Résumé : Ce papier présente une méthodologie de modélisation du contrôle des blocs fonctionnels définis par la norme CEI 61499 à l’aide d’automates temporisés. L’objectif est d’associer à chaque bloc fonctionnel un ensemble simple d’automates et de composer ces automates de façon similaire à la composition des blocs fonctionnels dans une application afin d’en vérifier la bonne exécution. Une application de commande répartie est utilisée pour illustrer la démarche et des exemples de vérification qui peuvent être effectuées.

Mots Clés : blocs fonctionnels, CEI 61499, vérification, automates temporisés

Abstract: This paper presents a methodology for the modelling of IEC 61499 function blocks behaviour using timed automata. The aim is to be able to associate basic timed automata with each functional block and to compose them in order to verify the execution of a network of functional blocks. The method and the verification that can be made are exemplified on a distributed control application.

Keywords: functional blocks, IEC 61499, verification, timed automata.

1. INTRODUCTION

The IEC 61499 (IEC TC65/WG6, 2003) standard specifies a model of applications and re-usable software components for the design of distributed industrial process control applications. This model is based on function blocks that are widely used in this domain and is then based on a dataflow approach. However, in order to further the standardisation of the function blocks and their reuse on various platforms the execution control flow is made explicit. For each block the control of the execution of the algorithms, according to input control events and the state of the block, is specified by a state machine. The global application is specified by connecting the data inputs and outputs of the blocks in order to define the data flows but also their control inputs and outputs for the control flows. It can be noticed that the standard is mainly concerned with the control part and that the specification of the algorithms is out of its scope.

When connecting such components it is difficult to determine the global behaviour of the execution and it is useful to get a model on which formal verifications can be made in order to guarantee the properties of the execution. In this paper an approach that used timed automata (Alur and Dill, 1990) and the UPPAAL tool (Larsen, et al., 1998) to provide such verifications is presented.

The IEC 61499 standard is briefly introduced in section 2 and some related work on verification of function blocks applications is presented in section 3. The IEC 61499 structure of an example of control application is given in section 4 to complete this presentation of the standard. The proposed formalisation of the behaviour with timed automata is then presented in section 5 and its application to the example of section 4 is described in section 6 where some examples of properties that can be verified are also presented.

2. IEC 61499 SPECIFICATION.

The specification of an application consistent with the IEC 61499 standard is based on two distinct steps. First, each component (function block) is specified and then the global application is build by connecting the blocks.
A block is specified, on the one hand by the algorithms part that specifies how the input data are used to generate the output data. This part is only considered from an external point of view in the standard that only includes the input and output data and algorithms declarations. On the other hand the block is also specified by its control part that defines how and when these algorithms are executed according to control input events.

This control is specified by a state machine called execution control chart (ECC). Each state of the ECC (called an EC state) can be associated with zero or more EC actions. Each EC action is associated with zero or one algorithm and zero or one output control event. The EC transitions are associated with zero or one input control event and zero or one Boolean condition on data. It can be cleared when its associated event occurs, its condition is true and its source state is active. When a transition is cleared its target EC state is activated, the algorithms associated with its EC actions are then executed, the output data value are changed and the output events are set to true. The standard specifies that this is a run to completion reaction; it means that if a transition can be cleared from the new active state, it is cleared when the algorithms execution is finished and the algorithms associated with the new active state are executed. The reaction stops when no more transition can be cleared, all the output events that where set during the reaction are then issued outside the block.

When an input control event occurs the input data are sampled and the reaction of the block is launched. All reactions are event driven, that is, if the previous reaction was stopped because the condition associated with a transition without event was false, and the input data have changed to values such that this condition becomes true, the block will not react until a new input control event occurs. If an input event occurs when the block is busy in a reaction the event is postponed and a new reaction is launched as soon as the current one finishes.

The application is then build with these blocks by connecting the data and the control events. Each output data may be connected to at most one input data and each control output event may be connected to at most one input control event. When the value of an output data is changed the value of the connected input data is immediately changed and when an output event is issued the connected input event immediately occurs.

The execution control of the application is then propagated through the blocks by the event connections. As the standard aims at dealing with distributed applications some specific interface blocks that propagate the data and control flows over networks are specified. Finally other specific blocks are specified to generate control events.

