HAL
open science

A static scheduling generator for the deployment of a component based application

Mohamed Khalgui, Xavier Rebeuf, Françoise Simonot-Lion

To cite this version:

Mohamed Khalgui, Xavier Rebeuf, Françoise Simonot-Lion. A static scheduling generator for the deployment of a component based application. Holger Giese, Oliver Niggemann. Postworkshop Proceedings of the 3rd Workshop on Object-oriented Modeling of Embedded Real-Time Systems, Heinz Nixdorf Institut, 2006, 3-939350-10-9. inria-00113450

HAL Id: inria-00113450

https://inria.hal.science/inria-00113450v1

Submitted on 13 Nov 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A static scheduling generator for the deployment of a component based application

Mohamed Khalgui, Xavier Rebeuf, Françoise Simonot-Lion
LORIA(UMR CNRS 7503) - INPL
Campus Scientifique B.P. 239
54506 Vandoeuvre-lés-Nancy cedex - France,
{khalgui ; rebeuf ; simonot}@loria.fr

Abstract

This paper proposes a static scheduling of an application designed using the IEC 61499 standard. In this standard, a function block (FB) is an event triggered component and an application is a FBs network. According to specifications, we propose temporal constraints on the application behavior. To verify these constraints, we propose to transform the application blocks into a particular tasks system with precedence constraints. The purpose is to exploit previous works on scheduling. In addition, we propose a schedulability analysis generating an accessibility graph of the application. This graph allows the construction of a static scheduling to use by a sequencer at run-time.

1 Introduction

Nowadays, several component based approaches have been proposed to develop safe control applications. They allow to model applications at design time [4]. Nevertheless, it is difficult to evaluate real-time behavior without modelling the execution support and its distribution.

The IEC 61499 standard [9] is one of the most known component-based methodology in the industrial field [4]. It allows to model applications as well as the execution support. A component (called Function Block and denoted by FB) is a reusable functional unit of software owning data.

A function block interacts with its environment thanks to event and data inputs and outputs. Event inputs trigger the function block activation while data inputs provide algorithms parameters. According to [8, 9], we allows events buffering in blocks. We suppose a buffer of size m ($m \geq 1$) in each application block. In this case, events loss depends on the buffer size. On the other hand, a control application is specified by a so called "function blocks network".

A control application has classically to respect end to end bounds [13] according to specifications. These bounds represent the maximum duration between the receive of stimulus from sensors and the activation of the corresponding actuators.

According to the standard, the application blocks are distributed on containers of devices called resources. A resource is a logic execution unit corresponding to time slots of the processing unit.

In a resource, the application blocks may share data and also interactions with physical processes. The standard imposes a non-preemptive execution between them. Due to this restriction, a mutual exclusion on these interactions does not have to be explicitly handled. We note that a FB execution can be preempted by another FB belonging to another resource.

In this paper, we synthesize the static scheduling of a centralized IEC 61499 application. The resource concept is not relevant to take into account in this work. We suppose as assumption an application located in only one resource of a device. We have to apply a non-preemptive policy to perform such synthesis.

To validate the temporal behavior of a control application, we propose to transform its FBs network into a particular tasks system with precedence constraints. The purpose is to exploit the previous researches in this field. This system is different from all those proposed in other researches. It allows the representation of all execution scenarios.

To avoid any events loss, we propose to compute deadlines for the different application tasks. A deadline defines the latest completion date of a task execution [19]. This computation must take into account the end to end bounds according to specifications.

To check the application feasibility, We propose

a schedulability analysis based on a graph construction. If all deadlines are met, we generate an off-line scheduling [24] to use by a sequencer at run time. This off-line scheduling specifies all the execution scenarios inside the resource. It is a direct acyclic graph (DAG) where each trajectory represents a possible blocks scheduling.

In the section 2, we present the IEC 61499 standard. Then we briefly present in the section 3 a behavioral characterization of a FB and a control application. In the section 4, we present the transformation approach of a FBs network into a particular tasks system. Then, we present in the section 5 the computation of blocks deadlines. In the section 6, we present the schedulability analysis of an IEC 61499 control application. Finally, we present in the section 7 a method generating priorities of events inside blocks.

2 The IEC 61499 standard

We present the main concepts of the IEC 61499 Function Blocks standard [9, 8]. This standard is an extension of the IEC 61131.3 [7] for the Programmable Logic Controllers (PLC). We can divide its description into two parts: the architecture description and the block temporal behavior through the events selection mechanism.

2.1 Architecture description

An application function block (FB) (figure 1) is a functional unit of software supporting some functionalities of an application. It is composed by an interface and an implementation.

The interface contains data/event inputs and outputs supporting the interaction with the environment. Events are responsible for the activation of the block while data contain valued information.

The implementation consists of a body and a head. The body is composed of internal data and algorithms implementing the block functionalities. Each algorithm gets values in the input data channels and produces values in the output data ones. They are specified in structured text (ST) language [7].

The block head is connected to events flow. It selects the sequence of algorithms to execute with regard to an occurrence of an input event. The selection mechanism of an event occurrence is encoded in a state machine called the Execution Control Chart (ECC). At the end of the algorithms execution, the ECC sends the corresponding output event occurrences.

Regarding that we are interested in the temporal behavior of the application, we will only focus on

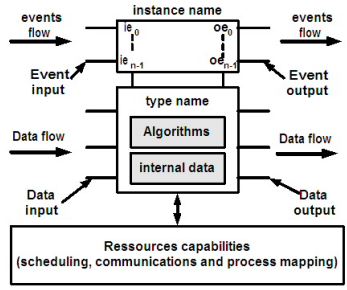

Figure 1. An IEC 61499 Function Block

event flows. Therefore, we suppose a complete synchronization between event and data flows.

