Control and Cybernetics

vol. **39** (2010) No. 1

Formal modelling of IEC 61499 function blocks with integer-valued data types*

by

Christian Gerber, Ioanna Ivanova-Vasileva and Hans-Michael Hanisch

Chair of Automation Technology, Institute of Computer Science
Martin Luther University of Halle-Wittenberg
Kurth-Mothes-Str. 1, D-06120 Halle/Saale, Germany

Abstract: During the last couple of years a lot of component- or object-oriented approaches have been presented to ease the implementation and reconfiguration of distributed control systems. Using these frameworks, the structure of the control objects follows the configuration of the mechanical components, and each object is instantiated as often as the corresponding device occurs in a plant. Having an object library, a control engineer will glue objects together, which means, in the context of the IEC 61499, connecting function blocks by event and data connections. This rapid way of implementation and reconfiguration makes also high demands on the verification process, which has to be done after each change. This means that the model of the plant has to be updated and the model of the distributed controllers have to be automatically generated to connect both in a closed loop. Both formal models should be modular to manage even large systems. Furthermore, the controller must be able to interact with the plant not only via Boolean data but via integer-valued data as well. Thus, the existing verification approaches have to be extended and an execution and a data processing model defined. The execution model includes the function block interface and the Execution Control Chart as well as their interconnections. For the arithmetical operations of adding and subtracting as well as the comparison of Boolean and integer-valued data it is shown how the data processing inside the function blocks has to be transformed to the formal model. Consequently, rules are defined for the transformation of the execution and data processing of function blocks with Boolean and integer-valued data. Due to the proposed separation, the resulting data processing model is not limited to IEC 61499 Basic Function Blocks.

As formal model, Net Condition/Event modules (NCEM) and structures are used. Modelling of the plant and the analysis of the resulting closed-loop behaviour are presented using a small, but realistic manufacturing system.

*Submitted: January 2009; Accepted: May 2009

Keywords: modelling, verification, net condition/event systems, IEC 61499, closed-loop behaviour.

1. Introduction

Object-oriented control implementation is a very common approach since the 1990s, meant to fulfill the requirements of modularity, reusability, flexibility, extendibility and reconfigurability of manufacturing systems (Maffezzoni, Ferrarini and Carpanzano, 1999; Zhang et al., 1999). Consequently, there exists a clear separation between the implementation of the control function, encapsulated inside the objects, and the mechanism of data exchange and synchronisation between the objects. This enables the control engineer to extend, modify and reuse previously tested and verified control functions and to interconnect objects together instead of implementing them from scratch. To support this, the paradigm of strict correspondence of each object to real mechatronic component is often used and the structure of the software objects is derived from the hierarchical structure of the mechanical components (Vyatkin, Karras and Pfeiffer, 2005). Thus, it is suitable to rapidly implement and reconfigure an even complex manufacturing system using an object-oriented approach (Brennan et al., 2008). But, nevertheless, the plant safety and the reliability of the implemented control has to be guaranteed in any case by integrating the verification of the closed-loop system into the control engineering practices (Vyatkin and Hanisch, 2003). Consequently, this means that also the formal models used have to support an object-oriented or modular modelling, and have to be automatically updated if the control system or the plant model encounter changes.

The concept of object-oriented control implementation could be realised with any kind of high level programming language, which would lead to various kinds of data exchange and synchronisation mechanism, differing from manufacturing system to manufacturing system. Thus, for everyday practice the Technical Committee 65 of the International Electronical Commission launched several standards, such as IEC 61131 and IEC 61499 to define the execution order and the mechanism of data exchange as well as the graphical representation. Thus, the programming language used is up to the runtime environment applied at the control devices, and the implemented objects should have everywhere the same behaviour. Using these definitions and the former publications about verification of the execution of 61499 function blocks (Vyatkin and Hanisch, 2000; Stanica and Gueguen, 2003; Bonfe and Fantuzzi, 2003; Frey and Hussain, 2006; Dimitrova, Frey and Batchkova, 2007) an execution and data processing model is defined in this contribution. Thereby, the view is extended from the *Non-Preemptive Multi-Threaded Resource* execution model, which is used for the known FBRT runtime as a reference implementation of the IEC 61499, and is discussed in more detail in Suender et al. (2006), to a parallel one as well as to the processing of integer-valued data.

Due to the semantics of the used modular formal model Net Condition/Event Systems (abbr.: NCES), defined in Pinzon et al. (2004), the modelled parallel execution of function blocks incorporates also the *Non-Preemptive Multi-Threaded Resource* execution model. Further, modelling and verification based on NCE modules have been successfully applied for modelling of IEC 61499 function blocks beyond Vyatkin and Hanisch (1999), Missal, Hirsch and Hanisch (2007), Iva-nova-Vasileva, Gerber and Hanisch (2007). Although it is often neglected, correct behaviour can only be verified in a closed-loop model that incorporates also a model of the plant, because the safety constraints and the desired production processes are specified by the manufacturing plant itself (Preusse and Hanisch, 2008). The contribution, therefore, describes the modelling of such closed-loop behaviour as well as verification of the resulting dynamic graph and the trajectories visualisation, in order to check whether they fulfill the specification or are a counterexample.

The interaction between the controller and the plant, as well as data processing inside the controller, are not only based on Boolean data, but on integer-valued as well. According to this, Heiner and Menzel (1998) proposed a binary representation of integer-valued data and applied it by a Petri net model of a *Carry-Ripple-Adder* to the verification of instruction list programs. Thereby, each variable is modelled in a binary form, each bit being represented by two places forming a place invariant. A similar approach is used for transforming the function block data in- and outputs and internal variables of integer-valued data types to basic NCE modules. All explanations of the presented NCE structures of the data processing model are written in a way to be easily adopted to the previously defined execution model of IEC 61499 basic function blocks, but they can be used also with any other execution model as long as the formal model is a basic NCE module and a binary representation of integer-valued data is chosen. Thus, it is possible to use the algorithm transformation rules at the end of Section 5 of this contribution also for transforming controllers with a different execution runtime than the one of the IEC 61499.

The paper is structured as follows. Section 2 describes briefly two different testbeds, used in the work. Section 3 defines the execution model and extends transformation rules to automatically generate formal models from function blocks following IEC 61499. The transformation of data in- and outputs with integer-valued data types are introduced. Based on this, the data processing model is specified in Section 4 for arithmetic operations of adding and subtracting, and for the comparison of Boolean and integer-valued data. The modelling of the plant is described briefly in Section 5, while Section 6 concludes and provides some directions for future studies.

2. Subject of the study

2.1. EnAS-demonstrator

The EnAS demonstrator, shown in Fig. 1 and described in detail in Gerber (2008a), consists of two identical plant modules, which are rotated by 180 to each other. Thereby, the conveyors form a circuit and transport the pallets cyclically from station to station in clockwise direction.

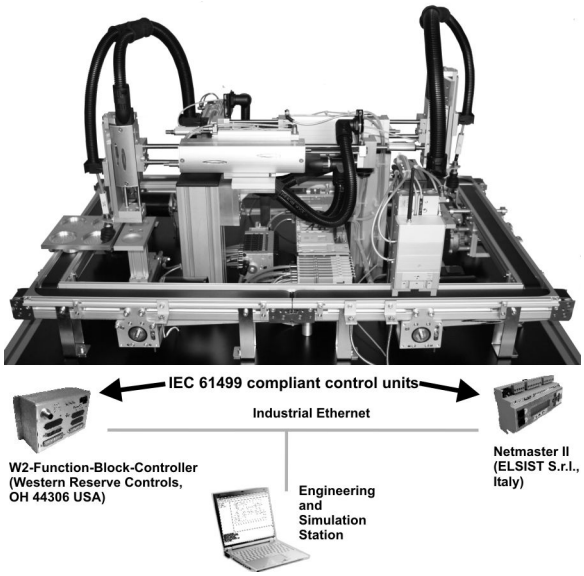


Figure 1. Application plant EnAS

The *jack station*, Fig. 2, can put workpieces from the sledge to the tin at the pallet or vice versa. It can also open a tin and lay the cap to the pallet or onto the tin again. To do this, the sucker of the jack station can drive to a high and low position as well as to three different horizontal positions. The middle position can only be reached if the jack station extends an additional pole. There is also a low-pressure sucker mounted, which may be moved to a lower position at the top of the workpiece or tin. By means of vacuum the cap or the workpiece is lifted safely. Through the combination of these possibilities several operations are realised.

The main function of the *gripper station*, Fig. 3, is to close the cap of all tins, which are previously reloaded by the jack station. It is also possible to lift a tin and hold it until the next pallet arrives. Through compressed air the gripper is moved up and down and thereby activates the sensors *gripper.up* and *gripper.down*.

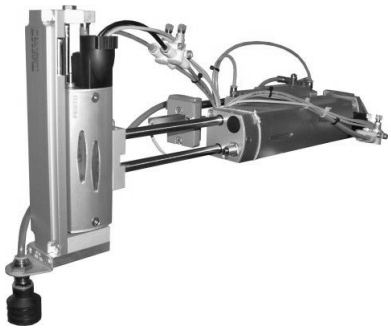


Figure 2. Jack Station

Figure 3. Gripper Station

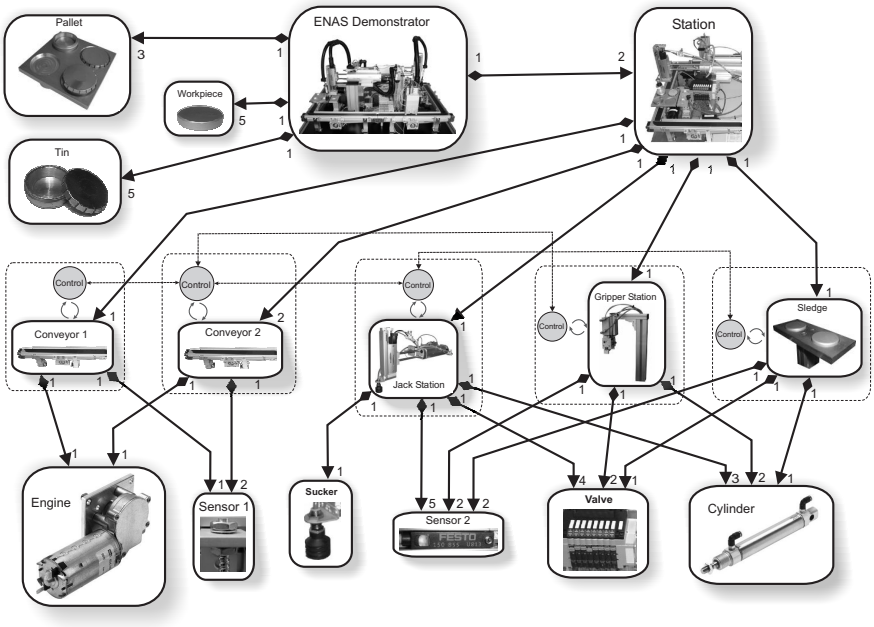


Figure 4. Structural description of the sample automation system

As described in the Introduction, the composition of the software objects is derived from the structure of the mechanical components (Fig. 4). This leads to the definition of automation objects and intelligent actors, which have some pre-programmed functionality. To represent the hierarchical structure of the mechanical components, as well as the automation objects, class diagrams of the informal modelling language UML are used. Thus, the EnAS Demonstrator is composed of two stations with one gripper station, one jack station, one

