Closed-Loop Modelling in Future Automation System Engineering and Validation

Valeriy Vyatkin, *Senior Member IEEE,* Hans-Michael Hanisch, *Senior Member IEEE,* Cheng Pang, *Member IEEE* and Chia - han Yang, *Non-member*

***Abstract*—This paper presents a new framework for design and validation of industrial automation systems based on systematic application of formal methods. The engineering methodology proposed in this paper is based on the component design of automated manufacturing systems from intelligent mechatronic components. Foundations of such components' information infrastructure are the new IEC 61499 architecture and the Automation Object concept. It is illustrated in this work how these architectures, in conjunction with other advanced technologies, such as UML, Simulink and Net Condition/Event Systems, form a framework that enables pick-and-place design, simulation, formal verification, and deployment with the support of a suite of software tools. The key feature of the framework is the inherent support of formal validation techniques achieved on account of automated transformation among different system models. The paper appeals to developers of automation systems and of automation software tools via showing the pathway to improve the system development practices by combining several design and validation methodologies and technologies.**

***Index Terms*—manufacturing automation, mechatronics, software reusability, software verification and validation**

I. Introduction

THE major challenge of embedded system design has been recently defined by Henzinger and Sifakis in [1] as the ability to achieve the following goals:

(1) encompass heterogeneous execution and interaction mechanisms for system components;
(2) provide abstractions that isolate the design sub-problems requiring creativity from those that can be automated;
(3) scale by supporting compositional, correct-by-construction techniques; and,
(4) ensure the robustness of the resulting systems.

In particular, it is emphasized that "shortcomings of current design, validation, and maintenance processes make software the most costly and least reliable part of embedded applications" and the main "culprit" of that is "the lack of rigorous techniques for embedded systems design". The conclusion of [1] states that "…embedded systems design requires a more holistic approach that integrates essential paradigms from hardware and software design and control theory". The key features of such an approach are "shift of the emphasis from design synthesis to design verification—the proof of correctness", and "frameworks supporting transformations across heterogeneous model boundaries".

Very similar challenges have been recognized and dealt with in the industrial automation domain (e.g. see [2]), which includes good deal of embedded control systems design for manufacturing applications. In automation, the central part of the design concerns with development and programmatic implementation of control logic, and the ultimate design success criterion is the correct behaviour of the controlled manufacturing system. This is ensured by verification and validation (V&V) of the controllers through debugging and testing of their code, using simulation, and applying correct-by-design techniques. In particular, computer simulation is widely used for prototyping automated manufacturing systems and, less often, for verifying actual controller code (e.g. Control Build Validation software suit [3]). However, even the simulation-based debugging cannot reveal all potential pitfalls of control logic, especially when it is decentralised. As a result, complimentary techniques of mathematical proof of correctness (known as formal verification [4]) are being actively researched. One such technique is model-checking [5-7]. Model-checking can be used to analyse the embedded control part, including the control logic and some details of the underlying run-time platform, or the entire closed-loop plant and controller system.

The first approach is most common in verification of classic embedded systems and business software. The environment is considered there as an input to the embedded control and some non-determinism introduced in such inputs can help in proving the robustness of the verified embedded system.

Contrary, control systems are usually analyzed using closed-loop models. Model-checking using only an open-loop model of the controller may help to identify some undesired controller reactions. However, it is by no means appropriate to verify the correct behaviour of the controlled object which indeed is the major point of concern. This simple truth has often been (and is still) misunderstood or even neglected. In fact, no liveness property can be proven by an open-loop model. It is basic knowledge in automation technology that closing the control loop significantly changes the behaviour of both the

Manuscript received November, 17, 2007. The work was supported in part by the research grant VAIAS of the German Ministry of Education and Research, and by the University of Auckland research grant SRF 3607893.

Valeriy Vyatkin is with the Department of Electrical and Computer Engineering, University of Auckland, New Zealand (phone: +64-9-3737599 ext. 89437, e-mail: v.vyatkin@auckland.ac.nz).

Hans-Michael Hanisch is with Institute of Information Sciences, Martin Luther University of Halle – Wittenberg, Germany.

Cheng Pang and John Yang are with the Department of Electrical and Computer Engineering, University of Auckland, New Zealand

plant and the controller. Therefore, any claim that an open-loop analysis of behaviour would prove the correctness of the plant behaviour under control is wrong. There is a growing recognition of the benefits of the closed-loop approach in the research community dealing with formal verification of industrial automation systems, as justified, for example, in [8-10]. The main problem for wider adoption of the closed-loop verification is the lack of model design methods, which can transform this activity from a kind of art to a systematic routine, well integrated into usual activities of control engineers.

This paper addresses the issue of model development and reuse to cater for the closed-loop verification and validation techniques in industrial automation domain. A model-integrated design framework is proposed, which follows the object-oriented design principles based on using intelligent mechatronic components for design of automated manufacturing machines and systems [11-13]. The framework facilitates the closed-loop V&V of complex systems with decentralised control logic by supporting simulation, formal-verification and code deployment. As the implementation concerns, the proposed ideas can be used to extend the existing engineering support software tools, or as a foundation of new toolsets, by combining system design with the closed-loop V&V capability.

The main motivation for the formal verification research in automation so far has been improving the overall safety and robustness by finding bugs or undesired features, which could not be revealed by testing, either on real or simulated machine. In this paper another hypothesis on the utility of formal verification is presented and justified in the context of reconfigurable and adaptive manufacturing systems [14]. Testing of new configurations of control systems obtained as a result of physical reconfiguration of automated manufacturing systems becomes the main bottleneck for their fast commissioning. Therefore, an automated verification can essentially contribute to systems' flexibility.

The paper is structured as follows. Section II presents the problem statement and outlines the goal of this work. Section III outlines the features of the proposed engineering framework and the corresponding data flows to justify the needs for component-based design and the corresponding requirements to automation system's internal component organisation. The most important gaps between the proposed framework and the existing model-based engineering concepts are identified in Section IV. The key role of the IEC 61499 standard in bridging the practices of control systems design, software design, and the industrial automation legacy is also elaborated in Section IV. Then, the overall modelling structure is proposed in Section V, which is the base for designing model templates and is common for the analytic, simulation, computation and verification models. The discrete-state modelling is discussed in Section VI. In particular, the concept of transforming the hybrid Stateflow model to the equivalent discrete model in timed Net Condition/Event Systems is presented. Section VII presents the rationale and details of using formal verification for proving the correctness of new system configurations obtained as a result of a reconfiguration process. Then, Section VIII briefly explains the set of software tools used in the prototype implementation of the framework and the first application's experiences are then analyzed. The paper is concluded with a short summary and a plan for future work in Section IX followed by acknowledgements and references.

II. Problem Statement

Computer-aided verification and validation of automation systems requires models of machines' behaviour. Building these from scratch for every machine is very effort-consuming and thus not practical. A structural model composition approach is required, which is not only modular, but also reflects the hierarchical structure of manufacturing machines and systems.

In addition, computer simulation cannot reveal all the design pitfalls and cannot cope effectively with distributed systems, and reconfiguration. Thus, it needs to be appended by the automated formal verification techniques. Formal verification, in general, requires different type of models than simulation.

Traditionally, the research effort on improving the efficiency of embedded automation systems design was concentrating around programming technologies, like controller design patterns, refinement of high-level specifications to executable code, and validation of embedded controllers, regarding them as yet another embedded computer application. As a result, there is no design framework which can efficiently support model development, use and re-use as a consistent part of the automation software development cycle.

This work aims to propose:

- an architecture and design flow of automation systems which can accommodate behavioural models at every object scale and design stage;
- systematic modelling methods, based on the use of model templates, automatic model transformation and their re-use; and,
- closed-loop verification and validation procedures integrated with controller development.

III. Model-Integrated Design

In this work it is assumed that a new system, such as a machine or a production cell, is built from available intelligent mechatronic components (IMC), which are provided by their vendors along with controllers programmed to perform a certain set of operations, and with the ability to communicate with each other by means of messages and shared data. The controllers can be represented in an abstract model form or in some programming language, and can reside and be executed in the IMCs' embedded control devices (if any).

