Formal models for the verification of IEC 61499 function block based control applications

Arndt Lüder, Christian Schwab, Marcus Tangermann, Jörn Peschke
Univ. of Magdeburg, Center Distributed Systems – CVS@IAF
Universitätsplatz 2; D-39106 Magdeburg; Germany
{arndt.lueder/christian.schwab/marcus.tangermann/joern.peschke}@mb.uni-magdeburg.de

Abstract

Industrial automation is currently on the cusp to the application of distributed systems based on distributed intelligence enabling distributed decision making within control. As one main technology within this field IEC 61499 based function block systems will be applied. But the usage of function block systems may cause problems which have to be avoided prior to the integration of a function block system in a control system. Hence, function block systems need to be examined using formal models and formal verification technologies. Therefore, formal models need to be created automatically during the function block system design. Within this paper an algorithm for automatic model design and its integration in the Function Block System Designer will be described.

1. Introduction

Industrial automation is currently at the crossroads. Conventional hierarchical organized and on central decision making based control architectures will be step by step replaced by distributed control systems based on distributed intelligence and distributed decision making. But this replacements requires a reasonable support during the design and maintenance phases of control systems. Here, the huge variety of conventional control programming systems such as STEP7, RSLogix, or PL7-PRO are not sufficient any more, since the conventional programming systems will not provide means for describing and maintaining distributed systems.

This problem has been faced by industry as well as by the research community. Within industry the new control specification and programming standard IEC 61499 [1] and new control programming strategies as for example integrated in the ProfiNet CBA based Imap [2] and the TransparentFactory [3] approach first industrial tools have been arisen to enable the application of distributed control systems based on distributed intelligence and distributed decision making. Within the academic world especially the IEC 61499 has been adopted as basement for the consideration of distributed control systems with respect to specification, implementation, and validation with the Function Block Development Kit (FBDK) [4], the tools around the CORFU approach [5,6], the Function Block System Designer (FBSD) [7] and the VEDA tool [8]. Within any case either industrial or academic the developed function block systems (FBS) can contain multiple errors and malfunctions. Prominent examples for such malfunctions are deadlocks created by pair wise on each other waiting functions blocks or hazards of different functions block sequences. Hence, formal models describing the behavior of are IEC 61499 based FBS are required to provide a mean for validation and verification. These models have to be automatically designed during and in parallel to the design process of FBS to ensure a continuous equivalence of the FBS and its formal model.

The aim of this paper is to describe a methodology to design such models in parallel to a FBS design tool. Here the Function Block Systems Designer (FBSD) from Magdeburg University is used as a reference.

Several different formal models for Discrete Event Dynamic Systems (DEDS) are available. During the last years one type of models has been proven as very efficient for modeling of FBS since it enables a direct expression of the block, data, and event expression, the Net-Condition/Event-Systems (NCES, also known as signal event systems) first introduced by Hanisch and Rausch [9] in 1995. For this type of models several methodologies for formal analysis have been developed and implemented [10,11]. Hence, this type of models is used within this approach of automatic modeling of FBS.

This paper is organized as follows. Within the second section IEC 61499 FBS and the NCES are introduced. The third section describes the automatic step by step translation of FBS to NCES and the fourth section provides information on the integration of the automatic model design within the FBSD tool. With a short conclusion and an outlook on further research within upcoming international research projects the paper is finished.

2. Models for distributed control systems

IEC 61499 based Function Block Systems as well as Net-Condition/Event-Systems are special types of models used to describe discrete event dynamic control systems. Within the following both types of models will be described in more detail.

2.1. Function block systems based in IEC 61499

The IEC 61499 standard is based on a fundamental module, the Function Block (FB), which represents a functional unit of control software, associated to a hardware resource of a control system. A FB instance is characterized by its type name and instance name, a sets of event inputs/outputs, a set of data inputs/outputs, an execution control chart (ECC), internal data, and internal algorithms.

The type and instance name is used to uniquely identify a FB, the event and data inputs and outputs are required for the interconnection of different FB to a FBS, while the ECC, the internal data, and the internal algorithms will describe the internal behavior of the FB.

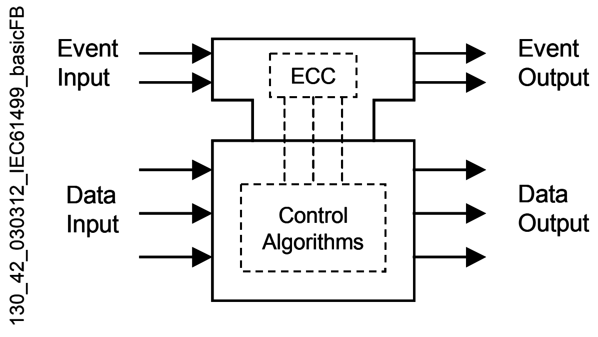


Figure 1. Basic structure of a Function Block

For the overall behavior of a FBS the ECC and its connection to events, data, and algorithms is of most importance. An ECC consists of states, transitions and actions, which invokes the execution of algorithms, which are associated to the ECC states, in response to event inputs. An incoming event will be read by the ECC and can enforce the invoking of an algorithm in dependence of the active state of the ECC, the possible state changes enforced by the incoming event, and internal bordering conditions of state changes described by transition guards. When the execution of an algorithm is invoked, the needed input and internal data values are read and new values for output and internal data may be calculated. Furthermore, upon completion of execution of an algorithm, the execution control part generates zero or more event outputs as appropriate which can be transmitted to other FB.

