Functional and temporal formal modelling of embedded controllers for intelligent mechatronic systems

Christoph Sünder*

Automation and Control Institute,
Vienna University of Technology,
Gusshausstrasse 27-29/376, 1040 Vienna, Austria
E-mail: suender@acin.tuwien.ac.at
*Corresponding author

Valeriy Vyatkin

Department of Electrical and Computer Engineering,
The University of Auckland,
Private bag 92019, Auckland, New Zealand
E-mail: v.vyatkin@auckland.ac.nz

Abstract: Formal verification of intelligent mechatronic systems needs to incorporate a detailed description of the system architecture in order to provide sufficient results. Especially

- the model of function blocks with precise semantics
- real-time behaviour has to be included.

This paper focuses on the IEC 61499 control applications for automation objects, the building blocks for intelligent mechatronic systems. A comprehensive approach for their formal description of control behaviour incorporating also the operating system policy and physical time as parameter for real-time behaviour is given on the basis of Net Condition/Event Systems for a typical control device configuration.

Keywords: intelligent mechatronic components; temporal specification; functional specification; formal description; net condition/event systems.

Reference to this paper should be made as follows: Sünder, C. and Vyatkin, V. (2009) 'Functional and temporal formal modelling of embedded controllers for intelligent mechatronic systems', *Int. J. Mechatronics and Manufacturing Systems*, Vol. 2, Nos. 1/2, pp.215–235.

Biographical notes: Christoph Sünder earned a Master's of Science in Electrical Engineering from Vienna University of Technology, Austria, in 2004 and his Dr. Sci. Degree in Electrical Engineering from the same University in 2008. Currently he is with Thales Rail Signalling Solutions GesmbH (Austria) as a consultant from IVM Engineering, responsible for safety management of main line interlocking systems. This paper is related to his engagement as research assistant at the Automation and Control Institute (ACIN), Vienna University of Technology. His research interests encompass distributed automation systems, zero-downtime evolution of production systems, verification and motion control.

Valeriy Vyatkin is a 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 and with Taganrog State University of Radio Engineering in Russia. His research interests 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. His specific expertise area is in distributed automation and the IEC 61499 standard.

1 Introduction

In recent years the situation in mechatronics industry has been changing dramatically. As the term mechatronics has been established as the combination of mechanical and electronic components, which determine the functioning of the system, for many years, new challenges and developments can be observed in the market as described in Bouyssounouse and Sifakis (2005, Section 28.2). Companies that have been established as simple machine builders, based on solid know-how of the mechanical engineering involved, move to embedded systems as the functionality of the machine is captured to a bigger extent in software code than in mechanical and electronics components.

The key findings presented in this survey can be summarised in the following statements:

- Components in mechatronic systems become programmable and their functionality is mainly expressed in software (so-called embedded software). Thus, the mechatronic systems are becoming more intelligent on account of embedded intelligence.
- The ratio of design costs for the software is steadily increasing and becomes the largest part of the overall costs (multi-technology departments are in charge of the overall system requirements).
- The control logic of mechatronic systems is getting more and more decentralised. Intelligence is added even to sensors and basic actuators.
- Mechatronic systems become networked and interact with the environment in order to fulfil their tasks (e.g., delegate tasks to the environment).

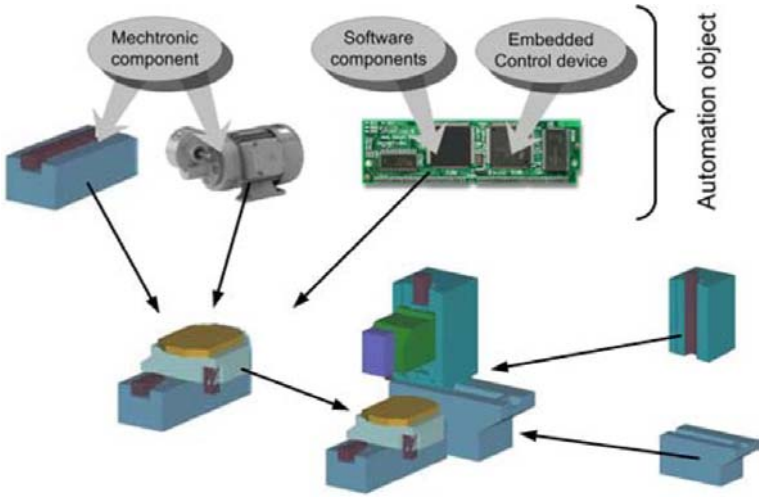
A mechatronic system is the result of the aggregation of components (as this is usual since many years), but the single components are able to compute logic by themselves; they do interact with other components within the mechatronic system and coordinate their actions. This scenario of intelligent mechatronic systems within the overall plant has been the basis for different approaches such as for instance reconfigurable manufacturing systems (Koren et al., 1999) or an open, object-oriented knowledge economy for intelligent industrial automation (Vyatkin et al., 2005). Within these approaches the common understanding of a component that incorporates computational power in order to act autonomously on the one hand and as basic element for the engineering with different views of its functionality on the other hand can be identified. As basic definition we will

use the term automation object, as it has been defined in Vyatkin (2003) to comprise three main elements:

- *Mechatronic component*: A physical functional device with sensors, actuators and electric circuits.
- *Embedded control device*: A computing device with interfaces to the sensors and actors as well as to the network.
- *Software components*: A set of data and control logic implementing various automation functions. These elements provide the autonomy and cooperation of the automation object.

The role of the automation object within the engineering of a mechatronic system is manifold. It starts from the representation of the mechanical parts of the component (e.g., CAD data) to the electrical or pneumatic circuit descriptions and the software functionality in terms of software code, visualisation, or interaction diagrams with the environment. Based on these different views the overall engineering process of a mechatronic system, which is as already stated above multi-technology point of view, can be established as composition of automation objects. To give a demonstrative picture of such an engineering process we refer to Strasser et al. (2005) who have investigated on engineering tools for this kind of design process. Figure 1 gives the schematic of the hardware representation of a production facility, with the basic components having included an embedded device with software components.

Figure 1 Hardware representation of reconfigurable production facilities (see online version for colours)



Source: Based on Strasser et al. (2005)

This paper aims at the special aspect of formal description and verification/validation of automation components within a mechatronic system. Due to the fact that the functionality of automation components is defined to a large extent by software components, the interaction of the mechanic and electronic components with the software components has to be considered very carefully. A comprehensive formal description of

the behaviour of the software components has to be taken into consideration in order to provide the basis for significant statements on the correctness of the automation component's functionality. The combination of these formal models of the different automation objects may be used as the basis for the evaluation of intelligent mechatronic systems. This work focuses on single automation objects as the basic building blocks.

The remainder of this paper is as follows: Section 2 will discuss the architecture of an automation component as well as the resulting requirements for its formal description and verification. A linear axis will be used for illustration, which includes an overview on the model (Section 3) and a detailed description (Section 4). Both functional and temporal behaviour will be taken into consideration. Finally, some formal modelling results are presented in Section 5 and related work is summarised in Section 6. The paper concludes with further work directions on this topic in Section 7.

2 Problem description

The focus of this study is verification of the behaviour of a control system composed of an embedded device, further referred to as 'controller', and a process (e.g., manufacturing or energy production), further referred to as 'plant'.