An overview of the design flow is presented in Fig. 1 and is explained as follows. It is assumed that the design will be computer-aided and facilitated by a software toolset, but not necessarily fully automatic. In the first design step of a new

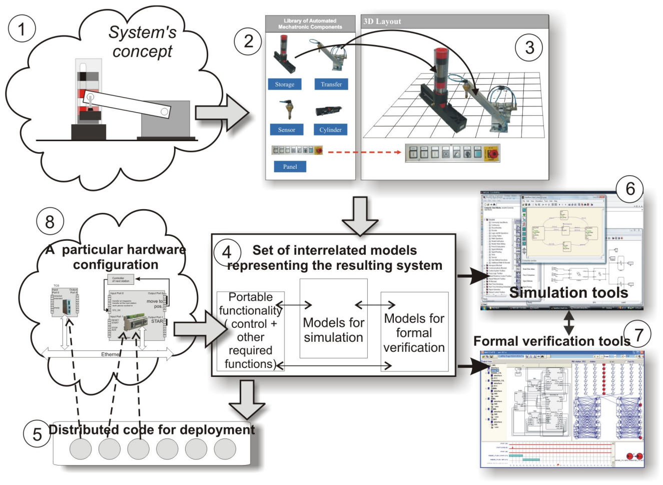
1
System's concept
2
Library of Automated Mechatronic Components
Storage
Transfer
Sensor
Cylinder
Panel
3D Layout
3
6
Simulation tools
8
A particular hardware configuration
4
Set of interrelated models representing the resulting system
Portable functionality (control + other required functions)
Models for simulation
Models for formal verification
Formal verification tools
7
5
Distributed code for deployment

Fig. 1. Sketch of the engineering framework supporting the integrated validation

system, a tool similar to the usual computer-aided design (CAD) tools will facilitate capturing the system's concept (item ① in Fig. 1) by picking IMCs from a library ② and placing them into the design space ③. It is also assumed that an IMC is provided by its vendor along with a repository of software components and models (further referred to as repository of the IMC intellectual property (IP) or IP repository), such as embedded controllers implementing its basic operations, rendering components visualizing its operations, and modelling components for simulation and verification. With the standardized interfaces and common protocols for inter-component communication, it will be possible to integrate IMCs one with another seamlessly in a plug-and-play way. The result of the integration is the overall system model consisting of a set of interconnected component models and integration models (e.g. protocols). Such system model enables closed-loop simulation and formal verification using the corresponding software tools ⑥, ⑦. Thus, the executable high-level specification of the system's controller will be assembled from the constituent embedded controllers possibly with some central controller-coordinator. The tool will facilitate mapping of the executable specification of the system's software onto a particular configuration of distributed networking hardware ⑧. Models for different verification and use activities (e.g. simulation, model-checking, execution) can be translated one to another, that is achieved by using common structure and interfaces in all model types. Verification will check whether the integration process has resulted in the desired behaviour of the entire system.

To illustrate the proposed ideas, a mechatronic system called Workpiece Distribution Station, or WDS, as shown in Fig. 1 ① will be used throughout the paper. The system consists of three mechatronic components: a Storage unit, a Transfer unit, and a control Panel with buttons. The Storage unit is composed from two parts: a Magazine storing a pile of workpieces and a Feeder shifting the lowest workpiece in the pile to the output position. The Magazine is equipped with a sensor that detects the presence of workpieces in the pile. The Feeder has two end position sensors: one for the retracted position and the other for the extended position of the Feeder's cylinder shaft. The Transfer unit grasps a workpiece at the output position of the Storage using its vacuum suction and carries it to the subsequent station. In our experiments two types of Transfer units with slightly different parameters are used. The first Transfer unit (L-Transfer) uses a stepper motor to drive the arm at a constant speed. The second Transfer unit (NL-Transfer) is driven by a DC motor whose speed changes continuously at the start-up and slow-down phases.

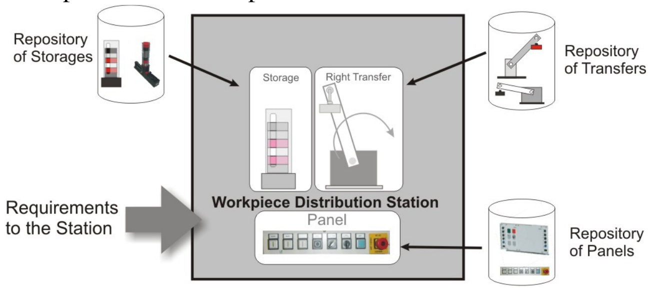

Fig. 2. Workpiece Distribution Station design scenario: fixed system structure with replaceable components

One particular design scenario of the WDS system is demonstrated in Fig. 2. The WDS has a fixed structure, of a Transfer unit picking up workpieces from the Storage and dropping them to some transportation system on the other side. There is a variety of available IMCs, each of which can be inserted into the system. However, the requirements of the system's behaviour do not change with the replacement of any mechatronic component. Validity of the same requirements needs to be verified for each new configuration. The requirements can include safety conditions, liveness (lack of deadlocks), functional correctness, and so on.

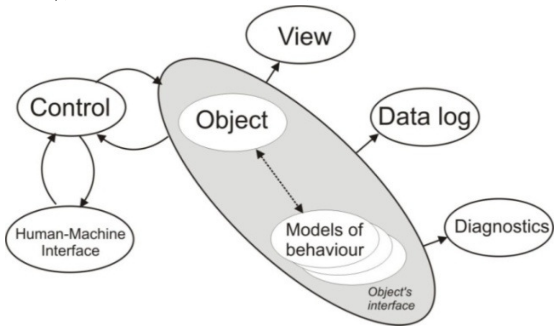

Fig. 3. MVC design pattern architecture

In our approach, the internal architecture of the components in the knowledge repository follows the object-oriented Model-View-Control (MVC) design pattern [15], adapted by Christensen in [16] to the domain of industrial automation and integrated with the IEC 61499 standard architecture [17]. According to the MVC pattern as indicated in Fig. 3, the core part of the IMC software is organized from two interconnected components:

- Autonomous (low-level) Controller, which implements a set of operations, published as services to be used directly or by higher level controller-coordinator, and

- Object, which provides an interface to the input/output signals of the IMC, or to one of the behavioural Models included in its IP repository.

The behavioural Models, provided in the IP repository, can be used for verification of the standalone IMC's behaviour, or as building blocks for creating the behavioural model of the whole system. The identical interface makes the Model component interchangeable with the Object component, thus providing an easy pathway from simulation to deployment. The combination of these two functions enables simulation of the system in closed-loop with the actual, ready for deployment control code. Moreover, the simulation model is obtained with a high degree of components' re-use.

Additionally, the View component supports interactive simulation by rendering the system's status based on the parameters provided by the Model. It also can be re-used in different deployment scenarios. Being connected to the real object instead of the model, the View component will render the object's status in real time. Other functions, such as Diagnostics and Database Logger are also fed by the data from the Object or the Model. In contrast, the Human-Machine Interface (HMI) component is connected in the closed-loop with the controller.

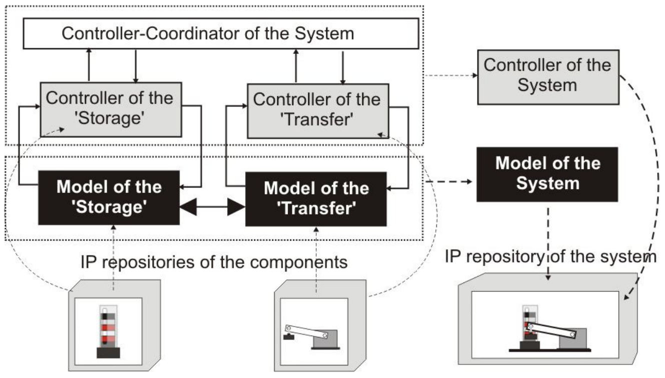
Controller-Coordinator of the System
Controller of the 'Storage'
Controller of the 'Transfer'
Controller of the System
Model of the 'Storage'
Model of the 'Transfer'
Model of the System
IP repositories of the components
IP repository of the system

Fig. 4. Component design of a manufacturing system from the mechatronic component

The MVC pattern allows precise closed-loop simulation and formal verification of complex mechatronic systems by re-using models of their constituent parts as seen in Fig. 4 for our illustrative example. Models and controllers of two mechatronic components are taken from their IP repositories and connected one with another in closed-loop. Models of different IMCs are connected with each other, via standardized interfaces, when such a connection is imposed by the system's structure.

As a result, the model of the system's behaviour can be designed with 100% re-use of the component models. The controllers enclosed in the knowledge repositories are intended to be re-used to the maximum extent, but some changes may be inevitable. The controller-coordinator in many cases will be required, although it can be avoided in some cases where each IMC has autonomous behaviour with loose dependence on the surrounding IMCs. In such cases, the IMCs' coordination can be purely interlock-based as proposed in [18]. The resulting system model can be used in the corresponding V&V activity. Finally, the controller and the model of the whole system can be included into the system's IP repository.

