A hybrid scheduling approach of centralized component based control applications

Mohamed Khalgui, Xavier Rebeuf, Françoise Simonot-Lion

A hybrid scheduling approach of centralized component 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 the IEC 61499 control applications. A function block (FB) is an event triggered component and an application is a centralized function blocks network. To validate its temporal behavior, we propose a hybrid scheduling approach combining off-line and on-line policies. The off-line policy allows the generation of pre-schedulings for the application. These pre-schedulings are transformed into recurring tasks. We verify then the on-line feasibility of these tasks using an existing schedulability condition.

Keywords. Component, Function Blocks, PLC, Real Time, Schedulability Analysis.

1 Introduction

The IEC 61499 standard [1] is a component-based methodology allowing to design control applications. A Function Block is an event triggered component [5] and a control application is a function blocks network. This application has to verify end to end delays according to specifications. In this paper, we suppose a centralized application in only one device.

According to the standard, a device contains one or more logic execution units called resources. A resource contains and serves application FBs interacting with one or more physical processes [1]. The standard imposes a non-preemptive execution between these blocks. Due to this restriction, a mutual exclusion on the interaction with these processes does not have to be explicitely handled.

On the other hand, the different resources have to be scheduled in the device. We have to apply a preemptive scheduling as required by several RTOS[8].

To satisfay all these requirements, we propose a scheduling approach combining off-line and on-line policies.

We first apply an off-line (non preemptive) policy to generate (if possible) a pre-scheduling of application FBs inside each resource. To apply this policy, these blocks are transformed into an actions system with precedence constraints [9]. At this step, we suppose that the resource gains 100% of the CPU. If such pre-scheduling is not feasible, then we conclude as soon as possible that the application is not also feasible [2, 3]. Note that such pre-scheduling will be used by a sequencer at run-time [7].

Once all the resources are feasible, we propose to consider them as OS tasks [8]. Regarding the conditional behavior of the application, we exploit the recurring task model to specify such OS tasks [6]. This model permits the representation of conditional real-time codes.

The application is then a set of OS tasks. We check their feasibility using a schedulability condition proposed in [6].

In the next section, we present the standard concepts. In the section 3, we present a temporal characterization of an IEC 61499 application. Then we present in the section 4 the scheduling approach.

2 The IEC 61499 standard

We present the main concepts of the IEC 61499 Function Blocks standard [1].

A function block (FB) is a functional unit of software supporting elementary 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. The implementation contains algorithms to execute when the corresponding events occur. The selection of the algorithm (when the corresponding input event occurs) is performed by a state machine called the execution control chart (ECC).

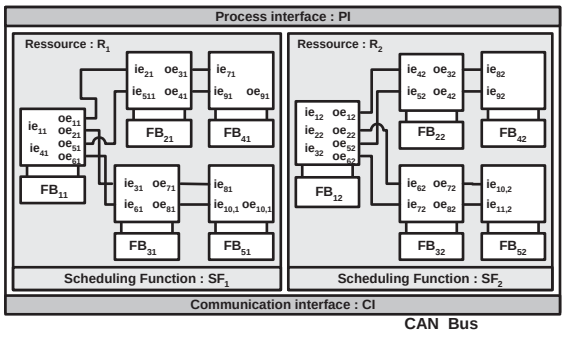

Figure 1: An IEC 61499 application

On the other hand, a control application is specified by a function blocks network. In [2], we described in detail the standard concepts.

To describe the execution support, a device is composed of one processing unit and interfaces (with sensors, actuators and the network). Moreover, each device is characterized by logical execution unit(s) called resource(s). A resource contains and serves application FBs : it defines "*the important boundary existing between what is within the scope of the IEC 61499 model and what is device (ie. OS) and networks (ie. communications protocols)*" [5].

Running Example. *We consider the following toy example of an IEC 61499 application distributed on two resources of a device (figure 1). We suppose that* ie_{22} *and* ie_{32} *are linked respectively to* oe_{91} *and* $oe_{10,1}$ *. We particularly present the behavior of the function blocks* FB_{11}*. When the ECC selects an* ie_{11} *(resp,* ie_{41}*) occurence, it asks the processor to perform the corresponding algorithm. When the scheduler signals the execution end, the ECC sends* oe_{11} *and* oe_{21} *(resp,* oe_{51} *or* oe_{61} *).*

3 Characterization of an IEC 61499 application

To validate its temporal behavior, we proposed in [2] a method transforming an IEC 61499 application into an elementary actions system with precedence constraints [9]. The purpose is to exploit classical results on the scheduling of real-time tasks.

An application action, denoted by *act*, corresponds to a FB algorithm activated by an input event *ie*. It is characterized by:

* *WCET(act)* (resp, *BCET(act)*) : the Worst (resp, Best) Case Execution Time of the algorithm.

* *pred(act)* : the action to execute before *act*. *pred(act)* belongs to the FB producing the output event connected to *ie*.

* *succ(act)* : a set of actions sets. Each actions set corresponds to a possible execution scenario (ie. only one actions set between all ones is performed). The actions of a set have to be executed once the execution of *act* is finished. These actions belong to FBs activated once the treatment corresponding to *ie* finishes. Note that *succ(act)* is constructed thanks to a static analysis of the ECC.