As a general prerequisite, the control logic is represented in terms of the IEC 61499 standard (IEC 61499-1, 2005), and the internal architecture of the device is characterised by its compliance with this standard. For the analysis purposes it makes sense to represent architecture of the device in multi-layered form.

Main structural element of the IEC 61499 architecture is the function block, which is a component, encapsulating data, some behavioural logic and data processing algorithms. Function blocks form applications, which can be allocated to devices. The standard provides mechanisms for building abstract models of computing control devices, which can be further subdivided into independent containers called resources. The IEC 61499 function blocks correspond to the application layer of the device architecture, as seen in Figure 2 (right).

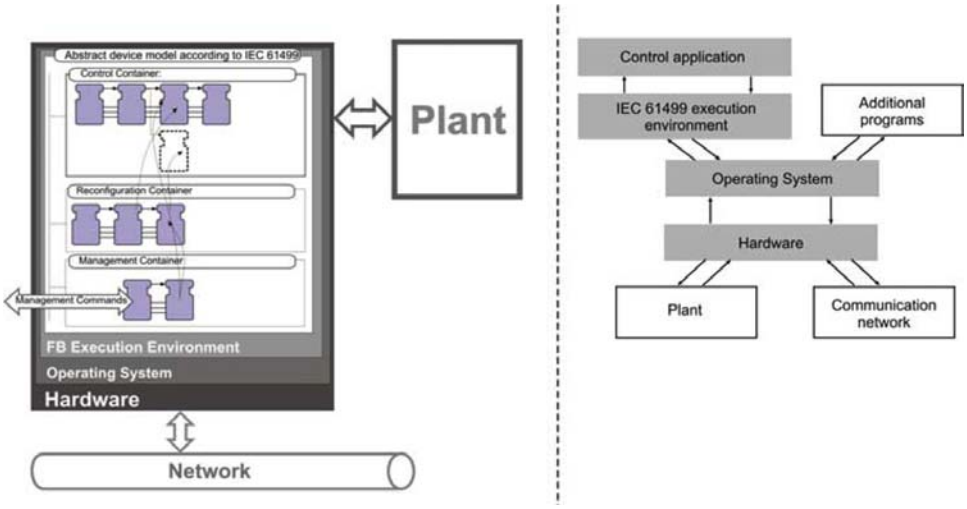
The device model is implemented by the FB execution environment (FBEE), which provides the mechanisms for executing function block networks. In particular, it implements an event passing mechanism from one function block to another, which is the basic execution control mechanism in IEC 61499.

The FBEE uses services of a Real-Time Operating System (RTOS), which, in general, can support execution of other applications concurrently with FBEE. Finally, the RTOS runs on a particular hardware platform and uses its services. The behaviour of the control device is influenced by the functions of all the layers described above. To verify its correctness, it is required to model the entire device, especially when fine real-time properties and deterministic behaviour of the whole device are concerned. Then powerful automatic validation techniques need to be applied in order to prove the logical and temporal correctness of this process.

The analytic model analysis techniques have limited applicability due to the presence of non-deterministic disturbances from the plant or from communication networks, leading to explosion in the number of different behaviour traces. Model based simulation can be helpful to ensure the correct behaviour within a single trace and to estimate its timing. However, all system traces, corresponding to different combinations of input disturbances, cannot be exhaustively checked by simulation. The formal verification by

model-checking (Clarke et al., 1999) is a technique that allows proving the correctness of a system's behaviour in presence of non-deterministic inputs. The exhaustive proof is performed by software tools, called model-checkers. In case if an incorrect state is found, the model-checker can supply description of the trace leading to such a failure.

Figure 2 Simplified structure of the IEC 61499-compliant device (left), and the layered model architecture of the device (right) (see online version for colours)



Due to the high model complexity and the complex device structure, the modelling approach needs to support modularity in order to derive a model of the entire device as a composition of constant and variable model parts. For a particular embedded device the details of the hardware, RTOS, and FBEE do not change from application to application, so they can be modelled once and later combined with the model of the control application, applying the model of reconfiguration on top. Moreover, the models need to be designed with the following requirements in mind:

Modelling of function blocks needs to take into account their precise semantics: The IEC 61499 standard defines a generic model of FBs that are executed according to their event connections. But the standard's definitions of event propagation and of FB execution are quite ambiguous as reported at Sünder et al. (2006), Čengić et al. (2006) and Dubinin and Vyatkin (2006). In addition, many important implementation details are not specified, leaving the decision up to the implementer. So, the standard's text is not sufficient to define unambiguous FB execution semantics, and the details of the implementation choice of the FBEE vendor needs to be modelled too.

Real-time behaviour: The correctness of a control application implemented by the device described in Figure 2 means the fulfillment of functional as well as of temporal requirements (Kopetz, 1997). Correspondingly, the correctness of an application can be proved only by taking into account its real time characteristics – e.g., 'the response time of the controller that is being re-configured should not exceed a maximum limit'. A model must capture real-time characteristics of the system if such characteristics of basic operations are known and included in it.

3 Model overview

The layered architecture of the system model (Figure 2 (right)) is structurally similar to the system (shown in the left side of the same Figure) and is explained in the following. For the purposes of this study a simple control system will be considered, which controls electric linear drive lifting some payload, see Figure 3.

Control application: The closed-loop controller is implemented as a function block application, following the traditional cascade control pattern with two control loops for speed and position. Correspondingly, there are two controller function blocks *Speed_CTL* and *Pos_CTL*, both being instances of the *P_CTL* function block type, implementing proportional control. Readings of the position sensors are delivered to the controllers by the 'Sensors' FB, and the value of speed is sent to the actuator via the 'Actuator' FB. The cascaded control loops are driven by two different clocks: the inner loop by the 5 ms clock (*CLOCK5 ms* FB), and the outer loop by the 50 ms clock (*CLOCK50 ms* FB). Both clocks are implemented using instances of the *RT_E_CYCLE* FB type, which identifies the event chain triggered by the *EO* event as a real-time constrained execution path.

Plant: Although we are mainly focusing on the processes within the control device, explicit modelling of its environment (i.e., a plant) is beneficial, as indicated by Hanisch (2004), Machado et al. (2003) and proven in Hanisch et al. (2006). Therefore, in this study, the system will be, in general, modelled as a closed-loop combination of plant and controller. Even when the plant is not explicitly presented, it should be noted that the modelling of the controller will allow its integration with the model of the plant. Precise modelling of plant dynamics may be not always possible by a particular modelling formalism, and it is important to ensure that even an abstracted discrete model of plant emits most of important events.

Communication network: As the scope of this paper is limited to a single control device, the influences of a communication network are abstracted to two different types:

- triggering of a function block execution by a message arrived from the network
- disturbances to the control application caused by processing of a message for any other program being executed concurrently with FBEE, within the same control device.

