A degraded scheduling generation of a component based application

Mohamed Khalgui, Françoise Simonot-Lion, Xavier Rebeuf

A tolerant temporal validation of components based applications

Mohamed Khalgui, Xavier Rebeuf, Françoise Simonot-Lion

INPL - LORIA(UMR CNRS 7503)

Campus Scientifique B.P. 239 54506 Vandoeuvre-lès-Nancy cedex. France.

[khalgui, rebeuf, simonot]@loria.fr

Abstract

This paper deals with control applications designed using the component-based standard IEC 61499. In this standard, a function block is an event triggered component and an application is a function blocks network. Supposing end to end delays on applications behavior, the hard temporal validation of FBs networks may be not feasible. We propose to weaken these delays by tolerance constraints deduced from specifications. Exploiting the (m, k) model to specify these constraints, we propose a schedulability analysis generating an off-line scheduling to use by a sequencer at run-time.

1 Introduction

The development of safety control applications is often a complex activity. Indeed, such applications have to be certified with regard to several functional and extra-functional properties. One of the most important property deals with Real Time behavior.

Several component based approaches have been proposed to develop such applications [12]. The IEC 61499 standard [1] is a component-based methodology allowing to design control applications as well as the execution support [8]. In the standard, the Function Block is defined as a reusable and event triggered component [6]. It is a functional unit of software owning data. A control application is designed as a "function blocks network" [5, 11].

In [3], we introduced end to end delays deduced from the application specification. Each delay represents the maximum duration between the receive of an external application event and the corresponding output event. Supposing an IEC 61499 application as a hard real time system [14], we proposed a schedulability analysis to validate its temporal behavior. To perform such analysis, we proposed a transformation approach of such application into a particular tasks system with precedence constraints. If all delays are verified, we generate an off-line scheduling to apply by a sequencer at run time.

In some cases, when the application is not schedulable, its execution is possible in a debased mode. For example, in a closed control loop, it can be possible to discard some sensors readings. In a FBs application, such degradation corresponds to discard some input event occurrences. Therefore, we propose to define an IEC 61499 application as a weakly hard real time system [9] by defining tolerant constraints on its behavior.

We classically specify a weakly hard real time system thanks to the (m, k) model [10]. A tasks system is under a constraint (m, k) such as $m < k$, if at least m among k consecutive instances of each task meet their deadlines.

In this paper, we define $(m,\ k)$ constraints on each end to end delay. We propose a tolerant schedulability analysis of an IEC 61499 control application. This analysis verifies these delays according to their $(m,\ k)$ constraints. If the application is schedulable, we generate an off-line scheduling containing only instances of tasks that meet their deadlines.

In the next section, we present the IEC 61499 standard. In the section 3, we present a temporal characterization of an IEC 61499 application [3]. Then we present in the section 4 the tolerant schedulability analysis.

2 The IEC 61499 standard

We present the main concepts of the IEC 61499 Function Blocks standard [1]. This standard is an extension of the IEC 61131.3 [2] for the Programmable Logic Controllers. We can divide its description into two parts: the architecture description and the block behavior through the events selection mechanism.

2.1 Architecture description

An application function block (FB) (figure1) is a functional unit of software that supports 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 function block while data contain valued information.

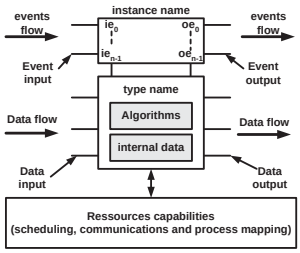

Figure 1: An IEC 61499 Function Block

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 programmed in structured text (ST) language [2].

The block head is connected to event flows. It selects the sequence of algorithms to execute with regard to an input event occurrence. The selection mechanism of an event occurence 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.

In the standard, a function blocks network defines the functional architecture of a control application. Each function block event input (resp. output) is linked to an event output (resp. input) by a channel. Otherwise, it corresponds to a global application input (resp. output). Data inputs and outputs follow the same rules.