In the standard, a function blocks network defines a functional architecture of a control application. Each event input (resp. output) is linked to an event output (resp. input) by a channel. If not, it corresponds to a global input (resp. output) of the network. Data inputs and outputs follow the same rules.

According to the standard, a device (industrial control system) is composed of one processing unit and interfaces (for sensors, actuators and the communication network). Moreover, it contains one or more containers called resources.

A resource contains application FBs interacting with physical processes. It defines *"the important boundary that exists between what is within the scope of the IEC 61499 model and what is device and system specific functionality. Issues such as operating system and communications protocols are outside the scope of the standard" [18]*.

The resource can be viewed as a logic execution unit corresponding to time slots of the processing unit. It provides a scheduling function, for its local FBs, applying a non-preemptive policy.

In this paper, we are interested in the validation of the temporal behavior inside a resource. We suppose then an application located in only one resource of a device. This assumption, representing the simple case, is well required to validate thereafter the temporal behavior of a distributed application on several resources of devices.

Running Example. *For all the continuation, we consider a simple toy example of an IEC 61499 ! (figure 2) to explain the proposed approach.*

This application is composed by four FBs. Each FB implements elementary functionalities (one for each input event). The application receives two external input events (i.e, ie_1 and ie_5) and can send five output ones (i.e, oe_4, oe_5, oe_6, oe_9 and oe_{10}).

According to specifications, the application has to

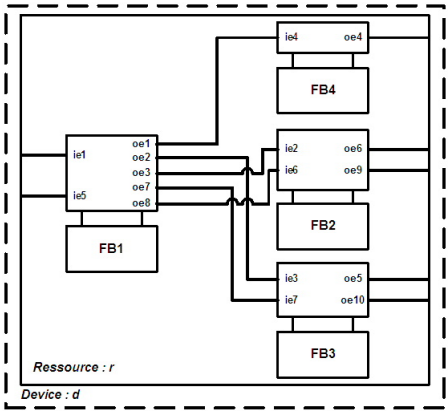

Figure 2. A control application fbn

respect end to end bounds between the receive of these external input events and the sent of output ones.

2.2 Temporal behavior of a FB

Let us turn to the internal behavior of a function block. The standard supposes that only algorithms execution spends time. In a given function block, the ECC is said idle if there is no algorithm to execute. Otherwise, the ECC is busy.

According to the standard [9], the FB contains a limited buffer (size $m \geq 1$) for input occurrences. The ECC behavior is devised into three steps :

* First, it selects one occurrence of an input event according to priority rules defined in the resource.

* It activates the algorithms sequence corresponding to the selected event. Then, it waits for the resource scheduler to execute this sequence.

* When the execution ends, it emits occurrences of corresponding output events. These occurrences are sent simultaneously or in exclusion. The emission depends on state variables of the FB.

We note that an algorithms sequence of a FB is atomic. The generation of events priorities is not specified in the standard. Therefore, it is up to the designer to fix such priorities [11]. We define at the end of this paper a method generating events priorities in a FB.

On the other hand, the ECC is specified by a state machine where each trajectory is conditioned by the reception of an input event, then the execution of an algorithms sequence and finally the sent of the corresponding output events.

Running Example. *We present the ECC behavior of the function block $FB1$ (figure 3).*

We distinguish two algorithms sequences to execute : Alg_1 and Alg_5. These sequences have to be executed when the corresponding input events occur. We note that the selection mechanism is performed thanks to state variables 'a' and 'b' of FB_1.

When the ECC selects an ie_1 occurrence, it asks ($!ex_fb$) the processor to perform the corresponding algorithms sequence Alg_1. When the execution is finished ($?end_ex$) and depending on the state variable 'a', the ECC sends oe_1 to FB_4 ***or*** *simultaneously oe_2* ***and*** *oe_3 to respectively FB_3 and FB_2.*

When the ECC selects an ie_5 occurrence, it waits also the processor to execute the corresponding algorithms sequence Alg_5. When it is finished, it sends oe_7 to FB_3 ***or*** *oe_8 to FB_2 depending on the internal variable 'b' (figure 3).*

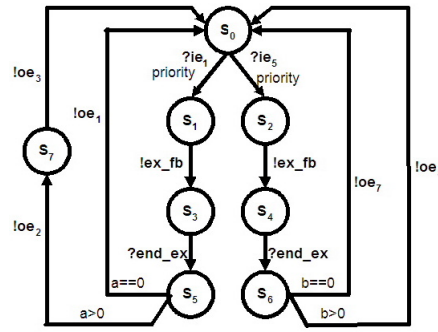

Figure 3. The ECC **behavior of** FB_1

3 Behavior formalization

To characterize the temporal behavior of a function block, we have to take into account the execution of its ECC. Indeed, the ECC selects not only algorithms to execute but also the output events to send. Nevertheless, the selection of transitions inside the ECC may depend on internal state variables. We propose to define sets of output events corresponding to all possible executions.

On the other hand, we classically define end to end bounds on the application behavior. These bounds represent the maximum duration between the receive of stimulus from sensors and the activation of the corresponding actuators. They are deduced from specifications.

In this section, we first present an abstraction of the function block behavior. Then we propose a formalization of end to end bounds. For all the continuation, we denote by fbn a function blocks network.

3.1 Function block behavior

We propose an abstraction of the function block behavior. The problem is to identify the possible output occurrences corresponding to an input one. Such association is specified in the ECC state machine. Nevertheless, firing a transition in the ECC can depend on internal variables of the block and also on input data. Therefore, we propose to identify the supersets of output occurrences that occur simultaneously.

For each trajectory of the ECC automaton (i.e. each possible execution), we associate a superset gathering all the output events occurring successively.

