly when a blocked funnel is sensed? and (*ii*) Time domain errors i.e. even if the PLC takes the right action; Was it done within the required time limit? For e.g. One such industrial automation test question covering the above points would be: if the funnel is blocked does the PLC stop the conveyor belt? - and does it do it in time?

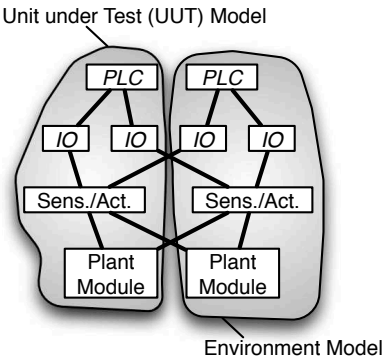

Figure 11: Another typical HIL case for automation.

Another typical automation testing scenarios is shown in figure 11: A whole plant module including its PLC, IO devices, and the plant is being tested.

For testing, two key questions have to be solved: (*i*) Where do the test cases come from? and (*ii*) Who defines whether a test case was successful or failed—this is the so–called test verdict. Generally speaking two solution approaches exist:

1-Model-Approach. 1–Model–Approach considers

one model for both test vector generation and code generation for the UUT (e.g. PLC); the test verdict is generated by comparing the test output with the prediction of the UUT model. The key issue with this approach is: since the implementation and the test cases are derived from the same model, only the code generator can be tested.

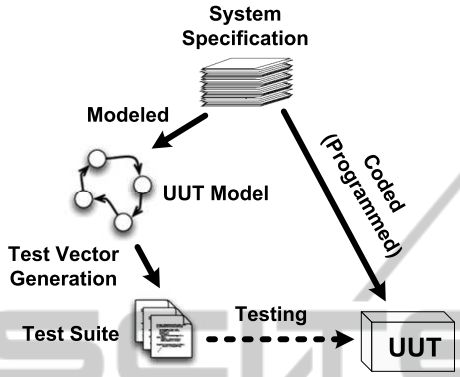

Figure 12: 2-Model-Approach.

2-Model-Approach. 2-Model-Approach (also see figure 12) on the other hand considers a separately implemented unit under test (UUT), i.e. the UUT in programmed using conventional programming languages (e.g. IEC 61131 languages, in case of PLC) based on an existing informal specification. In addition to this, a specific formal testing model is developed based on the same informal specification and is used for test vector generation; the test verdict is created by comparing the test output with the prediction of this test model. Since the generated vectors and the UUT code are not derived from the same model, this approach provides the necessary redundancy needed for testing. For our HIL test we use the 2-Model-Approach.

Our prototypical HIL test bed consists of a manually programmed (e.g. using IEC61131) UUT and the testing model. The test model can be obtained in two ways:

(*i*) by making use of the modular separation concept of AutomationML (as shown in figure 10) - i.e. in this case the environment model (overall model minus UUT model) is also the testing model. The simulation framework can then be used to simulate all the individual components (For e.g. other PLC models, IOs and plant models, etc.) of the test model. Such a testing scenario results in a closed loop test of the UUT.

(*ii*) by generating test vectors from a single automaton which is obtained by merging the automata of all the individual components of the environmental model. Such a test model can be used for an open loop test.

In both of the above two scenarios, the test model is used as a bases to extract 'abstract test input sequences' (our publication (Kumar et al., 2010b) details on the algorithmic concepts of test extraction) either for simulating the environmental model (closed loop) or for test execution (open loop). These extracted input sequences are first serialized and then built into appropriate industrial bus protocol frames (in our case ProfiNet IO frames). The HIL sends these sensor values to the UUT (PLC) at appropriate times, simulating the actual execution environment of the UUT taking the real time requirements into consideration. After the UUT reacts to these inputs, the outputs are constantly monitored and compared with the results predicted by the testing model.

Figure 13: A model factory.

Empirical Results. The HIL framework was used for both closed loop and open loop tests for our initial test studies using a MF shown in figure 13. This plant is used to produce and pack up bulk goods (e.g. popcorn). This modular plant comprises of 8 modules, e.g. a storage system, several transportation systems, a weighing mechanism, a bottling station, a robot, a heating facility, and a packaging module. The automation solution comprises of several distributed PLCs to control the plants, buses such as ProfiNet, and approx. 250 sensor and actuator signals.