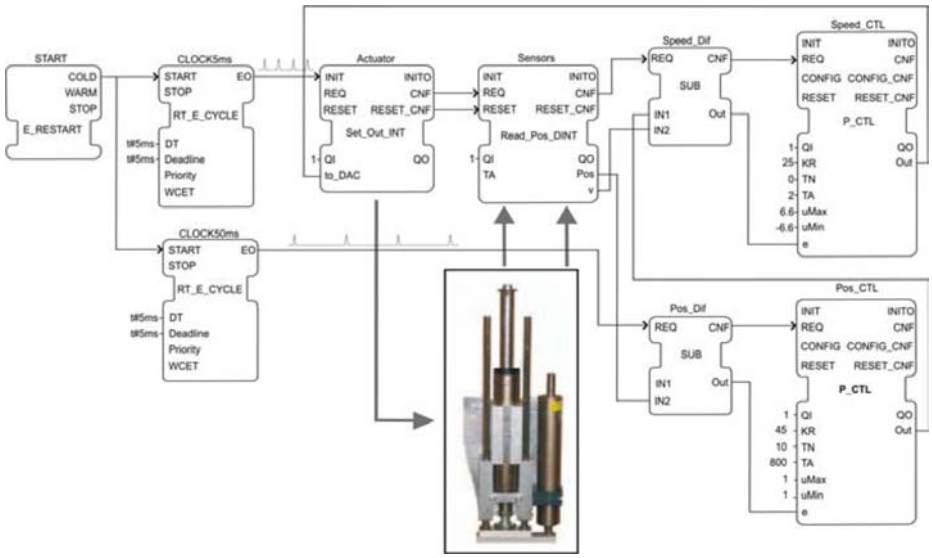
Hardware: This layer of the model represents only essential properties of the control device hardware, such as CPU speed, and parallel processing in multi-core processors. In the case study, the evaluation board phyCORE AT91M55800A (Phytec Messtechnik GmbH, 2003) was used. The board's CPU is ATMEL AT91M55800A microcontroller, which utilises the ARM7TDMI ARM thumb processor core. It is assumed that the variations in the execution time of code segments are negligible for the purpose of modelling.

Operating system: It is assumed that the FBEE uses one or more threads (lightweight concurrent processes) of the RTOS. The threads are characterised by their priorities. Their scheduling and inter-process interaction can be modelled explicitly for a particular RTOS. The combination of the RTOS model with the hardware model gives the model of a particular runtime platform of a control device. This part of the model also provides the

room for taking into account the impact of other programs being executed on the same control device in parallel with the IEC 61499 control application.

The open source operating system Embedded Configurable Operating System (eCos) is used in this study (Massa, 2003). The model captures characteristic features of the multi-threading implementation, along with the inter-thread interaction mechanisms. For instance, eCos provides the possibility to use a priority based preemptive scheduler with 32 priority levels and two basic scheduling paradigms for one (Bitmap) or more threads (Multi-Level Queue, MLQ) per priority level. Interrupts can be integrated into the application by use of callback functions, which are activated as soon as the corresponding interrupt occurs.

Figure 3 Function block application controlling the linear drive (see online version for colours)



FB execution environment (FBEE): The standardised details of the IEC 61499 execution model, along with implementation-dependent details, can be explicitly modelled and encapsulated in the model of execution environment. The details of such modelling using the modular language of Net Condition/Event Systems (NCES) stem from Vyatkin and Hanisch (1999), where NCES modelling of FB execution was proposed. In this work, the 4DIAC runtime environment (FORTE) has been used, which is available as open source project (4DIAC, 2008). FORTE is capable of executing IEC 61499 FB networks with real-time constraints and provides extensive reconfiguration possibilities. Zoitl et al. (2007) give a detailed description of the implemented real-time concept which is called event chain concept. The central elements are the sources of events (event sources, ES), which usually are triggered externally (e.g., from the network or a timer). Each ES triggers some FBs according to the connections within the application. The event chain is assigned to be executed within one thread of the operating system: real-time threads for constrained event chains and background threads for unconstrained event chains. Within a thread, the event propagation is characterised by the so-called event dispatcher concept. Each triggered input event is put into a queue. The FBs are executed according to the order of the events within this queue.

Additional programs: The impact of additional programs, such as a web server, being executed in a parallel thread, can be taken into account as thread of the RTOS as discussed earlier.

4 Formal model of an automation object

4.1 Modelling language

An important requirement for the formal modelling language is support of modularity enabling structural design of large models, following the multi-layer architecture presented in Figure 2. Moreover, the modelling language has to support efficient model encapsulation and re-use, since formal models of particular control device parts can be designed by their particular vendors.

Popular formal languages for modelling distributed systems are place-transition nets which exist in many dialects (the most famous are Petri Nets which stem from Petri (1962)). In such formalisms places are loaded with tokens, and a distribution of tokens across the places determines state of the model. Tokens can move from place to place through transitions, connected to the places by means of flow arcs. However, original Petri Nets are rather a mathematical than engineering language and the need to improve their expressiveness has been long recognised, for example, by Gomes and Barros (2005), who reviewed the Composition and Refinement/Abstraction mechanisms provided in various dialects of Petri Nets. Wurmus (2002) introduces CNET, a modular approach for the formal description and design of distributed control systems, which is described by a class of coloured Petri Nets. The NCES formalism (Rausch and Hanisch, 1995) is another attempt to add explicit modularity to Petri Nets. In NCES, modularised Petri Nets are augmented by event and condition arcs, helping to synchronise transitions and deliver information from places to transitions without incurring the token flow. This kind of nets with the model typing improvements from Vyatkin and Hanisch (2005) supports modular design, encapsulation and model re-use via instantiation of model types. Modules can be put together to larger modules, enabling a hierarchical architecture of the total model.

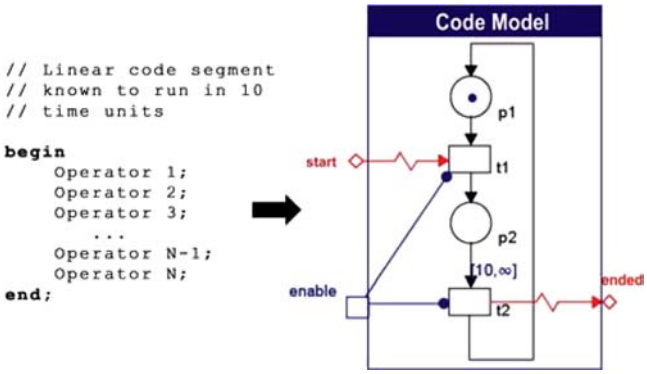
We will use the formalism of NCES for the description of our formal models. One of the reasons of that is availability of the tool chain (Vyatkin, 2008) for NCES modelling as well as for their verification. Another important aspect is the modelling of pre-emptive code execution (see discussion in the next section). We will discuss the modelling of the grey shaded elements of Figure 2 (right) according to the prerequisites FB execution semantics and real-time behaviour mentioned in Section 2 in the following subsections.

4.2 Hardware

The hardware of a control device is characterised by the speed of code execution, which is an essential property of the formal model. It is assumed, that the code structure is linear and each command has fixed execution time. Thus, the execution time for linear segments of code is constant, can be measured off-line and assigned to the model parameters, as proposed in Sünder et al. (2007), where static timing analysis of FB networks was achieved on account of using the runtime environment parameters.

Passing of time is modelled in NCES by adding time intervals of kind ($T1$, $T2$) to the token flow arcs from places to transitions, where $T1$-delay, and $T2$-deadline, are integer numbers, such that $0 \leq T1 < T2$. In our models $T2$ is always equal to ∞, so only delays are modelled. Figure 4 presents the idea of modelling of the delay caused by the execution of the linear program code. Suppose the code takes ten time units to execute. As soon as an event arrives from the event input *start*, the token jumps from $p1$ to $p2$. The internal clock of $p2$ starts counting. As soon as its value is equal to ten, the timed arc from $p2$ to $t2$ is enabled and an event at the event output *ended* is emitted. The above described scenario is valid, if the condition input *enable* is assigned the value *true* in all states until *ended* is emitted.