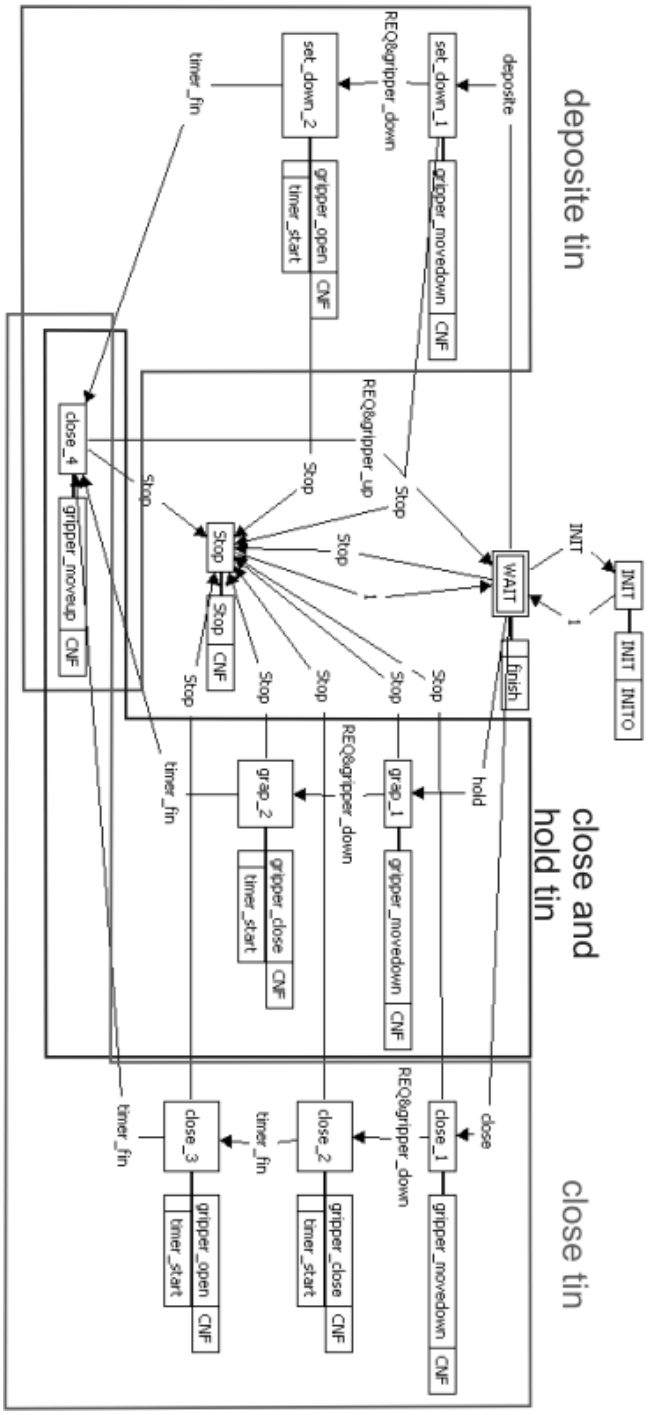


Figure 5. ECC of the gripper task controller

conveyor with one light barrier and two conveyors with two light barriers. The design and verification of distributed controllers, according to IEC 61499 standard, are some of the crucial topics for industry nowadays. In order to satisfy the initially mentioned requirements on manufacturing system, a hierarchical control approach with a multi-layered architecture was proposed by Missal, Hirsch and Hanisch (2007). This architecture includes distributed master controllers as a layered composition. This approach was developed by using the previously introduced definitions of automation objects.

Each basic automation object will get a fully reusable task controller, which controls all basic operations of this mechanical component. On the example of the gripper station, the ECC presented at Fig. 5 realises the three actions *close, hold* and *deposit* a tin (Gerber, Hirsch and Hanisch, 2009).

2.2. Servo Control System

Another common control example of industrial plants is a position control system, which could be found in disc drives, automotive products, robotics and process control. A simple position control system is shown in Fig. 6 and consists of a servomotor, an optical or magnetic position sensor and a controller. Thereby, the shaft position is detected by the position sensor and expressed in a 8 bit Gray code, depending on the desired solution precision. This signal will be decoded by the controller first and then subtracted from the provided reference position. Thus, the output of the 8 bit adder in subtract mode will be the position error, which is the input to an or logic to decide whether the motor should be turned on or off and in which direction. A more detailed description concerning the control and simulation of such a servo control system could be found in Readman (2005).

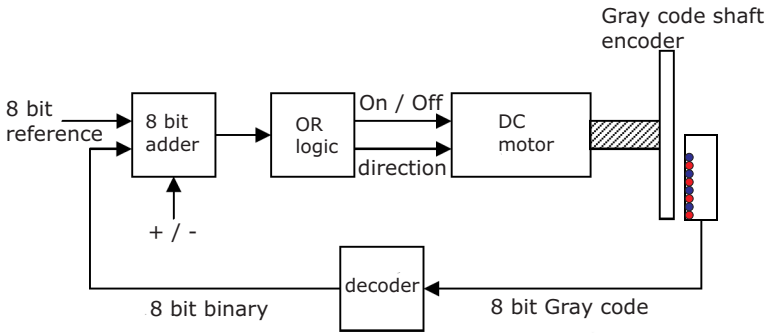


Figure 6. CE300 position control system

The challenges of the two presented control systems lie in the complexity of the execution and the data processing. In the distributed control system of the EnAS Demonstrator a hierarchical control structure with several mas-

ter and task controllers is used. This leads to a complex execution structure, but otherwise to a simple data processing with mainly Boolean values. Contrary to this, the servo control system has a linear execution structure from the decoder, to the adder and finally to the logic, but a complex data processing with integer valued data. Both control systems are implemented with function blocks following the IEC 61499 and should be verified using a closed-loop model. Therefore, a formal model divided into execution and data processing models of such function blocks will be described in the following.

3. Execution model

3.1. Interface with Boolean data inputs

The transformation of the ECC, presented in Fig. 5, and the associated algorithms can be done according to the rules 3 and 4 of Ivanova-Vasileva, Gerber and Hanisch (2007). Also the graphical transformation of the function block to a NCE module can be done according to rule 1. Only rule 2 for transformation of the interface has to be extended. As can be seen from Fig. 7, there is only one in- and one output event, connected with the data in- and outputs. Thus, the data values are only sampled and updated if the relevant event occurs.

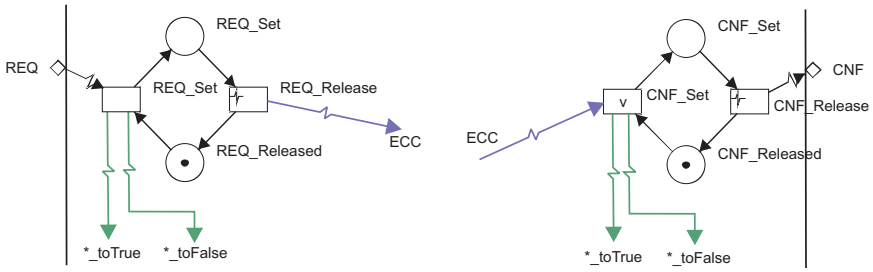
Figure 7. Task controller of the gripper station

Transformation rule 2 - Interface.

Transformation rule 2.1 - Event In- and Outputs:

The event in- and output of each function block are transformed to event in- and outputs of an NCE module and for basic function blocks this has to be extended by the NCE structure in Fig. 8.

It consists of places **_Set* and **_Released* and transitions **_Set* and **_Release*, connecting them. Furthermore, transition **_Release* gets the switching mode *instant*, which means it has to be fired before any spontaneous transition.



(a) Event Input (b) Event Output

Figure 8. NCE structure representing an event in- and output

For every data input associated to an event input or data output to an event output two event arcs have to be inserted with their source at **_Set* and their sinks at *+_toTrue* and *+_to False* where *+* stands for the name of the data in- or output and is represented by the attribute *Var* of the *With* tag, which is a child element of the *Event* tag.

If it is an event input, an event arc has to connect the input and the transition **_Set* and if it is an event output the event arc has to connect the transition **_Release* with the output.

Transformation rule 2.2 - Data In- and Outputs:

All data in- and outputs of each function block are translated to condition in- and outputs of an NCE module. For basic function blocks, the NCE structures shown in Fig. 9 and 10 have to extend the condition in- and outputs to model the sampling and updating of data.

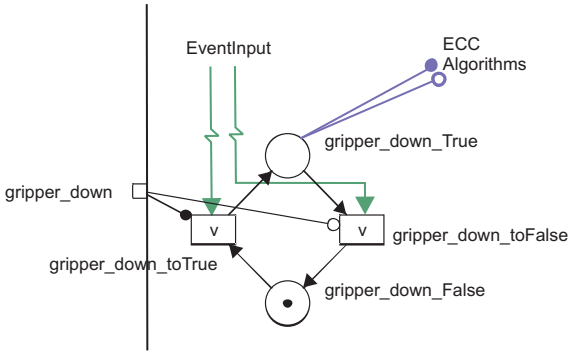


Figure 9. NCE structure representing a data input

The NCE structure consists of the places *+_True* and *+_False* as well as the connecting transitions *+_toTrue* and *+_toFalse*.

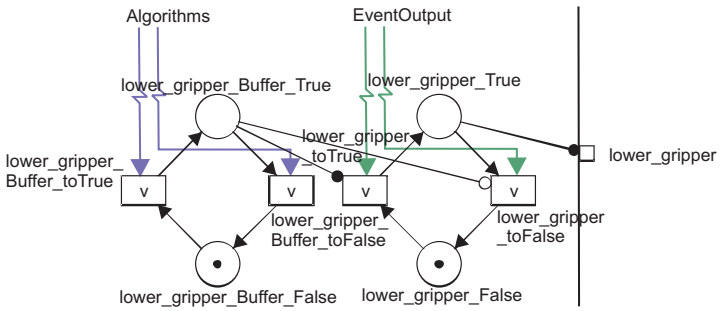


Figure 10. NCE structure representing a data output

The data input transformed to a condition input is connected by an inhibitor arc to the transition named *+_toFalse* and by a condition arc to the transition *+_toTrue*.