For the first results, we performed open loop test with just 2 modules of the MF. To test the credibility of the generated test vectors - 24 mutants (16 functional (i.e. actuator errors) and 8 timing errors) were introduced into the PLC control logic, which were positively identified. A test model for closed loop test with 2 modules of the MF were modelled using Modelica FMUs; these FMUs were simulated using the proposed simulation framework to imitate the execution environment of the PLC. Further, tests were conducted for ideal plant scenarios and with 4 functional and 2 timing errors; which were correctly detected by the HIL framework. Table 1 summaries HIL test's initial results. Even though, the first results are positive - further studies are needed to check complex test scenarios; For e.g. including all 8 modules of the MF and by including error with an exhaustive exploitation of mutant sample space.

Table 1: HIL test results.

Type of Test	Inserted mutants		Detected mutants
	Functional mutants	Timed mutants	
Closed Loop	4	2	6
Open Loop	16	8	24

4.5 Anomaly Detection

The detection of system deterioration and of non–normal system behaviour will prove essential for the improvement of system quality and reliability: Production facilities could detect failures earlier and inspections can be planned whenever first sign of deterioration appears.

The key idea is rather simple: Let M be a system model which captures all fault relevant system aspects; here we use the model from section 3. Whenever M's behaviour shows a discrepancy to the system's current behaviour, the user will be alarmed.

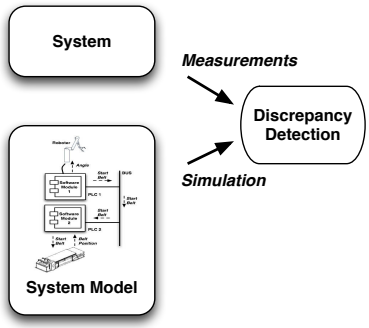

Figure 14: Detection of system deterioration.

This can also be seen in figure 14: During the runtime of the system, the simulations of the model M are constantly compared to the system behaviour and discrepancies are detected.

Generally speaking, the offline simulation from section 4 can be used for the anomaly detection. There are three main differences. The first one is, that the simulation is slowed down, so that the clock in the simulation matches the clock of the real world communication system. The second difference is, that a specialised simulation component is used, which is able to read all process data (inclusive time stamps) from a real plant. The third difference is, that the simulated process data and the real world process data are compared in every single simulation step. The idea is, that if the simulated process data are different to the real world process data, an anomaly is supposed to be detected (compare (Supavatanakul et al., 2006)).

This approach is currently tested for the application of anomaly detection: The predictions of the model are compared to measurements from the real plant; if a discrepancy occurs, an anomaly is signalled. For this, the MF discussed in section 4.4 is used.

For a first verification of the approach, we only use 3 of the plant modules and we define 3 types of errors: a blocked sensor, a sensor communication error, and a deterioration effects of a conveyer belt. Then, for each of the 4 cases (3 errors, one OK case), we recorded 30 production cycles and tried to detect the errors. All errors have been identified correctly.

5 CONCLUSIONS

Even though the development of the PIM editor and the Simulation Framework is still in progress, this software tool chain already enables a user to create an abstract and modular PIM model of a production plant and to store it in an AutomationML based file format. AutomationML is well suited for this purpose, because of its hierarchical CAEX structure and its extendable interface concept. These interfaces are currently used to describe the behaviour of the individual components in the plant model by state automatons as (self executing) FMUs, but the modular concept of Simulation Components also allows for different approaches.

The usage of the FMI is at the moment a necessary requirement to execute these automatons in the Simulation Framework. The Simulation Framework reads the required communication connections from the AutomationML plant model and instantiates all needed Simulation Components with the corresponding in- and output lists of process data. The process data then are managed by the Process Data Manager, which enables the individual Simulation Components to communicate among each other, and which provides them with the process data appropriate to the clock of the corresponding components.

For a demonstration purpose, this tool chain already had been used for the offline simulation of a production plant, a HIL test of a single component (a PLC) and for the anomaly detection (diagnosis) in an operating production plant. Future work will extend the PIM editor with the functionality of mapping real hardware to the model. As a result, the editor will be a PIM/PSM editor then.

REFERENCES

Allen, P., editor (2002). *The OMG's Model Driven Architecture*, volume XII of *Component Development Strategies, The Monthly Newsletter from the Cutter Information Corp. on Managing and Developing Component-Based Systems*.

Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T. A., Ho, P.-H., Nicollin, X., Olivero, A., Sifakis, J., and Yovine, S. (1995). The algorithmic analysis of hybrid systems. *THEORETICAL COMPUTER SCIENCE*, 138:3–34.

Brecher, C., Fedrowitz, C., Herfs, W., Kahmen, A., Lohse, W., Rathjen, O., and Vitr, M. (2008). Durchgängiges Production Engineering Potenziale der digitalen Fabrik. In Brecher, C., Schmitt, F. K. R., and Schuh, G., editors, *Wettbewerbsfaktor Produktionstechnik: Aachener Perspektiven*. Aachen AWK.