In all the continuation, we denote by Σ (resp σ) the actions set of the application (resp in a resource R). We denote by $first(\sigma)$ (resp $last(\sigma)$) the actions set with no predecessors (resp successors) in σ. We denote also by act_i^j the j-th instance of the action $act_i \in \Sigma$.

In this paper, we suppose periodic input events of the application. Therefore, each action *act* belonging to $first(\Sigma)$ is activated periodically. We characterize such action by a release time r , a period p , a jitter j (the maximum deviation of the period) and a deadline d.

To respect end to end delays, we proposed in [2] a method processing deadlines for the different actions of the application. Moreover, we proposed in [4] their temporal characterization.

4 Scheduling approach

In this section, we present the scheduling approach of an IEC 61499 application combining off-line and on-line policies. This approach remains compliant with the standard while allowing a preemptive scheduling between resources. Moreover, it reduces the context switching during execution. Such reduction is often needed by several RTOS regarding their restriction in the tasks number to schedule [8].

4.1 Pre-scheduling step

To validate the temporal behavior of a resource R, we apply a schedulability analysis of its application FBs [2, 3]. This analysis is based on a construction, in a hyper period, of an accessibility graph. This hyper period is composed by two behavioral modes : *the non-stationary* mode and the *stationary* one. Note that the stationary mode will occur periodically[2, 3].

If the resource is feasible, we generate a static scheduling as a direct acyclic graph (DAG). In this graph, each trajectory specifies a possible execution scenario. A state of the graph specifies the execution start time of an action instance [2, 3].

In this paper, we consider the generated static schedulings as pre-schedulings [7]. Indeed, the execu-

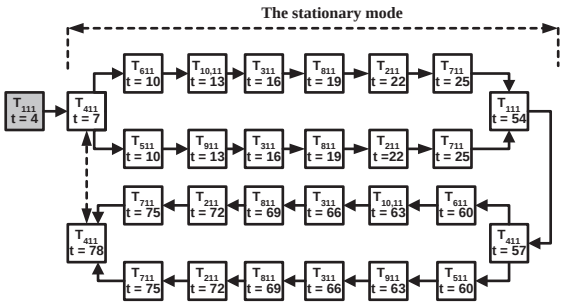

Figure 2: A pre-scheduling

tion of a pre-scheduling may be preempted to execute a pre-scheduling of another resource.

Let S_R be the pre-scheduling of a resource R. We denote by $successor(act_i^j)$ (resp $predecessor(act_i^j)$) the instances set following (resp preceding) the instance act_i^j in S_R.

Running example. *We take as example the resource R_1. We check the feasibility of the corresponding set of actions. We generate a pre-scheduling depicted in the figure . This pre-scheduling is a DAG where each state specifies an action instance to execute.*

4.2 Transformation into OS Tasks

We propose in this section a method transforming the generated pre-schedulings into OS tasks.

Regarding their conditional structure, we exploit the recurring real-time task model to specify these pre-schedulings [6]. This model was introduced to represent conditional real time codes [6]. The application becomes then a set of OS tasks.

A recurring task Γ is represented by a task graph $G(\Gamma)$ and a period $P(\Gamma)$. The task graph $G(\Gamma)$ is a direct acyclic graph (DAG) with a unique source vertex (denoted by τ_0) and a unique sink vertex. Each vertex of this DAG represents a subtask (denoted by τ) and each edge represents a possible flow of control.

A vertex of Γ is characterized by a WCET and a deadline d. An edge (u, v) is characterized by a real number p(u, v) denoting the minimum amount of time that must elapse after vertex u is triggered and before vertex v can be triggered.

For sake of clarity, we encode the graph structure using the following functions,

- pred(τ) : a set of subtasks such as only one has to be executed before τ in Γ.
- succ(τ) : a set of subtasks such as only one has to be executed once the execution of τ is finished.

We propose to transform the pre-scheduling S_R into two recurring tasks Γ and Γ'. The task Γ implements the stationary behavior whereas the task Γ' implements the non-stationary one.

Regarding that the stationary behavior is periodic, the corresponding recurring task Γ is also periodic with the same period. We denote by $t(\tau)$ the triggering-time of the subtask τ (the task Γ is activated at $t(\tau_0)$).

A transformation solution is to associate each subtask to an instance of an action. Nevertheless, this transformation produces recurring tasks with a lot subtasks. This solution increases the complexity of the schedulability analysis [6]. Therefore, we propose to merge a sequence of actions instances into a unique subtask.

Let $traj(S_R)$ be a trajectory of S_R. We implement a subtask τ of Γ as follows,

$$\tau = act_0^e, ..., act_{k-1}^f$$

such as,

- $\forall i \in [0, k-1]$, $act_i^h \in traj(S_R)$.
- $\forall i \in [0, k-2]$, $succ(act_i^h) = \{\{act_{i+1}^l\}\}$.
- $act_{k-1} \in last(\sigma)$ or $cardinality(successor(act_{k-1}^f)) \geq 1$.