Running Example. *For all the continuation, we consider the following running example (Figure 2) of a control application composed by four FBs.*

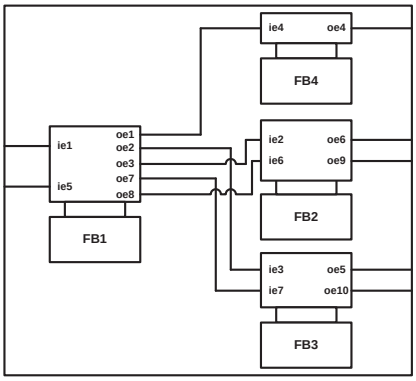

Figure 2: A control application

2.2 Events selection mechanism

In a function block, the ECC is said idle if there is no algorithm to execute. Otherwise, the ECC is busy. According to the standard [1], the FB contains an internal buffer to store input occurrences. The ECC behavior is divided into three steps:

- First, it selects one input event occurrence according to priority rules.
- It activates the algorithms sequence corresponding to the selected event. Then, it waits for the scheduler to execute this sequence.
- When the execution ends, it emits corresponding output event occurrences.

We note that an algorithms sequence is atomic and the scheduling policy is non preemptive. On the other hand, the policy of events priorities is not specified in the standard. Therefore, it is up to the designer to fix such policy for each funtion block. Note that the ECC is specified as a state machine where each trace is composed by a waiting of an input event, invocations of algorithms and sending of output events.

Running example. *We present the ECC behavior of the function block FB_1 (Figure 3). The selection mechanism is performed thanks to a state variable $'priority'$.*

When the ECC selects an ie_1 occurence, it asks $(!ex\_fb)$ the processor to perform the corresponding algorithms sequence. When the scheduler signals the execution end $(?end\_ex)$, the ECC sends oe_1 to FB_4 or simultaneously oe_2 and oe_3 to respectively FB_3 and FB_2. Sending of these output events depends on FB_1 state variables. In the same way, when the ECC selects an ie_5 occurence, it waits the processor to execute the corresponding algorithms sequence. When it is finished, it sends oe_7 to FB_3 or oe_8 to FB_2.

3 Temporal characterization of an IEC 61499 control application

In [3], we proposed a schedulability analysis to validate the temporal behavior of a hard IEC 61499 application. To perform such analysis, we proposed a transformation approach of such application into a particular tasks system with precedence constraints. This transformation lets to process tasks deadlines according to end to end delays described in specifications.

3.1 Transformation into a task model

We define an application task T as a FB execution activated by an input event occurence ie. This task is characterized by:

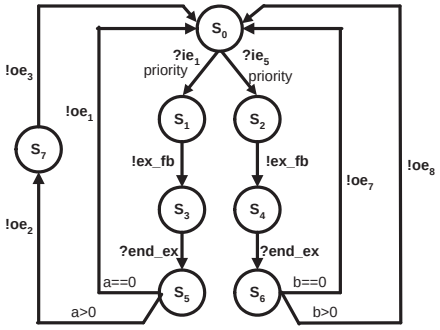

Figure 3: The ECC behavior of FB_1

- $WCET(T)$: the worst case execution time of the algorithms sequence corresponding to ie.
- $pred(T)$: the task that must be executed in the application before the T execution.
- $succ(T)$: a set of tasks sets. All the elements of a set correspond to tasks to execute once the execution of T is finished. Note that each set corresponds to a possible execution scenario (ie. only one tasks set between all ones of $succ(T)$ is performed).

We denote by T_i^j the j-th instance of the task T_i. We define $first$ (resp $last$) the set of tasks that they have not a predecessor (resp successors).

Supposing process control application in closed loop, we define an arrival law for each input event. Such control is based on periodic readings from sensors to compute commands for actuators. Therefore, each task T belonging to first is activated periodically. We characterize such task by a release time r, a period p, a jitter j (the maximum deviation of the period) and a deadline d.

Running example. *In the example, the application contains seven tasks {T1, T2, T3, T4, T5, T6, T7}.*

We note that, succ(T_1) = {{T_2, T_3}, {T_4}}, succ(T_5) = {{T_6}, {T_7}}.

To specify the causality between tasks, we define a trace tr as a tasks sequence,

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

such as,

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

In this paper, we classically focus on non reentry traces [4] : the execution of the $k-th$ instance of a trace cannot occurs before the execution end of the $(k-1)-th$ instance. Otherwise, the system is not feasible.