By properly connecting more then one FB, a FBS generating a distributed application can be defined. The event flow between ECCs of FBs determines the scheduling and execution of the algorithms and thereby the behavior of the complete FBS based control application.

An application can be distributed among several devices. A device uses the causal relationship specified by the application FBs to determine the appropriate responses to events, which may include communication and process events, utilising the different resources associated to the devices.

To give the described FB structure a more understandable background the FB example for controlling a pushed system as depicted in Figure 2 will be examined.

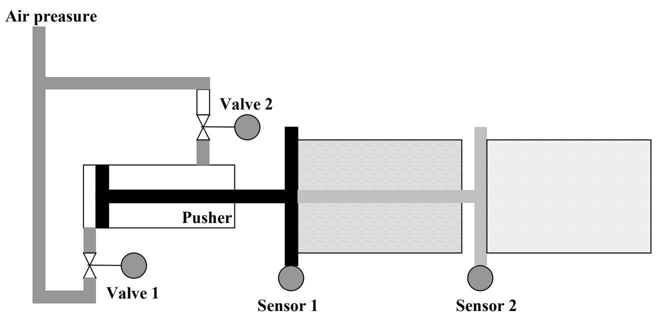


Figure 2. Pusher example

This pusher can be actively extended and retracted by using the valves 1 and 2. Within a FB dedicated to control the pusher for both activities as special algorithm can be implemented opening and closing the valves depending on the pusher position.

Additionally the pusher can reach an error state, i.e. the extension or retraction process will fail by seizing it. This case will also require a special algorithm handling the failure.

As the result of the mentioned behavior the FB controlling the FB can be structured as given in Figure 3.

Within this figure the ECC consists of 3 states. The “Wait” state is used for process initialization and as waiting state before an object in front of the pusher has to be moved. The “Move” state is used to move an object by extending and retracting the pusher. The “Error” state is used to handle the error of a seized pusher.

The FB has one internal variable “ObjectPushed” used to determine whether the pusher has moved an object or not. It will be used within transition guards to decide whether the pusher has been seized or not and thereby whether the error state has to be entered or not.

The FB can be integrated in a FBS by using the event and data inputs and outputs. For example, the “MoveEvent” event input can be used to enforce the moving of a work piece and the “ErrorEvent” event output will signalize that the pusher has seized up.

2.2. Net-Condition/Event-Systems

Extending Petri nets by incoming and outgoing signals is by no means new. Some classical concepts from Petri net application to discrete event controller design use such signal extensions. These extensions, however, do not provide means for interconnecting several separate Petri nets with incoming and outgoing