Let consider IE (resp OE) the set of input (output) events of fbn. In the same way, let consider IE_{FB} (resp OE_{FB}) the set of input (resp output) events of a function block FB. In addition, let tr be a trajectory ECC ECC_{FB}. we denote by,

* $IE(tr)$ the input event occurring in tr

* $OE(tr)$ the set of output events occurring in tr

We propose a function *follow* associating to an input event $ie \in IE_{FB}$, the sets of simultaneous output events.

$$\text{follow}(FB,ie) = \{OE(tr)/\ ie = IE(tr),\ tr \in ECC_{FB}\}$$

Running example. *In the example, we associate for ie_5 two sets of output events. These sets correspond to the trajectories of the ECC starting from the transition triggered by $?ie_5$.*

$$follow(fb_1, ie_1) = \{\{oe_1\}, \{oe_2, oe_3\}\}$$
$$follow(fb_1, ie_5) = \{\{oe_7\}, \{oe_8\}\}$$
$$follow(fb_4, ie_4) = \{\{oe_4\}\}$$

3.2 fbn temporal constraint

According to specifications, a real time application must often respect temporal constraints as end to end bounds. We associate, in this paper, such bounds to a FBs network.

We formalize in this section the dependance between function blocks. We propose a function $cause$ that specifies causalities between an event input of a FB and the corresponding output of another one regarding the FBs network. Note that $effect$ specifies the opposite function associating to an output event, the input event target of the sent occurrences.

Running example. *From the link between $FB1$ and $FB4$, one can deduce :*

$$cause(ie_4) = oe_1;\ effect(oe_1) = ie_4$$

On the other hand, we define in fbn the set $inputs$ (resp, $outputs$) of input (resp, output) events such as each event is not linked to another one.

$$inputs = \{ie \in IE\ /\ cause(ie) \notin OE\}$$
$$outputs = \{oe \in OE\ /\ effect\ (oe) \notin IE\}$$

In this paper, we suppose periodic events of *inputs*. We are based on the model proposed in [19] to characterize such events by a release time r, a period p, a jitter j (the maximum deviation of the period) and a constant deadline d.

Running example. *In the example, we have the following sets :*

$$inputs = \{ie_1, ie_5\}$$
$$outputs = \{oe_4, oe_5, oe_6, oe_9, oe_{10}\}$$

We propose the function *bound* encoding all the end to end bounds of fbn. *bound(ie,oe)* denotes the maximum duration between the release time of an ie occurrence ($ie \in inputs$) and the sent of an oe one ($oe \in$ outputs). It is directly deduced from the application specifications.

Running Example. *In the treated example, bound(ie_1,oe_4) specifies the maximum duration that can take the treatment of Alg_1 and Alg_4.*

We suppose the following constraints deduced from the specifications :

- *bound(ie_1,oe_4) = bound(ie_1,oe_5) = 20,*
- *bound(ie_1,oe_6)= bound(ie_5,oe_{10}) = 25,*
- *bound(ie_5,oe_9) = 23,*

4 Transformation into a Task Model

In this part, we propose to transform fbn into a tasks system S with precedence constraints [2]. This system is different from all those proposed in other researches.

We first define a task as an execution of a FB. Then we define a trace as a sequence of tasks.

4.1 Task definition

An application task T corresponds to the execution of a function block activated by an occurrence of an input event ie. This task implements the corresponding algorithms sequence.

We define the function *generate(ie)* associating for an input event ie the corresponding task T. Note that *is_ generated_ by(T)* is the opposite function of generate(ie). In addition, we denote by *Task* the tasks set of S.

Let set_{OE} be a set of output events. We define the function $target(set_{OE})$ associating for set_{OE} the following set of tasks,

$$\text{target}(set_{OE}) = \{T \in Task \ / \ \exists \ oe \in set_{OE}, \ oe = cause(is_generated_by(\text{T}))\}$$

We propose, the following characterization of a task T,

$$\text{T}=\{ \ r, j, p, d, WCET, \ BCET, \ \ pred, \ succ\}$$

such as,

- r (release time), j (jitter), p (period), d (deadline) : Temporal parameters according to the model proposed in [19]. We propose, in the next section, a method processing these parameters.
- WCET (resp BCET) : the worst (resp best) case execution time of the algorithms sequence corresponding to ie. It can be evaluated using the code and the characteristics of the execution support [23].
- pred : the task that must be executed in fbn before the execution of T . It corresponds to the execution of the FB producing cause(ie) (ie = is_generated_by(T)).
- succ : a set of tasks sets. Each tasks set corresponds to a possible execution scenario (ie. only one tasks set between all ones is performed). The tasks of a set are to be executed once the execution of T is finished. They belong to FBs activated once the ie treatment finishes.

$$succ(T) = \{set_T \subset \text{Task} \ / \ \exists \ set_{OE} \in follow(fb, is_generated_by(T)), \ set_T = target(set_{OE})\}$$

Running example. *In the proposed example, we distinguish seven tasks T_i corresponding to ie_i (i $\in$ [1, 7]). The task T_1 corresponds to the algorithms sequence Alg_1. The predecessor of T_4 and the successors of T_1 are as follows,*

$$pred(T_4) = T_1$$
$$succ(T_1) = \{\{T_2, T_3\}; \{T_4\}\}$$

When the T_1 execution is finished, two scenarios are possible : Either we execute T_2 and T_3 or we execute T_4.

We define *first* (resp *last*) as the set of tasks with no predecessors (resp successors). The set *first* (resp *last*) corresponds to $inputs$ (resp $outputs$),

$$first = \{T \in Task \ / \ pred(\text{T}) \notin \text{Task}\}$$
$$last = \{\text{T} \in \text{Task} \ / \ succ(T) = \emptyset\}$$

4.2 Trace definition

To specify causalities between tasks, we define in S a trace tr as a tasks sequence,

$$tr = T_0, T_1, \ldots, T_{n-1}$$

such as,

- $T_0 \in first, T_{n-1} \in last$
- $\forall \ i \in \ ]1, n-1\ ], T_{i-1} = pred(T_i)$

We define the trace concept to just specify end to end bounds. A trace represents then a possible execution part of the application when an event belonging to $inputs$ occurs.

