TRANSFORMATION OF IEC 61499 CONTROL SYSTEMS TO FORMAL MODELS

I. Ivanova-Vasileva, C. Gerber, H.-M. Hanisch

Chair of Automation Technology, Institute of Computer Science, Martin Luther University of Halle-Wittenberg, Karth-Mothes-Str. 1, 06120 Halle/Saale, Germany, tel: +49 345 55 25974, email: ioanna.ivanova@informatik.uni-halle.de, christian.gerber@informatik.uni-halle.de, hans-michael.hanisch@informatik.uni-halle.de

Abstract: This paper is focused on transforming a real case IEC 61499 control system to a formal Net Condition/Event System (NCES). Thereby, rules are defined and implemented as XSLT scripts and SWI-Prolog rules, which allow an automatic generation of formal NCES modules from the given function blocks. These function blocks incorporate data processing by means of algorithms together with a model of execution control. The goal of the study is to verify the correct behaviour of the IEC 61499 controller. This is realized within a closed-loop model that comprises also a model of the plant.

Keywords: IEC 61499, Net Condition/Event System, NCES, transformation rules

INTRODUCTION

Modelling and verification of distributed controllers following the IEC 61499 International Standard has become a research issue since the early publications about verification of their execution control [1,2,3]. This contribution does not focus as exactly as the preceding work on the execution control but extends the view and presents models that also incorporate data processing by means of algorithms together with a simplified model of execution control. The goal is to verify the correct behaviour of the controller. Although it is often neglected, correct behaviour can only be verified in a closed-loop model that incorporates also a model of the plant. The contribution therefore describes the modelling of such closed-loop behaviour.

It is structured as follows. Section 2 describes briefly the testbed that is used for the work. Section 3 defines transformation rules to automatically generate formal models from function blocks following IEC 61499. The approach for formal analysis and verification is presented in Section 4. Section 5 concludes the work and provides some directions for future studies.

SUBJECT OF THE STUDY

The station Distribution of the modular production system from FESTO shown at figure 1 is used as a testbed for the study.

The Distribution station shown at figure 2 is a simple automation system that includes two intelligent mechatronic actors, namely an automated storage of workpieces and a swivel passing the workpieces to other processing units. As seen in figure 3, each of the mechatronic actors is complex and could be built up from simpler ones. This means that the automated storage consists of the structural units magazine and pusher.

The pusher shifts a workpiece forward from the lower position of the magazine to the position from where the workpiece can be picked up by the swivel. After the pusher returns to its initial position, the pile of workpieces in the magazine falls down. Thus, the lower place of the magazine becomes occupied again.

Before the pusher moves forward, the swivel has to move away to the following station and will return if the workpiece is pushed to the front position. Then the vacuum is turned on, the workpiece is picked up and is transferred to the following station. Afterwards, the swivel returns to its start position.

According to [4] the basic idea of object-oriented engineering in automation is that the structure of mechanical units should be followed by the structure of the software components.
This leads to the definition of automation objects and intelligent actors, which have some preprogrammed functionality. To represent the hierarchical structure of all automation objects, class diagrams of the informal modelling language UML as shown in figure 3 should be used. Each complex automation object is composed from other complex or basic automation objects. The basic automation objects are those located at the lowest level. For example, a pneumatic cylinder or a sensor, which does not include any other automation objects, is a basic automation object.

![Diagram of the sample automation system](image)

**Figure 3: Structural description of the sample automation system**

Based on the described hierarchy, the IEC 61499 function block model of the Distribution station is developed. It consists of the basic function block *Distribution_Control_FB* and the composite function block *Distribute_FB*. According to [5,6] the basic function block represents a master control of the station, which starts certain preprogrammed actions of automation objects and waits until the completion. The composite function block *Distribute_FB* includes all preprogrammed automation objects, which get a certain event that starts an action. The completion of each action is indicated by the output events *MAG_CTRL_READY* and *TRANSF_CTRL_READY*.