IV. Enabling Technologies: Related Work

There are existing integrated model-based engineering frameworks, such as Matlab/Simulink and Unified Modelling Language (UML), but still they do not cover the full spectrum of needs and thus cannot be used consistently throughout the whole engineering process as outlined in the previous section.

UML accumulates several practices from software engineering on refinement of specifications to executable code but lacks the support of thinking in the control engineering's way. Consequently, UML has no means to support simulation with necessary precision and weakly supports the system's deployment to distributed embedded targets.

In contrast, Matlab/Simulink provides sufficient control engineering support with code generation for some embedded targets but lacks support of automation architectures, such as IEC 61131-3 [18]. Reverse engineering is difficult as well as the mapping to distributed architectures.

None of these frameworks does support formal verification per se. There are some research works on generation of formal models from both UML and Simulink representations, for example [19], but they do not cover the full range of V&V activities needed in a consistent engineering framework. Thus, reverse engineering from the formal models back to UML or Simulink models has not been sufficiently worked out.

In addition to UML and Simulink, the proposed engineering framework was influenced by and partially relies on the following concepts and technologies:

- The IEC 61499 function blocks architecture [17],[18] which provides portable high-level executable specification framework for distributed automation. As a result, a complex distributed system combining control, visualization, simulation, data logging, and so on can be defined in a single "language" and then can be analyzed and deployed.
- The ideas of mechatronic architectures and model-integrated mechatronics as described in [11-13] and [21], worked out in detail for discrete-state modelling of mechatronic systems in [19, 22], and [23].
- The concept of Automation Object (AO), following [24] and [25], and Semantic Web technologies, in particular the Web Ontology Language (OWL) [26], which can be used as a mechanism for automatic integration of AOs.
- The progress in discrete-state formal verification methodologies and tools, and modular formal languages, in particular Net Condition/Event Systems [8]. There are other modular formal modelling languages, which can be used in the discrete-state verification, for instance input/output automata [27], CNets [28], and MCFSM [29];
- The OOONEIDA approach [30] which suggests re-use of the models from IP repositories of individual mechatronic components when a new system is created. The IEC 61499 standard and the Automation Object concept offer an architectural framework for such repositories.

The central element of the proposed framework is the IEC 61499 function block (FB) architecture. A FB is a component encapsulating data, algorithms of data processing, interfaces consisting of event and data inputs/outputs, and an execution control function. There are two types of function blocks:

- basic FBs, where a state machine implements the execution control; and,
- composite FBs, where the execution control is implicitly determined by the order of event and data interconnections among the constituent function blocks.

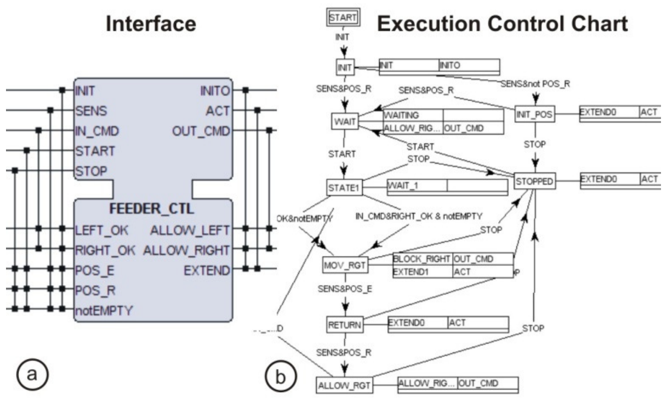

Fig. 5. An example of a basic function block definition (controller of the Storage component): (a) interface, composed of event and data inputs and outputs with associations between events and data; (b) Execution Control Chart: a state machine describing reaction on events and call of algorithms;

The controller of the Storage mechatronic component in Fig. 5 is an example of a basic FB. The controller implements the interlocking protocol from [18], which enables easy integration with other objects on the right and on the left from the Storage.

If compared to UML, the function block architecture provides a similar level of abstraction for definition of basic components, namely event-driven state machines and algorithms in various programming languages. Its distinctive features include the support of traditional PLC programming languages (of the IEC 61131-3 standard [19]) inside the algorithms, explicit component interfaces, composite components and sub-applications defined as networks of other components. In our opinion, the most important feature of the IEC 61499 architecture is the bridging between traditional computing architectures and control architectures. Being the architecture for embedded computing systems, the IEC 61499 supports the block diagram way of thinking of control engineers. Therefore, the use of function blocks to encapsulate intellectual property related to a mechatronic component provides a better re-use mechanism and makes the concept "What You Verify is What You Run" feasible.

When it comes to the deployment to distributed embedded targets, the use of IEC 61499 provides the following benefits. First, the function blocks are portable and have platform-independent execution semantics. Second, unlike plain software code, they include additional means for specification and documentation of their behaviour, such as Execution Control Charts (ECC) in basic FBs. These features simplify the creation of "parameterized templates" of functional and behavioural models, which can be generated on demand for a particular mechatronic device and for particular states of the system development workflow.

The IEC 61499 architecture provides the means to add or remove the simulation capabilities dynamically to the real device by substituting, appending, or modifying the device's management and scheduling functions. Thus, both simulation models and other software parts, such as control, can be implemented in a coherent way within the same architecture. Models of this form can be used for off-line simulation, as well as online for predictive control, as proposed in [31].

A system configuration in IEC 61499 is a model of an application deployed to computing devices, which can also be used for exhaustive simulation of a particular distributed system taking into account details of inter-device communications. Thus, the use of the IEC 61499 architecture enables verification of complete distributed control systems. The key elements of the system models in IEC 61499 are device and resource types and communication and service interface function blocks. These model elements can capture such fine details as the execution time (depending on CPU performance), memory size, network characteristics, scheduling policy due to specific network protocols, and so forth. It will be shown in the next sections that each of these model elements can be further formally modelled in a discrete-state formalism. The resultant discrete-state model will have the same structure as the original function block system, but can be used for model-checking.

V. MODELLING

The IMC's models required to be included in the IP repository can be classified in the following categories:

- Functional models consist of description and implementation of control algorithms, communication protocols, visualization functions, and so on. In the proposed framework, the IEC 61499 architecture is used as an intermediate functionality encapsulation language to represent such models.
- Behavioural models capture the uncontrolled behaviour of the object. Since there is no single modelling language which could be efficiently used in all the required V&V activities, at least three different forms of model representations seem to be necessary:
    1. A *hybrid model* in form of a hybrid automaton [33] is a mathematically rigorous analytic model capturing both continuous and discrete dynamics of the object. A hybrid automaton is a state machine where state transitions happen as discrete events, while continuous variables change according to some differential equations assigned to the states. A closest computer language is the Stateflow diagram of Matlab/Simulink.
    2. *Executable behavioural model* in form of basic and composite function blocks of IEC 61499 is used for off-line simulation in closed-loop with the actual controller code, or even for embedded online simulation, enabling predictive control.