By transforming a data output, there is also a buffer for internal value change by any algorithm created. The names of the places are *+_Buffer_True* and *+_Buffer_False* and the transitions are named *+_Buffer_toTrue* and *+_Buffer _toFalse*. The place *+_Buffer_True* is connected with an inhibitor arc to the transition *+_toFalse* and by a condition arc to the transition *+_toTrue*.

According to rule 2.1, the transitions *+_toTrue* and *+_toFalse* are the event sinks of the outgoing event arc from the transition **_Set*.

3.2. Interface with integer-valued data inputs

To proceed with the hierarchical verification of the distributed master controller design pattern, proposed in Missal, Hirsch and Hanisch (2007), the several master controllers have to be transformed. If associated task controllers can perform several actions, the master controller has to coordinate the timed as well as the causal action performing with the other master controller.

As shown in Fig. 11, the timed coordination will be done by the event in- and output *Gripper* and *GripperO* and the causal coordination by the event qualifier *action*. The transformation of the interface can be done according to the defined rules, except for the data input *action*. Therefore, rule 2.2 has to be improved to the following:

Transformation rule 2.2.1 - Integer-valued Data In- and Outputs.

For each data in- or output with an integer-valued data type, rule 2.2 is repeated n times, where n represents the number of bits used for the data type. After the "+"sign the string *"_2pn"* is inserted. To complete the transformation a place with the name of the data input and the capacity of the highest possible value has to be inserted. This place is connected through flow arcs of the arcweight 2^n to the transitions named *+_2pn_toTrue* and *+_2pn_toFalse* and

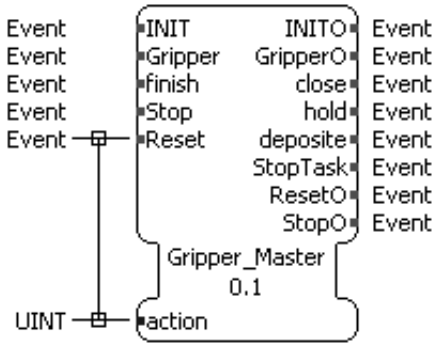


Figure 11. Master controller of the gripper station

lies at the post of the first and at the pre of the second transition. The result for the data input *action* is shown in Fig. 12, and for a possible data output *OUT*, in Fig. 13.

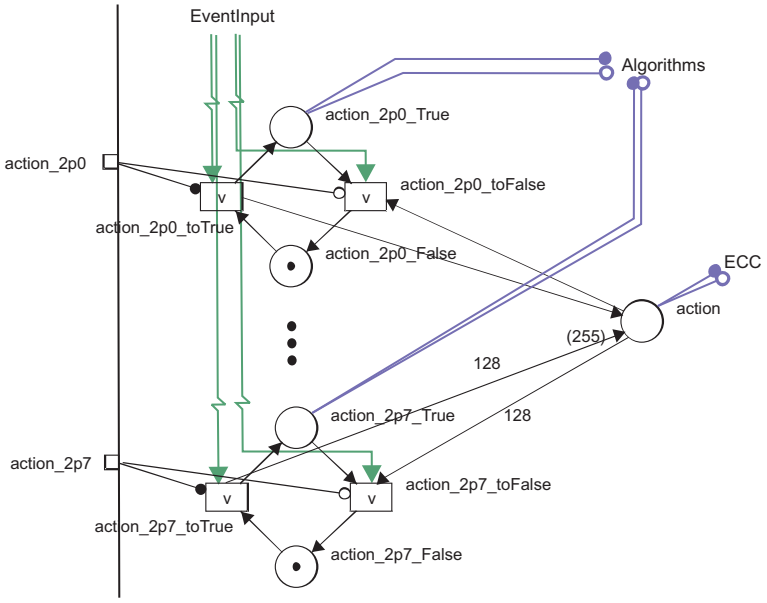


Figure 12. NCE structure representing a data input with integer-valued data type (8 bits)

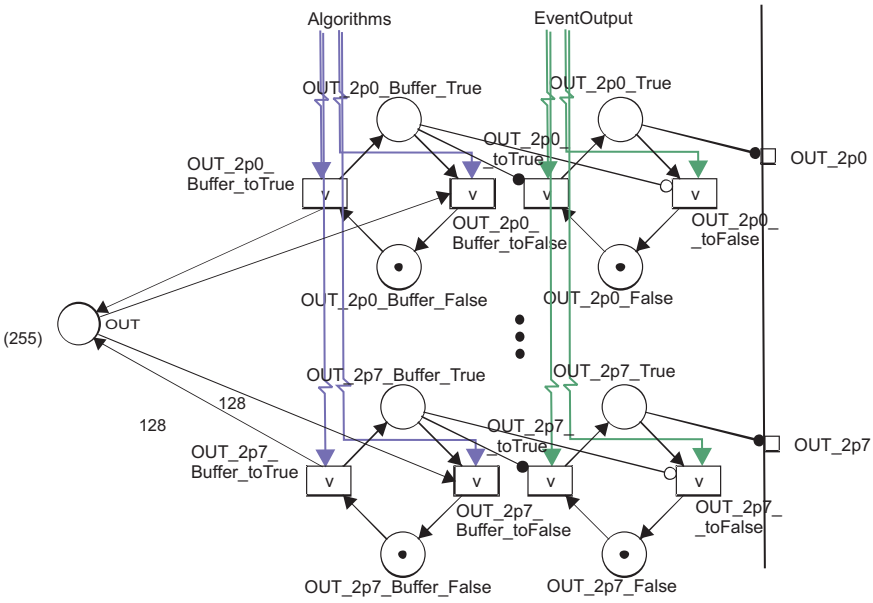


Figure 13. NCE structure representing a data output with integer-valued data type (8 bits)

3.3. Interconnection between interface and ExecutionControlChart

Inside the ECC the integer-valued data in- or output can be part of a guard condition controlling the clearing of an ECTransition. Therefore, rule 3.3 for transforming an ECTransition of Ivanova-Vasileva, Gerber and Hanisch (2007) has to be extended to model every possible condition, consisting of an event input or a guard condition only, or of combination of both, Dubinin and Vyatkin (2008).

For example, in this approach, a condition is used consisting of the event input *Gripper* and a guard condition checking the value of the data input *action*:

$$Gripper \ \wedge (action = 0 \vee action = 3 \vee action = 6).$$

This means that this ECTransition will clear only if it receives the mentioned event and if the data input has the value 0 OR 3 OR 6. For modelling this, the brackets have to be solved first, by the use of the distribution law. This leads to the disjunctive term:

$$Gripper \ \wedge action = 0 \vee Gripper \ \wedge action = 3 \vee Gripper \ \wedge action = 6.$$

Each conjunctive term of this disjunction can be modelled by transitions, which are controlled by the combination of event, condition and inhibitor arcs.

In our example, the ECTranstion from the ECState *Start* to *hold* is transformed to the transitions *START_hold_0*, *START_hold_1* and *START_hold_2*, which are controlled by an event arc from the transition *Gripper_Release* and a condition and an inhibitor arc from the place *action* with the arc weights according to Fig. 14.

Transformation rule 3.3 - ECTransitions

By using the law of distribution and de Morgan all brackets have to be solved to get n conjunctive terms connected through disjunctions. Each conjunctive term has to be modelled by a transition between the *§_Finished* place of the previous ECState and the *$_Run* place of the following ECState with the name as concatenation of the names of both ECStates and the string *_n*.

If the first variable of a conjunctive term is an event, an event arc from the transition **_Release* to the current transition has to be inserted, otherwise the switching mode of the current transition has to be set to *instant* (Fig. 14). The remaining part of the term represents a Boolean equation of internal variables and data in- and outputs, which can be modelled with condition and inhibitor arcs connecting the place *+_True* and the current transition.

If there is a comparison inside a Boolean equation between a variable X with integer-valued data type and a defined value i, then the following rules have to be taken into account.

Transformation rule 3.3.1 – $X > i$:
In order that X be greater than i, a condition arc with arc weight $i + 1$ has to be connected to the transition with its source at place X.

Transformation rule 3.3.2 – $X \geq i$:
In order that X be greater or equal i, a condition arc with arc weight i has to be connected to the transition with its source at place X.

Transformation rule 3.3.3 – $X < i$:
In order that X be lower than i, an inhibitor arc with the arc weight i has to be connected to the transition with its source at place X.

Transformation rule 3.3.4 – $X \leq i$:
In order that X be lower or equal i, an inhibitor arc with the arc weight $i + 1$ has to be connected to the transition with its source at place X.

Transformation rule 3.3.5 – $X = i$:
In order that X be equal i, a condition arc with the arc weight i and an inhibitor arc with the arc weight $i + 1$ have to be connected to the transition with their sources at place X.

Transformation rule 3.3.6 – $X \neq i$:
In order that X not be i, transformation rules 3.3.1 and 3.3.3 have to be combined. Thus, an inhibitor arc with the arc weight i has to be connected to one transition and a condition arc with arc weight $i + 1$ to a second transition. Both arcs have as their source place X. The two transitions are a part of the pre

and post of a place with two tokens and the capacity of four. Furthermore, they are connected via event arcs to the resulting transition with the event mode *or*.