We denote by $Traces$ the traces set in S. We denote also by start(tr) the first task of the trace tr.

To avoid any functional problem, we suppose non reentry traces [14, 20] : the execution of the $k-th$ instance of a trace must not start before the execution end of the $(k-1)-th$ one. More precisely, the period of the first task is higher than the corresponding end to end bound.

In a closed control loop, a new reading from a sensor cannot be done before the activation of the corresponding actuators.

Running Example. *In the example, we distinguish five traces. Each trace specifies a possible application behavior.*

$$tr_1 = T_1, T_2; \ tr_2 = T_1, T_3; \ tr_3 = T_1, T_4; \ tr_4 = T_5, T_6; \ tr_5 = T_5, T_7$$

Finally, we define in the system S an operation op_i as the set of traces having the same first task T_i. It specifies all possible executions of fbn when T_i is activated.

$$op_i = \{tr \in Traces \ / \ start(tr) = T_i\}$$

At run-time, some tasks of op_i have to be executed each time T_i is activated. Some others have to be executed depending on the execution of their predecessors. We characterize the execution of a task as follows,

Definition. Let T be a task of an operation op_i. We say that the task T is *principal* if it is executed each time T_i is executed.

More precisely, the task T is principal if it is directly executed when its predecessors are completely executed.

$$\forall \ T' \in pred^*(T), \ \text{cardinality}(succ(T')) = 1$$

Running example. *In the example, we distinguish two operations op_1 and op_5 (figure 4),*

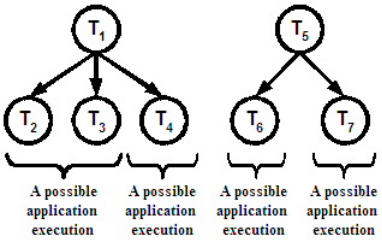

Figure 4. The application operations

$$op_1 = \{tr_1, tr_2\};\ op_5 = \{tr_3, tr_4, tr_5\}.$$

The operation op_1 specifies two possible application executions. According to the execution of the task T_1, we execute either T_2 and T_3 or T_4.

The task T_1 is executed periodically. It is then a principal task. The execution of the tasks T_2, T_3 and T_4 depends on the state variable 'a' of FB_1. They are not principal tasks.

5 Deadlines computation

To validate the temporal behavior of the application, we present in this section an approach to compute tasks deadlines. A deadline represents the latest completion date of a task execution [19].

A deadline computation is based on end to end bounds described in specifications. Moreover, it must avoid any events loss in buffers of blocks.

5.1 deadlines computation basing on end to end bounds

Let tr be a tasks trace of S. We classically define $bound(tr)$ as the end to end bound of the trace tr. This temporal constraint corresponds to the bound between the activation of $first(tr)$ and the end of the $last(tr)$ execution.

We define d the deadline of a task $T \in tr$. We define d^{bound} the upper bound deadline to guarantee the end to end bounds of traces containing T. d^{bound} has to take into account the time for executing all the successors belonging to $succ(T)$ before their respective deadlines. We process this deadline as follows,

$If\ T \in last,$

$$d^{bound} = bound(tr)$$

$Otherwise,$

$$d^{bound} = min_{T_i \in succ(T)}\{d_i - \sum_{T_j \in succ(T)}^{d_j \leq d_i} T_j.WCET\}$$

To respect the end to end bounds of all traces containing T_i, the deadline d_i has to respect the following condition

$$\forall T_i \in S,\ d_i \leq d_i^{bound}.$$

Running example. *We suppose the following worst case and best case execution times of the different tasks,*

Task	T_1	T_2	T_3	T_4	T_5	T_6	T_7
WCET	4	4	5	4	3	6	5
BCET	3	2	4	3	2	4	2

We process the tasks deadlines as follows,

- $d_2^{bound} = d_4^{bound} = bound(tr_1) = bound(tr_3) = 20$
- $d_3^{bound} = bound(tr_2) = 25$
- $d_1^{bound} = min\{d_4^{bound} - T_4.WCET,\ d_2^{bound} - T_2.WCET - T_3.WCET\} = 11$
- $d_6^{bound} = bound(tr_4) = 23$
- $d_7^{bound} = bound(tr_5) = 25$
- $d_5^{bound} = min\{d_6^{bound} - T_6.WCET,\ d_7^{bound} - T_7.WCET\} = 17$

5.2 Deadlines computation taking into account buffers size

Denoting by *m* the size of buffers in the application blocks, the events loss occurs when the number of active tasks at a given time is larger than m.

To avoid this problem, we first propose to characterize the release time of each application task. We propose then to compute deadlines in the order to keep such number lower than m.

Note that this method can also be used to determine the sizes of buffers if they are not fixed.

5.2.1 Temporal characterization

Let T_i be an application task such as $T_i \in first$. The task T_i is periodic regarding the periodicity of readings from sensors.

We characterize the temporal behavior of an application block as follows,

Definition. Let fb be a function block of fbn. We define a *"hard activity duration"* of fb, a duration during which all the non principal tasks are periodically activated.

More precisely, the hard activity duration of a block corresponds to the worst case when all the non principal tasks are executed as if they are principal.

To avoid any events loss in the corresponding buffer, we must compute tasks deadlines in a such duration.

Running example. *In the example, we suppose T_2 and T_6 (resp T_3 and T_7) as principal tasks in FB_2 (resp FB_3) to compute their deadlines d_2 and d_6 (resp d_3 and d_7).*

Considering non reentry traces, all the application tasks are periodic. Let tr = T_0,....,T_{n-1} be a trace of S. We characterize the temporal behavior of $T_i \in tr$ (i$\in$ [1 , $n-1$]) as follows.