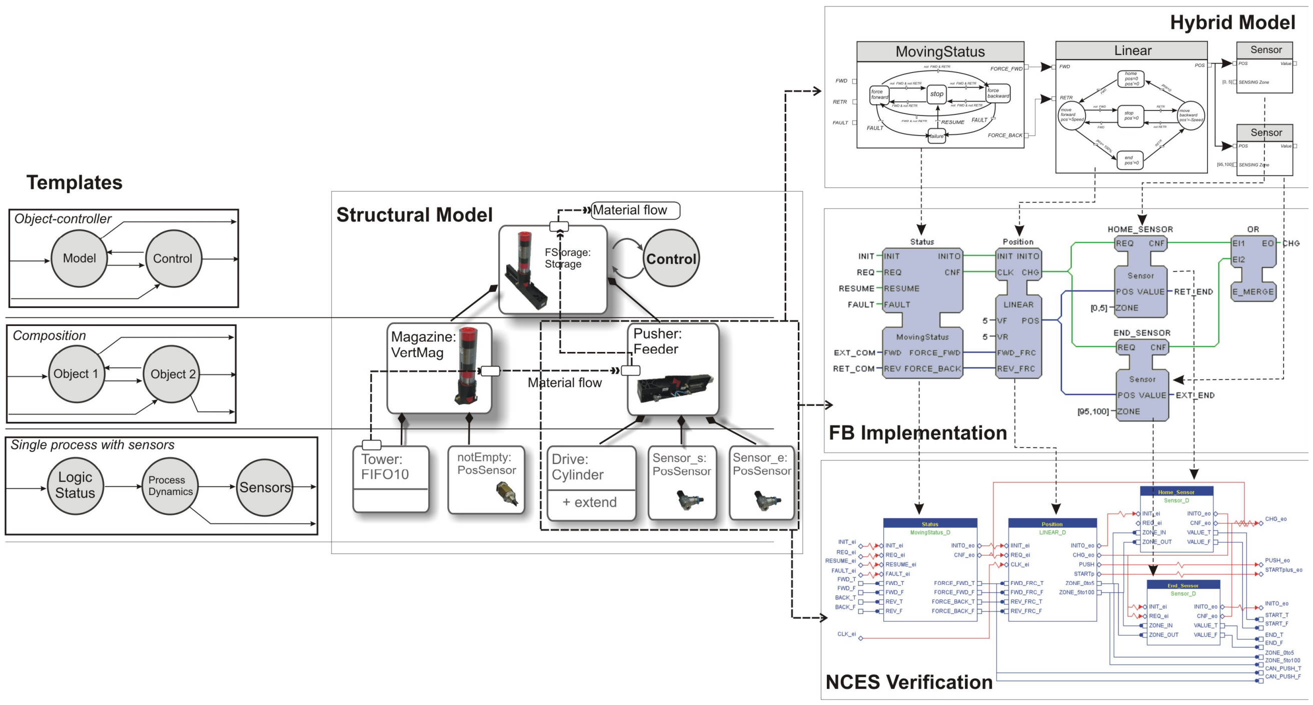

Fig. 6. Structural model of the instance FStorage_w_CTL of the class Storage with Control, the design templates, corresponding to different system levels.

3. The *discrete-state model* is mainly needed for formal verification by model-checking, which is very computationally hard for the hybrid models. In our case it is represented in the form of the modular formal language called Net Condition/Event Systems (NCES) [8], which is a well established modular discrete-state and discrete-time place-transition formalism similar to Petri nets [34], but with additional elements for modelling communication. NCES preserve the structure of distributed systems specified in a modular form of Stateflow diagrams and IEC 61499 function blocks and enables their formal verification. NCES have been used in a number of formal verification projects, for example in the one described in [23].

The set of required functions is determined by the use-case scenarios of mechatronic components, which are: (i) integration to a system, (ii) testing, simulation and deployment, and (iii) exploitation and maintenance. The latter scenario may include re-configuration of the system.

A mechatronic component may consist of other mechatronic components. Such a composite component has an additional description model in form of Mechatronic Object Diagrams (MOD) similar to object diagrams in UML. A conceptual example of such a diagram is presented for the Storage in Fig. 7. According to the MOD, the Storage can be seen as a hierarchical 3-level structure. The Storage object (level 1) is composed of two simpler mechatronic components, namely a Magazine and a Feeder (level 2). The Feeder, in turn, consists of a linear drive based on a pneumatic cylinder and two sensors indicating end positions of the drive (level 3). Behavioural models in each level follow the structural templates shown in the left part of Fig. 7. Thus, the Feeder's model is composed according to the template "Single process with sensors" which fits for purpose of modelling simple mechatronic objects. The right part of the figure shows the behavioural models of the Feeder obtained by application of this template. The Simulink block diagram, generated according to the same template is presented in Fig. 7, (right, top), with the correspondence between the Simulink and the function block models illustrated by arrows.

According to the template the model is composed of three types of elements explained as follows:

1) Logic status of the process, such as "Moving status" for the cylinder moving along a linear axis. This part of the model corresponds to the logic part of the mechatronic component and describes how to convert the control signals to the operation status. Therefore, this part is purely logical and can be implemented as a finite state machine (FSM).
2) Model of dynamic properties, such as "Linear" in Fig. 7. This part provides the state of the model that can be represented by a number of numeric parameters. Their evolution is best described by differential equations. On the other hand, reaction on logic control signals requires purely discrete description mechanisms like FSM. Therefore, such models are best described as hybrid automata, whose states can be associated with invariant functions as follows. The values of continuous system parameters are assigned according to the invariant functions of the form $I(x, t) = 0$ associated with a state, where x represents a numeric system parameter, and t denotes local clocks of the state. For example, in Fig. 7 the "Move forward" state of the Linear model describes the dependency *Pos':=Speed* (first derivative of the coordinate (*Pos*) is assigned to the input parameter Speed), i.e. $I(Pos,t) \equiv Pos' - Speed$.
3) Sensors physically present in the modelled object signal

that certain parameters are within a certain range of values. For instance, the end-position sensors of the Feeder indicate the piston's position in the intervals [0%, 5%] and [95%, 100%] of its working interval.

The model's elements are connected to each other via graphical links representing data flow. For example, the model of the dynamics "Linear" passes the numeric value POS (an integer within the interval [0, 100]) to the models of both sensors.

Such a modular model structure enables re-use of model components. For example, there could be two almost identical pneumatic cylinders, different only in their control signals. One cylinder could require separate signals for moving in each direction, while in the other one the high level of the same signal commands to move the piston forth and the low level drives it backwards. In this case, the models will be different only in the "Moving Status" element. Another benefit of such a modular structure is the opportunity to model explicitly malfunctions of certain elements, such as sensors, and see how the control logic reacts on it. The malfunctions can be modelled by adding non-determinism to the behaviour of some sensors. For doing that one just needs to substitute the model types of certain components, for example change type of the function block instance HOME_SENSOR from "Sensor" to some new type "Sensor_with_Fault", and program it accordingly.

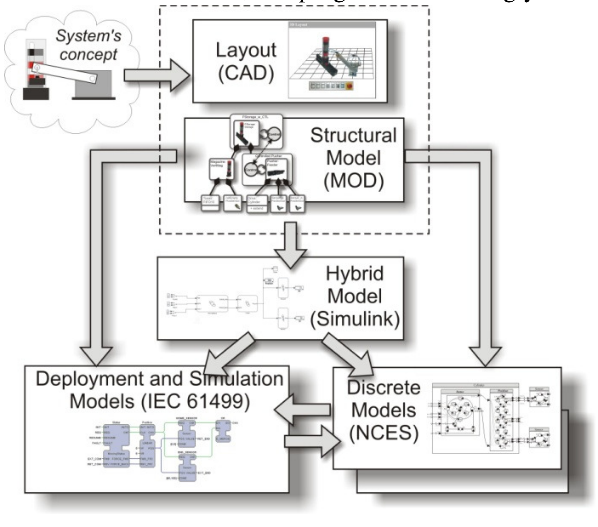

Fig. 7. The data-flow in the model co-design and model transformation

The template-based model design need be complemented with automatic model transformations in order to reduce the model-development effort. The flow of possible model transformations is illustrated in Fig. 8. At first, the CAD tool creates the structural model of the system captured in the MOD form. Then, the MOD can be refined into a Simulink block diagram based on a particular model template. Both the Simulink block diagram model and the MOD can be the source for automatic generation of the function block model and modular discrete-state model (e.g. in NCES). In addition, the function block model can also be used to generate the NCES model [35] and vice versa. The model-transformation capability will add more freedom to the designer, allowing original model development in the language best fit to the actual problem.

On the other hand, the MOD can also include descriptions of various relations between the components, for example directions and properties of material flow, as shown in Fig. 6. These descriptors can be used for automatic generation of the material passing parts of the model, for example, between Magazine and Feeder, or between Storage and Transfer. Discrete material flow between components is implemented using the Producer-Consumer protocol with buffer of size 1. The implementations of the protocol in two of three modelling languages (NCES and function blocks) are provided in Fig. 8. The NCES model is simplified and does not include passing of the workpiece type from module to module. The working scenario of the protocol is presented in Fig. 8(c).