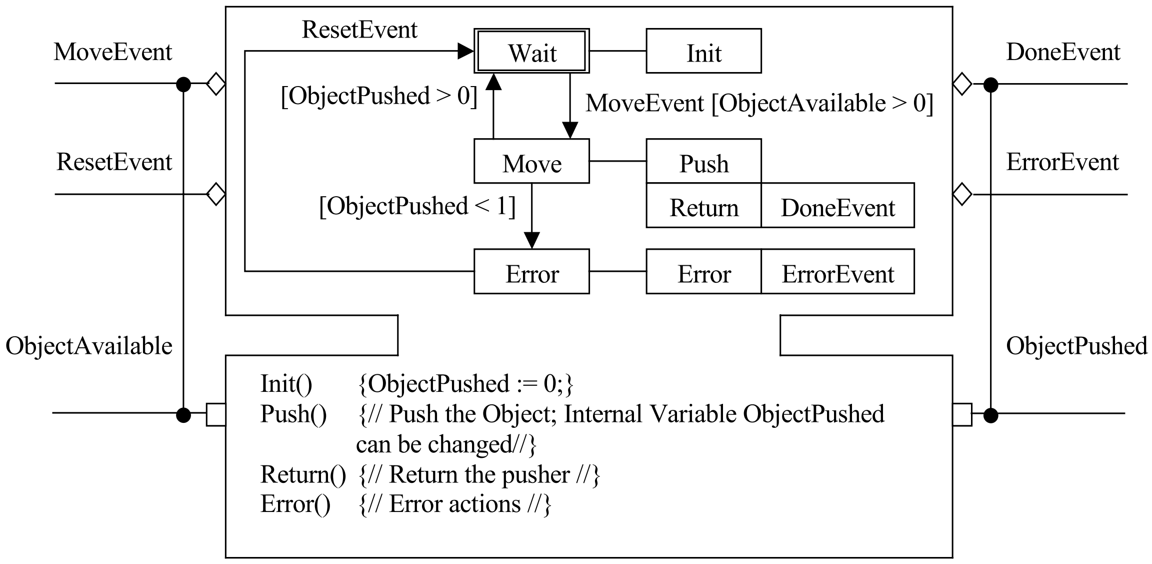

MoveEvent
ResetEvent
ObjectAvailable
ResetEvent
Wait
Init
[ObjectPushed > 0]
MoveEvent [ObjectAvailable > 0]
Move
Push
[ObjectPushed < 1]
Return
DoneEvent
Error
Error
ErrorEvent
DoneEvent
ErrorEvent
ObjectPushed
Init() {ObjectPushed := 0;}
Push() {// Push the Object; Internal Variable ObjectPushed can be changed//}
Return() {// Return the pusher //}
Error() {// Error actions //}

Figure 3. Function block to control the pusher

signals to a new model which has the same characteristics. Such means for interconnecting several subsystems to a new system, however, are needed if a automatic model design for FBS is intended. The modelling framework has to provide means to compose models of individual FB models by event and data connection describing means to a consistent model. These type of signal are integrated in the Net-Condition/Event-Systems (NCES).

The term "Net Condition/Event System" (abbr.: NCES) is somewhat misleading since it does not relate to 1-bounded Petri nets which are usually called Condition/Event systems as well. The terms "Conditions" and "Events" refer to two different types of signals interconnecting subsystems. The idea of defining these two types of signals came from R. Sreenivas and B. Krogh [12]. In the original paper condition and event signals are defined over a time axis where time takes its values from a subset of real numbers. Condition signals are piecewise constant over time. They "jump" to new values at certain points in time. Event signals, in contrast to this, have nonzero values only at certain points in time. They can be thought of as pulses of zero length. Signal propagation delays do not exist for this signals.

In NCES this idea of two types of signals has been used, but here in a strictly causal way, i.e. as untimed signals. Hence, the semantics differs slightly. Condition signals carry information about states or substates, and event signals carry information about state transitions. Furthermore, for the case of modelling FBS the NCES definition can be restricted to binary systems.

In the following a more informal definition of structure and behaviour for one Net Condition/Event Module (abbr.: NCEM), the combination of NCEM to a NCES, and the translation of NECS to NCEM are given. A more formal definition of NCES can be found in [10].

The basis of the formal definition of a NCEM is formed by the NCE structure, describing the internal static part of the module, the input and output structure of the module, as the static interface of the module, and the marking by tokens, as the dynamical part of the module.

A NCE structure contains two nonempty and disjoint sets of places and transitions as well as three sets of arcs connecting them. The first set is the set of ordinary arcs connecting places and transitions as well as transitions and places to describe the token flow within the net as known from Petri nets. The second set is given by condition arcs connecting places and transitions to describe additional enabling conditions of transitions depending on markings of places. The third type of arcs is unusual for Petri nets. This type is formed by event arcs connecting transitions with transitions describing the joint firing of transitions under special conditions. It is supposed that the set of event arcs will not form any cycle.

A NCE structure describes the structural basis for modelling purposes but contains no facilities for modular modelling by modules and signals. The integration of the input and output signal concept of Condition/Event systems will be possible only by definition of an input and output structure. The input set to a NCE structure is formed by a set of condition inputs and a set of event inputs. Both sets are connected by condition input arcs and event input arcs to transitions only. The output set is formed by condition outputs and event outputs which are connected by condition output arcs and event output arcs with places (in the case of condition outputs) and transitions (in case of event outputs). The values of condition outputs is determined by the places of the NCE structure, and the values of the event outputs by the transitions of the NCE structure.

A NCEM is completed by the marking. A marking of an NCE structure is given by a mapping of the set {0, 1} to the set of places.

Now the NCEM structure is complete. Places and transitions, resp., are denoted by the usual symbols of Petri nets. The same applies for the flow arcs and the tokens. Condition signals are represented by arcs with a black dot instead of an arrowhead, and event signals are represented by arcs with zigzag symbols. Condition inputs and outputs are represented by small boxes whereas event inputs and outputs are represented by small diamond symbols. In the following figure a small NCEM example is given.

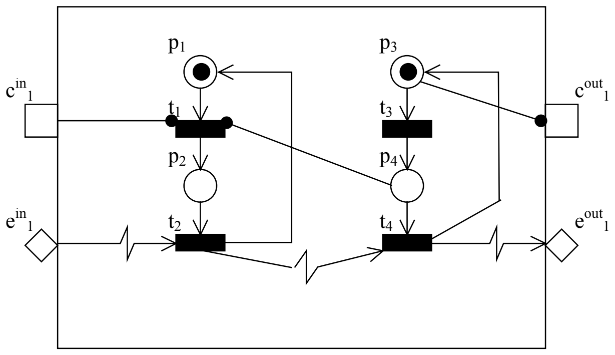


Figure 4. NCEM example

The dynamic of the system is defined by enabling of transitions and the firing rule. The two types of signals have an important influence on the enabling and firing rule. Condition input signals and condition arcs enable transitions to fire whereas event input signals and event arcs force transitions to fire. As a result, we get the following enabling and firing rules for NCES.