- $r_i = r_0 + \sum_{k=0}^{k=i-1} BCET(T_k)$
- $j_i = d_{i-1} - \sum_{k=0}^{k=i-1} BCET(T_k)$
- $p_i = p_0$

The earliest activation date of T_i occurs when each previous task in the trace is executed as soon as possible ($\sum_{k=0}^{k=i-1} BCET(T_k)$). The latest activation date occurs when the previous task ends just in time (i.e. at its deadline). The difference between the earliest and the latest date corresponds to the jitter (figure 5).

5.2.2 Evaluating the Hyper Period

To validate the application, we classically verify the respect of deadlines in a hyper-period H [22]. Therefore, we propose to compute these deadlines in the same hyper-period.

Let lcm be the least common multiple of the tasks periods. Let $T_{max} = \{r_{max}, p_{max}, j_{max}\}$ and $T_{min} = \{r_{min}, p_{min}, j_{min}\}$ be two tasks of $first$ such as,

$$\forall T_i \in first,\ r_{min} + j_{min} \leq r_i + j_i \leq r_{max} + j_{max}$$

As we treat non reentry traces, we can exploit the result on the hyper period proposed for the schedulability analysis of asynchronous systems [17]. By analogy with our case, the analysis may be done in $[r_{min} + j_{min},\ r_{max} + j_{max} + 2.lcm]$.

5.2.3 Deadlines computation of a FB

Let fb be a function block containing q tasks. To compute the deadlines of its tasks, we suppose the hyper period as a hard activity duration. All the non principal tasks are then supposed as principal ones in a such period.

According to the previous temporal characterization, each task $T_a \in fb$ is activated periodically. Moreover, T_a belongs also to a trace tr of S.

$$tr = T_0, ..., T_a, ..., T_{n-1}$$

Let $T_{a,k}$ be the $k-th$ instance of the task T_a in the hyper period (figure 6). Let $T_{e,h}$ be the $(m+1)-th$ instance activated in fb after $t(T_{a,k})$.

To avoid any events loss in the buffer, the execution of $T_{a,k}$ must finish before the activation of $T_{e,h}$. Otherwise, $m+1$ input events occur during the $T_{a,k}$ execution, whereas the buffer size is m.

We denote by $d_{a,k}^{loose}$ the deadline bound of the instance $T_{a,k}$ to avoid any events loss in fb (figure6). It corresponds to the earliest activation date of $T_{e,h}$.

$$d_{a,k}^{loose} = r_e + h.p_e - (r_0 + k.p_0)$$

Considering that there exist $(\lfloor \frac{r_{max}+2.lcm-r_i}{p_i} \rfloor + 1)$ instances of each task $T_i \in fb$ in the hyper period, we compute the corresponding deadline bound d_i^{loose} as follows,

$$d_i^{loose} = min\{d_{i,k}^{loose}, k \in [0, \lfloor \frac{r_{max}+2.lcm-r_i}{p_i} \rfloor]\}$$

In the same way, we compute the deadlines of tasks belonging to each block of fbn. We suppose the hyper period as a hard activity duration for each one of them.

Finally, to avoid any events loss in a buffer, the deadline of each task has to respect the following condition,

$$\forall\ T_i \in S,\ d_i \leq d_i^{loose}$$

5.3 Deadlines generation method

In this subsection, we propose a method processing deadlines for the different tasks of the system S. These deadlines have to respect

- d^{loose} bound to avoid any events loss in blocks.
- d^{bound} bound to satisfy end to end bounds.

Contrary to d^{loose}, the computation of d^{bound} is based on the successors deadlines of the task. Therefore, such deadlines have to take into account the corresponding d^{loose}. We propose a computation method in two steps.

Let tr be a trace of S as follows,

$$tr = T_0, ..., T_{n-1}$$

first step. We compute the deadline d^{loose} of each task $T_i \in tr$, ($i \in$ [0, $n-1$]).

Second step. We compute the real deadline thanks to the proposed bound formula,

- $d_{n-1} = min\{d_{n-1}^{loose}, bound(tr)\}$

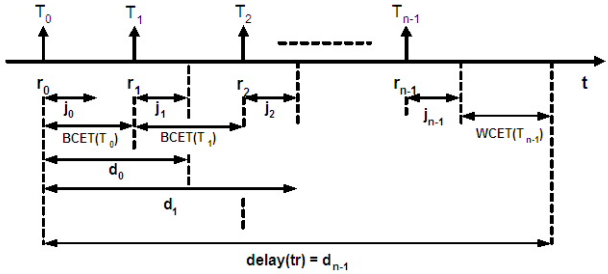

Figure 5. Temporal characterization of tr

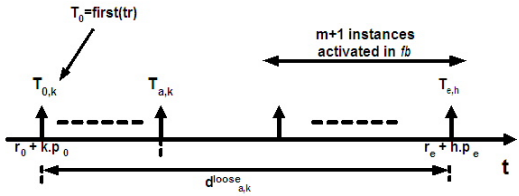

Figure 6. The scenario of the instances arrivals

- $\forall i \in [0, n-2]$,

$$d_i = \min\{\, d_i^{loose}, d_i^{bound} \}$$

with,

$$d_i^{bound} = min_{T_k \in succ(T_i)} \{\, d_k - \sum_{T_j \in succ(T_i)}^{d_j \leq d_k^{loose}} \text{WCET}(T_j) \}\}$$

To conclude as soon as possible the infeasibility of the application, we propose the following schedulability condition,

Proposition. (Schedulability condition)

Let consider a tasks system S specifying an IEC 61499 application. The system S is infeasible if

$$\exists\, T_i \in S, d_i < WCET_i$$

Running example. *In the example, we propose the following temporal characteristics of the tasks belonging to first. We suppose that their jitters are null. We deduce also the temporal characteristics of the tasks T_3, T_4, T_6 and T_7.*