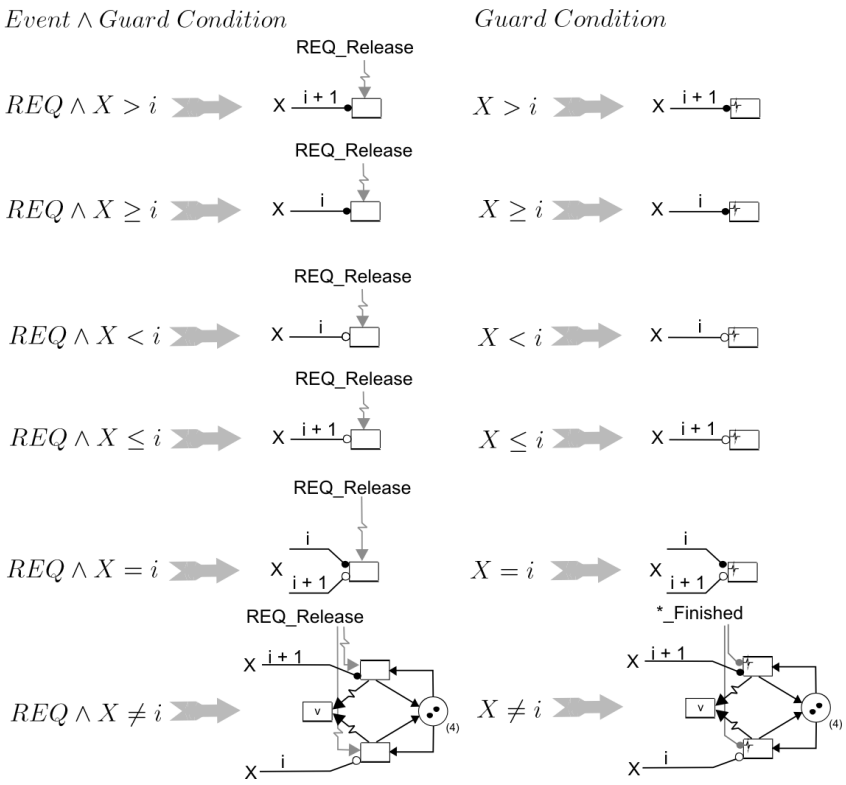


Figure 14. Transformation rule 3.3 - ECTransitions

4. Data processing model

With the defined execution model of function blocks, following the IEC 61499 in the previous section and the transformation rule 4 for algorithms, defined in Ivanova-Vasileva, Gerber and Hanisch (2007), one can get a formal model of the presented task and master controller of the EnAS Demonstrator, because the master controller uses the integer-valued data only in a guard condition of an ECTransition. But if the integer-valued data is processed like it is done by the servo control system in Fig. 15, it is not feasible, because the transformation of data processing is limited to set, reset and negate Boolean variables. Thus, a data processing model including all necessary operations of the servo control system will be derived from the proven methods of informatics in the following.

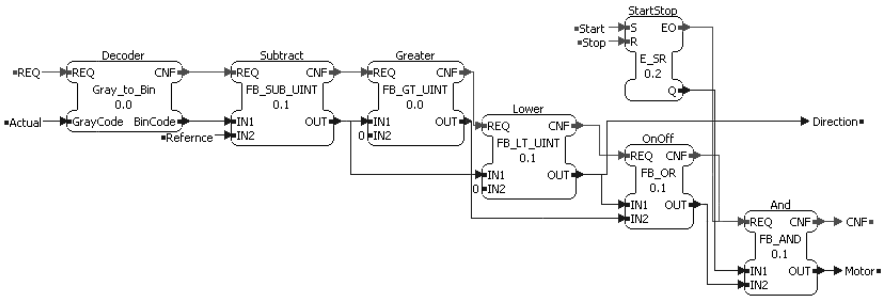


Figure 15. Control of the servo control system

As known from Becker, Drechsler and Molitor (2005), Schiffmann and Schmitz (2001), the basic operation of all four mathematical operations is the adding of n-digit binary numbers, which have to be realized first. Instead of direct subtraction of one value from another, the negated value of the second one and an additional carry bit can be added. The multiplication of two values can be done by adding the first value n times to zero, where n equals the second value, and division is n times subtracting the second value from the first until zero and n will be the result.

In the presented examples $IN1$ and $IN2$ will be used as input variables and due to the execution model described in the previous section and derived from the approach of Heiner and Menzel (1998), they are represented as place invariants of the places *IN1_2pi_True* and *IN1_2pi_False* as well as *IN2_2pi_True* and *IN2_2pi_False*. The variable to store the result is the data output OUT of a function block and due to the defined execution model of function blocks a buffer of OUT will be changed during the internal data processing. This is done by forcing the transitions *OUT_Buffer_2pi_toTrue* and *OUT_Buffer_2pi_toFalse*. By occurrence of the associated output event the actual buffer value is published.

4.1. Carry-ripple-adder

In a first sketch, the modelling of a *Carry-Ripple-Adder* (CR), also known as *Carry-Chain-Adder* (CCR) will be presented. The CR calculates the sum OUT_i and the carry c_i from the lowest to the highest digit i as follows ($\oplus \ldots XOR$):

$$OUT_i = IN1_i \oplus IN2_i \oplus c_{i-1}$$
$$c_i = IN1_i IN2_i \vee c_{i-1}(IN1_i \oplus IN2_i).$$

Because of calculating bit by bit the result and the carry for the next bit, n steps have to be fired at the formal NCE model of the *CR*. Each adding step is triggered from the algorithm by an event shown grey inside Fig. 16. Thereby, only one transition of the modelled adding step will be condition enabled to

switch the resulting bit of the modelled output buffer OUT_i to true or false. The conditions modelled at the triggered transitions are done according to Table 1 by condition and inhibitor arcs connected to the place *IN1_2pi_True* or *IN2_2pi_True*. Inside our model the left 4 or at the first step the left 2 transitions are representing the cases where the result of OUT_i becomes *true* and the others switch the result to *false*.

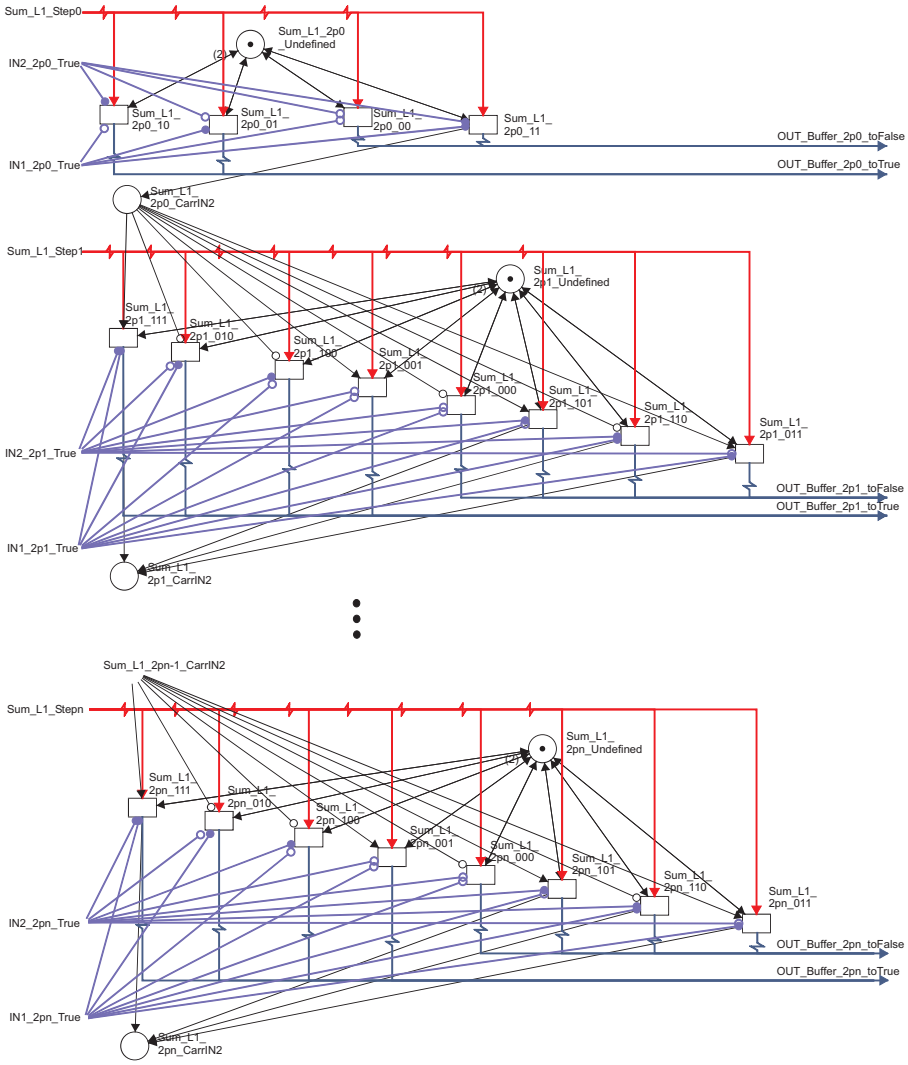


Figure 16. Carry-ripple-adder

Table 1. Carry-ripple-adder

	IN2	IN1	Carry
$OUT_i =$ **true**	1	1	1
	1	0	0
	0	1	0
	0	0	1
$OUT_i =$ **false**	0	0	0
	0	1	1
	1	1	0
	1	0	1
$c_i =$ **true**	1	1	1
	0	1	1
	1	1	0
	1	0	1

4.2. Carry-lookahead-adder

One point to cope with the state explosion problem during the reachability calculation is the use of efficient NCE structures. To do this the linear number of fired steps should be reduced in the following to a logarithmic one, by deriving the NCE model of a *Carry-Lookahead-Adder*. The main idea of Ladner and Fischer was to improve the calculation of the carry bits. Thereby, the attribute *generate* $g_{i,j}$ or *propagate* $p_{i,j}$ is evaluated for each digit block $[i, j]$ with $j \leq i < n$.

First, this attribute is evaluated for the block $j = i$ as follows.

$$p_{i,i} = IN1_i \oplus IN2_i$$
$$g_{i,i} = IN1_i IN2_i.$$

Afterwards, the attributes of the blocks $[i, k+1]$ and $[k, j]$ will be merged until $j = 0$, by the following rules:

$$p_{i,j} = p_{i,k+1} p_{k,j}$$
$$g_{i,j} = g_{i,k+1} \vee p_{i,k+1} g_{k,j}$$

The attribute *generate* is left hand stable, which means that if the most significant block $[i, j]$ has the attribute *generate*, then the block $[i, 0]$ has it also. So, inside the NCE model there will be only a place named †*_gi0*, and any transition evaluating the *generate* attribute to *true* will have a post arc to this place. Thus, also the transitions named †*_11_gii* evaluating the *generate* attribute at the first step by checking the digit i of variable $IN1$ and $IN2$.

As shown for *Step n* in Fig. 17 the *propagate* attribute is modelled by the place †_*pij*[1] at every evaluation step. Only at the first step this place has the capacity of two.

By merging the blocks $[i, k+1]$ and $[k, j]$ the token at the place †_*pkj* or †_ *gkj* will condition enable only one transition, which switches the state of the modelled attribute of the new block $[i, j]$ to *propagate* or *generate*. During the evaluation process, the n tokens of place †_*add* flow directly to the place †_*gi0* or through place †_*pii* to †_*gi0* or to †_*pi0*.

Using this information, the resulting sum of digit i is:

$$OUT_i = p_{i,i} \oplus (g_{i-1,0} \vee p_{i-1,0} c_{-1}).$$

Due to the modelling of an adder the input carry c_1 will be zero, which reduces the formula above to $OUT_i = p_{i,i} \oplus g_{i-1,0}$ and the resulting truth table to Table 2.