Figure 4 NCES model of physical time by timed flow arcs (see online version for colours)

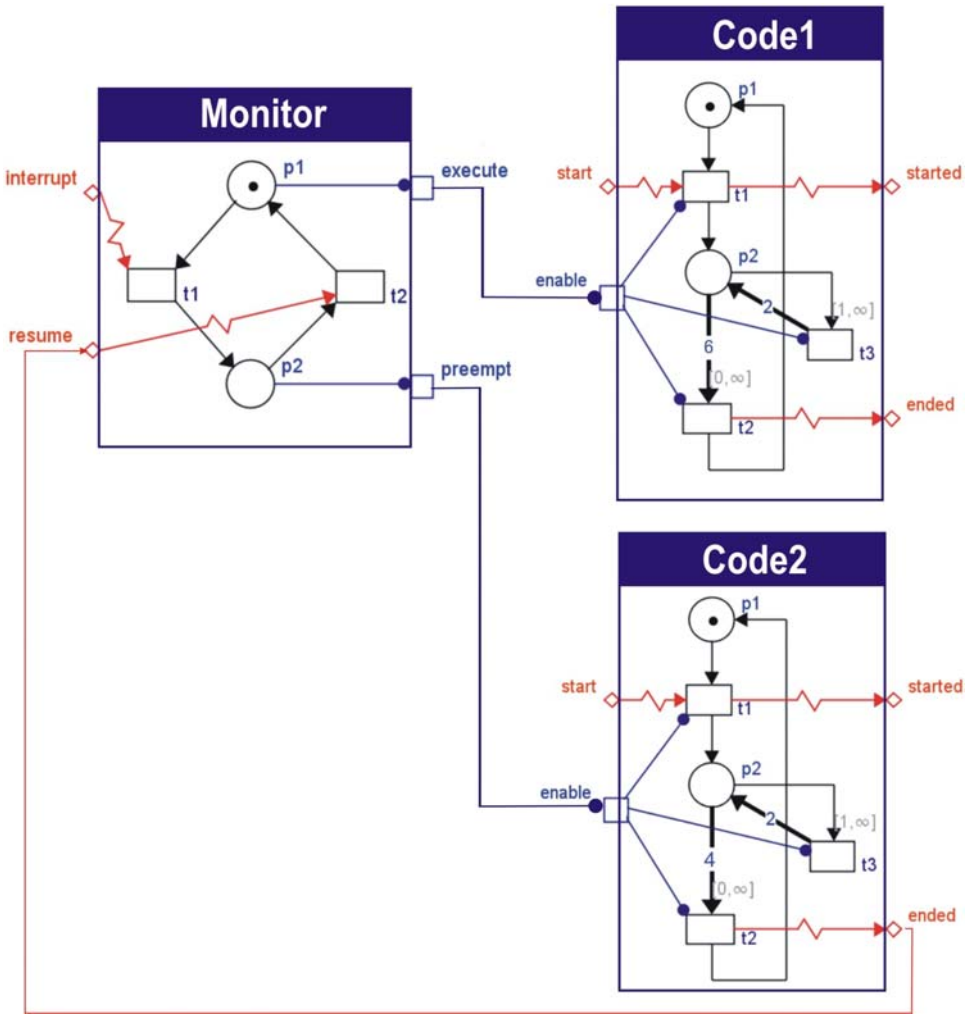


However, the linear execution of code may be disrupted due to several reasons, such as interrupts or higher priority threads. An interruption can be modelled by the change of the condition input *enable* from *true* to *false*. However, the internal clock of place $p2$ will continue counting; regardless of the value on the condition input *enable*. Therefore, the time delay representing the physical execution time would not be correct in case of an interruption of the code execution.

An enhanced model of linear code execution which correctly handles interruptions is depicted in Figure 5 in NCES modules *Code*1 and *Code*2. The time delay is represented by a (numeric) marking of place $p2$. As soon as the place $p2$ receives a token, the internal clock starts counting. By use of transition $t3$, which is 'looped back' to place $p2$ by a timed arc with delay one, each time unit one token is added to place $p2$. The arc weight (multiplicity) of the flow arc from place $p2$ to transition $t2$ (for example, in *Code*1 it is equal to six corresponds to the desired time delay. As soon as this number of tokens is achieved in place $p2$, transition $t2$ fires and $p2$ remains unmarked. If the value of the condition *enable* changes to *false* as a result of an interruption, the time counting will be interrupted as well. The whole model in Figure 5 illustrates modelling of pre-emption of the *Code*1 execution by *Code*2 execution. It is assumed that *Code*1 takes five time units to run, and *Code*2 takes three time units to run. The scenario is illustrated in Figure 6 by means of timing-state diagram, representing parameters of the model. First the *Code*1 starts activated by the event input *start*. The module *Monitor* has two places: $p1$ stands for the state allowing execution of *Code*1, and $p2$ for the state where it is pre-empted by *Code*2. Suppose the interrupt occurs when two fifth of the *Code*1 have been executed, i.e., after two time units. The marker moves from $p1$ to $p2$ and the increment of marking

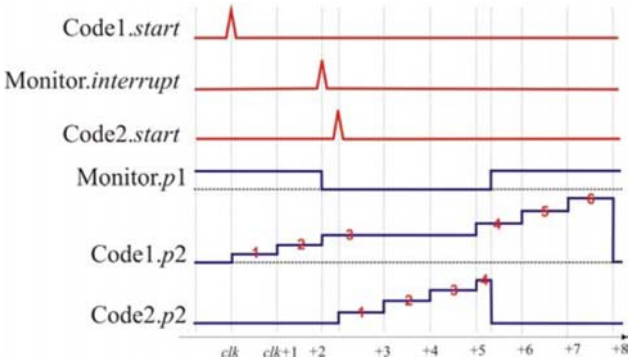
in *Code1.p*2 is blocked. Then an event input *Code2.start* receives an event and the similar counting takes place in *p*2 of *Code*2. When the marking four is reached after three time units, *Code*2 terminates, and *Code*1 resumes and takes three more time units.

Figure 5 Model of pre-emption: *Code*1 is interrupted by *Code*2 (see online version for colours)



Summarising this approach, the execution time of code structures is modelled by time delays, which are derived as a result of measuring actual execution time of particular program parts. Due to interrupts or higher priority threads, the execution may be disrupted at any time. However, using the proposed enhanced model of the execution modelled by delay, delays in pre-empted modules will be handled correctly.

Figure 6 Diagram showing discrete states of the model and values of discrete time (see online version for colours)



4.3 Operating system

The RTOS is the basis for the interaction of all programs of the control device. The main part of it is the scheduler, which takes care of the execution of the different threads within the system. Additionally, also the interrupt handling and the corresponding execution of callback functions has to be handled. For the formal models of the operating system, a clear description of its interfaces is necessary to provide simple composable modules for different configurations. There are two kinds of interfaces that need to be distinguished:

- an interface between a thread and the scheduler
- an interface between different priority levels within the scheduler.