3. RELATED WORK.

A lot of work has been done in the field of verification of control and PLC applications (see e.g. (De Smet et al., 2001) or (Mader and Wupper, 2000)). Generally this work considers the problem of the verification of programs specified in one of the IEC 61131-3 languages and their consistency with the behaviour of the plant. The work presented here is concerned with the formalization of the execution control and this problem of validation of the application is not considered as it can be seen below. This part is, then, more specifically devoted to work on IEC 61499 standard.

Vyatkin and Hanisch (2000) have proposed a formalisation based on Net/Condition Event Systems, which are an extension of Petri nets. This extension introduces the concept of module: a Petri net specifies the behaviour of a module that can be composed with others modules by condition and event outputs and inputs. The condition signals correspond to places that can be tested and the event signals corresponds to transitions that can be used to synchronize other transitions. The verification of models expressed with this formalism can be performed with the tools SESA and VEDA. In order to model IEC 61499 semantic, modules modelling each input and output events, an invocation manager, an execution manager and the ECC are introduced. As the structure of the model is very closed to the one of the block and of its execution and because of the composition mechanism this lead to rather complicated models.

Wurmus (2002) also presents an approach based on a variant of Petri nets (CNET) that is less complete and based on a practical implementation. The verification is then oriented to the synchronisation of the blocks and the absence of deadlocks.

Faure et al. (2002) propose a model based on the synchronous dataflow language Signal and its associated verification tool (Leguernic et al., 1991) to verify IEC 61499. This approach that uses the notion of clock and constraints on clocks of Signal to synchronize the input and the ECC takes into account the sequential aspects of the execution but not its timed aspects that are not considered.

4. THE EXAMPLE APPLICATION.

The example considered to complete this presentation and later to exemplify the methodology of formal modelling and verification is based on the control of the flow and temperature of an air stream. The general structure of the model of the plant and the controller is given figure 1.

With this structure of the controller it is possible to use a distributed application: the control of the flow (controller C1) is implanted in one resource and the control of the temperature (controllers C2 and C3) in another one. A communication link is used to implement the data flow between C1 and C3.
The IEC 61499 specification of the control of the flow and the temperature are shown figure 2 and figure 3, where the dotted lines represent the data flows and continuous lines the control event flows. The block C1, C2 and C3 are the implementation of the corresponding controllers of figure 1. The blocks RC1(2) and WC1(2) are interface blocks that perform the reading and the writing of plant variables by means of the AD/DA board. The blocks Requester and Responder are a couple of distribution blocks defined by the standard to perform the propagation of the data and events flows between two resources. The Split block is used to split the event issued by the bloc C1 to input events of the blocks WC1 and Requester in order to launch their simultaneous execution. The blocks Sampler1(2) are timer blocks that periodically generate an event that is the source of a reaction of the application. The functions of the Filter and Adder blocks are clearly given by their name. It can be noticed that in this specification the execution of the Adder and consequently of the WC2 blocks is launched either by the executions of the C2 or C3 blocks that are asynchronous. Finally the Verif1(2) blocks are used to insure that a new execution of C1(2) is not launched before the value issued by the previous one is written on the output of the system.

5. FORMAL MODELING OF IEC 61499 COMPONENTS.

This formal modelling is based on IEC TC65/WG6 (2000) and is consistent with this document that first specified the behaviour of blocks and applications. Its main aim is to define a modular translation scheme in order that a formal model may be associated with each functional block and that the global model of an application may be mainly built by composing these basic models. An other important aim is to define a model that is fully consistent with the external behaviour of the block but not necessary to its internal structure. For example, as the specification of the algorithms are not in the scope of the standard, they are not taken into account in the modelling and consequently neither are the data that are abstracted. This abstraction means firstly that algorithms will be modelled by their processing time and secondly that a different reaction will be considered for each value (true or false) of conditions on the data. Of course, this abstraction introduces some spurious reactions but most verification results are still worthwhile.

5.1. The function block.

The basic component of the specification is the function block (FB) and of course as far as control is concerned the main part is the ECC. However it is not sufficient to consider this ECC, especially because modularity is emphasized and each FB will be associated with a set of automata: one for the ECC, one for each input event and two additional ones for synchronization.