Table 2. Carry-lookahead-adder

OUT_i	$p_{i,i}$		$g_{i-1,0}$	
0	0	—○	0	$p_{i-1,0}$→
1	1	→	0	$p_{i-1,0} g_{i-1,0}$ —○
				$p_{i-1,0}$→
0	1	→	1	$g_{i-1,0}$→
1	0	—○	1	$g_{i-1,0}$→

According to the number of rows, the modelling as an NCE structure is done by five different transitions with inhibitor and flow arcs from place †_*pii* and flow arcs from place †_*pi-10* or †_*gi-10*. Only the transition representing the second row has two inhibitor arcs to the mentioned places. This means that the block $[i-1, 0]$ absorbs any carry. The names of the transitions are †_*ag_p_2pi*, †_*p_a_2pi*, †_*p_p_2pi*, †_*p_g_2pi* and †_*ag_g_2pi*. The place †_*add* is the post place of all transitions. Thus, the number of tokens there will be n at the start of the calculation and at the end again.

4.3. Subtraction

With the use of the NCE structure representing the basic operation of adding it is easy to get an NCE structure for subtracting $IN2$ from $IN1$. This is because the negation of the integer-valued data $IN2$ can be done by connecting the condition and inhibitor arcs of the first evaluating step instead to the place

[1]Here and further on, the sign † is used to denote the algorithm name and the line number, if not otherwise explicitly stated.

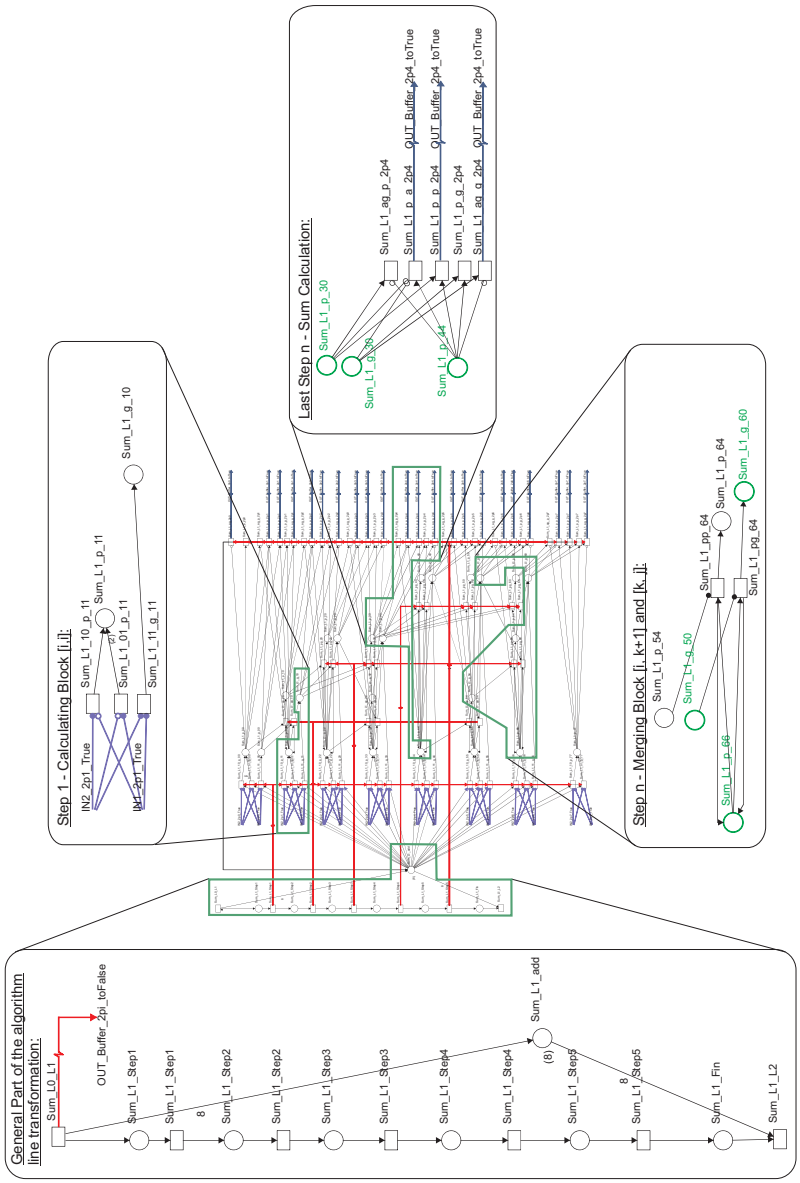


Figure 17. Carry-lookahead-adder

IN2_2pi_True to *IN2_2pi_False* and to calculate the sum with an input carry c_{-1} equal to *true*. This will change the formula of OUT_i to $OUT_i = p_{i,i} \oplus (g_{i-1,0} \vee p_{i-1,0})$ and the truth table to Table 3.

Table 3. Carry-lookahead-adder: subtraction

OUT_i	$p_{i,i}$		$g_{i-1,0}$	$p_{i-1,0}$	
0	0		0	0	not modelled
1	0	—○	0	1	$p_{i-1,0}\rightarrow$
1	0	—○	1	0	$g_{i-1,0}\rightarrow$
1	0		1	1	not possible
1	1	→	0	0	$p_{i-1,0}g_{i-1,0}$ —○
0	1	→	0	1	$p_{i-1,0}\rightarrow$
0	1	→	1	0	$g_{i-1,0}\rightarrow$
0	1		1	1	not possible

The first row must not be modeled, because the token has remained at the place †*_add*, and the sum bit OUT_i is switched to *false* at the beginning of the calculation. Furthermore, it will not be possible for a block $[i, 0]$ to be generating and propagating at the same time. As can be seen in Fig. 19, the number of used transitions is equal to the adder model, only the destination of the event arc is different, due to the value of OUT_i. The used names are also the same.

In the formal model of the servo control system *IN2* would get the value of the decoded actual position of the shaft and *IN1* has the value of the reference position. The result X of the subtraction is the position error.

4.4. Comparison

To map the comparison to the formal NCE structures one has to check if it is done between Boolean or integer values and also if it is done between a variable and a static value or between two variables. A Boolean comparison, e.g. whether the value of a variable is *true* or *false* or whether two variables are equal, can simply be modelled by condition and inhibitor arcs. The comparison of an integer-valued data with a static integer value can be modelled as presented in the execution model at the ECTransitions. In this paper we introduce the comparison between two integer-valued variables.

According to the previous description of transforming function blocks to formal models both variables are available as binary values. Thus, the solution will be a multistage one. At first, a decision is taken for every bit i if $IN1_i$ is greater or smaller than $IN2_i$ according to the truth Table 4.

Next, it must be checked if the most significant decision is “greater” or “smaller”. This will switch the result of the whole comparison accordingly. If variables are equal, no decision can be made, and it is still undefined. Fig. 18

shows the NCE structure model of a greater-than-comparator for integer-valued data.

Table 4. Truth table

$IN1_i$	$IN2_i$	$IN1_i = IN2_i$	$IN1_i < IN2_i$	$IN1_i > IN2_i$
0	0	*true*	*undefined*	*undefined*
0	1	*false*	*true*	*false*
1	0	*false*	*false*	*true*
1	1	*true*	*undefined*	*undefined*

By entering the comparing model, n tokens are added to the place †_*undefined*, because n bits have to be checked. The next fired transition †_*CompareBits* is the event source of the transitions †_*2pi_toGreater* and †_*2pi_toLower* to distinguish whether $IN1$ is greater or smaller than $IN2$ at bit i. Modelling of "greater" and "smaller" is done conform to Table 4 by inhibitor and condition arcs. Every fired transition takes one token from the place †_*undefined* , and only if both variable values are equal, all tokens remain there. Afterwards, the most significant decision about greater and smaller turns the result of the comparator accordingly. Modelling is done by a transition at the post of the place †_*2pi_Greater* and †_*2pi_Lower* and connecting them by inhibitor arcs with all places †_*2pj_Greater* and †_*2pj_Lower* and $j > i$. Firing one of these transitions removes the token from the preplace and stores one token at the place †_*Greater* or †_*Lower* and again one at †_*undefined*. Further, the token is removed from all places †_*2pk_Greater* or †_*2pk_Lower* and $k < i$, and transferred to †_*undefined*, by triggering the post transitions with the event mode $\vee$. Thus, at the end, place †_*Fin* is marked, n tokens are at the place †_*undefined* again and depending on the value of $IN1$ and $IN2$, the place †_*Greater* or †_*Lower* is marked also.

Depending on the transformed algorithm, the result can be used as needed, but it has to be made sure that all tokens are removed from the given NCE structure model.

4.5. Derived modelling rules

Before the presented examples and the derived rules could be used to model the data processing of IEC 61499 function blocks, a lexical analysis had to be done for each algorithm by an a priori defined formal grammar to identify each statement and the used variables. Due to the freedom to choose any kind of programming language, the lexical analysis is out of the scope of this contribution, but as result a collection of statements is received. This collection could be transformed by the following rules to a formal data processing model in NCE structures. Further, these rules will extend rule 4 for transformation of

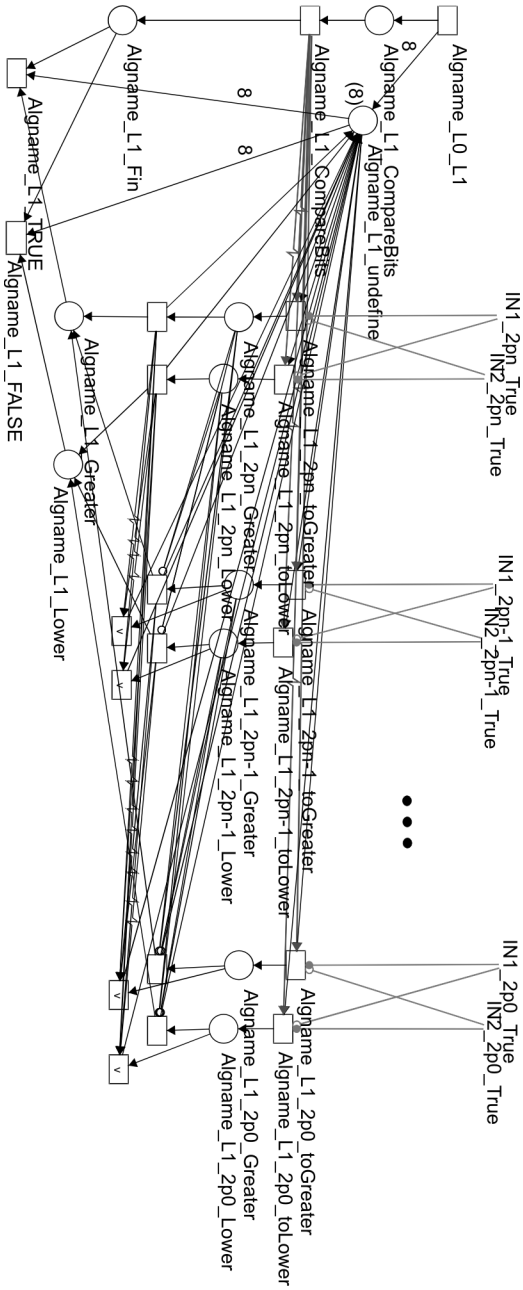


Figure 18. Greater-than-comparator

algorithms of Ivanova-Vasileva, Gerber and Hanisch (2007) and will be implemented in an automatic data processing modeller of IEC 61499 function blocks with integer-valued data types. The extended rule is still split into the general and conditional part, and rule 4.1 for the general part remains untouched to provide the connections between the execution and data processing model. In the conditional part one has to distinguish between Boolean and integer-valued data, which leads to the following rule:

Transformation rule 4.2 - Conditional Part:

Each line i of the algorithm is activated by a transition named *AlgName_Li-1_Li* and finished by a place named *AlgName_Li_Fin*. Further, the name of each transition and place gets a prefix consisting of the algorithm name and the line number *AlgName_Ln* (abbr. †), where n stands for the current line number.