A transition is enabled, if its predecessor places over ordinary arcs are marked, its successor places over ordinary arcs are not marked, the predecessor places over condition arcs are marked, the condition inputs connected by connection input arcs to the transition are true, the event inputs connected by event input arcs to the transition are true and finally all predecessor transitions over event arcs are enabled. A set of transitions is enabled, if all of its transitions are enabled, all transitions have disjoint sets of predecessors places with respect to ordinary arcs, all transitions have a disjoint set of successor places with respect to ordinary arcs, and for all transitions with incoming event arcs the predecessor transitions are within the set of enabled transitions. Only a maximal set of enabled transitions is allowed to fire.

If the firing rule is applied to the example the following sets of transitions will fire and the following states are reached. Within the initial state only transition $\{t_3\}$ is enabled. After its firing the state $\{p_1,p_4\}$ is reached. Within this state $\{t_1\}$ is enabled if , and only if, c^{in}_1 is true. If $\{t_1\}$ is fired the state $\{p_2,p_4\}$ is reached. Within this state the transition set $\{t_2,t_4\}$ is enabled if, and only if, e^{in}_1 is true. Only this set will fire in this state. $\{t_2\}$ will not fire alone. If $\{t_2,t_4\}$ fires the initial state is reached again.

Now the behaviour of a NCEM has been described. A NCES will be developed out of NCEM by connecting its inputs and outputs with appropriate connection arcs as expressed in Figure 5. Therefore a NCES is defined as a

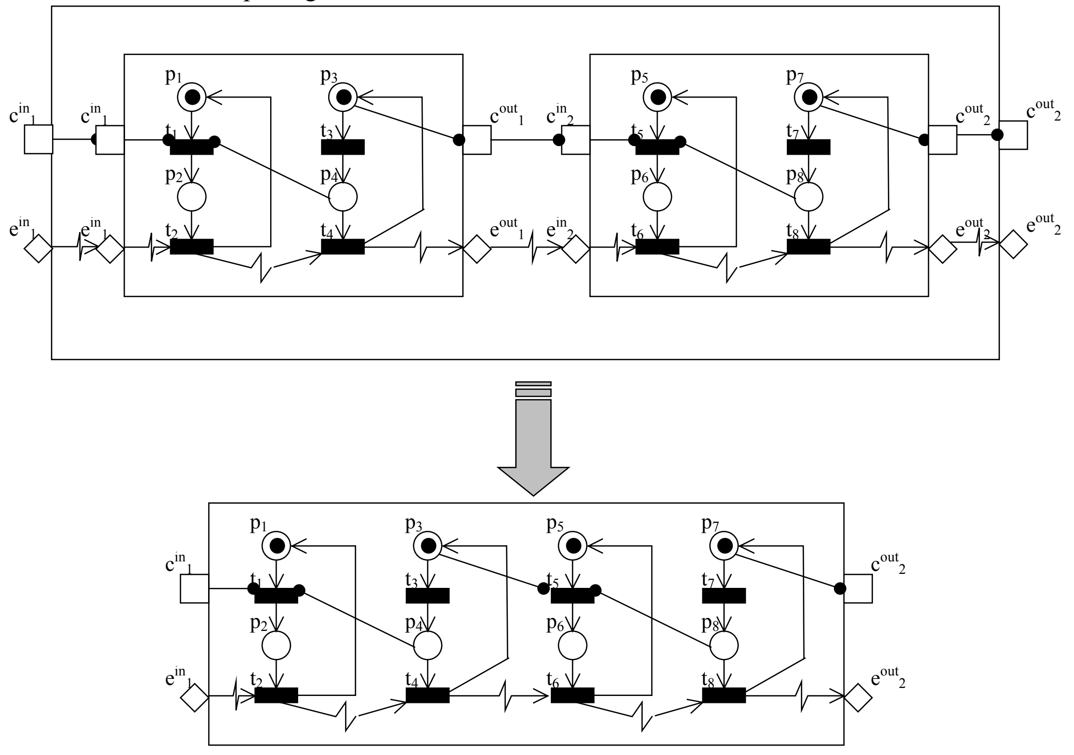


Figure 5. Example of NECS transformation to NCEM

set of NCEM, a set of overall condition inputs, a set of overall condition outputs, a set of overall event inputs, a set of overall event outputs, a set of condition connection arcs connecting local NCEM condition outputs with local NCEM condition inputs, a set of event connection arcs connecting local NCEM event outputs with local NCEM event inputs, a set of condition input connection arcs connecting global NCES condition inputs with local NCEM condition inputs, a set of event input connection arcs connecting global NCES event inputs with local NCEM event inputs, a set of condition output connection arcs connecting local NCEM condition outputs with global NCES condition outputs, and finally a set of event output connection arcs connecting local NCEM event outputs with global NCES event outputs.