Running example. *In the example, we distinguish five traces $tr_1 = T_1, T_2; tr_2 = T_1, T_3; tr_3 = T_1, T_4; tr_4 = T_5, T_6$; and $tr_5 = T_5, T_7$.These traces specify all the possible application behaviors.*

Finally, we define an operation op_i as the set of traces having the same first task T_i. We characterize an operation op_i by the following tasks set,

$$op_i = \{T_m / T_i \in first \wedge T_m \in succ^*(T_i)\}$$

Running example. *In the example, we distinguish two operations $op_1 = \{tr_1, tr_2, tr_3\}$* and *$op_5 = \{tr_4, tr_5\}$.*

3.2 End to end delays

Let tr be a tasks trace of a control application. We classically define $delay(tr)$ as the end to end delay of the trace tr.

We define d the deadline of a task $T \in tr$. d has to take into account the time for executing all the successors belonging to $T.succ$ before their respective deadlines.

If $T \in last$, then

$$d = delay(tr)$$

Otherwise,

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

4 Tolerant schedulability analysis

According to the previous transformation, an operation op_i defines all the possible behaviors when T_i is activated. These behaviors are not foreseable off-line. Moreover, each execution of op_i must respect the corresponding traces delays. Therefore, we propose to define the tolerance on the operations.

Let define (m_i, k_i) the tolerance constraint for the operation op_i. We define that an operation occurrence meets its deadlines if all its possible traces meet their delays. The application is feasible if at least m_i among k_i ($m_i < k_i$) consecutives instances of op_i meet all their deadlines.

We propose a schedulability analysis taking advantage of such tolerances. This analysis is based on the construction of an accessibility graph in a hyper-period H [13].

The accessibility graph models all the possible trajectories of the application scheduling. Each trajectory represents a scheduling of application traces. A state of a trajectory contains a selected task instance to execute among all active ones. We apply the EDF policy to perform such selection. In particular, if an active instance $T_i^j \in op_h^j$

misses its deadline, then we evaluate in the op_h history the maximum number of ocurrences missing their deadlines. If the tolerance constraint (m_h,k_h) remains not violated, then we perform a back track in the graph to remove the occurrence op_h^j.

4.1 Evaluating the Hyper Period

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

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

We can exploit the result on the hyper period proposed for the schedulability analysis of asynchronous systems [7]. According to this result, the analysis is done in $[r_{min} + j_{min}; r_{max} + j_{max} + 2.lcm]$..

Let G be the accessibility graph to construct. We define a tasks state C of G as follows.

$$C = \{S, T_m^n, t\}$$

where,

- S: a set of tasks instances to execute.
- T_m^n: the selected instance between all active ones of S according to the EDF policy.
- t: the start time of the T_m^nexecution.

4.2 Accessibility graph construction

We construct the accessibility graph as follows. We apply for each state $C = \{S, T_m^n, t\}$ having no successor the following rules,

- **Rule 0 : stop condition.**

If $t > r_{max} + j_{max} + 2.lcm$, Then we stop the current trajectory construction.

- **Rule 1 : Constraints verification.**

If there exists an instance $T_i^j \in S$ ($T_i^j \in op_h^j$) missing its deadline, then the instance op_h^j is failed. In this case, we evaluate how many number of failed instances among the last k ones of op_h. Two cases occur,

> If such number exceeds m_h, then the constraint (m_h, k_h) of op_h is violated.
>
> Else, we cut all the graph states containing tasks instances of op_h^j.

- **Rule 2 : Construction of new states.**

We construct a following state for each set of $succ(T_m^n)$ (belonging to op_q^n). In particular, if T_m belongs to the $last$, then we start a new instance op_q^{n+1} of the operation op_q.

4.3 Formalization

In this part, we formalize the tolerant schedulability analysis of an IEC 61499 application.

Definition. We propose the following functions used later to perform the analysis. Let $C = \{S, T_m^n, t\}$ be a tasks state of G.

- $follow(C)$: defines the tasks state following C in the same trajectory.

$$follow(C) = C'$$

where

$$C' = \{S', T_q^p, t'\} / \exists T_k^h \in S', pred(T_k) = T_m$$

- $loose(op_i^n)$:defines the set of failed instances of op_i^n among the k_i last ones.

$$loose(op_i^n)$$

$$=$$

$$\{l \in [max\{0, n - k_i\}, n] / \exists T_p^l \in op_i^l, t + WCET(T_p) > d(T_p)\}$$

The algorithm applying the analysis is applied recursively as follows. The step 0 lets to construct the first tasks state of the graph. The step1 is applied recursively for the graph states without successors.