First it is checked if Boolean or integer-valued data is used in the algorithm line by checking for each variable if there exists a place named *+_True*, where *+* is the name of the variable. If it is true, only Boolean data is used, and this line has to be transformed according to rule 4.2.1, otherwise by 4.2.2. Both rules have subrules to set the variables of this type to certain static values, to perform value assignment by given equations and to compare them.

Transformation rule 4.2.1 - Boolean-valued Data:

4.2.1.1 - Reset - Set Data: If the Boolean equation only sets or resets a Boolean data output, an event arc has to connect the transition activating this line with the transition *+_Buffer_toFalse* to reset it, or to the transition *+_Buffer_toTrue* to set it.

To negate the Boolean value of the variable, an event arc has to connect the transition *+_Buffer_ toFalse* and another one *+_Buffer_toTrue*. Thereby, the symbol *+* stands for the name of the data output or internal variable.

4.2.1.2 - Boolean equation: In Boolean equations the law of distributivity and de Morgan have to be used to resolve all brackets of the term after the equation sign to get n conjunctive terms connected by disjunctions. Each conjunctive term is modelled by a transition with the switching type *instantaneous* and named †_j, where j stands for the number of the conjunctive term, and every transition is connected by an event arc to the transition *+_Buffer_toTrue*. Furthermore, there is a transition with the switching type *spontaneous* connected by an event arc to the transition *+_Buffer_toFalse*, which will only fire if none of the other transitions is condition enabled. All these transitions have as preplace the place of the algorithm line before and as post place the place of this algorithm line.

4.2.1.3 - Boolean comparison: A Boolean comparison checks if the value of two terms is equal or not. Similar to the transformation of Boolean equations all brackets of both term have to be resolved first by the law of distributivity and de Morgan to receive n conjunctive terms connected by disjunctions.

If the left and right term consist only of a single variable, the transformation of an *equal to* comparison is done by inserting two transitions with the switching type *instantaneous* and connecting one of them by condition and the other by inhibitor arcs to the places *+_True*, where + stands for both variable names. A *not equal to* comparison of these two variables is done by inserting also two transitions with the switching type *instantaneous* and connecting the first with an inhibitor arc to the place *+_True* of one variable and by a condition arc to the place *+_True* of the other variable. The second transition is connected vice versa by signal arcs. Both transitions have as preplace the place of the algorithm line before. At the post area of both transitions is the place †*_True*. Furthermore, transitions with the switching type *spontaneous* connecting the place of the algorithm line before and †*_False* have to be inserted.

If the left and right terms consist of n conjunctive terms, the transformation is separated into the steps that model the calculation of the value of each term and compare the values.

1. Two places †*_TL_undefined* and †*_TR_undefined* have to be inserted. These places are connected by the transition *AlgName_Li-1_Li* to the place of the previous algorithm line. The transformation of the left n conjunctive terms results in n *instantaneous* and one *spontaneous* transition. The *instantaneous* transitions are connected by condition and inhibitor arcs to the places *+_True*, according to the represented conjunctive term, where + stands for the variable name. At the pre area of every transition place †*_TL_undefined* is located, and at the post area the place †*_TL_True*. The *spontaneous* transition has no incoming signal arc and connects the places †*_TL_undefined* and †*_TL_False*. The transformation of the right n conjunctive terms is done in a similar way.
2. If an *equal to* comparison is transformed, a transition with the places †*_TL_True* and †*_TR_True* at the pre area and a transition with the places †*_TL_False* and †*_TR_False* have to be inserted and connected at the post area to the place †*_True*. Further, a transition connected to the places †*_TL_True* and †*_TR_False* and a transition connected to the places †*_TL_False* and †*_TR_True* have to be inserted. Both have at the post area the place †*_False*.

In a *not equal to* comparison the post flow arcs of last four transitions are exchanged.

Transformation rule 4.2.2 - Integer-valued Data:

4.2.2.1 - Set Integer-valued Data: To set a variable with an integer-valued data type to a certain value n, event arcs have to connect the transition of this algorithm line with the transition *+_2pi_Buffer_toFalse* or *+_2pi _Buffer_toTrue* according to the binary representation of the certain value.

4.2.1.2 - Add Integer-valued Data: For transforming the adding of two n-bit integer-valued variables $IN1$ and $IN2$ and as result OUT, there have to be inserted first a sequence of $2 + log_2(n)$ places and transitions named †_*Stepi*. The place †_*Step1* is located at the post area of the transition activating this algorithm line, and the transition †_*Stepj* is at the pre area of the place †_*Fin* ($j = 2 + log_2(n)$). Next, the place †_*add* has to be inserted, having a pre arc with the arc weight n to the transition activating this algorithm line and a post arc with the same arc weight to the transition reaching the next algorithm line. Further, the transition activating this line has event arcs to every transition *OUT_Buffer_2pi_toFalse*.

Now, the model of the Carry-lookahead-adder has to be inserted, step by step as described in Section 4.2.

1. For every bit i three transitions, named †_*10_p_ii*, †_*01_p_ii* and †_*11_g_i0* and having the place †_*add* at the pre area and an incoming event arc from the transition †_*Step1* have to be inserted. The first two transitions are connected to the place †_*p_ii*, which has a capacity of two, by a post flow arc. The third transition has a post flow arc to the place †_*g_i0*. The first transition has a condition arc to the place *IN2_2pi_True* and an inhibitor arc to the place *IN1_2pi_True*, and the second transition vice versa. The third transition has a condition arcs to both places.
   Inserting of the transition †_*11_g_i0* and the place †_*g_i0* and all connecting arcs is neglected at the most significant bit.
   At the least significant bit the pre arcs of place †_*p_ii* have the arc weight 2.

2. For every bit i with

$$i < n \wedge i = 2^*2^{x*}y + 2^x + z \wedge$$
$$x < log_2(n) \wedge y < \frac{n}{2^{x+1}} \wedge z < 2^x$$

   two transitions, named †_*pp_ij* and †_*pg_ij*, with an incoming event arc from the transition †_*Stepx*, as well as a place named †_*p_ij* have to be inserted ($j = 2^*2^{x*}y$).
   At the pre area of both transitions is the place †_*p_ik*. If $i = k$ then this place is also at the post area of the transition ($k = 2^*2^{x*}y + 2^x$).
   Further, the transition †_*pp_ij* has a post flow arc to the place †_*p_ij* and a condition arc to the place †_*p_lj*. The transition †_*pg_ij* has a post flow arc to place †_*p_i0* already inserted at step 1 and a condition arc from place †_*g_lj* (l=k-1).
   At the most significant bit this is neglected.

3. For every bit i five transitions named †_*ag_p_2pi*, †_*p_a_2pi*, †_*p_p_2pi*, †_*p_g_2pi* and †_*ag_g_2pi* with a post arc to the place †_*add* and an event arc from the transition †_*Stepj* have to be inserted ($j = 2+log_2(n)$). Every transition with the prefix †_*p* has a pre flow arc to the place †_*p_ii* and all

other an inhibitor arc to this place. The transitions †*_p_a_2pi*, †*_p_p_2pi* and †*_ag_g_2pi* get an event arc to the transition named *OUT_Buffer_2pi_toTrue*.
Further, the transitions with the prefix †*_ag_p* and †*_p_p* have a pre flow arc to the place †*_pj0*, the transitions †*_p_g* and †*_ag_g* have a pre flow arc to the place †*_gj0*, and the transition †*_p_a* has two inhibitor arcs to the places †*_pj0* and †*_gj0* ($j = i - 1$).

4.2.1.3 - Subtract Integer-valued Data: To transform the subtraction of one n-bit integer-valued variable $IN1$ from $IN2$ and storing the result at OUT, point 1 and 3 of the previous adding rule have to be changed. In 1 the destination of the condition and inhibitor arcs has to be changed from *IN2_2pi_True* to *IN2_2pi_False*. In 3 the event arcs to the transition *OUT_Buffer_2pi_toTrue* have to be changed as follows:

- †*_ag_p_2pi* new event arc
- †*_p_p_2pi* no event arc

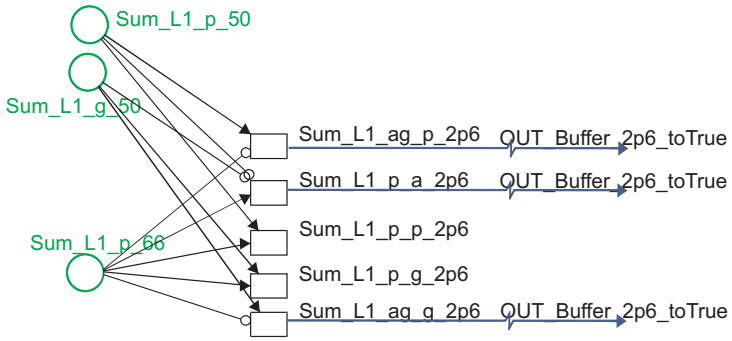


Figure 19. Changes for substraction in the Carry-lookahead-adder

4.2.1.4 - Compare Integer-valued Data: The transformation of the comparison of two n-bit integer-valued variables $IN1$ and $IN2$ to the formal model is done by inserting the places †*_CompareBits* and †*_Fin* and the connecting transition †*_CompareBits*. Further, the place †*_undefined* with the capacity of n is inserted and connected to the transition reaching this line by a pre arc with the arc weight of n.

Now, the model of the comparator, as described in Section 3, starting with the creation of the places †*_Greater* and †*_Lower* has to be inserted.