A set of NCEM which has been connected to form a NCES can be combined to a NCEM by replacing each chain of arcs from a place over a condition output and a condition input to a transition be a condition arc and from a transition over an event output and an event input to a transition by an event arc. Additionally the arc chains from an global input over an local input to a place or transition will be replaced by an appropriate input arc and chains from places or transitions over local outputs to global outputs will be replaced by appropriate output arcs. This is also depicted in Figure 5.

Following this composition process each NCES can be analysed by using its appropriate NCEM.

NCES have been used in several formal ways to analyse or calculate control systems. Within a huge set of papers the way how to analyse NCEM models to verify state and/or firing sequence related properties have been examined. Here technologies as the consideration of the state space [13], unfoldings [14], and structural analysis [15] have been used. The models also have been used to automatically generate specifications and finally control code [16].

3. Transformation of FBS to NCES

The generation of Petri net based models of FBS has been considered before (for example [17]). One remaining problem is the generation of models in parallel and in consistence with the development process of FBS which is necessary to enable a fast, online simulation and validation of modelled FBS..

This can only be achieved by providing mechanisms and modelling technologies handling the different building blocks and the different entities within a FBS.

Following the aims of modelling of FBS the resulting NCES models have to be useable for evaluating the overall system behaviour and the absence or presence of special system conditions. Therefore the ECCs of all FB, its connections among each other, its relations to local variables, and its relations to the FB algorithms have to be modelled.

Following the definition of FBS the following basic modelling steps need to be provided:

- Modelling of individual states and transitions within an ECC,
- Modelling of transition guards and events initiating by transitions,
- Modelling of the execution of algorithms in relationship to the events triggered and the data local variables changed by the algorithms, and
- Modelling of connections between different FB to an FBS.

Based on this modelling steps an automatic model design during the FBS design is possible.

3.1. Modelling of ECC states and ECC transitions

A state within an ECC is characterised by the two main phases of behaviour within the state. If the state is activated the algorithms attached to this state will be executed in a given order. After the algorithm execution the state will remain until one of its successor transition will occur. To model these two phases two different places are necessary. A transition within an ECC is executed without a time consumption. Hence, transitions have to be modelled by one transition only. Following these both facts a state-transition-system of an ECC will be modelled by a system of place-transition-place and transition system as depicted in the following figure.

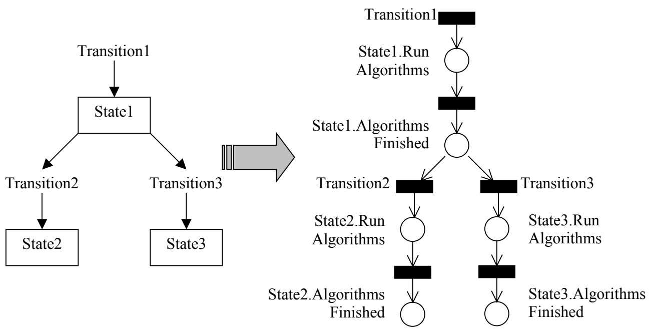


Figure 6. Modelling of ECC states and transitions

3.2. Modelling of transition execution

ECC transitions will be executed if the incoming events associated with the transition and the transition guard are true. Hence, the guard as well as the incoming events have to be modelled.

In case of the incoming event the modelling step is trivial. It will be modelled by an incoming event signal with an event input and incoming event arcs.

The guard consisting of logical conditions related to internal variables and external data signals will be modelled by translating incoming data signals to condition inputs and condition input arcs and internal variables to condition arcs from places modelling the internal variable. The binary character of this signals will not be a problem since the fact, that a variable will fulfil a guard condition will be modelled by an individual place only marked if the guard condition is given. The translation of the execution rules is depicted in Figure 7.

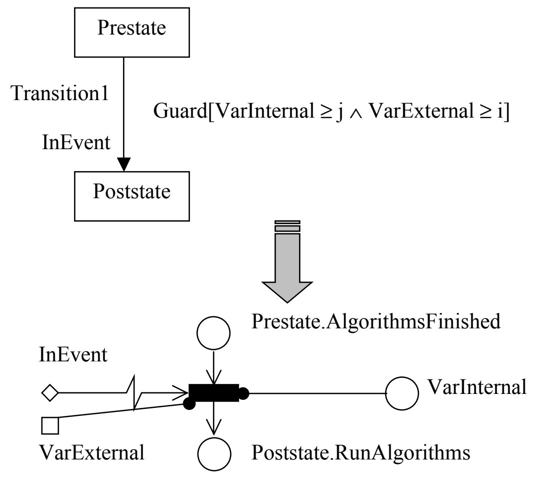


Figure 7. Modelling of ECC transition execution

3.3. Modelling of algorithm execution

Algorithms will be executed, if the state they are attached to, is active. They will be executed in a fix sequence and if all algorithms are finished the events attached to the algorithms will be fired.