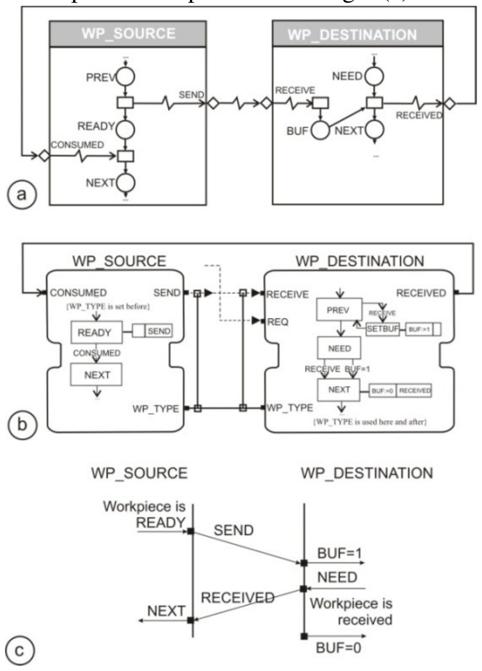

Fig. 8. The discrete "token" passing protocol between two models of mechatronic components, implemented in a) NCES, b) function blocks; c) a working scenario

VI. Discrete-State Modelling and Model-Generation

Discrete-state model-checking of an automation system can be used to prove the controller's safety (i.e. avoidance of undesired behaviour), correctness (compliance with the specifications) and robustness (i.e. correct reaction on some unexpected behaviour of the plant). Considerable progress has been achieved recently in modelling of programmable controllers using different discrete-state formalisms, both traditional PLC based (a survey is presented in [36]) and distributed, based on IEC 61499 (e.g. [37, 38]). Some of such approaches take into account fine but important details of program execution in embedded control devices, such as data interfacing, reaction on external events, multi-threading, and dynamic reconfiguration (e.g. [39]), The control logic (in a particular environment of operating system, runtime environment, etc.) must be verified in closed-loop with the model of the plant, as discussed in Section I. Having the explicit model of the plant can be beneficial for many reasons, even to reduce the complexity of model-checking. By introducing certain non-determinism in particular parts of the plant model, for example to the models of sensors, actuators, or communication lines, one can expose the controller to a more realistic range of inputs than it can be done in the simulation.

In the proposed framework the NCES formalism is used for discrete-state modelling. Its modularity can preserve the structure of discrete-state model in the analytic and simulation models. Thus, the controller part of the closed-loop model can be automatically generated. The plant part still has to be designed manually. Then, the modular discrete-state NCES model of the system can be used for state-space model-checking. Moreover, similar to function blocks, the modular nature and support of type instantiation allow existing NCES models to be re-used with minor changes.

In order to facilitate the development of plant models, the model-transformation and structural templates are applied. NCES models of dynamic properties are generated from hybrid Stateflow models using the straightforward discretization technique, presented in the following. In short, the discretization, for a given grid of the model parameter values $\{x_1, x_2,\ldots, x_n\}$, finds the corresponding clock readings $\{t_1,t_2,\ldots, t_n\}$ in which the hybrid model's parameter takes these values. Then the clock readings are used to parameterize the discrete timed model.

Let us assume that the original hybrid automaton HA is defined by state transition rules of the following kind: $S_i/I(x,t_i) \xrightarrow{x=\mathrm{D}} S_{i+1}$, where each state S_i has associated clocks t_i, which are reset to zero when the state becomes active, x is a continuous parameter, and $I(x,t)=0$ is a function defining an invariant while S_i is an active state. The predicate $x=\mathrm{D}$ defines the guard condition of the transition, i.e. transition to S_{i+1} occurs when x reaches the value D.

The discretization transformation of the HA to a discrete NCES model is done via an intermediate timed automaton (TA): HA + discretization info → TA → NCES. Consider the case of spatial discretization, when the interval of length D (domain of a model's parameter, e.g. coordinate) is divided into N subintervals of equal length. The corresponding timed discrete model needs exact values of times corresponding to the coordinate grid points.

The original hybrid automaton HA is transformed first to a timed automaton TA by converting each transition rule of the HA to a sequence of rules $S_i = s_0^i \xrightarrow{t=T(1)} s_1^i \xrightarrow{t=T(2)} \ldots \xrightarrow{t=\mathrm{T(N)}} s_{\mathrm{N}}^i = \mathrm{S}_{i+1}$ with the guard conditions defined by the function $T(i) = I_{\mathrm{t}}^{-1}\left(\frac{\mathrm{D}}{N}\mathrm{i}\right)$, where $I_t^{-1}(a)$ is a root of equation $I(a,t)=0$ for a given a, i.e. $I_{\mathrm{t}}^{-1}(x)$: $I\left(I_t^{-1}(x),x\right)=0\ \forall x$ (in case if the equation is not easily solved analytically, a numeric solution can be used).

For example, let's consider the linearly moving shaft of the cylinder in Fig. 10. For the state "Moving forward" the invariant function I is $I(x,t)=x(t)'+Speed=0$. The solution of this equation is $\mathrm{T}(i) = I^{-1}(x(t_i)) = \frac{\mathrm{D}\cdot i}{N\cdot Speed}$. Assuming D=100, the number of discretization grid points N=4 and *Speed*=5, the time values to be used in the transition conditions of the TA will be: $\mathrm{T}(i) = \{5i|\ i=1,4\} = 5, 10, 15, 20$.

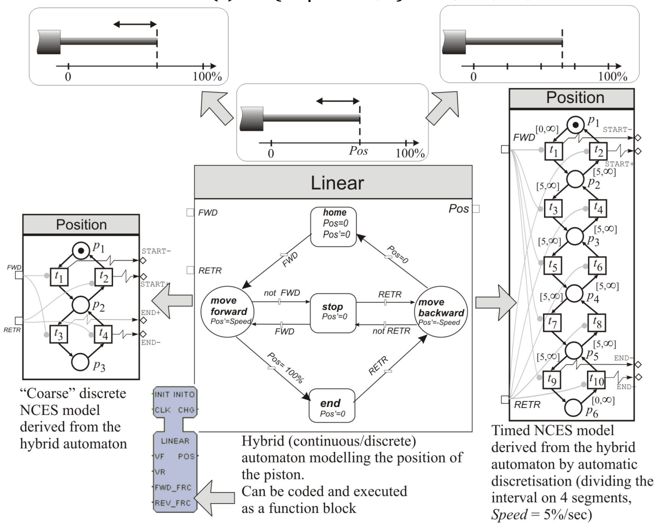
0
100%
0
Pos
100%
0
100%
Linear
FWD
Pos
RETR
home
Pos=0
Pos'=0
move forward
Pos'=Speed
not FWD
FWD
stop
Pos'=0
RETR
not RETR
move backward
Pos'=-Speed
Pos=0
Pos=100%
end
Pos=0
Position
FWD
RETR
START-
START
END-
END
"Coarse" discrete NCES model derived from the hybrid automaton
INIT INITO
CLK CHG
LINEAR
VF POS
VR
FWD_FRC
REV_FRC
Hybrid (continuous/discrete) automaton modelling the position of the piston.
Can be coded and executed as a function block
Position
[0,∞]
[5,∞]
Timed NCES model derived from the hybrid automaton by automatic discretisation (dividing the interval on 4 segments, Speed = 5%/sec)

Fig. 9. Using the model of dynamic properties of the cylinder (hybrid state machine encapsulated in function block LINEAR) for automatic generation of discrete models.

The use of NCES is especially beneficial for modelling of distributed systems, composed of several of such timed automata "running" concurrently. Each timed automaton TA can be trivially coded in terms of timed NCES, by encoding each state by a place and associating the *i*-th arc with the interval [T(i), ∞] (in timed NCES an interval [l, h] is associated with a place-transition arc where l denotes the lower bound of the permeability time interval and h corresponds to the upper bound).

The transformation is illustrated in Fig. 9 for the "Linear", hybrid automaton model. In this example, the role of x is played by *Pos*, and the state invariant looks like: *Pos*'+*Speed* = 0. The hybrid "Linear" module is used as the source for two purely discrete-state models, explained as follows.

The one on the left-hand side is a very simplified 3-state model of the linear movement that distinguishes three positions: home (place p1), end (p3), and in between (p2). This model distinguishes static and dynamic states of the drive, but it does not reflect precisely the location of the piston while in movement.

The model on the right is obtained by the discretisation of the cylinder's interval into 4 segments of equal length, assuming the Speed=5% of the interval per time unit. A token in places p2, p3, p4, p5 models the position of the cylinder in one of the intervals. Using this model, it will be possible to distinguish between positions of the cylinder.

Both models can be used in place of the "Linear" module in the network in Fig. 7 (right, bottom). Depending on the used model, the precision of model-checking and its complexity (i.e. the number of reachable states) will be different. The discretisation is an engineering trade-off which obviously decreases the precision of continuous parameters. In return it allows checking closed-loop systems, having reasonably complex controllers and reasonably detailed plant models. The hybrid "Linear" model can be also easily implemented as the corresponding function block.