Interface between thread and scheduler: A thread can be configured for one of the 32 different priorities of the eCos scheduler. The thread can either suspend itself, or claim resources for execution. The scheduler issues execution time to the thread. A very simple interface is sufficient for the interaction between scheduler and thread (we describe the interface from the scheduler's side):

- event input for recognition of the suspension of the thread
- event input for recognition of the thread's request to become active
- condition output for assigning execution control to the thread.

Interface between priority levels within the scheduler: A modular model of the scheduler can be achieved by splitting up the control of different priority levels. The functionality of one priority level is determined by the type of scheduler (bitmap or MLQ). Each priority can be interrupted by a higher priority. If no thread within the priority level wants to become active, control is passed to the next lower priority. The interface is characterised as follows:

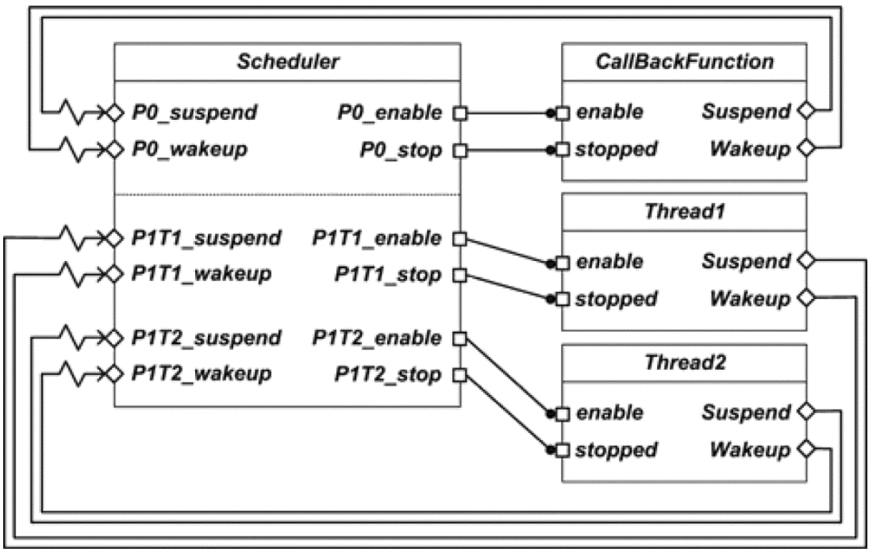
- activation of the scheduler priority, which means that the priority level assigns execution control
- hand over execution control to the next lower priority, if the priority level does not need it

- hand over execution control to a higher priority, if it has claimed for it
- input for the condition (suspended, want to be active) of the next lower priority
- input for the condition (suspended, want to be active) of all higher priorities
- output for the condition (suspend, want to be active) of the priority itself.

Callback functions: A callback function can be handled in the same way as a thread, since it disrupts the execution of all lower prior threads and callback functions (there exist also an order within the callback functions).

Figure 7 depicts the overall configuration of an RTOS consisting of two threads *Thread*1 and *Thread*2 (hosted on the same priority level) and one callback function *CallbackFunction* (e.g., for a timer). At this level the scheduler *Scheduler* and the different threads/callback functions are interrelated by the simple interface mentioned above. Additionally, based on the functionality programmed further interactions between these elements (e.g., *CallbackFunction* calls some method within *Thread*1, as this takes place typically for an ES trigger) can be modelled. The internal realisation of the scheduler can be established based on modules for each of the priority levels.

Figure 7 RTOS configuration with three tasks and two priority levels



Real-time behaviour: The RTOS influences the temporal behaviour of the control device to a very big extent. The execution interruption of threads and external events (callback functions) according to scheduling policy is one main aspect. But also the time consumption for the scheduling activities needs to be considered. Table 1 summarises the most important parameters measured for the given hardware (phyCORE AT91M55800A). These are the time necessary to switch between execution contexts (thread switching time) as well as the time required to switch to and from the idle state (thread suspension, thread resumption).

Table 1 Real-time behaviour of actions within eCos (hosted on phyCORE AT91M55800A)

Thread switching time	82.0 µs
Thread suspension	10.1 µs
Thread resumption	13.3 µs

4.4 Function block execution environment

The 4DIAC runtime environment utilises different threads and callback functions of the operating system. The real-time constraint execution of event chains is mapped onto different threads. The external events, which trigger the event chains and therefore execution of FB networks, are assigned to the various sources of interrupts, which are integrated by the callback functions. Therefore, the mapping of the runtime environment functions to the RTOS threads is almost static, except for the allocation of priorities based on the real-time constraints defined in a FB application. We will consider two items of FORTE in more detail: the event dispatcher and the mutual exclusion mechanism for the access to the event dispatcher.

Event dispatcher: The event dispatcher is the central element of each thread corresponding to event chains in FB networks. It represents an event queue, which stores each input event that is issued within the FB network as well as the occurrence of an external event. In case of external events, only an identification of the event source (ES-ID) is put into the event dispatcher. This ES-ID is handled in the same manner as input events and causes the execution of the ES FB. As soon as the event dispatcher is not empty, the thread claims for execution time. Figure 8 depicts the NCES model of a thread including an event dispatcher *EventDispatcher* and the access for external events. For each event, which may be put into the event dispatcher, the following interface is provided by *EventDispatcher* (Figure 8 only mentions the interface for one event):

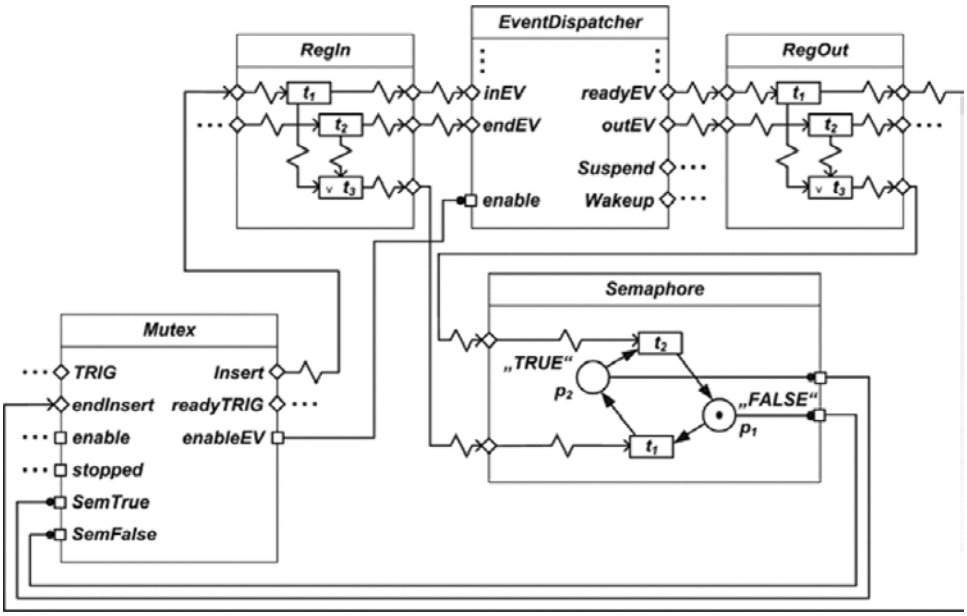
- The insertion of an event is represented by the input event *inEV*, which will be acknowledged by the output event *readyEV*.
- If a FB is called for execution, the corresponding output event *outEV* is issued by the *EventDispatcher*. After finishing the execution of the FB, control flow returns to the event dispatcher via the input event *endEV*.