Task	T_1	T_5	T_2	T_3	T_4	T_6	T_7
r	1	3	4	4	4	5	5
p	25	25	25	25	25	25	25

We Compute for the different tasks the corresponding deadlines according to the proposed method,

First step. *To avoid any events loss in $FB_1, FB_2, FB_3 and FB_4$, we compute for each task the deadline d^{loose} in the hyper period $[1, 53]$. We obtain the following values.*

Task	T_1	T_5	T_2	T_6	T_3	T_7	T_4
d^{loose}	25	25	25	25	25	25	50

Second step. *Applying the proposed method, we obtain the following real deadlines that avoid any events loss and guarantee the respect of end to end bounds,*

- d_2 = *min*$\{\, bound(tr_1), d_2^{loose}\}$ = *20*
- d_3 = *min*$\{\, bound(tr_2), d_3^{loose}\}$ = *25*
- d_4 = *min*$\{\, bound(tr_3), d_4^{loose}\}$ = *20*
- d_1 = *min*$\{\, d_1^{loose}$, *min*$\{\, d_4$-$WCET(T_4)$, d_2-$WCET(T_2)$- $WCET(T_3)\, \}\, \}$ = *11*
- $d_6 = min\{bound(tr_4), d_6^{loose}\}$ = *23*
- d_7 = *min*$\{\, bound(tr_5), d_7^{loose}\}$ = *25*
- d_5 = *min* $\{d_5^{loose}$, *min*$\{\, d_6$ - $WCET(T_6)$ - $WCET(T_7)\}\}$ = *17*

All the processed deadlines are higher than the corresponding $WCET$. Therefore, we use them to perform the schedulability analysis of the application.

6 Schedulability analysis

In the scheduling theory of real-time systems, two interests exist : periodic systems [16, 1] and systems with precedence constraints [15, 3]. Even these two fields are separately rich in results, there are few results where both aspects are treated together.

Until today, only one work studied the case of systems with precedence constraints, periodicity and end-to-end bounds [5]. Nevertheless, the used task model is not well expressive to specify all execution scenarios of an application.

We propose then a schedulability analysis [1] of a tasks system S based on the proposed task model. This analysis validates the temporal behavior according to specifications. The schedulability criterion is then the respect of all tasks deadlines.

Based on the method processing these deadlines, the analysis applies the EDF policy [25] to verify all bounds. We propose to construct an accessibility graph [22] in the proposed hyper period. The accessibility graph is a set of scheduling trajectories. Each trajectory represents a possible scheduling of a traces set. We apply the EDF algorithm during each trajectory construction to verify end to end bounds of the corresponding traces. A trajectory specifies then a possible behavior of the application.

When the application is feasible, we generate an off-line scheduling as a DAG to use by a sequencer at run-time. This DAG is well required to abstract the resource behavior at run-time.

6.1 Accessibility graph generation

Let G be the accessibility graph constructed during the schedulability analysis of S. We construct this graph in the proposed hyper period $[r_{min} + j_{min}, r_{max} + j_{max} + 2.lcm]$. We define a tasks state C of G as follows :

C={ set, T, t} Where,

- set : a tasks set of S to execute
- T : a selected task to execute between all the active ones of set. We apply the EDF policy to perform such selection
- t : the start time of the T execution

We propose the following rules to apply during the graph construction. The first rule allows to construct the first tasks state C_0.

- **Rule 0.** the first tasks state is characterized as follows,

  $$C_0 = \{ set_0, T_{min}, t{=}r_{min}{+}j_{min}\}$$

  Where, set_0 contains all the tasks belonging to first ($set_0 = first$).

We generate then step by step the different tasks states in the different G trajectories as follows.

Let C_i = { set_i, T_i, t_i} be a state in the graph G

- **Rule 1. if** $t_i \geq r_{max} + j_{max} + 2.lcm$ **Then** we stop the current trajectory construction. This trajectory is a possible scheduling of the application.
- **Rule 2. if** $\exists\ T_j \in set_i\ /\ d_j < t_j$ **Then** the system S is infeasible.
- **Rule 3. if** $T_i \notin last$ **then** Let us suppose that $succ(T_i)$ contains k tasks sets.

  $$succ(T_i)= \{ts_0,...,ts_{k-1}\}$$

  We construct k tasks states C_0,...., C_{k-1} target of C_i as follows,

  $$\forall \text{ j} \in [0, k-1], C_j = \{set_j, T_j, t_j\}$$

  1. $\forall$ j $\in$ [0,$k-1$], set_j = $set_i \setminus \{T_i\} \cup ts_j$
  2. $t_j = t_i + WCET(T_i)$
- **Rule 4. if** $T_i \in$ last. Let tr be the trace containing T_i . We construct $C_j = \{set_j, T_j, t_j\}$ target of C_i as follows

  1. S_j = $S_i \setminus \{T_i\} \cup$ first(tr)
  2. $t_j = t_i + WCET(T_i)$

6.2 Algorithm

We propose the algorithm applying the approach. This algorithm is based on a recursive function *generate()* (table 1). The application feasibility is concluded if we successfully generate all the possible trajectories.

Based on a graph construction, the proposed analysis is optimal regarding the optimality of EDF [25]. Therefore, if we conclude the infeasibility of the application then no other approach concludes the reverse.

To calculate the algorithm complexity, we denote by m the number of all operations to schedule. Let

$Bool$ **generate(C :** $tasks_state$**, first :** $tasks_list$**,** $tasks$ **:** $tasks_list$**,** $time$**:** $integer$**)**
Begin
$T_1 : task$; $C_1 : tasks_state$; $result : bool$;
$result \leftarrow true$;
if($C.t \geq time$) //$time$ = 2.lcm + r_{max} + j_{max}
then return($true$);
for each task $T_1 \in C.S$
if $deadline_violated$ (T_1)
then $return$ ($false$);
$C.T \leftarrow apply_EDF(C.S)$;
while($ts \in C.T \rightarrow succ$ and $result$)
create(C_1); C_1 .S $\leftarrow C.S \setminus$ C.T $\cup ts$;
C_1.t $\leftarrow C.t$+ $T_2.WCET$;
$result \leftarrow$ generate(C_1 , $first$, time)
return result;
End.