**TRANSFORMATION OF THE IEC 61499 SYSTEM**

This paper is intended to show the formal verification of the function block model by means of transforming it to a formal model. As formal modelling language we use the known Net Condition/Event Systems [7]. They provide means for a modular and graphical modelling in a way that is intuitively understandable by any engineer. The basic patterns are so called NCES modules that encapsulate some kind of causal behaviour. These NCES modules have an interface, which provides information about internal state changes and internal states. State changes in one NCES module can trigger state changes in other NCES modules, if they are connected by event arcs. Beside this, the internal state of one NCES module can enable a state change in another NCES module if they are connected by condition arcs.

If we compare Net Condition/Event Systems with the IEC 61499 specification, we have the following similarities:

- Causal behaviour is encapsulated in function blocks or NCES modules and is hidden to their environment.
- Function blocks or NCES modules, respectively, have a signal interface that clearly distinguishes between events and data (events and conditions).
- The structure of the system is represented by the structure of the function blocks or NCES modules.
- The causal behaviour of the control system is represented by the behaviour of the interconnected function blocks and process as well as by the closed-loop NCES model.

It is therefore very natural to map IEC 61499 function blocks to formal NCES modules and to interconnect them exactly in the same way as the function blocks are interconnected at the application or resource model [8]. To interconnect the once translated function blocks the tool TNES Editor is developed at the Chair of Automation Technology at the University of Halle-Wittenberg. By the means of the tool, the translated function blocks can be imported as NCES modules and graphically connected. To store the created formal NCES model, the TNES Editor uses the open XML-based TNES Markup Language.

Based on the previous work [1,2,3,9,10], the following transformation rules are proposed, with the constraints that each event in- or output of a function block is associated with all data in- or outputs. Furthermore, at the current state, there are only Boolean variables used in the data flow.

**Transformation rule 1 – Graphical representation:** The graphical representation of the NCES modules is similar to the one known from IEC 61499 function blocks as it is shown at figure 4.

The function blocks consist of head and body. The head is connected to the event flow via event inputs and outputs and the body to the data flow via data inputs and outputs as shown at figure 4.

![Graphical representation of function blocks and NCES modules](image)

**Figure 4: Graphical representation of function blocks and NCES modules**

**Transformation rule 2.1 – Event In- and Outputs:** The event in- and outputs of each function block are transformed to event in- and outputs of a NCES module. Only at the basic function blocks the event output has to be extended by a net structure to merge several events. This is built up by a marked place with the capacity 2 and a transition with the event mode *OR* connected with a pre and a post flow arc to the place and an event arc to the event output. Both are named *merge*, and the symbol *represents the name of the event output.

**Transformation rule 2.2 – Condition In- and Outputs:** All data in- and outputs of each function block are translated to condition in- and outputs of a NCES module. At basic function blocks, the condition output is extended by a net structure representing a Boolean value and a condition arc from the True place to the condition output.

**Transformation rule 2.3 – Internal Boolean Variables:** Boolean variables can have the value True or False and are therefore represented by two places with names *True* and *False*, two transitions with the event type *OR* and flow arcs, connecting them. The transitions have to be named *TrueFalse* and *FalseTrue*. The event type *OR* is necessary, because one or more algorithms can change the value of the Boolean variable. If there is an initial value defined, the place representing it has to be marked with one token, otherwise the place *False* is initially marked. The symbol *represents the name of the Boolean variable.
Transformation rule 3.1 – ECStates: For each ECState it has to be checked if it is the initial state and if there is an ECAction associated or not. The initial ECState is always represented by a place with the same name as the ECState and a marking of one token at it. If there are no ECActions associated with the ECState, it can be represented by a combination of 2 places, 1 transition and 2 flow arcs. The places are named * Run and * Finished and are connected by flow arcs with a transition named * Algs_complete. The symbol * represents the name of the ECState. If there are ECActions associated with the ECState, it is also represented by the places * Run and * Finished, but between both places are a place and a transition inserted that represents the ECActions. As shown at figure 5, the ECState CTRLS_READY with no ECAction is translated to the place CTRLS_READY_Run and CTRLS_READY_Finished and a transition with two flow arcs, connecting both places.