The execution of the overall thread is controlled by the number of events within *EventDispatcher*. As soon as an event is inserted into the queue, the output event *Wakeup* is issued. The suspension of the thread will be issued by the event dispatcher via the output event *Suspend*, if there are no more events within the queue and the current execution of the FBs has been finished.

Mutual exclusion: In such cases, as when the timer callback function issues an ES-ID to the event dispatcher, a critical inter-process interaction occurs. It may happen, that the thread itself as well as the callback function use the event dispatcher simultaneously. The implemented solution is the establishment of a critical region and a mutual exclusion with priority inversion for the occurrence of a higher prioritised invocation (such as the timer interrupt). Figure 8 depicts the principal NCES model for this configuration. The critical region *EventDispatcher* surrounded by modules for registration of event flow (*RegIn* and *RegOut*), which control the module *Semaphore*. Based on this information the

invocation of *EventDispatcher* by a callback function can be blocked (*MUTEX*) and the current execution of the event dispatcher will be finished (only within the critical region) before continuing the execution of the higher priority callback function.

Figure 8 NCES model of the event dispatcher and the access via mutual exclusion policy



Real-time behaviour: In order to describe the time consumption of the execution of event chains next to the pure execution time of each FB instance additional parameters of the FBEE have to be taken into consideration. Based on the event dispatcher concept for event propagation, these are the time necessary for insertion and fetching of events to and from the event dispatcher. Table 2 depicts the measurement values for FORTE based on the eCos operating system and the phyCORE AT91M55800A microcontroller board.

Table 2 Real-time behaviour of actions within FORTE

Insertion of an event	7.0 µs
Fetching of an event	11.1 µs

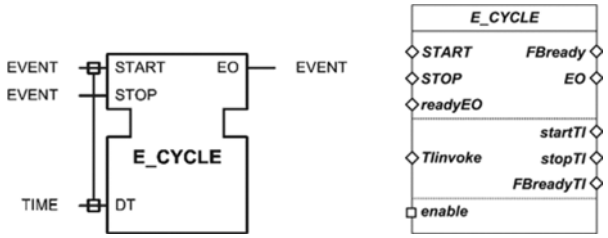
4.5 Control application

The control application consists of IEC 61499 FB networks. The formal modelling of FB networks can be subdivided into several issues:

- the event interface
- the execution control chart (ECC) of basic FBs
- data interface and algorithms, as well as
- composite FBs.

Event interface: The transformation of the event interface needs to be adapted to the implementation of the runtime environment. Figure 9 depicts the interface of the FB 'E_CYCLE' (a) and the corresponding NCES model (b). As the event dispatcher fetches input events of FBs for execution, direct mapping of IEC 61499 input events to NCES events is possible. The end of the execution of a FB is indicated by the event output *FBready*. By use of this event the execution control is handed back to the event dispatcher. In case of an output event, there exists also an appropriate NCES output event. This event is used to put all connected input events into the event dispatcher. A confirmation is given by a related input event (*readyEO* as confirmation of *EO* in Figure 9(b)).

Figure 9 The FB type E_CYCLE represented (a) in IEC 61499 and (b) as NCES model



(a) (b)

The second part of the NCES model provides the interface to the callback function used for the timer. Each FB utilising the timer functionality may start (*startTI*) or stop (*stopTI*) the related counter within the callback function. If the FB instance will be invoked by the timer, first the ID of the FB instance is put into the event dispatcher (see description above). Then the corresponding output from the event dispatcher (*outEV*) is connected to *TIinvoke*. The execution control is handed back to the event dispatcher via the event output *FBreadyTI*.

Execution control chart: The ECC is an event driven state machine, therefore generation of the corresponding NCES model is straightforward. It must be ensured, however, that the formal model precisely follows the execution flow as implemented in the runtime environment. For instance, after the invocation of the FB by an input event, all transitions with their origin at the active ECC state are evaluated one by one. As soon as one transition clears, the ECC state will be changed and the corresponding actions are executed, again one by one. Therefore, timing parameters can be extracted and used to parameterise the formal models to achieve the same real-time behaviour for the verification. These parameters are related to the runtime environment and do not need to be measured for each basic FB separately.

Data interface and algorithms: NCES is limited in representing data types other than Boolean. In case if an integer variable takes a limited number of known values, it can be modelled by the corresponding number of NCES places. These places can be connected by condition arcs to the parts of the model corresponding to the operations using the data values. When the NCES model is generated from an FB, all operations with the data can be identified and if they consist only in comparing a variable with a constant, or assigning it a constant value, such cases can be handled quite easily.

In the general case – as done in Pang and Vyatkin (2007) – a NCES place can hold the number of tokens equal to the value of the modelled variable, but the use of this approach is quite bulky.

Composite FBs: A composite FB consists of a FB network by itself. The borders of the composite FB are transparent for the events, only the latching of data based on the event data associations has to be modelled. The event interface elements of the component FBs have to be connected through its borders to the event dispatcher of the resource or ES.

Real-time behaviour: The time consumption of one FB instance may be measured separately for each instance. This is necessary for instance for Service Interface FBs which do have any kind of interaction included. For basic FBs (and composite FBs) the definition of the standard can be used to describe the execution flow within the FB and provide appropriate timing parameters. These parameters are independent of the FB instance. Table 3 summarises different parameters for basic FBs, which need to be used according to the concrete type of FB.

Table 3 Real-time behaviour for execution of basic FBs in FORTE

Time for invocation of FB	7.9 µs
Overhead for WITH connected data inputs	57.1 µs
Latching of data input (INT)	7.0 µs
Evaluation of ECC transition (event)	7.0 µs
Evaluation of ECC transition (condition)	7.6 µs
Time for algorithm execution	Specific for FB
Sending output event	Included acc. to Table 2
Latching of data output (INT)	71.2 µs

5 Prototypic formal model of an automation object

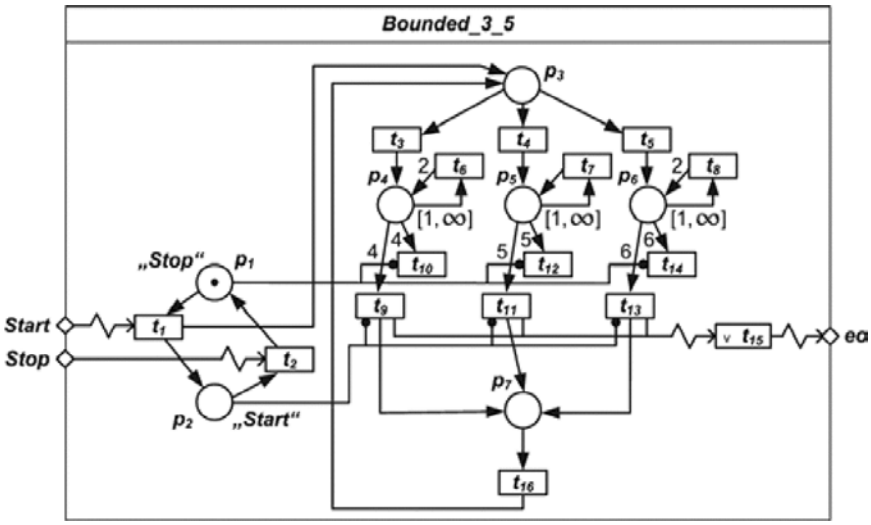
As an example for an automation object we will investigate the linear drive already described above. According to the structure of an IEC 61499-compliant device, each element within the layered model architecture has to be provided as formal model. The grey shaded elements have been described already in Section 4. In addition, also the environment has to be taken into consideration. Herein we will distinguish between the model of the plant and other elements which describe some interrelation with the system environment (communication network, additional programs).