The input events. When an input occurs a reaction of the ECC is requested but if the ECC is busy the event has to be memorised and may be used later. The model that is associated with each input event of a block is then given figure 4. This automaton has 2 main locations and two main synchronisation channels. The channel External_event is used to synchronize with the outside of the block when the event occurs and the channel Internal_event is used to synchronize with the ECC reaction (see figure 6) when the event is taken into account. The channel ECCInvocationReq is used to request the reaction of the block (see figure 7). The location requestingECC is then a committed location that is introduced in

\[^1\] In UPPAAL a committed location is a location that can not remain active as time elapses.
UPPAAL in order to synchronize the External_event and the ECCInvocationReq. The channel notEvent is used by the model of the ECC to determine the reaction that is to be launched when an invocation is requested and the clearEvent channel is used to reset the events that where not used when this invocation is processed. The location action is then active when an external event has occurred and no reaction has been launched since then.

All the channels are parameters of the model and have to be specified when creating an instance for each input event. Most of them are specified at the FB level and the External_Event is specified at the application level.

The ECC. The automaton that models the ECC is the core of the modelling as it is synchronized with all the others. As the ECC is a state machine its modelling may seem quite straightforward, however, as the semantic is based on a run to completion view, the basic element is not really the EC state but the reaction to an invocation. Such a reaction consists of a sequence of deactivations and activations of EC state, algorithm executions, outputs event setting leading to a stable state that is a state that will remain active until a new input event occurs. The output events are all issued at the end of the reaction. This concept of reaction has to be introduced because some transitions are not associated to any event and is very similar to transient evolutions in Grafcet (Guéguen, and Bouteille, 2000; Marcé, and Le Parc, 1992).

The modelling of the ECC is based on three types of elements or patterns. First the waiting locations and the invocation synchronization, second the reaction selection part and third the reaction modelling.

The source of a reaction is an EC state that can be stable. Generally these states are the source of at least one transition with an associated event. In the first step, a location is then associated to such EC state. For example if the ECC of FB C1 (that is given figure 5) is considered, a location will be associated with EC state Start, where the FB can wait for an event but not with EC states C1 or Init. As a reaction is launched by an invocation request these locations are the source of a unique transition synchronized on the InvokeECC channel of the FB with the automaton of figure 7.

The target of this transition is a set of committed locations and transitions that is more or less complex according to the number of relevant input events. For each event that is associated with a transition which source is the considered EC state a committed location and two downstream transitions that are synchronized with the automaton of the event (see figure 4) are introduced in order to test whether the event has occurred or not. These patterns are composed in a sequential way and all the relevant events are tested one after the other. This part selects the reaction, if any, that must be launched and at the end of this selection part committed locations and synchronized transitions are introduced to reset all the input events. For the ECC example of figure 5 this selection part consists of the locations Decision1 and Decision2 of the automaton of figure 6 and the resetting part consists of locations Clear1, Clear2 and Clear3 and the transitions synchronized on the channel resetEvsC1 with the automaton of figure 7.

The modelling of the reaction itself is also based on three parts. The first one is the synchronization with the scheduler of the resource that controls which FB executes its algorithms and is composed of one location and one transition. For the example this part

consists of locations SchedAwait1 for the reaction to event EviC1 and SchedAwait2 for the reaction to event Init and the transitions synchronized on channel ExecC1. The second part is also composed of one location and one transition and is used to model the execution of all algorithms of the reaction. This is a timed location that is associated with a lower and upper value of the execution time of the reaction that are specified by the invariant and guard concepts of UPPAAL (see for example location processAlgC1). The last part of the reaction is then the issuing of output events. According to UPPAAL constraints, in this part an urgent location and a transition synchronized on the corresponding channel is introduced for each output event (see for example location issueOE1 in figure 6). The target of the last transition of this sequence is then the location modelling the stable EC state reached by the reaction. A new synchronization with the invocation request can then take place.

ECC invocation request mechanism. The ECC invocation model automaton is introduced for each FB to memorise the occurrence of events and to
launch the reaction selection and the input event resetting even if the occurred events are not relevant. It has the general structure in figure 7 where the synchronisation channels are parameters that are instantiated for each FB.