Transformation rule 3.2 – ECActions: The ECAction consists of two parts – an algorithm and an output event, which is generated if all algorithms have completed their execution. This means if one or more ECActions are associated to one ECState, a transition and a place named * sched_Algs are inserted. The transition is connected to the place * RUN, and the place is connected to the transition * Algs_complete. Furthermore, an event arc to the Run transition of each algorithm of the associated ECActions has to be inserted with the event source transition * sched_Algs. Also a condition arc from the _Wait place of each event connected algorithm to the transition * Algs_complete, has to be inserted. If there are output events defined, an event arc has to connect the transition * Algs_complete and the transition _merge of the event output net structure.

Adopted to the example of figure 5, the ECState MOV2E1_START with one ECAction is translated to the places MOV2E1_START_Run, MOV2E1_START_Finished and MOV2E1_START_sched_Algs and the two transitions MOV2E1_START_sched_Algs and MOV2E1_START_Algs_complete. The transition MOV2E1_START_SCHED_Algs has an event arc to the transition MOV2E1_Run. The transition MOV2E1_START_Algs_complete gets a condition signal from the place MOV2E1_Wait and sends an event signal to the transition REQ MOV2E1_merge.

Transformation rule 3.3 – ECTransitions: As shown at figure 6, between the ECStates there are ECTransitions, which allow the switch from a prestate to a poststate, if the associated condition gets true. The ECTransition is always transformed to a transition that is 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.

Figure 6: ECTransition (Switch between the Pre- and Poststate)

At the condition part of the ECTransition it has to be checked if the first value represents an event input, a data input or an internal variable. If it is an event input, an event arc from the event input of the NCES module to the translated transition has to be inserted. The remaining part of the condition represents a Boolean equation of data inputs and internal variables, which can be modelled with condition and inhibitor arcs.

Transformation rule 4 – Algorithm: The structure of a

Figure 5: Transformation of the ECStates into NCES places and transitions
translated algorithm can be divided into a general and a conditional part. The conditional part depends on the content or the invariant of the algorithm.

**Transformation rule 4.1 – General Part:** The general part consists of the places *Wait, *Run and *Finished and 3 transitions, connecting the places *Wait → *Run, X → *Finished and X → *Wait. Thus, every algorithm is a closed-loop net segment with an initial marking at the place *Wait.

**Transformation rule 4.2 – Conditional Part:** The conditional part is placed between the place *Run and the transition before the place *FINISHED (Figure 7) of the general algorithm structure. Each line of the algorithm invariant is represented by a spontaneous transition and a place. To set a Boolean output or internal variable to a special value, an event arc has to connect the transition of this invariant line with the transition VarName_TrueFalse to set it to FALSE or to the transition VarName_FalseTrue to set it to TRUE. To negate the Boolean variable, one event arc has to connect the transition VarName_TrueFalse and another one to VarName_FalseTrue.

In our example the conditional part represents an algorithm invariant with two lines, which set the Boolean output variable swl_to_te to TRUE and swl_to_ma to FALSE. This conditional part is presented here as Structured Text, but it is also possible to calculate the invariant of algorithms written in Instruction List, Ladder Logic as well as Java, Delphi or C++.

**Transformation rule 5 – Hierarchy:** The formal NCES model has the same hierarchy as the one of the function block model as well as the hierarchical structure of the automation objects (Figure 8).

**Transformation rule 6 – Function Block Network:** Using the transformation rules 1,2 and 5 gives us the capability to transform the interface and the hierarchy of composite function blocks. Thus, we only have to define a rule for translating the Function Block Network.

A composite function block consists of several basic or composite function blocks, which have to be translated first and inserted into the NCES module representing the composite function block. Each event arc inside the composite function blocks is translated to an event arc in the NCES module, with same source and end point. The data arcs are handled the same way, but they are converted to condition arcs.

**Figure 7: Structure of the Algorithm**

As a result of this rule and rules 1,2 and 5, one achieves for the function block network of the composite function block, shown at figure 9, the NCES modules shown below.

**Transformation rule 7 – Resource Model:** The IEC 61499 resource model consists of a function block network with the function block START of the type E, RESTART. This special and implementation dependent function block has to be modelled once manually as a NCES module. Inside the NCES module are a marked and an unmarked place, connected through a transition. Further, an event arc connects the transition and the event output Cold.