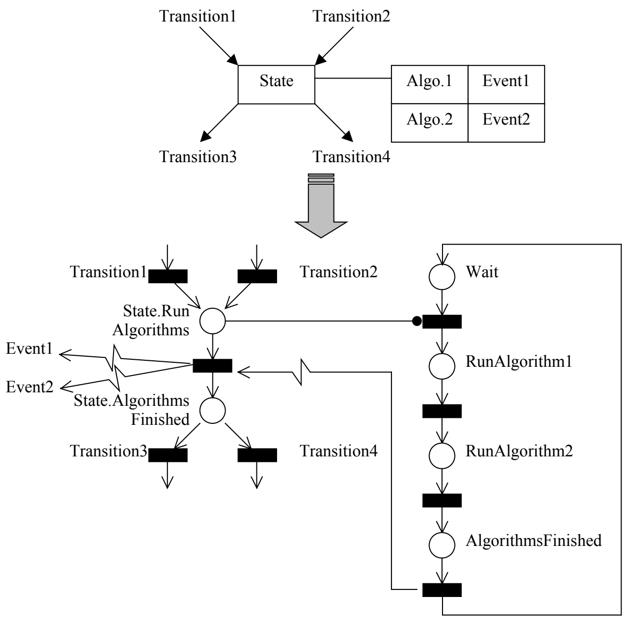


Figure 8. Modelling of algorithm execution

To model this behaviour a direct link from the activity of the algorithms to the activity of a state has to be developed. This will be reached by modelling a condition arc from the place indicating the execution of the state attached algorithms to the transition indicating the start of the algorithm execution. Additionally the sequence of the algorithms has to be modelled. This will be done by a cycle of places and transitions with one place indicating the waiting for the start of the algorithm execution, a set of places indicating the execution of the individual algorithms, and one place indicating the finalisation of all algorithms. The transition taking a token from the place indicating the finalisation of all algorithms and placing it on the place indicating the waiting for the start of the algorithm will be connected to the transition demarking the place indicating the execution of all algorithms. All invoked events will be modelled by event arcs outgoing from the transition indicating the finalisation of all algorithms. This translation of event execution is depicted in the Figure 8.

During the execution of algorithms the internal variables, which can be part of the transition guards, may be changed. This can be modelled by connecting the places modelling the internal variables with transitions whose enabling depends on the activity of an algorithm execution indicating place. This is depicted in the following figure.

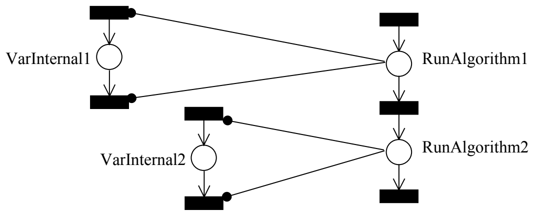


Figure 9. Modelling of dependencies of internal variables on algorithm execution

The outgoing data connections from the internal data can be modelled by using data outputs and outgoing condition arcs from the individual places modelling the internal data.

3.4. Modelling of connections between FB

The modelling of connections by event / data signals between FB is straight forward. Since the nature of event connections is equal to the nature of event connection arcs and the nature of data signals is equal to condition connection arcs they will be modelled in that way.

3.5. FB translation example

If we consider the example of the pusher, the function block controlling the pusher the NCEM model of the FB is given in Figure 10.

4. Integration of the automatic model generation in the FBSD tool

As mentioned in the introduction different tools for developing FBS based control systems are under development. One of these tools is the Function Block System Designer (FBSD) which is based on the results of the TORERO project [18] and has been implemented using the ECLIPSE framework [19].