**Event input resetting model.** This automaton is introduced because there is no broadcast mechanism in UPPAL. It is used to broadcast the resetting event

![Fig. 7. ECC invocation request automaton.](image)

**Event input resetting model.**

issued by the ECC model at the end of selection part to all input event models of the FB. It has then a transition synchronized on the ClearAllEvent channel of the block and one synchronized on the ClearEvent channel of each input event and is very simple as it can be seen on figure 8 for the example.

5.2. The application and resource modelling.

The previous set of automata is attached to each function block to model its execution control. Composing these sets of automata with synchronisation channels and, building a new automaton to model the scheduling, then model a specific application.

**Control flow connection.** In the function block diagram the control flow is specified by connecting an output event to an input event. In the formal model this connection is realised by instantiating to the same synchronisation channel, the channel of the output event used in the automaton modelling the ECC (e.g. figure 6.) and the External_Event channel of the input events model specified figure 4.

**Scheduling function.** An automaton that ensured that only one location modelling the execution of the algorithms of the functional blocks is active models the scheduling. In the current version of this work, this automaton is simple and only implements this function (e.g. figure 9). This corresponds to a non-pre-emptive and without priorities scheduling policy that could be improved in further work for example by using stopwatch automata (Abdeddaim and Maler, 2002) for pre-emptive scheduling.

6. FORMAL VERIFICATION.

This modelling approach can be applied to the example of figures 2 and 3 in order to check some properties of the possible executions. Actually some blocks are not modelled as described above as they are specific. The first ones are the Requester-Responder blocks, and at a first level of analysis this communication link is abstracted by the delay introduced in the event flow and modelled by the automaton of figure 10.a. The second ones are the sampler blocks that have no input event and are modelled by the automaton of figure 10.b. For this application, the timed automata network is then composed of 55 automata that are the instantiations of only 12 UPPAAL templates; as for example 14 instances of the automaton of figure 4 are included in the model.

This automata network can be analysed and a first property that can be checked is that there is no deadlock in the model. An other important property is that the control algorithms are executed; it means that the location processAlg1 of figure 6 and the relative locations for the other controllers become active. This can be verified by checking with UPPAAL that the predicate E<(C1.processAlg1) is true. Actually if the Verif FB specified figure 11 is used this property is false as the output event OE is never issued because of the loop in the control flow. In order to correct this error that is introduced by the reuse of a block in a bad context, this block has to be substituted by the one specified figure 12.

An other important property that can be checked is that the sampling period is consistent with the processing time of the application and that the output relative to a sampling event is really applied before a new sampling event is issued. This property can be verified by checking that the predicate A[not(RC1.processAlg and Verif1.S1) is true.
7. CONCLUSIONS AND FURTHER RESEARCH.

This paper presents a formalisation of the behaviour of the execution of IEC 61499 function blocks. As it may be quite difficult to understand the global behaviour of an application built by composing such blocks this formalisation and the associated verification tool are useful. The proposed formalisation is based on a modular modelling and uses timed automata in order to take into account the execution time of the algorithms. This behaviour modelling of IEC 61499 function block is basically based on the concept of reaction rather than state in order to get simpler models. Finally some examples of properties that can be verified and their relationship with the application are given.

It may seem that this formalisation is rather complex but the automata are very simple and most of them are specified as instances of generic templates. A tool has been developed that translates the standardised XML specification of an application to the timed automata network in order to make it easier. From the verification complexity point of view it may be noticed that most locations are untimed and that there is only one clock for each function block.

Of course some points have to be considered further. The abstraction of data and its effects on verification results is among the most important ones. For example this formalisation is useful to detect some deadlocks that are structural but it can not detect deadlocks linked to conditions on data. An other point is the relationships that can be made between the properties of the execution that can be checked with this approach and the properties of the application of control of plant that can be verified with other approaches. The scheduling of the application and the improvement of the modelling of more complex scheduling algorithms is a point that has been studied in other contexts and that has also to be included in this work.

Finally the confrontation of these principles to real industrial applications would be very reach. Of course the number of blocks would introduce some complexity but it is not sure that the ECC of each block would itself be very complex.

REFERENCES