1. Two transitions named †*_2pi_toGreater* and †*_2pi_toLower* as well as two places †*_2pi_Greater* and †*_2pi_Lower* are inserted for every bit i. At the pre area of every transition the place †*_undefined* is located and additionally every transition has an incoming event arc from transition †*_CompareBits*. Further, transition †*_2pi_toGreater* has a condition arc

to the place *IN1_2pi_True* and an inhibitor arc to *IN2_2pi_True*. The transition †*_2pi_toLower* is connected, vice versa, by the signal arcs.

2. For every bit i two transitions named †*_2pi_Greater* and †*_2pi_Lower* are inserted, connecting the places †*_2pi_Greater* and †*_Greater* and the places †*_2pi_Lower* and †*_Lower*. Every transition has an incoming inhibitor arc from all places †*_2pj_Greater* and †*_2pj_Lower* ($j > i$).
Except at the most significant bit, two transition are inserted with the event mode $\vee$ having the place †*_2pi_Greater* or †*_2pi_Lower* at the pre area and an incoming event arc from the transitions †*_2pj_Greater* and †*_2pj_Lower* ($j > i$).
All transitions have at the post area the place †*_undefined*.

5. Plant modelling

Each manufacturing system consists of several mechanical components, like conveyors, cylinders, valves, tanks and storages, as well as digital and analogue sensors and actuators. The sum of all components describes the physical equipment of the plant, which changes the properties of the workpieces during the processing. To be applicable to an engineer, the formal model should provide a modular way of modelling the plant and workpiece behaviour. Each module encapsulates the discrete and uncontrolled behaviour of the represented mechanical units. This means every behaviour, feasible at each physically possible state under each order of arbitrary assignment of the actuator state has to be described, as shown in Hanisch, Kemper and Lueder (1999).

During the last decade a library of often used NCE modules encapsulating the uncontrolled behaviour of several mechanical components was built up to facilitate the manual development of the plant models as described in Vyatkin and Hanisch (2003) by interconnecting these modules with event and condition arcs. This composition of large models from smaller ones is obvious to any engineer, who has ever modelled a system in a block diagram oriented way.

The structural description of the used plant is shown in Fig. 4. The gripper station consists of two valves, two cylinders and two sensors. For each plant part a separate timed NCE module is used and interconnected inside the *gripper* module by event and condition arcs (Fig. 20).

Depending on the *true* and *false* state of the control output, the connected pneumatic valve switches to *ON* or *OFF*. This enables the flow of compressed air into or out of the corresponding cylinder, which extends or retracts. Doing so, the gripper moves up and down or is closed and opened again. The up and down movement turns the positioning sensors immediately to *ON* or *OFF*, if the end positions are reached. Also, if a closed gripper starts moving up or finishes its down moving, a tin is immediately taken from or deposited on the pallet. For modelling this synchronous behaviour event arcs are used. Despite this, a tin is only closed correctly after the gripper is closed for a certain period

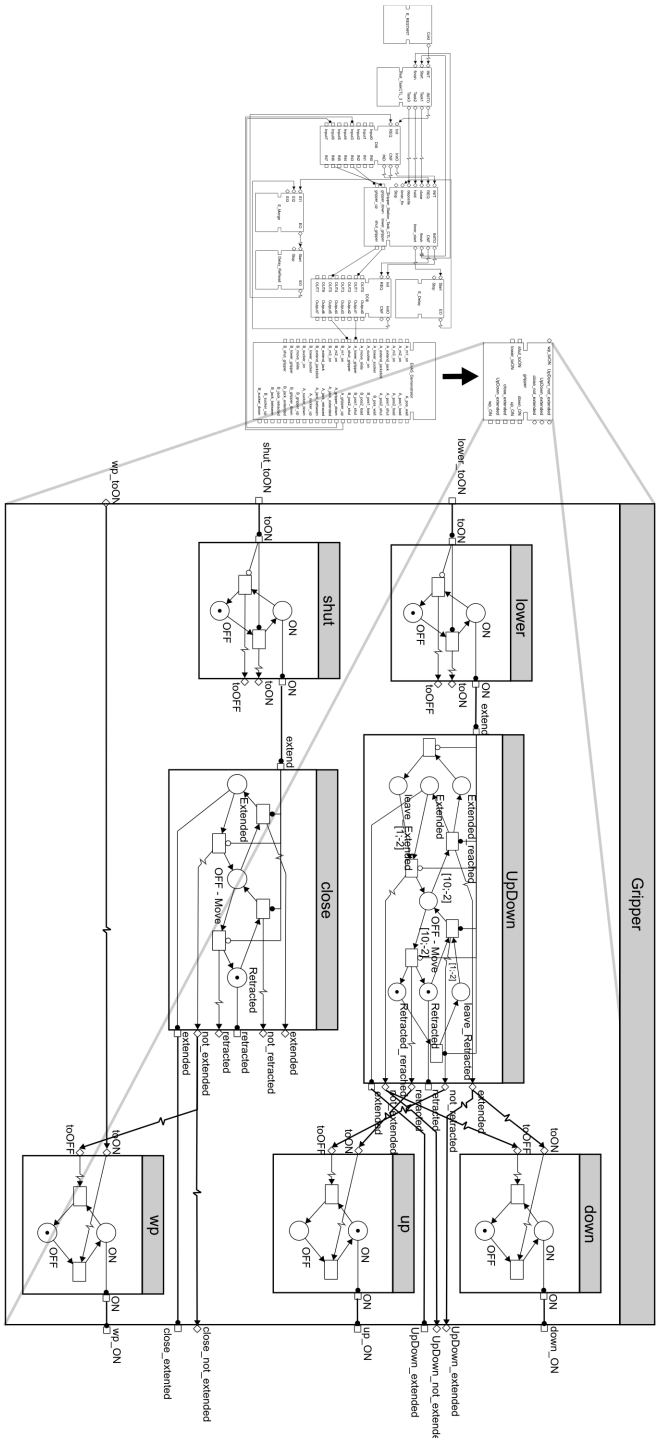


Figure 20. TNCES model of the interconnected master and task controller with gripper station

of time, which results in modelling the connection between the closing cylinder and the tin attribute by a condition arc. Also the physical gripper can interact in different plants or even at the same plant with different kinds of workpieces and at our plant with three different pallets with different loading states. This workpiece behaviour is modelled inside other NCE modules to make the *gripper* module reusable.

The modelling of the uncontrolled behaviour of the jack station is done similarly to the gripper station, but according to Fig. 4 there are four modules of the type valve, three modules representing a cylinder and five sensor modules used and interconnected by condition and event arcs. A model of a conveyor of the EnAS Demonstrator is obtained by interconnecting an engine and a conveyor module by event arcs. This is done because both have a mechanical connection, and if the engine starts to rotate, also the conveyor starts to move. Both modules have inside a place invariant of the places *ON* and *OFF*, because the engine could only be switched on or off, and accordingly the conveyor. The movement of the three pallets is modelled in a separate module and is enabled by a condition signal from the six different conveyors. Each position sensor is connected by event arcs to the three pallets module, which announce an event if they reach the sensor position and if they leave it. Also the loading state of the tins on the pallets is modelled separately, and, combined with movement modules, it represents the appropriate workpiece behaviour. As result a hierarchical NCE module, describing the physical plant interconnected with a material model, is achieved.

To get the model of the plant more realistic and to cope with the state explosion problem, discrete times are introduced to the cylinder and pallet module. These discrete times describe the time relations and not the exact times between the different plant modules and the control. Due to the not known execution times for algorithms and the event propagation, only the input scan gets a discrete time at the control model. This ensures also that the control is faster than the plant. To illustrate a significant problem of the different temporal behaviour of the plant and the control, consider the following example. During the movement of the cylinder, which takes approximately 1 second, the control unit rereads its inputs roughly every 5 ms and checks if something changed or not. If not, it waits until the next reread. According to the semantics of timed NCE modules and systems, the sequence of the input reread would occur 200 times without any important information for the model checking. Thus, only discrete times with a value representing the time relations between the different actions like cylinder and pallet movement and input reread are used. Therefore, time delays are used in the controller models that are three to five times smaller than the smallest time delay in the plant modules. On the other hand, this ensures that the dynamic graph is kept as small as possible.

To model the shaft encoder of the servo control system, an 8-bit Gray code counter is modelled as basic NCE module. The upward counting describes the left and the downward the right movement of the shaft. As known from infor-

matics, the state change of bit i occurs only if the bit $i-1$ is true and every less significant bit is false. This could be modelled by condition and inhibitor arcs, and the resulting module is afterwards connected to the transformed function blocks, as shown in Fig. 21.

6. Conclusion and future work

With the transformation rules of Ivanova-Vasileva, Gerber and Hanisch (2007) and their extensions described in this contribution it is possible to get a formal execution and data processing model including integer-valued data processing and comparison for almost every basic and composite function block at the application level as well as their interconnections. The behaviour of the presented data processing models is derived from the proven methods of informatics, and a further validation was done by simulation using the closed-loop model of the servo control system.

Despite the automatic function block transformation, the plant model has to be created up to now manually, but it is facilitated by using premodelled timed NCE modules for actuators and sensors as well as for other plant equipment. Both formal models are interconnected by condition arcs, and using the TNCES-Workbench, implemented inside the expert system SWI-Prolog, a dynamic graph for the closed-loop behaviour can be calculated.

For the closed-loop system of the modelled plant and the task controller of the gripper station the result is a graph with 1347 states. At a first sketch a visual verification by the user like it was done in Ivanova-Vasileva, Gerber and Hanisch (2007) may not be suitable, but if the graph layout algorithm *neato* (Kamada and Kawai, 1989) from the graph visualization software *Graphviz* is used and important steps including transitions of the plant model are marked with different colours, it becomes feasible for this system like presented in Gerber (2008b).