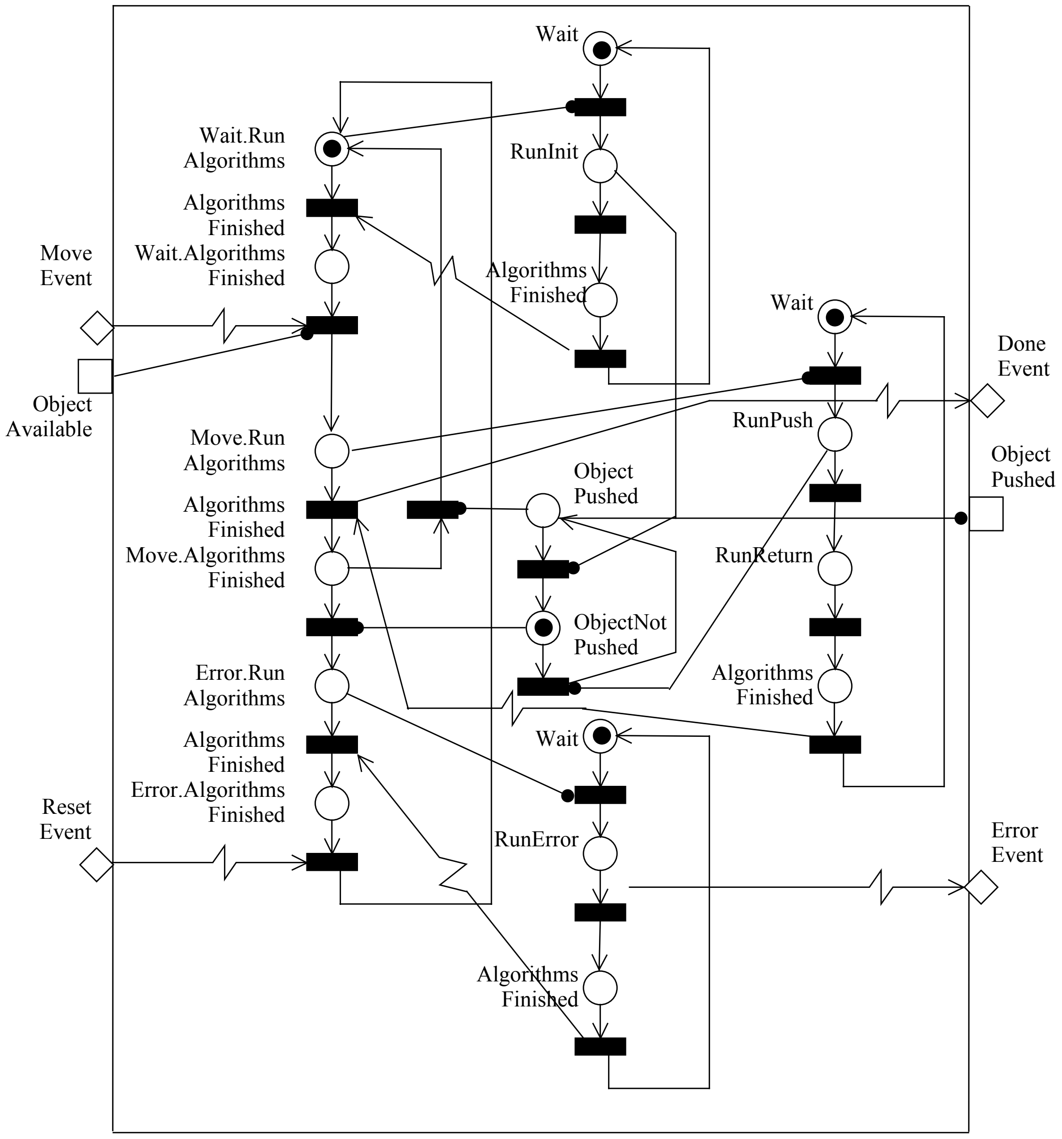
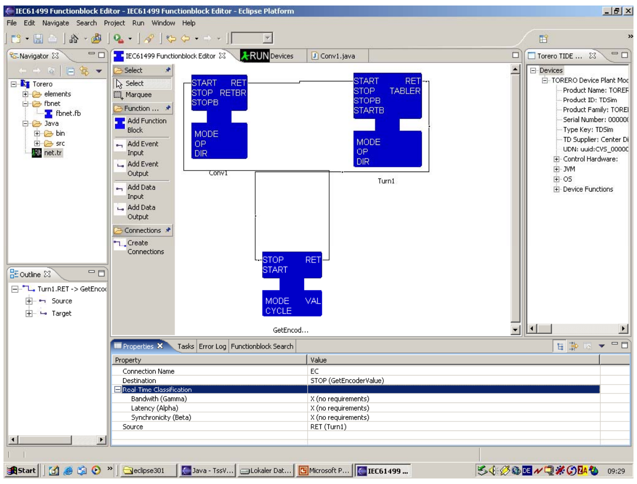


Figure 10. NCEM model of the pusher controlling function

Within the TORERO approach a control system is implemented using pre-implemented and customized IEC 61499 function blocks. Therefore a function block editor as well as a system of additional ECLIPSE plug-ins handling the automatic control code design, distribution, and deployment have been developed. The FBSD works on an integrated data model for all parts of the control application design process including FBS model data and control code. The automatic modelling of FB by NCES models will be also integrated within this data model. Thereby, the FB data set will be extended by a data set containing the relevant NCEM information as well as the interconnections to other FB describing NCEM. Using this data model in the near future at first a NCES model based simulation and later on a NCEM based analysis plug-in for the FBSD will be implemented. An screenshot of the current version of the FBSD is given in Figure 11.



Property	Value
Connection Name	EC
Destination	STOP (GetEncoderValue)
Real-Time Classification	
Bandwith (Gamma)	X (no requirements)
Latency (Alpha)	X (no requirements)
Synchronicity (Beta)	X (no requirements)
Source	RET (Turn1)

Figure 11: FBSD user interface

5. Outlook

Within this paper a methodology for automatic modelling of IEC 61499 compliant function block systems by Net-Condition/Event-Systems has been presented. Using this models it will be possible to formally validate and/or verify the behaviour of function block systems to ensure its correctness within control systems.