VII. Formal Verification for New Configuration Testing

The verification experiment described in this section follows the design scenario illustrated earlier in Fig. 2. Once the new system is designed from the corresponding IMCs, its simulation and verification models will be automatically “assembled” and then the verification can be performed. The examples of the invariant conditions describing the allowed or forbidden situations for the whole system are as follows:

1) The Transfer and workpiece in the Feeder never collide;
2) The Transfer is not attempting to return to the Storage with a workpiece when there is another workpiece in the extended Feeder. Such a situation can happen when the delivery area of the Transfer gets occupied while it moves, so the controller of the Transfer may decide to go back; and,
3) A workpiece is released by the Transfer only at the unloading position, opposite to the Storage unit.

The verification will check the validity of the model’s behaviour against the specifications, highlighting those invariants which failed for the given model. Then the simulation models will be used to illustrate the failures by playing back the animated behaviour leading to the failure. Following this way, errors in controllers can be easily detected to avoid the time consuming testing process.

In particular, problems can arise when integrating mechatronic components with controllers not tailored to each other, but designed according to an interlocking protocol. Slight differences in the protocol implementation in one communicating side can lead to completely different behaviour of the system. In the described experiment, one of the used Storage units had a controller incorrectly setting the interlocking condition “ALLOW_RIGHT”, which enabled Transfer to approach right after it has started pushing the workpiece from the left position, rather than on arrival to the right position. Conducted simulations did not reveal any problem in the system’s behaviour. However, the model-checking has shown a situation when the collision of the Transfer and the workpiece can happen. The problem shows itself only when a very light workpiece is followed by a very heavy one in the magazine. As illustrated in Fig. 10 (a), in this case the Transfer returns back to the Storage before the workpiece arrives, and the workpiece hits the sucking nozzle of the Transfer. The figure shows the *x* coordinate of both Feeder and Transfer as a function of time. Two trajectories are shown for the Transfer, one for the light workpiece (gray line) and the other (thick black line) for the heavy. The Feeder’s trajectory is shown for the case of the heavy workpiece.

The specification of the collision needs to take into account that the Transfer is already in the “potential collision area”, while the Feeder enters its potential collision area (Fig. 10(b)). One sees in Fig. 10(a) that the collision does not happen if a heavy workpiece follows another heavy one. The model-checking of the discrete-state timed NCES model explores all possible combinations of workpiece sequences in the magazine and includes the scenario leading to the collision.

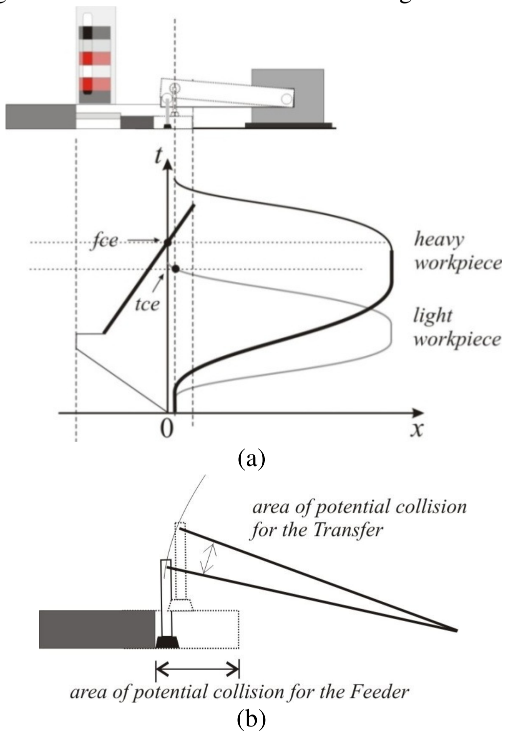
t
fce
tce
heavy workpiece
light workpiece
0
x
(a)
area of potential collision for the Transfer
area of potential collision for the Feeder
(b)

Fig. 10. The use of hybrid model analysis to find proper discretization intervals including the collision events. a) *fce*=feeder collision event, *tce*=transfer collision event; b) enlarged collision area

In real-life applications it is expected that the reasons leading to the failures will be fixed by the designer manually (in the code of controllers). As a result of this process, the system will be commissioned only if it is free of any errors and fully complies with the specifications.

The structural and semantic similarities of NCES and function blocks help to identify not only the overall design pitfalls but also the specific function blocks causing the problem. The similarity of the NCES and the FB models can be seen from Fig. 6. The NCES model has the same topology as the original function block network while each constituent module also has a similar interface compared to its counterpart function block.

Although the hybrid model-checking is computationally complex to be used for comprehensive model analysis of closed-loop models, especially with a sophisticated controller part, it can be extremely useful in the validation framework if applied in a limited scale, which can be illustrated in the sample system as shown in Fig. 10.

The events of entering the potential collision areas are determined by the coordinate of each object and need to be included in the behaviour of their respective NCES models. Provided that the position in those models is generated by the discretized counterpart of the “Linear” model (from Fig. 9), the timing intervals on the arcs need to include the time of such event occurrences. The collision points can be derived absolutely formally by exposing the system’s hybrid model, which can be obtained following the pattern in Fig. 4 as a combination of the “black box” models of the Transfer and

Feeder dynamics plus models of their controllers, to the CheckMate model checker [40] that works with Matlab/Simulink Stateflow.

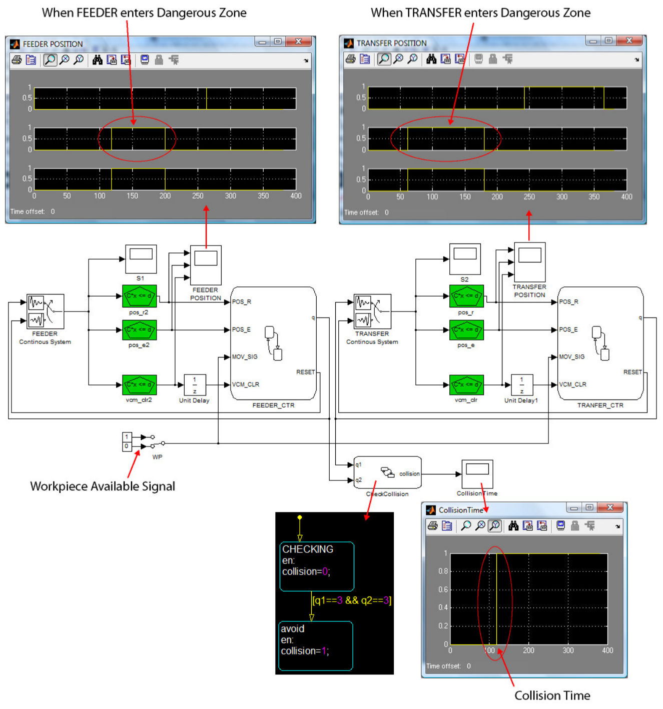

Fig. 11. Full CheckMate model for finding collision time of the FEEDER-TRANSFER system

Fig. 11 presents a sample CheckMate model which may be used to find the collision point of the system once it is modelled properly in Simulink. This model is built in order to verify whether the collision between the Feeder and Transfer will happen. The CheckMate model is constructed by combining standard Simulink blocks (Finite State Machine Block (FSMB)) with two custom block types: Polyhedral Threshold Bloc (PTHB) sets up the constraints regarding to the positions of Feeder and Transfer (i.e. "tate" presents "Transfer at Extended position" and "fatr" presents "Feeder at Retrieved position" etc). The Switched Continuous System Blocks (SCSB) sets the initial values and the parameters relating to this verification scenario. FSMB contains the hybrid state models of the closed-loop control system that is verified. For each type of workpiece the corresponding movement duration (from start to the entrance of the collision area) can be derived and included in the timed NCES model as described in Section V.

In order to find the collision time of the Feeder and Transfer units, Matlab/Simulink and CheckMate were used to model both the plants and the controllers of the two units. The model is built based on the assumption where the collision happened when both Feeder and Transfer enter the "dangerous zone" (as per Fig. 11). There are several ways one can find the collision time with Matlab/Simulink package, one of which is to use CheckMate as demonstrated here. In Fig. 12, the closed-loop model of the system containing Feeder and Transfer is built, and the scope monitors the positions of the two units and plots them in respect to time.

In the scenario described in Fig. 12, the collision happened at time 118.75 sec after arrival of "workpiece available signal". This is when both Feeder and Transfer enter the dangerous zone. Therefore the discretisation rate must be chosen so that this collision point at time 118.75 sec will be included in the grid.