Further, an interesting trajectory can be visualized by drawing a Gantt chart including the marking of places. An example for the closed-loop system of the gripper performing the action to close a tin is shown at Fig. 22. The figure shows several groups like sensor with the up sensor draw in dark and the down in light grey or the lower group showing the state of the corresponding actuator. The group event shows the received events INIT, REQ (dark grey) and the published event CNF (light grey). Using the group ECC, the current execution state schedule algorithm (white), waiting for completion (grey) and idle (dark grey) of each modelled ECState can be checked. This Gantt chart can now be compared with the provided specification as Symbolic Timing Diagrams (Preusse and Hanisch, 2008; Schloer, Josko and Werth, 1998) or Timing Diagrams (Fisler, 1999; Vyatkin and Bouzon, 2008). The same can be done for the closed-loop systems of the modeled plant with the task controller of the jack station and with the task controller of the conveyor. Thereby, dynamic graphs

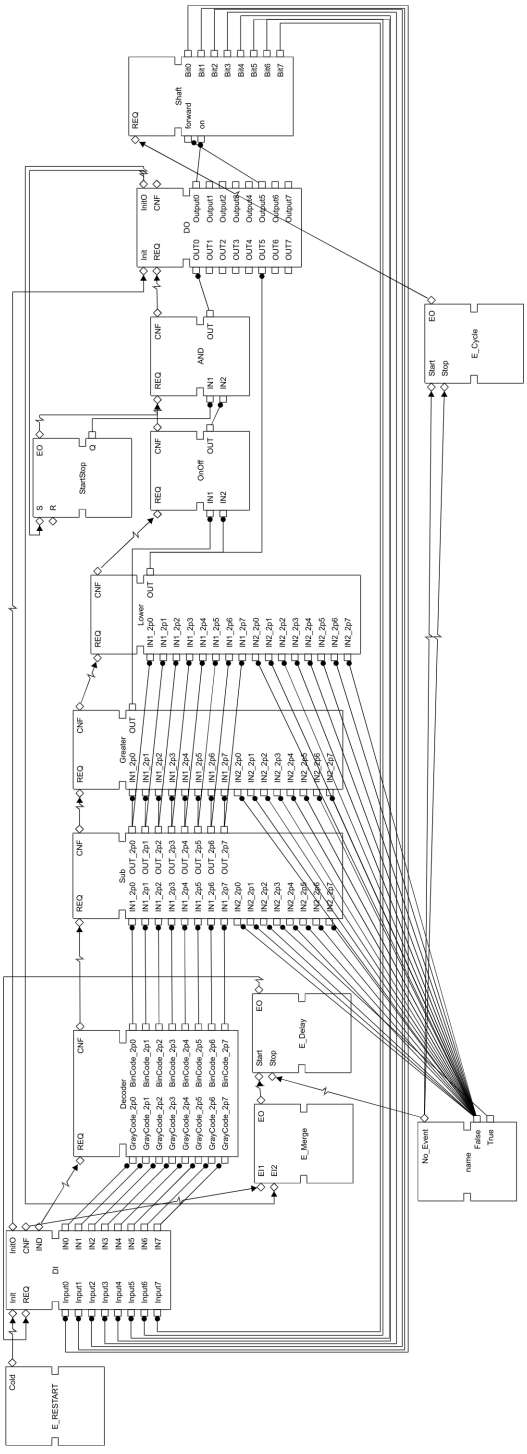


Figure 21. TNCES model of the closed-loop servo control system

of 2699 states for the jack station and 624 states for the conveyor are calculated and visually verified as described above.

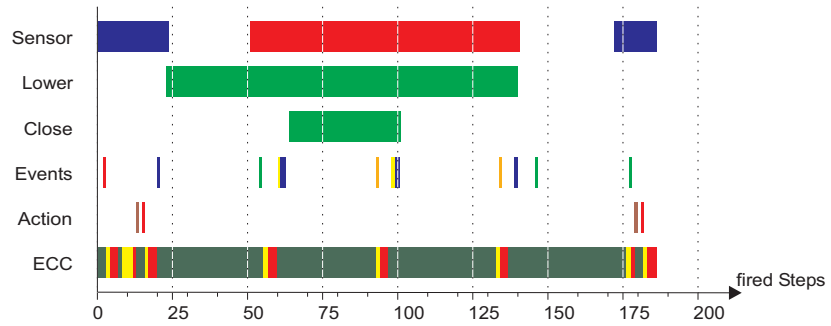


Figure 22. Visualisation of a trajectory of the gripper system

If the considered system gets even larger, at the next step of the hierarchical verification, a formal verification by temporal logic with CTL and eCTL formulae have to be done. Therefore, rules have to be defined in future work to transform the dynamic graph to a Kripke structure, for which verification tools like NuSMV, Uppall and so on exist. The counterexample to the provided formal specification can be visualised as a Gantt chart, as well, for finding the failures faster.

For completing our work, we will fully implement the presented rules of modelling the function block interface and the connection to the ECC inside the expert system SWI-Prolog and the resulting TNCES-Workbench will be made available to the public until the middle of the year 2009.

Acknowledgements

This work was partially supported by the cooperative project EnAS funded by the German Ministry for Commerce and Industry (BMWI) under reference 01MG566 and by the Deutsche Forschungsgemeinschaft under reference Ha 1886/16-1.

References

Becker, B., Drechsler, R. and Molitor, P. (2005) *Technische Informatik - Eine Einführung.* Pearson Studium, München.

Bonfe, M. and Fantuzzi, C. (2003) Design and verification of mechatronic object-oriented models for industrial control systems. In: *International Conference on Emerging Technologies and Factory Automation (ETFA'03)*, II, 253–260. Proceedings, Lisbon, Portugal.

Brennan, R., Vrba, P., Tichy, P., Zoitl, A., Suender, C., Strasser, T. and Marik, V. (2008) Developments in dynamic and intelligent reconfiguration of industrial automation. *Computers in Industry* **59** (6), 533–547. 10.1016/j.compind.2008.02.001.

Dimitrova, D., Frey, G. and Batchkova, I. (2007) Sequential control at the supervisory level of batch plant using signal interpreted Petri nets. In: *International Conference Automatics and Informatics (CAI'07)*, V–17–V–20. Proceedings, Sofia, Bulgaria.

Dubinin, V. and Vyatkin, V. (2008) On definition of a formal model for IEC 61499 function blocks. *EURASIP Journal of Embedded Systems* **2008**, 1–10. 10.1155/2008/426713.

Fisler, K. (1999) Timing diagrams: Formalization and algorithmic verification. *Journal of Logic, Language, and Information* **8** (3), 323–361. 10.1023/A:1008345113376.

Frey, G. and Hussain, T. (2006) Modeling techniques for distributed control systems based on the IEC 61499 standard – current approaches and open problems. In: *8th International Workshop for Discrete Event Systems (WODES 2006)*, 176–181. Ann Arbor, Michigan, USA.

Gerber, C. (2008a) Enas project demonstrator. http://aut. informatik.uni-halle.de/forschung/enas_demo.

Gerber, C. (2008b) Verification of the gripper station. http://aut.informatik.uni-halle.de/forschung/enas_demo/gripper.

Gerber, C., Hirsch, M. and Hanisch, H.M. (2009) Automatisierung einer energieautarken Fertigungsanlage nach IEC 61499. *Automatisierungstechnische Praxis*, 03/09, 44–52. München, Germany.

Hanisch, H.M., Kemper, P. and Lueder, A. (1999) A modular and compositional approach to modeling and controller verification of manufacturing system. In: *14th IFAC World Congress*, 187–192. Proceedings, Beijing, P.R. China.

Heiner, M. and Menzel, T. (1998) Instruction list verification using Petri net semantics. In: *International Conference on Systems, Man, and Cybernetics*, 716–721. Proceedings, San Diego.

Ivanova-Vasileva, I., Gerber, C. and Hanisch, H.M. (2007) Transformation of IEC 61499 control systems to formal models. In: *International Conference Automatics and Informatics (CAI'07)*, V-5-V-10. Proceedings, Sofia, Bulgaria.

Kamada, T. and Kawai, S. (1989) An algorithm for drawing general undirected graphs. *Information Processing Letters* **31** (1), 7–15. 10.1016/0020-0190(89)90102-6.

Maffezzoni, C., Ferrarini, L. and Carpanzano, E. (1999) Object-oriented models for advanced automation engineering. *Control Engineering Practice*, **7** (8), 957–968. 10.1016/S0967-0661(99)00074-X.

Missal, D., Hirsch, M. and Hanisch, H.M. (2007) Hierarchical distributed controllers - design and verification. In: *International Conference on*

Emerging Technologies and Factory Automation (ETFA'07), 657–664. Proceedings, Patras, Greece.

Pinzon, L., Jafari, M.A., Hanisch, H.M. and Zhao, P. (2004) Modeling admissible behavior using event signals. *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics*, **34**, 1435–1448. 10.1109/TSMCB.2004.825915.

Preusse, S. and Hanisch, H.M. (2008) Specification and verification of technical plant behavior with symbolic timing diagrams. In: *3rd International Design and Test Workshop (IDT'08),* 313–318. Proceedings, Monastir, Tunisia.

Readman, M. (2005) *Servo Control Systems 2: Digital Servomechanisms.* TQ Education and Training Ltd. http://control-systems-principles.co.uk/whitepapers/servo-control-systems2.pdf.

Schiffmann, W. and Schmitz, R. (2001) *Technische Informatik 1.* Springer Verlag.

Schloer, R., Josko, B. and Werth, D. (1998) Using a visual formalism for design verification in industrial environments. In: *Lecture Notes in Computer Science*, **1385**, 208–221. Springer-Verlag. 10.1007/BFb0053491.

Stanica, M. and Gueguen, H. (2003) A timed automata model of IEC 61499 basic function blocks semantic. In: *Euromicro European Conference on Real-Time Systems (ECRTS'03), 385–390.* Proceedings, Porto, Portugal.

Suender, 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) Usability and interoperability of IEC 61499 based distributed automation systems. In: *International Conference on Industrial Informatics (INDIN'06)*, 31–37. Procedings, Singapore.

Vyatkin, V. and Bouzon, G. (2008) Using visual specifications in verification of industrial automation controllers. *EURASIP Journal on Embedded Systems.* 10.1155/2008/251957.

Vyatkin, V. and Hanisch, H.M. (1999) A modeling approach for verification of IEC 1499 function blocks using net condition / event systems. In: *Proc. Int. Conf. on Emerging Technologies and Factory Automation (ETFA'99)*, Catalonia, Spain, 261–269.

Vyatkin, V. and Hanisch, H.M. (2000) Modelling of IEC 61499 function blocks as a clue to their verification. In: *Design and optimization of intelligent machine tools,* 59–68. Proceedings, Karpacz, Poland.

Vyatkin, V. and Hanisch, H.M. (2003) Verification of distributed control systems in intelligent manufacturing. *Journal of Intelligent Manufacturing* **14** (1), 123–136. 10.1023/A:1022295414523.

Vyatkin, V. and Hanisch, H.M. (2005) Reuse of components in formal modelling and verification of distributed control systems. In: *International Conference on Emerging Technologies and Factory Automation (ETFA'05)*, 129–134. Proceedings, Catania, Italy.

Vyatkin, V., Karras, S. and Pfeiffer, T. (2005) An architecture for automation system development based on IEC 61499 standard. In: *International Conference on Industrial Informatics (INDIN'05)*, 13–18. Proceedings, Perth (Piscataway - Western Australia).

Zhang, J., Gu, J., Li, P. and Duan, Z. (1999) Object-oriented modeling of control system for agile manufacturing cells. *International Journal of Production Economics* **62** (1-2), 145-153. 10.1016/S0925-5273(98)00227-8.