Table 1. The recursive function generate()

p_i be the traces number of the operation op_i ($i \in [$ 0,$m-1$]). Let q_i be the tasks number of the longest op_i trace.

The maximum number of tasks states to construct in the graph is α*β where,

- α = $\Pi_{j=0..m-1} p_j$ is the trajectories number in the accessibility graph.
- β=$\Sigma_{j=0}^{m-1} q_j$ is the longest trajectory.

To process the problem complexity, we suppose that the biggest number of trajectories in the accessibility graph is n. We suppose moreover that all trajectories contains n tasks states. The complexity of the problem is then O(n^2).

Running example. *In the example, we perform the proposed algorithm to validate the temporal behavior of the application.*

By constructing the accessibility graph in the hyper period [1,53] , the algorithm constructs the trajectories specifying the different execution scenarios. We present a part of such graph (figure 7).

Applying the Rule 0, we construct the first tasks state C_0 =$\{\{T_1,T_5\},T_1,t = 1\}$. *Then, we apply the proposed rules to construct the remainder states.*

We successfully construct the graph and we prove the application feasibility.

In addition, we deduce from the accessibility graph an off-line scheduling as a DAG. In fact, identical branches of the tree are merged. We show in each state of this graph the selected task and the start time of its execution. This graph, used by a sequencer at run-time, abstracts the resource behavior (figure 8).

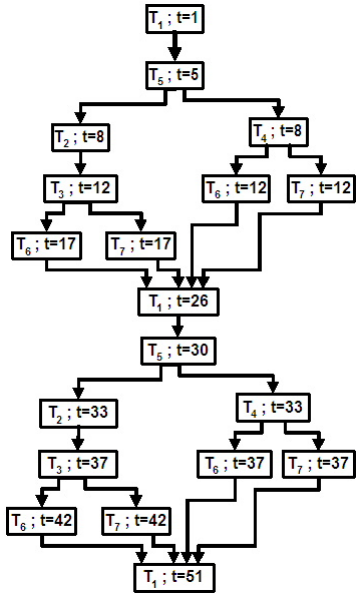

Figure 8. The-schedulability-analysis

7 Definition of events priorities policy

Now we have to go back to the FB behavior. The selection of events occurrences (by the ECC) must be based on the corresponding tasks deadlines. Indeed, the occurrence to select must correspond to the task that has the earliest deadline.

Regarding that the ECC is unaware of these temporal properties, We propose therefore to exploit the previous schedulability analysis to generate events priorities for each block. The scheduler receives then at a given time t from an ECC the adequate task to execute that has the earliest deadline.

This proposition allows to guarantee the conformity between the internal behavior of a FB and the scheduler behavior inside the resource.

As the schedulability analysis is performed in the hyper period [$r_{min} + j_{min}$, r_{max}+j_{max}+2.lcm] , we propose to generate for each block the order of occurrences to select in a such duration.

Let $ie_{i,m}$ and $ie_{j,n}$ be two occurrences to select, we note that $ie_{i,m} \ll ie_{j,n}$ if the ECC has to select $ie_{i,m}$ before $ie_{j,n}$.

Running Example. *According to the generated accessibility graph, we deduce the following events priorities for* ECC_1 *:* $ie_{11} \ll ie_{51} \ll ie_{12} \ll ie_{52} \ll ie_{13} \ll ie_{53}$

The state machine ECC_1 *must select these occur-*

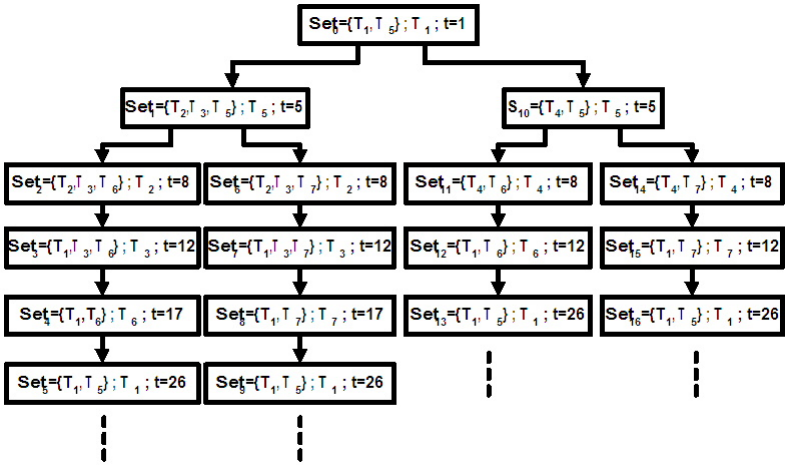

Figure 7. The-accessibility-graph

rences in a such order to guarantee the correct composition with the scheduler.

8 Conclusion

This paper proposes a contribution to develop an industrial control application according to the IEC 61499 standard. This application is located in a single resource of a device. We classically suppose end to end bounds as bounds on the application behavior according to specifications.

We combine the two versions of the standard by supposing a buffer of size $m \geq 1$ in each block. The events loss appears in a block when the number of new events is larger than the buffer size.

To exploit the previous researches on scheduling, we propose to transform the application into a particular tasks system with precedence constraints. this system is different from all those proposed in other works. It allows to model all the possible execution scenarios of the application.

To avoid any events loss, we propose a method processing deadlines for tasks. This method takes also into account the end to end bounds.

To validate the application behavior, we propose an optimal schedulability analysis based on the construction of an accessibility graph. If the application is feasible, then we generate an off-line scheduling to use by a sequencer at run-time.