Step 0 : We denote by C_0 the first tasks state of G,

$$C_0 = \{S_0, T_{min}^0, r_{min} + j_{min}\}$$

The set of instances S_0 of C_0 is as follows,

$$S_0 = \{T_j^0 / T_j \in first\}$$

Step 1. Let $C = \{S, , t\} \in G / \neg follow(C)$ be a state of G having no successor. We distinguish four cases,

- If $t > r_{max} + j_{max} + 2.lcm$ then we stop the current trajectory construction (Rule 0).
- There exists a task instance T_j^n of S that cannot meet its deadline. Let denote by op_i^n the instance containing T_j^n.

$$\exists T_j^n \in S / C.t + WCET(T_j) > d(T_j)$$

If $card(loose(op_i^n)) \geq (k_i - m_i)$

Then the (m_i, k_i) constraint is violated

Otherwise, we remove the instance op_i^n from the graph G. Let denote by $C_j = \{S_j, T_i^n, t_j\} \in G$.

$$G = G \setminus follow^*(C_j)$$

$$S_j = S_j \setminus \{T_i^n\} \cup \{T_i^{n+1}\}$$

- Let T_p^q ($T_p \in op_h$) be the selected instance in S (according to EDF policy).

If T_p does not belong to the set $last$ ($T_p \notin last$),

Then we construct new tasks states following the current state C as follows.

Let suppose that $succ(T_p) = \{ts_0, ..., ts_{k-1}\}$

$G = G \cup \{C_i, i \in [0, k-1] / C_i = \{S_i, T_i, t_i\}\}$

Where,

** $S_i = S \setminus \{T_p^q\} \cup ts_i^q$

** $t_i = t + WCET(T_p)$

If T_p belongs to the set $last$ ($T_p \in last$),

Then we start the instance op_h^{q+1}.

$$G = G \cup \{C_i / C_i = \{S_i, T_i, t_i\}\}$$

Where,

** $S_i = S \setminus \{T_p^q\} \cup \{T_h^{q+1}\}$

** $t_i = t + WCET(T_p)$

We propose the following algorithm performing the tolerant analysis. This algorithm is based on the recursive function generate(). For sake of conciseness, we don't display the following functions used in the algorithm,

* Source (T_m^n): defines for an instance T_m^n of an operation instance op_h^n the class $C_k = \{S_k, T_h^n, t_k\}$.

* Free(C): deletes all the tasks classes constructed from the class C.

Let n be the number of operations to schedule. Let p_i be the traces number of the operation op_i ($i \in [0, n-1]$). Let q_i be the tasks number of the longest op_i trace. The complexity of the algorithm is with the order of $O(\alpha.\beta.\Phi)$ where $\alpha = \Pi_{j=0..n-1} p_j$ is the trajectories number in the accessibility graph, $\beta = \Sigma_{j=0}^{n-1} q_j$ is the longest trajectory and $\Phi = \Sigma_{j=0}^{n-1}(k_j - m_j)$ the maximum number of back tracks during the analysis.

Finally, the algorithm analyzes the schedulability by analyzing all the possible cases of the graph construction. If it deduces that the application is not schedulable then no other method can deduce the reverse. We deduce the following proposition.

Proposition. The tolerant schedulability analysis of an IEC 61499 control application is optimal.

Running the example. *We apply the proposed algorithm to analyse the schedulability of fbn . We propose the following temporal characteristics of T_1 and T_5 .*

Algorithm 1 Schedulability analysis

Bool generate(C : tasks_state, first : tasks_list, oper : operation_list, time : integer, L : instances_list)
Begin
T_1 : task; op : operation; L_1 : tasks_list; C_1 : tasks_state;
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 op $\leftarrow$ get_operation(T_1, oper);
    if (m_k_violated(op))
      then $L_1 \leftarrow$NULL; return false;
    else add_instance(T_1 , L);
if(L $\neq$ NULL) **then** return false;
C.T$\leftarrow$ apply_EDF(C.S);
while(ts $\in$ C.T$\rightarrow$ succ)
  create(C_1); C_1.S$\leftarrow$C.S\{C.T}$\cup$ ts ;
  $C_1.t \leftarrow C.t + T_2.WCET$; $L_1 \leftarrow NULL$;
  if(not generate(C_1 , first, operations, time, L_1))
    then L$\leftarrow$L$\cup$ L_1;
    for each T_k in L and source(T_k)=C
    ;
    if(T_k)
    then $T_1 \leftarrow$ root_op(T_k , C.S);
      // T_1 = root(op); op = op(T_k);
      T_1.r$\leftarrow$$T_1$.r+$T_1$.p ; free(C); L$\leftarrow$L \ {$T_1$};
      return(generate(C, first, operations, time, L));
    else return false;
return true;
End.