DASSAULT SYSTEMES (2011). Dymola. http://www.dymola.com.

Drath, R., Weidemann, D., Lips, S., Hundt, L., Lüder, A., and Schleipen, M. (2010). *Datenaustausch in der Anlagenplanung mit AutomationML*. Springer.

Gall, H. (2008). Functional safety IEC 61508 / IEC 61511 the impact to certification and the user. In *AICCSA '08: Proceedings of the 2008 IEEE/ACS International Conference on Computer Systems and Applications*, pages 1027–1031, Washington, DC, USA. IEEE Computer Society.

Graeser, O. and Niggemann, O. (2009). Planung der Übertragung von Echtzeitnachrichten in Netzwerken mit Bandbreitenreservierung am Beispiel von Profinet IRT. In *Echtzeit 2009 - Software-intensive verteilte Echtzeitsysteme GI-Fachauschuss*, Boppard, Germany.

IEC 62424 (2008). Festlegung für die Darstellung von Aufgaben der Prozessleittechnik in Fliessbildern und für den Datenaustausch zwischen EDV-Werkzeugen zur Fliessbilderstellung und CAE-Systemen.

Khronos Group (2011). Collada - 3d asset exchange schema. http://www.khronos.org/collada/.

Kühn, W. (2006). Digital factory: simulation enhancing the product and production engineering process. In *WSC '06: Proceedings of the 38th conference on Winter simulation*, pages 1899–1906. Winter Simulation Conference.

Kumar, B., Niggemann, O., and Jasperneite, J. (2010a). Statistical models of network traffic. In *International Conference on Computer, Electrical and Systems Science*.

Kumar, B., Niggemann, O., and Jasperneite, J. (2010b). Test generation for hybrid, probabilistic control models. In *Entwurf komplexer Automatisierungssysteme (EKA 2010)*. Magdeburg, Germany.

Mellor, S., Scott, K., Uhl, A., and Weise, D. (2004). *MDA Distilled: Principles of Model-Driven Architecture*. Addison Wesley.

Microsoft (2011). Managed extensibility framework. http://mef.codeplex.com/.

Modelica Association (2011). Modelica. https://www.modelica.org/.

Modelisar project (2011). Functional mock-up interface. http://functional-mockup-interface.org/.

MSDN (2011). Visual studio visualization and modeling sdk. http://code.msdn.microsoft.com/vsvmsdk.

Niggemann, O. and Otterbach, R. (2008). Durchgehende Systemverifikation im Automotiven Entwicklungsprozess. In *Tagungsband des Dagstuhl-Workshops Modellbasierte Entwicklung eingebetteter Systeme IV (MBEES)*, Schloss Dagstuhl, Germany.

Niggemann, O. and Stroop, J. (2008). Models for model's sake: why explicit system models are also an end to themselves. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*, pages 561–570, New York, NY, USA. ACM.

PLCopen (2011). http://www.plcopen.org/.

PNO (2007). Profinet specification iec 61158-5-10 (v2.1).

Sproston, J. (2000). Decidable model checking of probabilistic hybrid automata.

Stein, S., Kühne, S., Drawehn, J., Feja, S., and Rotzoll, W. (2008). Evaluation of OrViA Framework for Model-Driven SOA Implementations: An Industrial Case Study. In *6th International Conference on Business Process Management*, Milan, Italy.

Steiner, P. and Schmidt, F. (2003). Anforderungen und Architektur zukünftiger Karosserieelektroniksysteme. In *VDI Berichte Nr. 1789*.

Strasser, T., Sunder, C., and Valentini, A. (2008). Model-driven embedded systems design environment for the industrial automation sector. In *Proceedings of the 6th IEEE International Conference on Industrial Informatics*, Daejeon, South Korea.

Streitferdt, D., Wendt, G., Nenninger, P., Nyssen, A., and Lichter, H. (2008). Model driven development challenges in the automation domain. In *Computer Software and Applications, 2008. COMPSAC '08. 32nd Annual IEEE International*, pages 1372–1375.

Supavatanakul, P., Lunze, J., Puig, V., and Quevedo, J. (2006). Diagnosis of timed automata: Theory and application to the damadics actuator benchmark problem. *Control Engineering Practice*, 14(6):609–619.

Szulman, P., Assmann, D., Doerr, J., Eisenbarth, M., Hefke, M., Soto, M., and Trifu, A. (2005). Using ontology-based reference models in digital production engineering integration. In *Proceedings of the 16th IFAC WORLD CONGRESS*, Prague.