We are currently working to propose a fault tolerant schedulability analysis of an IEC 61499 application [26, 6]. The purpose is to authorize a limited number of deadlines to be missed.

We also plan to check the feasibility of a control application distributed on several resources of a device. According to the standard, an on-line preemptive policy can be applied to schedule blocks belonging to different resources. Thanks to this paper contribution, we plan to consider the off-line scheduling of each resource as an OS task [26].

The application is viewed then as a set of OS tasks. We plan to apply a schedulability condition checking the on-line preemptive scheduling of these tasks in the device.

In addition, we plan to extend our researches by supposing a distributed application on several devices. Such extension imposes to take into account the communication interfaces and the networks bounds.

References

[1] Audsley. N. C, Burns. A, Richardson. M. F, Tindell K and Wellings, A. J. ”Applying new scheduling theory to static priority pre-emptive scheduling”. Software Engineering Journal. 1993

[2] Babanov. A, Collins. J, Gini. M, ”Scheduling tasks with precedence constraints to solicit desirable bid combinations”. International Conference on Autonomous Agents. Melbourne, Australia. 2003.

[3] Blazewicz. J ”Scheduling dependent tasks with different arrival times to meet deadlines”. Modelling and Performance Evolution of Computer Systems. 1976

[4] Crnkovic. I, Larsson. M, ” *Building reliable component-based software systems* ”. Artech House. London. ISBN 1- 58053-327-2.

[5] Cucu. L, Sorel. Y, ”Schedulability condition for systems with precedence and periodicity constrainsts

without preemption". RTS2003, 11th conference on Real-time and embedded systems, France, 2003.

[6] Hamdaoui. M, Ramanathan. P, "A Dynamic Priority Assignment Technique for Streams with (m, k)-Firm Deadlines", IEEE Transactions on Computers, Vol. 44, No. 4, Dec.1995, pp. 14431451.

[7] International Standard IEC 1131-3. " *Programmable Controllers Part 3* ". Bureau Central de la commission Electrotechnique Internationale. Switzerland. 1993.

[8] International Standard IEC TC65 WG6. " *Function Blocks for Industrial Process Measurements and Control Systems* ". Committee Draft. 2003.

[9] International Standard IEC TC65 WG6. " *Industrial Process Measurements and Control Systems* ". Committee Draft. 2004.

[10] Jeffay. K, Stanat. D. F, Martel. C. U, "On Non- Preemptive Scheduling of Periodic and Sporadic Tasks", Proc. of the 12th IEEE Real-Time Systems Symposium, San Antonio, December 1991, pp. 129-139

[11] Khalgui. M, Rebeuf. X, Simonot-Lion. F, " *A behavior model for IEC 61499 function blocks* ". Third Workshop on Modelling of Objects, Components and Agents. Denmark. 2004.

[12] Khalgui. M, Rebeuf. X, Simonot-Lion. F, " *A schedulability analysis of an IEC61499 control application* ". 6th IFAC International Conference on Fieldbus Systems and their Applications. Fet2005. Mexico 2005.

[13] Kim. H. S, Shroff. N. B, "The notion of end-to-end capacity and its application to the estimation of end-to-end network delays", Computer Networks: The International Journal of Computer and Telecommunications Networking, Pages: 475 - 488, Volume 48 , Issue 3 (June 2005).

[14] Klein. M H, Ralya. T, Pollack. B, Obenza. R, Harbour M G. " *A practioner's handbook for real-time analysis* ". Guide to Rate Monotonic Analysis for Real-Time Systems". Kluwer Academic Publisher, 1993.

[15] Lawler. E. L "Optimal sequencing of a single machine subject to precedence constraints". Management Science. 1973

[16] Lehoczky. J. P, Sha. L, Ding. Y, "The rate monotonic scheduling algorithm : exact characterization and average case behavior". Proceedings of the IEEE Real-Time Systems Symposium. 1989

[17] Leung. J, Whitehead. J, " *On the complexity of fixed-priority scheduling of periodic real-time tasks* ". Performance Evaluation 2 (1982), 237250.

[18] Lewis. R, " *Modelling Control systems using IEC 61499* ". The Institution Of Electrical Engineers. ISBN 0 85296 796 9.

[19] Liu. C. L and Layland. J. W (1973) Scheduling Algorithm for Multiprogramming in a Hard Real-Time Environment, Journal of ACM 20 46-61

[20] Liu. J W S, " *Real-Time Systems* ". Prentice Hall, 2000.

[21] Naedele. M, "Fault-Tolerant Real-Time Scheduling under Execution Time Constraints", Sixth International Conference on Real-Time Computing Systems and Applications (RTCSA'99), Hongkong, 1999.

[22] Pailler. S, Choquet-Geniet. A, "Off-Line scheduling of real-time applications with variable duration tasks", 7th Workshop on Discrete Events Systems, pp. 373-378, France, 2004.

[23] Petters. S. M, Betts. A, Bernat. G, "A New Timing Schema for WCET Analysis", 4th international workshop on worst-case execution time analysis (WCET2004) in conjunction with the 16th Euromicro Intl Conference on Real-Time Systems (ECRTS2004). Italy. 2004.

[24] Schild. K, Würtz. J, "Off-Line Scheduling of a Real-Time System", Proceedings of the 1998 ACM Symposium on Applied Computing, SAC98, 1998, ACM Press.

[25] Stankovic. J. A, Spuri. M, Ramamrithan. P, Buattazzo. G. C, "Deadline Scheduling For Real-Time Systems", ISBN : 0-7923-8269-2, Kluwer Academic Publichers

[26] Hiroaki Takada and Ken Sakamura, "μITRON for Small-Scale Embedded Systems", IEEE MICRO, vol.15, no.6, pp.46-54, Dec. 1995.