With this NCES module and the presented rules, it is possible to transform a whole IEC 61499 resource and receive the causal control behaviour (Figure 10).

**Figure 8: Hierarchy of the FBs and the NCES models**

**Figure 9: Composite FB transformation**

**Figure 10: Transformation of the resource model**

ANALYSIS

Before verifying the whole control system in closed loop, the already transformed formal NCES model of the causal control behaviour has to be connected to the plant model by condition arcs.
For this purpose a model of the causal plant behaviour and the different workpiece states has to be created. Every plant module of the Distribution station is modelled according to its causal physical behaviour. Exemplarily shown at figure 11 is the model of the pusher module, which consists of the NCES modules drive, sensor_s and sensor_e.

Figure 11: Model of the plant – Pusher

The workpiece is modelled separately for each plant module and respectively the workpiece position at every moment. This results in two NCES models of the workpiece behaviour – Workpiece_in_Magazine and Workpiece_in_Transfer, according to the automation object interacting with the workpiece. Figure 13 shows the model of the workpiece at the plant module transfer.

For reachability calculations of the created NCES system, an expert system implemented in SWI-Prolog or the tool SESAl [12] can be used. To verify the NCES system with computational tree logic, at the current state only the SESAl tool can be used. But before running SESAl all modelled inhibitor arcs have to be replaced by a special net construct described in [13]. The result of the reachability calculation is the reachability graph presented at figure 13 with 521 states. This means that the closed-loop model of the plant and the control can reach 521 defined states, which represents the causal behaviour of the controlled system.

The layout of the reachability graph is done with the tool dotty [14] and drawing zdraw [15], which allows at the same time a bird view and the zooming to special graph details. The graph detail a) shows the event splitting at the initialization to the several subcontrollers of the magazine, the pusher and the transfer control. Afterwards, the graph spreads to a lot of different states, which represents the different progresses of the three concurrently executed initialization algorithms. If all algorithms got ready and the modelled start button becomes true, shown at the graph detail b), the programmed action sequence runs. Because the start button can be pressed at every time and is also modelled in NCES in that way, every reachable state is duplicated with a pressed and released start button. Thus, modelling of the whole panel with a reset and stop button and a switch to allow the communication will lead to an even huger reachability graph without gaining any more causal information of the system. Also the parallel execution of algorithms and the splitting of control events spread the reachability graph. This will make it impossible or needs a lot of experience of the user to verify the calculated reachability graph by his visualization. Thus, only an automatic verification with a formal description of the specified plant behaviour will be practical. The result will be TRUE or FALSE, and if it is FALSE, the transformed control has to be changed until the result of the verification becomes TRUE. For formal specification the computational tree logic and its extension can be used.

Figure 12: Reachability graph of the NCES model
a) Parallel execution  b) Sequential execution
CONCLUSION AND FUTURE WORK

In this paper a real case study of the transformation of IEC 61499 control systems to formal models using Net Condition/Event Systems is presented. There are transformation rules proposed, which have been proven at the closed-loop model of the Distribution station of the multi production system of FESTO. The implementation of the rules is done in a first sketch without any graphical representation as XSLT scripts, which transform files with the defined markup language of the IEC 61499-2 into XML files with TCNES markup language.

These transformation rules give us a great number of research capabilities in the future. The very next step in our work is the further development of the transformation of basic function blocks with event inputs and outputs, connected with several data inputs and outputs. Also further development of the XML Transformation has to be done with a graphical representation of the NCES modules for the TCNES Editor by implementing them as SWI-Prolog rules.

As following steps of the work, service interface function blocks (SIFBs) should be inserted into the model and for more generality of the rules, they should be extended to a binary coding of INTEGER and REAL variables. This will be done by representing them as 2^0, 2^1, 2^2... 2^n condition signals. This will make it possible to adopt the transformation of function blocks to other plants and it gives us the capability to transform the whole FBDK library into NCES modules.

ACKNOWLEDGEMENTS

This work was supported partially by the cooperative project VAIAS funded by the German Ministry for Education and Research (BMBF) and by the Deutsche Forschungsgemeinschaft under reference Ha 1886/16-1.

REFERENCES