To verify all delays during the feasibility analysis of the OS tasks, an instance act_m^n ($act_m \in last(\sigma)$) must be a last instance of a subtask τ. According to the EDF policy, the deadline of τ is then the deadline of act_m^n. We denote by $firstsub(\tau)$ (resp $lastsub(\tau)$) the first (resp last) instance of the subtask τ.

Let $firstact(\Gamma)$ be the instances set of S_R with no predecessors to execute in the stationary mode. We propose the following rules to construct the task Γ. The first rule constructs the first subtask, whereas the second one is applied recursively to construct the other subtasks.

Rule 0. First subtask construction.

If $cardinality(firstact(\Gamma)) = 1$ ***then*** $\{\tau_0\} = firstact(\Gamma)$.

Otherwise*, we construct in $G(\Gamma)$ a virtual subtask τ_0 as follows,*

- $WCET(\tau_0) = 0$.
- *For each state $act_i^j \in firstact(\Gamma)$, we construct a subtask τ_k such as $(\tau_0, \tau_k) \in G(\Gamma)$ and $p(\tau_0, \tau_k) = 0$.*

Rule 1. Subtasks construction.

Let $\tau_i \in \Gamma$ be a subtask with no successors yet such as the last instance is with successors in S_R,

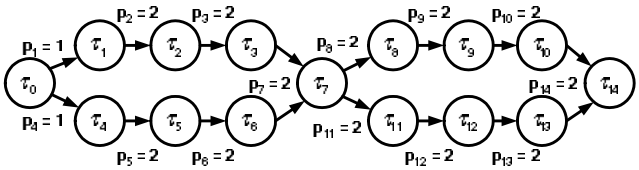

Figure 3: A recurring task

$\tau_i \in \Gamma / succ(\tau_i) = \emptyset \wedge successor(lastsub(\tau)) \neq \emptyset$,

We construct for each instance $act_0^q \in successor(lastsub(\tau_i))$ *a new subtask* $\tau_j \in succ(\tau_i)$ *such as* $first(\tau_j) = act_0^q$. *We suppose that* $\tau_j = act_0^q, ..., act_k^p$. *This new subtask is temporally characterized as follows,*

- ***The ready time*** $t(\tau_j)$ *is equal to the execution start time of* act_0^q *(processed in the pre-scheduling):* $t(\tau_j) = t(act_0^q)$
- ***The minimum amount of time*** $p(\tau_i, \tau_j)$ *is equal to the difference between the triggering times of* τ_j *and* τ_i *:* $p_j = t(\tau_j) - t(\tau_i)$
- ***The deadline*** d_j, *corresponds to the deadline* d_k *of the instance* act_k^p.
- ***The execution requirement*** $WCET(\tau_j)$ *is the sum of the WCET of the actions implementing* τ_j,

We note finally that we follow the same method to construct Γ'.

Running example. *We propose the OS task* Γ *to abstract the behavior of the resource* R_1 *(figure 3). The instances of actions* act_{611} *and* $act_{10,11}$;act_{311} *and* act_{811}; act_{211} *and* act_{711} *are respectively in the subtasks* τ_1, τ_2, τ_3.

4.3 Feasibility analysis

Once all the pre-schedulings are transformed into OS tasks, we analyze their feasibility using a schedulability condition proposed in [6]. We apply this condition in a well defined hyper-period. If it is satisfied, we conclude the feasibility of the application.

Note that in practice, [6] proposes an interesting technique to compute such hyper period.

5 Conclusion

This paper proposes an approach to validate the temporal behavior of an application running on a programmable logic controller. This approach combines off-line and on-line policies.

The off-line policy allows to validate the internal behavior of each resource. This validation generates a pre-scheduling as a DAG.

Once all resources are feasible, we propose to transform the corresponding pre-schedulings into OS tasks. We exploit the recurring task model to specify these tasks. To apply an on-line scheduling of these OS tasks, we apply an existing schedulability condition validating their feasibility.

In our future works, we plan to generalize this approach to take into account the distribution on several devices. Such extension imposes to take into account the communication interface inside each device and the networks delays. In addition, we plan also to propose a method deploying a such application in several devices. This method must be based on "placement" heuristics.

References

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

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

[3] Khalgui. M, Rebeuf. X, Simonot-Lion. F, "A tolerant temporal validation of components based applications". 12th IFAC Symposium on Information Control Problems in Manufacturing. France. 2006.

[4] Khalgui. M, Rebeuf. X, Simonot-Lion. F, "A schedulability condition of an IEC 61499 control application with limited buffers". OMER3. Paderborn. Germany. 2005.

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

[6] Sanjoy Baruah. Dynamic and static priority scheduling of recurring real-time tasks. Real-time Systems. 24 (1), pp. 93-128. 2003.

[7] W. Wang, A.K. Mok,"Pre-Scheduling: Balancing Between Static and Dynamic Schedulers", UTCS Technical Report RTS-TR-02-01.

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

[9] Stankovic. J, Spuri. M, Ramamritham. K, "Deadline Scheduling for Real-Time Systems ". Publisher: Kluwer Academic Publishers. ISBN : 0792382692.