VIII. Tools and Experiences

The V&V framework for Intelligent Mechatronic Components outlined in this paper has been partially implemented in a number of software tools. In its current state, a mechatronic system can be initially modelled in function blocks using FBench [41], which is an open-source integrated extensible development environment for engineering, simulating, analyzing, and deploying IEC 61499 applications. The function block models of controllers are then automatically translated into the functionally equivalent NCES models by the NCES model generator plug-in of FBench, while the plant models are manually created in the Visual NCES Editor (ViEd) [42]. ViEd is a full-featured editor for designing and editing NCES models represented in XML files, which can be stored in a common repository to accelerate the modelling process. The NCES models are then assembled and visually analyzed and verified using the model checkers implemented in the Visual Verifier (ViVe) [42] with safety and liveness properties, for example, specified in both computational tree logic [43] and predicate logic. ViEd and ViVe have been integrated and form the Visual Verification framework. Furthermore, the Web Ontology Language and the Protégé tool [44] are used as a framework to define interrelations between models as proposed in [45].

The presented design methodology has been applied in several research projects, starting from the testbed consisting of three mechatronic stations whose implementation was reported in [46], and extending the scale by an order of magnitude as described in [31, 48-50]. The design templates and model transformation can be efficiently applied to the graphical models of Simulink, function blocks, and NCES using the graphs transformation approach supported, for example, by the AGG software tool [51].

Both qualitative and quantitative benefits were meant to be achieved and the conducted laboratory experiments prove these expectations. The qualitative benefits are in enabling the design of more intelligent automation systems having better reconfiguration potential than the existing systems, and higher reliability and robustness. This is a clear advancement as compared to the state of the art in industry, where the simulation in the loop with the actual, ready to be deployed code is not common. The use of IEC 61499 function blocks and of the MVC pattern enables this scenario, and the model transformations allow the application of model-checking in the cases when analytic methods and hybrid verification are not applicable for complexity reasons. Another qualitative benefit is the possibility of running "embedded simulation" in the deployed configurations, providing systematic use of the predictive control concept. The quantitative benefits are in reduced design effort for creation of new systems and of their reconfigurations.

IX. Conclusions and Outlook

This work attempted to outline the methodology of holistic automation systems design which inherently supports a range of verification and validation scenarios as a part of system development. The architecture of the corresponding design framework is presented, and the underlying models are proposed. It is shown how the IEC 61499 architecture provides the executable specification level that enables this methodology. In particular, it is shown which features are essential and how they are to be applied in order to implement the holistic design concept and benefit from it.

In order to integrate models of plant behaviour in all verification and validation scenarios, a method for structural design of such models is proposed. The method is implemented in the form of graph templates supported by standardized interfaces and protocols and model transformation techniques. The latter allow deriving from analytic Simulink/Stateflow diagram both an executable function block simulation model and modular discrete-state model suitable for state-space model-checking. The discretization algorithm is presented in a constructive way along with a method for deriving the discretization interval using the hybrid model (in Matlab Simulink/Stateflow and CheckMate). Summarizing, the reported results of the on-going research contribute to solving the grand challenge [1] by developing a way to encompass heterogeneous execution and interaction mechanisms for system components (analytic, simulation and verification models and scenarios), provide abstractions that isolate the design sub-problems requiring human creativity from those that can be automated (e.g. automatic model-generation for models of controllers and model transformation for models of plant), which enables correct-by-construction models (developed following the proposed templates and patterns), and eventually ensures the robustness of the entire system.

However, the amount of work to completely solve the "embedded grand challenge" in the industrial automation domain is still enormous and cannot be accomplishedd by the work of one group of researchers. Our efforts will be focusing on the development of standardized data models supporting the model types discussed in this paper, and integration of the software tools in an open tool chain implementing the design flow from Fig. 1. The framework is intended to be applied to more complicated industrial systems in order to demonstrate its tangible benefits.

References

[1] T. A. Henzinger and J. Sifakis, "The Discipline of Embedded Systems Design", IEEE Computer Magazine, vol. 40, pp. 32-40, 2007.

[2] H.-M. Hanisch, "Closed-Loop Modeling and Related Problems of Embedded Control Systems in Engineering," in *Proc Intl Conf Abstract State Machines: advances in theory and practice (ASM'04)*, Lutherstadt Wittenberg, 2004, pp. 24-28.

[3] TNI-Software, "Control Build Validation," [Online (2008, April)]: http://www.tni-software.com/fr/produits/controlbuild/index.php.

[4] Wikipedia, "Formal Verification," (2008, April) [Online]. Available: http://en.wikipedia.org/wiki/Formal_verification.

[5] E. M. Clarke, O. Grumberg, and D. A. Peled, Model Checking. Cambridge: The MIT Press, 1999.

[6] G. Frey and L. Litz, "Formal methods in PLC programming," *in Proc* 2000 IEEE Conf Systems, Man, and Cybernetics, pp. 2431-2436, vol.4.

[7] W. Farn, "Formal verification of timed systems: a survey and perspective," Proceedings of IEEE, vol. 92, pp. 1283-1305, 2004.

[8] H.-M. Hanisch and A. Lüder, "Modular modeling of closed-loop systems," in *Proc of Colloquium on Petri Net Technologies for Modeling Communication Based Systems*, Berlin, Germany, 2000, pp. 103-126.

[9] J. M. Machado, B. Denis, J. J. Lesage, J. M. Faure, and C. L. Ferreira da Silva, "Increasing the efficiency of PLC program verification using a plant model," in *Proc Intl Conf on Industrial Engineering and Production Management (IEPM's 2003),* Porto, Portugal, 2003, p. 10-16.

[10] V. Vyatkin and H.-M. Hanisch, "Verification of Distributed Control Systems in Intelligent Manufacturing," *Journal of Intelligent Manufacturing*, vol. 14, pp. 123-136, 2003.

[11] V. Vyatkin, "Intelligent mechatronic components: control system engineering using an open distributed architecture," *in Proc of 2003 IEEE Conf on Emerging Technologies and Factory Automation*,pp.277-284,v.2

[12] J. L. M. Lastra, "Reference Mechatronic Architecture for Actor-based Assembly Systems", *Doctor of Philosophy Thesis*, Tampere University of Technology, Tampere, Finland, 2004.

[13] K. Thramboulidis, "Model-integrated mechatronics - toward a new paradigm in the development of manufacturing systems," *IEEE Transactions on Industrial Informatics*, vol. 1, pp. 54-61, 2005.

[14] Y. Koren, U. Heisel, F. Jovane, T. Moriwaki, G. Pritchow, H. Van Brussel, and A. G. Ulsoy, "Reconfigurable Manufacturing Systems," *CIRP Annals*, vol. 48, pp. 527-540, 1999.

[15] "Model-View-Controller design pattern," (2008, April) [Online]. Available: http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html.

[16] J. H. Christensen, "Design patterns for systems engineering with IEC 61499," in *Proc Conf Verteilte Automatisierung*, Magdeburg, Germany, 2000.

[17] IEC 61499-1, Function Blocks - Part 1 Architecture, International standard, International Electrotechnical Commission, 2005.

[18] V. Vyatkin, IEC 61499 Function Blocks for Embedded and Distributed Control System Design, Instrumentation Society of America, USA, 2007.

[19] International Electrotechnical Commission, Programmable Controller - Part 3: Programming Languages, IEC 61131-3 Standard. Geneva: International Electrotechnical Commission, 1993.

[20] M. Bonfe and C. Fantuzzi, "Design and verification of mechatronic object-oriented models for industrial control systems," *in Proc 2003 IEEE Conf on Emerging Technologies and Factory Automation, (ETFA'03),* pp. 253-260 vol.2.

[21] [19]M. Bonfé and C. Fantuzzi, "Mechatronic objects encapsulation in IEC 1131-3 norm," *in Proc of IEEE Intl Conf on Control Applications, 2000*, pp. 598-603.

[22] V. Vyatkin and H.-M. Hanisch, "Re-use in Formal Modeling and Verification of Distributed Control Systems," *in Proc. 2005 IEEE Conf on Emerging Technologies and Factory Automation,* Catania, Italy, 2005.

[23] H.-M. Hanisch *et al.*, "Formal validation of intelligent-automated production systems: towards industrial applications," *Intl Journal of Manufacturing Technology and Management*, vol. 8, pp. 75-106, 2006.

[24] International Electrotechnical Commission, Automation Objects for industrial-process measurement and control systems - IEC SB3/TC 65, working draft. Geneva, 2002.

[25] R. W. Brennan, L. Ferrarini, J. L. M. Lastra, and D. V. Vyatkin, "Automation objects: enabling embedded intelligence in real-time mechatronic systems," *Intl Journal of Manufacturing Research*, vol. 1, pp. 379-381, 2006.