The described methodology will be integrated in the near future within the Function Block System Developer tool developed at Center Verteilte Systeme at Otto-von-Guericke university Magdeburg, Germany.

Within further maybe joint research activities like the EC funded research project PABADIS'PROMISE and the German national research project AgentAut the developed methodology will be applied to analyse the behaviour of agent and function block based distributed control systems.

More information on the FBSD and the mentioned projects can be found under [20].

Acknowledgement

The work presented within this paper was partially supported by the European Commission under the 5th Framework Program within the TORERO project (IST 2001-37573).

References

[1] IEC: International Standard IEC 61499 - Function Blocks - Part 1 Architecture, www.iec.ch.

[2] Siemens AG: Distributed intelligence with PROFINET, Technical documentation, http://www2.automation.siemens.com/profinet/html_76/produkte/imap.htm, 2005.

[3] Schneider Electric: Transparent Factory – internat based technologies for the factory floor, Technical documentation, http://www.schneider-electric.com.au/Products/Automation/TF_RealTime/WhatIs.htm, 2005.

[4] J. Christensen (Rockwell Automation): Function Block Development Kit (FBDK) – Download and User Manual, http://www.holobloc.com/doc/fbdk/gettingstarted.htm, 2005.

[5] K. Thramboulidis: Development of distributed industrial control applications – The CORFU framework, IEEE Int. Workshop on Factory communication systems, Vasteras, Schweden, 2002, Proceedings, pp. 39-46.

[6] K. Thramboulidis: Model-integrated mechatronics – towards a new paradigm in the development of manufacturing systems, IEEE Transactions on Industrial Informatics, Vol. 1, No. 1, 2005, pp. 54-61.

[7] Kalogeras, A. P., Prayati, A., Schwab, C., Tangermann, M., Ferrarini, L., Papadopou-los, G.: Integrated Web Enabled Control System Design Methodology. IEEE International Workshop on Factory Communication Systems – WFCS, Vienna, Austria, 2004, Proceedings.

[8] Vyatkin V., Hanisch H.-M.: Component design and validation of decentralized reconfigurable control systems with IEC61499, International Symposium on Advanced Control of Industrial Processes, June 2002, Kumamoto, Japan, Proceedings, pp. 215-220.

[9] M. Rausch, H.-M. Hanisch Netz-Condition/Event-Systeme, 4. Fachtagung Entwurf komplexer Automatisierungssysteme, Braunschweig, Juni 1995, Proceedings, pp. 55-71.

[10] H.-M. Hanisch, A. Lüder: A Signal Extension for Petri Nets and its Use in Controller Design, Fundamenta Informaticae 41, IOS Press, 2000, pp. 415-431.

[11] A. Lüder, H.-M. Hanisch: Supervisoury synthesis for petri nets and partial order specification, Klüver Academic Publisher, Journal on Discrete Event Dynamic Systems, 2001.

[12] R.S. Sreenivas, B.H. Krogh: On condition/event systems with discrete state realizations, Discrete Event Dynamic Systems-Theory and Application, 1991, 2(1), pp. 209-236.

[13] H.-M. Hanisch, A. Lüder, M. Rausch: Controller Synthesis for Net-Condition/Event-Systems with Incomplete State Observation, Computer Integrated Manufacturing and Automation Technologie (CIMAT 96), Grenoble, France, May 1996, Proceedings, pp. 351-356.

[14] A. Lüder, H.-M. Hanisch: Synthesis of admissible behavior of Petri nets for partially ordered specifications, Workshop on Discrete Event Dynamic Systems (WODES2000), Gent, Belgium, Aug. 2000, Proceedings, Kluver Accademic Publisher, ISBM 0-7923-7897-0, pp. 409-420.

[15] D. Schwanke, A. Lüder, H.-M. Hanisch: Dynamic Behavior and the Deadlock-Trap Property of Signal/Event Nets, Workshop on Concurrency Specifications and Programming 1998, Berlin, Germany, 1998, Proceedings, pp. 215-220.

[16] A. Lüder: Formaler Steuerungsentwurf mit modularen diskreten Verhaltensmodellen, PhD-thesis, Dept. of Engineering Science, Martin-Luther-University of Halle-Wittenberg, 2000, Logos Verlag, Berlin, 2000.

[17] Vyatkin V., Hanisch H.-M: Verification of Distributed Control Systems in Intelligent Manufacturing, Journal of Intelligent Manufacturing, special issue on Internet Based Modeling in Intelligent Manufacturing, vol.14, N.1, 2003.

[18] TORERO project consortium: TORERO project homepage, www.uni-magdeburg.de/iaf/cvs/torero, 2005.

[19] J. Arthorne, C. Laffra, *Official Eclipse 3.0 FAQ,* Addison-Wesley Professional, 2004.

[20] Center Verteilte Systeme: Team homepage, www.uni-magdeburg.de/iaf/cvs, 2005.