Plant model: The formal model of the plant has to describe the temporal behaviour of the movement of the axis. Herein appropriate descriptions, which usually already exist for the design of the closed-loop control circuit, need to be modelled by means of NCES. The linear axis used for this automation object depicted in Figure 3 can be described by the transfer function which uses the velocity reference value as input and the position applied to the linear axis as output. As a simplification the velocity control application is added to the model of the plant in this case. The application necessary for the velocity closed-loop control is hosted within a separate thread (according to the real-time constrained operation), and as it provides a linear execution of events its overall

execution behaviour can be modelled as one linear execution of code (of course including also the possibility of interruption according to the scheduling policy of the operating system). The model represents an abstraction of the total system model.

Interrelation with the system environment: Another open issue within the layered model architecture regards to the communication network as well as other programs within the control device. If we are not interested in the details of these elements, a rough estimation of their behaviour can be used for the verification of the automation object. As basis event occurrence patterns can be included in order to describe a typical (or perhaps also the worst case) situation for communication requests. A typical pattern is for instance the bounded model, which limits the occurrence of consecutive events by a lower bound, the minimal inter-arrival time T_{min}, and an upper bound, the maximum inter-arrival time T_{max}. Figure 10 depicts the NCES model for the occurrence of an event according to the parameters $T_{min} = 3$ and $T_{max} = 5$ (time units in the formal model). Next to the occurrence of events at the event output *eo* the module includes two event inputs *start* and *stop* in order to control the occurrence of events within this model. The model further includes time delays which would be capable to handle interruptions as depicted in Figure 5. Therefore, this model may be used either for the representation of external events from i.e., the communication network, but may be also used to describe different execution lengths of e.g., additional programs within the control device.

Figure 10 NCES model for bounded event occurrence with minimal (three time units) and maximal (five time units) inter-arrival time



Analysis of temporal behaviour: Next to the verification of correct behaviour of the control device the above described methodology may be used also during the design for enhanced analysis. For instance the bounds of execution time for a specific part of the control application (according to our example for instance the position control) can be analysed based on different environmental conditions. The need for such enhanced methodologies is stated for instance in Bouyssounouse and Sifakis (2005, Section 7.4) as follows:

> "Testing is often performed to measure real-time execution time and response time e.g., to check resource utilisation or obtain an estimation for the worst case execution time. However, using this approach is very problematic because it is difficult to obtain safe and accurate bounds."

By use of the above described incorporation of real-time measurements into the system model together with a comprehensive model of the internal behaviour of all elements of the control device it is possible to achieve more detailed and meaningful checks and analysis of temporal behaviour.

6 Related work

There do exist different approaches related to the above mentioned methodology of incorporation of functional and temporal behaviour into intelligent mechatronic systems, which can be split up into two parts:

- verification of IEC 61499 under enhanced environment conditions
- verification including the real-time behaviour of a system.

Verification of IEC 61499: Vyatkin (2006) describes especially the modelling of execution semantics of IEC 61499 FBs by use of NCES. These enhancements of Vyatkin and Hanisch (1999) concentrate on the correct order of actions within a FB as well as the propagation of events over the network by use of a scheduler, which provides sequential operation of events. There is no runtime environment available for these models; further directions are a software implementation as well as a hybrid hardware/software implementation using Field Programmable Gate Arrays. Khalgui et al. (2004) propose a state machine model compliant to the standard IEC 61499. To avoid unpredictable behaviour in the case of simultaneous occurrences of events, they propose to design an offline scheduling of FB execution. They verify the scheduling correctness using the state machine model. By use of this scheduler, a hard-coded execution model of a runtime environment can be implemented. Čengić et al. (2006) describe their formal model of the runtime environment FUBER, which they have developed based on interacting finite automata in Supremica. In this case the formal description includes many aspects of the runtime behaviour. For instance, the event execution model specifies that each FB instance must wait for another instance to finish its event handling before it can begin its own event handling. Incoming events of a FB instance are stored in a queue; all FB instances waiting for execution are also handled in another queue. By use of such a detailed formal description of the runtime behaviour, they are able to proof in many details the behaviour of the FUBER implementation. Physical time is not mentioned in their approach. As the implementation of FUBER is based on Java, the virtual machine as well as the underlying operating system needs to be included to the models for the consideration of physical time.

Verification including real-time behaviour: The topic of formal description of real-time applications and especially RTOS and their applications has been discussed in several applications. Corbett (1996) aims at the formal description of Ada programs, with special attention to its concurrency and real-time constructs. For scheduling of tasks the task dispatching policy has been modelled. The formal description is based on constant slope hybrid automata, whereas a transition represents a code region and execution time is

modelled with an appropriate delay before its (instantaneous) transition. Cofer and Rangarajan (2003) describe the verification of the DEOS real-time operating system by use of SPIN model checker. In detail, the rate monotonic scheduler policy is implemented and analysed in contrast to an event-triggered system environment. Waszniowski and Hanzalek (2003) depict their model of a RTOS with timed automata. They claim that timed automata theory is not suitable to model pre-emption; therefore they focus on cooperative scheduling. The formal description includes for instance inter process communication via semaphores or context switching time. Krakora et al. (2004) utilise the combination of the RTOS with communication via CAN for verification of a distributed control application.

7 Conclusions

Mechatronic systems emerge towards intelligent systems, which consist of components that are capable to provide enhanced functionalities. These so-called automation components are foreseen to increase engineering efficiency, but on the other hand do need more comprehensive models in order to prove the correctness of single components as well as overall intelligent mechatronic systems. This work provides a detailed description of a methodology to describe the functional as well as the temporal behaviour of automation components, based on a comprehensive modelling approach for all parts within the overall system architecture including their execution time.

The proposed methodology does not only enhance the expressiveness of model checking for proving the properties of a given system, but may be used also for better analysis during system design. Resource utilisation or the bounds of execution time of certain parts of the application can be analysed based on the overall model of the system architecture.

Acknowledgement

The authors want to thank Alois Zoitl for his support on implementation details of the automation object under consideration. We also want to express our gratitude to Tarik Ferhatbegovic, Ivo Gosetti, Christian Hanni, Richard Mandl, and Wei Zhang, who provided detailed considerations of specific items during their project works and master theses at the Automation and Control Institute, Vienna University of Technology.

References

Bouyssounouse, B. and Sifakis, J. (Eds.) (2005) *Embedded Systems Design: The ARTIST Roadmap for Research and Development*, *Lecture Notes in Computer Science* (*LNCS 3436*), Springer-Verlag, Berlin/Heidelberg, ISBN 978-3-540-25107-1.

Čengić, G., Ljungkrantz, O. and Akesson, K. (2006) 'Formal modeling of function block applications running in IEC 61499 execution runtime', *Proceedings of the 11th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'06)*, Prague, Czech Republic, September, pp.1269–1276.

Clarke, E.M., Grumberg, O. and Peled, D.A. (1999) *Model Checking*, The MIT Press, Cambridge MA, USA, ISBN 0-262-03270-8.