* r_1 =1, p_1 =30, j_1 =0.
* r_5 =2, p_5 =60, j_5 =0.

We suppose the following (m, k) constraints for op_1 and op_5 : $(m_1,\ k_1) = (1,\ 2)$ and $(m_5,\ k_5) = (1,\ 1)$.

We suppose the following end to end delays and worst case execution times,

Trace	Tr_1	Tr_2	Tr_3	Tr_4	Tr_5
delay	30	30	30	25	25

Task	T1	T2	T3	T4	T5	T6	T7
WCET	8	7	8	7	9	8	9

Therefore, the deadlines of the first tasks are as follows, $d_1 = 15$ and $d_5 = 18$.

By constructing the accsessibility graph in the hyper period [1,122] , the algorithm verifies all the application delays according to (m, k) constraints. The operation instance op_1^0 cannot respect all its delays. Indeed, the instance op_5^0 has priority according to the EDF policy.

Considering the constraint $(m_1,\ k_1)$ is not violated by the loose of op_1^0, we continue then the analysis by treating T_5^0 . We present a part of such graph (figure 4).

Finally, we conclude that all end to end delays are verified according to their (m, k) constraints. Indeed, we can

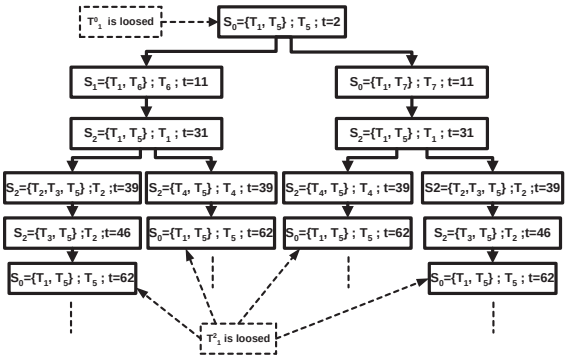

Figure 4: The accessibility graph

treat in the hyper period an instance of op_1 among two consecutive ones and we can also treat each instance of op_5. The application is feasible and we generate an off-line scheduling from the accessibility graph as proposed in [3]. This scheduling contains only tasks respecting their deadlines.

5 Conclusion

This paper proposes a contribution to develop an IEC 61499 control application. This contribution allows a controlled degradation of its behavior.

Supposing the application as a weakly hard real time system, we weaken its end to end delays thanks to the (m, k) model. We propose a tolerant schedulability analysis to verify such delays according to their (m, k) constraints. This analysis deducing by construction the application feasibility is optimal. If the application is schedulable, we generate an off-line scheduling containing all tasks meeting their deadlines.

We are currently working to find heuristics reducing the number of states in the accessibility graph in the order to reduce the combinatory explosion. Moreover, we are working to propose an on-line algorithm performing a non idling scheduling of such application.

On the other hand, we plan to propose hard and tolerant schedulability analyses of an IEC 61499 application distributed on several devices. Such extension imposes to take into account the communication interface inside each device and the network delays.

References

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

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

[3] Khalgui. M, Rebeuf. X, Simonot-Lion. F, "A schedulability analysis of an IEC 61499 control application". FET 05. Mexico. 2005.

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

[5] http://www.holobloc.com.

[6] http://www.ifak-md.de/wg7/.

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

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

[9] G. Bernat, A. Burns and A. Lamosi, Weakly-Hard Real-Time Systems, IEEE Transactions on Computers, 50(4), pp.308-321, April 2001.

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

[11] T. Blevins, Ch. Diedrich, F. Russo, L. Winkel,“Function block applications in control systems based on IEC 61804”. ISA transactions 43 (2004) 1-0.

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

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

[14] N.C. Audsley, A. Burns, M.F. Richardson, A.J. Wellings, "Hard Real-Time Scheduling: The Deadline-Monotonic Approach", Proceedings 8th IEEE Workshop on Real-Time Operating Systems and Software. Atlanta, USA. 1991.