[26] W3C Recommendation 10th February 2004, "OWL Web Ontology Language Guide," (2008, April) [Online]. Available: http://www.w3.org/TR/2003/PR-owl-guide-20031215/.

[27] N. Lynch and M. Tuttle, "Hierarchical Correctness Proofs for Distributed Algorithms," in *Proc 6th ACM SIGACT-SIGOPS Symp on Principles of Distributed Computing*, Vancouver, Canada, 1988, pp. 137-155.

[28] N. Hagge and B. Wagner, "A new function block modeling language based on Petri nets for automatic code generation," *IEEE Trans on Industrial Informatics*, vol. 1, pp. 226-237, 2005.

[29] E. W. Endsley and D. M. Tilbury, "Modular verification of modular finite state machines," in *Proc 2004 IEEE Conference on Decision and Control (CDC'04),* pp. 972-979, vol.1.

[30] V. Vyatkin, J. H. Christensen, J. L. M. Lastra, "OOONEIDA: An Open, Object-Oriented Knowledge Economy for Intelligent Distributed Automation,"*IEEE Trans on Industrial Informatics,* vol.1, pp. 4-17, 2005.

[31] G. Black and V. Vyatkin, "An Ontology-based Reconfiguration Agent for Intelligent Mechatronic Systems," *in Proc 4th Intl Conference on Holonic and Multi-agent systems in Manufacturing*, Regensburg, Germany, 2007.

[32] "Storage-Transfer Case study," (2008, April) [Online]. Available: http://www.fb61499.com/StorageTransfer.html.
[33] T. A. Henzinger, "The theory of hybrid automata," in *Verification of Digital and Hybrid Systems, M. K. Inan and R. P. Kurshan, (Eds),* New York: Springer-Verlag, 2000, pp. 265-292.
[34] M. Zhou and K. Venkatesh, "Modelling, Simulation and Control of Flexible Manufacturing Systems: A Petri Net Approach". New Jersey: World Scientific Publishing Company, 1999.
[35] C. Pang and V. Vyatkin, "Automatic Model Generation of IEC 61499 Function Block Using Net Condition/Event Systems,*" in Proc 2008 IEEE Conf on Industrial Informatics (INDIN'08)*, Daejeon, Korea
[36] M. Bani Younis and G. Frey, "Formalization of Existing PLC Programs: A Survey," in *Proc IEEE/IMACS Multiconference on Computational Engineering in Systems Applications (CESA'03),* Lille, France, 2003.
[37] C. Schnakenbourg, J. M. Faure, and J. J. Lesage, "Towards IEC 61499 function blocks diagrams verification," in *Proc 2002 IEEE Conference on Systems, Man and Cybernetics*, vol.3.
[38] M. Stanica, "Behavioral Modeling of IEC 61499 Control Applications." *Doctor of Philosophy Université de Rennes*, Rennes, France, 2005.
[39] C. Sunder, H. Rofner, V. Vyatkin, and B. Favre-Bulle, "Formal description of an IEC 61499 runtime environment with real-time constraints," *in Proc 5th IEEE Conf on Industrial Informatics,* Vienna, Austria, 2007, pp. 853-859
[40] "Checkmate hybrid model checker," (2008, April) [Online]. Available: http://www.ece.cmu.edu/~webk/checkmate.
[41] The University of Auckland, "FBench - Open Source Function Block Engineering Tool," (2008, April) [Online]. Available: http://oooneida-fbench.sourceforge.net/.
[42] The University of Auckland, "Visual Framework for Verification of Function Blocks," (2008, April) [Online]. Available: http://www.fb61499.com/valid.html.
[43] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM Trans. Program. Lang. Syst*., vol. 8, pp. 244-263, 1986.
[44] The University of Stanford, "The Protégé Project," (2008, April) [Online]. Available: http://protege.stanford.edu.
[45] Y. Al-Safi and V. Vyatkin, "An Ontology-based Reconfiguration Agent for Intelligent Mechatronic Systems," *in Proc 4th Intl Conference on Holonic and Multi-agent systems in Manufacturing*, Regensburg, Germany, 2007.
[46] V. Vyatkin, H.-M. Hanisch, S. Karras, and X. Cai, "IEC 61499 as an architectural framework to integrate formal models and methods in practical control engineering," in Electric Automation SPS/IPC/Drives, K. Bender et al.(Eds.) Heidelberg, Hüthig, 2002, pp. 310-318.
[47] M. Hirsch, C. Gerber, H. M. Hanisch, and V. Vyatkin, "Design and Implementation of Heterogeneous Distributed Controllers According to the IEC 61499 Standard - A Case Study," in *Proc 5th IEEE Intl Conf on Industrial Informatics*, 2007, pp. 829-834.
[48] A. R. Sardesai, O. Mazharullah and V. Vyatkin, "Reconfiguration of mechatronic systems enabled by IEC 61499 function blocks," *in 2006 Australasian Conf on Robotics and Automation*, Auckland, New Zealand.
[49] Martin Luther University of Halle-Wittenberg, "FESTO Modular production system model," (2008, April) [Online]. Available: http://aut.informatik.uni-halle.de/forschung/testbed/index.de.php.
[50] The University of Auckland, "Intelligent mechatronic testbed," (2008, April) [Online]: http://www.ece.auckland.ac.nz/~vyatkin/mitra_lab.html.
[51] G. Taentzer, "AGG: A Graph Transformation Environment for Modeling and Validation of Software," in Applications of Graph Transformations with Industrial Relevance. Vol. 3062/2004: Springer Berlin/Heidelberg, 2004, pp. 446-453.

Valeriy Vyatkin is graduated with a Diploma Degree in Applied Mathematics from Taganrog State University of Radio Engineering (TSURE), Taganrog, Russia in 1988. He holds Dr. Sci. degree (1998) and Ph.D. (1992) earned at the same University, and Dr. Eng. (1999) degree earned at Nagoya Institute of Technology, Nagoya, Japan, in 1999.

Currently he is Senior Lecturer with the Department of Electrical and Computer Engineering at the University of Auckland, New Zealand. His previous faculty positions were with Martin Luther University of Halle-Wittenberg in Germany (Assistant Professor, 1999- 2004), and with TSURE (Senior Lecturer, Professor, 1991-2002). He has been IEEE Senior Member since 2004.

Research interests of Dr. Vyatkin are in the area of industrial informatics, including software engineering for industrial automation systems, distributed software architectures, methods of formal validation of industrial automation systems and theoretical algorithms for improving their performance. The specific expertise area of Dr. Vyatkin is in distributed automation and the IEC 61499 standard.

Hans-Michael Hanisch is graduated with a Diploma Degree in Chemical Engineering at Polytechnical Institute of Merseburg, East Germany, in 1982. In 1987 he earned the degree Dr.-Ing. (PhD) with "summa cum laude" from the same institute. From 1987-1991 he was a Senior Research Assistent (PostDoc position) at the same institute before he changed to the Department of Chemical Engineering, University of Dortmund, where he was funded by a Habilitation Grant of the Deutsche Forschungsgemeinschaft (German Research Foundation). From 1994 - 1999 he was a Professor at the Institute of Automatic Control, Dept. of Electrical Engineering, University of Magdeburg as the Head of the Discrete Control Systems Laboratory. He received the Habilitation (qualification as University Teacher) at the Department of Chemical Engineering, University of Dortmund in 1995.

From 1998 – 1999 he was the Head of the Institute of Automatic Control at University of Magdeburg. He stayed as a visiting researcher at the Department of Industrial Engineering, Rutgers University, USA, in 1996, 1998 and 2003. Since January 2000 he has been a Full Professor at the Department of Engineering Science and later at the Institute of Computer Science at Martin Luther University of Halle-Wittenberg and the Head of the Automation Technology Laboratory. Since 2001 he has been a Senior Member of IEEE.

Research interests of Prof. Hanisch are: discrete event modelling techniques, controller synthesis and implementation, verification, Batch processes, Flexible manufacturing systems, Process optimization, fault detection and error recovering, distributed systems.

Cheng Pang is currently a Ph.D. candidate and senior teaching assistant in the Department of Electrical and Computer Engineering at the University of Auckland, New Zealand. He holds BE (2005) and ME (2007) from the same university. His current research interests are in the field of industrial informatics, especially the design and development methodologies and formal verification of IEC 61499 function block applications, and the development of software tools for engineering function blocks.

Chia-han Yang is currently a PhD student in the Department of Electrical and Computer Engineering at the University of Auckland, New Zealand. He holds a BE (2006) degree from the same university. His current research relates to simulation and validation for distributed control systems.