Cofer, D.D. and Rangarajan, M. (2003) 'Event-triggered environments for verification of real-time systems', *Proceedings of 35th Winter Simulation Conference (WSC'03)*, December, New Orleans LA, USA, pp.915–922.

Corbett, J.C. (1996) 'Timing analysis of Ada tasking programs', *IEEE Transactions on Software Engineering*, Vol. 22, No. 7, July, pp.461–483.

4DIAC (2008) *Framework for Distributed Industrial Automation and Control*, Profactor Produktionsforschungs GmbH, Online available: http://www.fordiac.org, March.

Dubinin, V. and Vyatkin, V. (2006) 'Towards a formal semantic of IEC61499 function blocks', *Proceedings of the Fourth IEEE International Conference on Industrial Informatics (INDIN'06)*, Singapore, July, pp.6–11.

Gomes, L. and Barros, J.P. (2005) 'Structuring and composability issues in Petri nets modelling', *IEEE Trans. on Industrial Informatics*, Vol. 1, No. 2, pp.112–123.

Hanisch, H.M. (2004) 'Closed-loop modeling and related problems of embedded control systems in engineering, lecture notes on computer science (LNCS 3052), *Proceedings of 11th Int. Workshop on Abstract State Machines (ASM'04)*, Springer-Verlag, Berlin Heidelberg, Lutherstadt Wittenberg, Germany, May, pp.6–19.

Hanisch, H.M., Lobov, A., Martinez Lastra, J.L., Tuokko, R. and Vyatkin, V. (2006) 'Formal validation of intelligent-automated production systems: towards industrial applications', *International Journal on Manufacturing Technology and Management (IJMTM)*, Inderscience Enterprise Ltd., Vol. 8, Nos. 1–3, pp.75–106, ISSN 1368-2148.

IEC 61499-1 (2005) *Function Blocks – Part 1: Architecture*, 1st ed., International Standard, International Electrotechnical Commission, Geneva.

Khalgui, M., Rebeuf, X. and Simonot-Lion, F. (2004) 'A behaviour model for IEC 61499 function blocks', *Proceedings of Third Workshop on Modelling of Objects, Components, and Agents*, Aarhus, Denmark, October, pp.71–88.

Kopetz, H. (1997) *Real-Time Systems: Design Principles of Distributed Embedded Applications*, Kluwer Academic Publisher, Boston, ISBN 0-7923-9894-7.

Koren, Y., Heisel, U., Jovane, F., Moriwaki, T., Pritschow, G., Ulsoy, G. and van Brussel, H. (1999) 'Reconfigurable manufacturing systems', *Annals of the CIRP*, Vol. 48, No. 2, pp.527–540.

Krakora, J., Waszniowski, L., Pisa, P. and Hanzalek, Z. (2004) 'Timed automata approach to real time distributed system verification', *Proceedings of Fifth IEEE International Workshop on Factory Communication Systems (WFCS'04)*, Vienna, Austria, September, pp.407–410.

Machado, J.M., Denis, B., Lesage, J.J., Faure, J.M. and Ferreira da Silva, J.C.L. (2003) 'Increasing the efficiency of PLC program verification using a plant model', *Proceedings of Int. Conf. on Industrial Engineering and Production Management (IEPM'2003)*, Porto, Portugal, 10pp.

Massa, A.J. (2003) *Embedded Software Development with eCos™*, Prentice-Hall Professional Technical Reference, Pearson Education, ISBN 0-13-035473-2, Upper Saddle River, New Jersey, USA.

Pang, C. and Vyatkin, V. (2007) 'Towards formal verification of IEC61499: modelling of data and algorithms in NCES', *Proceedings of the Fifth IEEE International Conference on Industrial Informatics* (*INDIN'07*), Vienna, Austria, July, pp.879–884.

Petri, C.A. (1962) *Kommunikation mit Automaten*, PhD Thesis, University of Bonn (in German).

Phytec Messtechnik GmbH (2003) *phyCORE-AT91M55800A*, Hardware Manual L-618e_32003.

Rausch, M. and Hanisch, H.M. (1995) 'Net condition/event systems with multiple condition outputs', *Proceedings of INRA/IEEE Symposium on Emerging Technologies and Factory Automation*, Magdeburg, Germany, October, Vol. 1, pp.592–600.

Strasser, T., Fessl, K., Hämmerle, A. and Ankerl, M. (2005) 'Rapid reconfiguration of machine-tools for holonic manufacturing systems', *Proceedings of 16th IFAC World Congress*, Prague, Czech Republic, July, p.6.

Sünder, C., Rofner, H., Vyatkin, V. and Favre-Bulle, B. (2007) 'Formal description of an IEC 61499 runtime environment with real-time constraints', *Proceedings of the Fifth IEEE International Conference on Industrial Informatics (INDIN'07)*, Vienna, Austria, July, pp.1123–1129.

Sünder, C., Zoitl, A., Christensen, J.H., Vyatkin, V., Brennan, R.W., Valentini, A., Ferrarini, L., Strasser, T., Martinez-Lastra, J.L. and Auinger, F. (2006) 'Interoperability and useablity of IEC 61499', *Proceedings of the Fourth IEEE International Conference on Industrial Informatics (INDIN'06)*, Singapore, July, pp.31–37.

Vyatkin, V. (2003) 'Intelligent mechatronic components: control system engineering using an open distributed architecture', *Proceedings of Ninth IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'03)*, Lisbon, Portugal, September, pp.277–284.

Vyatkin, V. (2006) 'Execution semantics of function blocks based on the model of net condition/event systems', *Proceedings of the Fourth IEEE International Conference on Industrial Informatics (INDIN'06)*, Singapour, July, pp.874–879.

Vyatkin, V. (2008) *Block Design, Visual Framework for Verification of Function Blocks*, Online available: http://www.fb61499.com/valid.html, March.

Vyatkin, V. and Hanisch, H.M. (1999) 'A modeling approach for verification of IEC1499 function blocks using net condition/event systems', *Proceedings of the Seventh IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'99)*, Barcelona, Spain, September, pp.261–270.

Vyatkin, V. and Hanisch, H-M. (2005) 'Re-use in formal modelling and verification of distributed control systems', *Proceedings of IEEE Int. Conference on Emerging Technologies and Factory Automation (ETFA'05)*, Catania, Italy, September.

Vyatkin, V., Christensen, J.H. and Martinez Lastra, J.L. (2005) 'OOONEIDA: an open, object-oriented knowledge economy for intelligent industrial automation', *IEEE Transactions on Industrial Informatics*, Vol. 1, No. 1, pp.4–17.

Waszniowski, L. and Hanzalek, Z. (2003) 'Analysis of real time operating system based applications', *Proceedings of First International Workshop on Formal Modeling and Analysis of Timed Systems (FORMATS'03)*, Marseille, France, September, pp.219–233.

Wurmus, H. (2002) *CNET – Komponentenbasierter Entwurf Verteilter Steuerungssysteme Mit Petri-Netzen*, PhD Thesis, University of Hannover (in German).

Zoitl, A., Grabmair, G., Smodic, R. and Strasser, T. (2007) 'An execution environment for real-time constrained control software based on IEC 61499', *Proceedings of the Fifth IEEE International Conference on Industrial Informatics (INDIN'07)*, Vienna, Austria, July, pp.853–859.