Component based deployment of industrial control systems: a hybrid scheduling approach

Mohamed Khalgui, Xavier Rebeuf, Françoise Simonot-Lion

▶ **To cite this version:**

Mohamed Khalgui, Xavier Rebeuf, Françoise Simonot-Lion. Component based deployment of industrial control systems: a hybrid scheduling approach. 11th IEEE International Conference on emerging Technologies and Factory Automation, Sep 2006, Prague, République Tchèque. inria-00113460

HAL Id: inria-00113460

https://inria.hal.science/inria-00113460

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.

Component based deployment of industrial control systems : a hybrid scheduling approach

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

Abstract

This paper deals with the IEC 61499 standard. A function block (FB) is an event triggered component and an application is a distributed FBs network on several devices. We consider these devices as multi-tasking PLCs. To validate the temporal behavior inside a device, we propose a hybrid scheduling approach combining off-line and on-line policies. This approach transforms application blocks into tasks thanks to the off-line policy. Then, it checks their required on-line feasibility. On the other hand, we propose also a temporal charcaterization of exchanged messages between devices to check also their online feasibility.

Keywords. Component, Function Blocks, PLCs, Real Time, Scheduling.

1 Introduction

Nowadays, the development of safety control applications becomes more and more complex. Indeed, they have to be certified with regard to functional and extra-functional properties [1]. Since they control critical processes, one of the most important property deals with Real Time behavior.

In addition, the "Time to market" delay between the application design and its commercialization is often shorter [2].

A way to address these constraints can be the reuse of existing components. The application is viewed then as a composition of components.

Several component based approaches have been proposed to model the components composition and also the execution support [2, 3]. Among these approaches, the IEC 61499 standard is the most known one in the industry [4, 5]. It provides wide librairies of existing software components called Function Blocks (FBs) [6].

The Function Block is an event triggered component. It is a functional unit of software owning data. A control application is a FBs network specifying dependencies between these blocks at run-time [5].

The execution support is a network of devices [6]. A device contains logical execution units called resources. A resource gathers application blocks interacting with one or more physical processes. The standard imposes a non-preemptive execution between these blocks. Due to this restriction, a mutual exclusion on such interactions does not have to be explicitely handled.

The application deployment is then a FBs distribution on several resources of devices. According to specifications, these blocks have to verify end to end response time bounds (denoted by eertb). These bounds are between the receive of stimulus from sensors and the activation of the corresponding actuators [7].

Several works have been proposed to validate the temporal behavior of a FBs network [8, 9]. Nevertheless, they don't take into account characteristics of the execution support.

In previous works, we considered the application deployment on a usual industrial equipment: the mono-tasking Programmable Logic Controller. This execution support cannot handle preemption. Therefore, we can model this execution support as a mono-resource device.

In [10, 11], we proposed a method constructing a static scheduling of FBs satifying the eertb constraints. Such scheduling is a Direct acyclic Graph (DAG) where each node is a FB execution (with its own activation date and its own deadline) and each edge is a control flow. In this DAG, a fork corresponds to a condition according to the global state of the resource. Note that this scheduling corresponds to an idling policy.

Nowadays, a new generation of a multi-tasking PLC is available. Its main feature is the on-line preemptive scheduling of tasks. Therefore, we can model this execution support as a multi-resources device. In this case, we propose to associate one task to one resource.

To validate the temporal behavior of a distributed application, we have to check the feasibility in each device and also on the network connecting them.

To validate the temporal behavior inside each device, we propose a hybrid approach combining off-line and on-line policies.

This approach has first to transform blocks of each re-

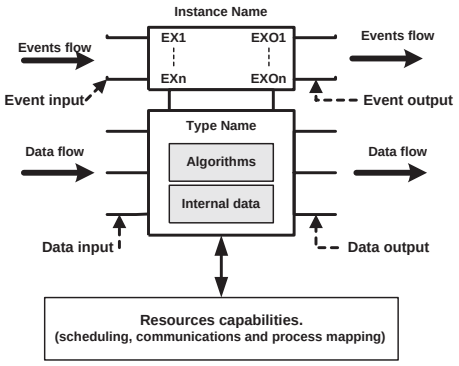

Figure 1. A function Block.

source into a task. It applies, in particular, the proposed approach that constructs their static scheduling [10, 11]. This scheduling can be viewed as a pre-scheduling [12]. To encode the resource task, we propose to exploit the recurring real-time task model [13]. A recurring task is a DAG allowing the representation of conditional real-time codes.

Once all the resources of the device are correctly transfomed into recurring tasks, we have to check their on-line feasibility. We propose to apply the schedulability condition in [13].

To check the on-line feasibility on the network, we propose a temporal characterization of exchanged messages between devices. This characterization is based on the execution of the application blocks. It allows to apply any schedulability condition according to the fixed scheduling policy on the network [14, 15].

In the next section, we present the standard, the proposed extensions and the considered assumptions. Then, we present a characterization of an IEC 61499 application. The section 4 presents the hybrid approach to apply in each device. Finally, the section 5 presents the temporal validation in a network of devices.

2 The IEC 61499 standard

We present the main concepts of the IEC 61499 Function Blocks standard [4, 5]. To validate the temporal behavior of the application, we propose some extensions and assumptions to take into account in all the continuation.

2.1 Concepts

A function block (FB) (figure 1) is a unit of software providing functionalities of an IEC 61499 application [5]. It is composed by an interface and an implementation.

The interface contains data/event inputs and outputs supporting interactions with the environment. Events are responsible for the activation of the FB while data contain valued information [5].

The implementation of the block contains algorithms to execute when the corresponding events occur [5].

The selection of an algorithm to execute is performed by a state machine called the execution control chart (ECC). The ECC is also responsible for sending output events at the end of the algorithm execution. In [10, 11], we describe in detail the FB behavior.

In the standard, a control application is specified by a FBs network. In this network, each FB event input (resp. output) is linked to an event output (resp. input) by a channel. Otherwise, it corresponds to a global input (resp. output). Data inputs and outputs follow the same rules [5].

The execution support architecture is defined by a devices network. A device is composed of one processing unit and interfaces (with sensors, actuators and the network). Moreover, it 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)" [6].

2.2 Extensions and assumptions

To analyze the behavior of a FB, we characterize the corresponding algorithms by worst (resp, Best) case execution times WCET (resp, BCET). Moreover, we consider that output events can be sent (by the ECC) simultaneously or in exclusion according to specifications.

To validate the temporal behavior of the application, we only focus on the events flow. We suppose, in all the continuation, a complete synchronization between events and data flows. Indeed, when an event occurs in the corresponding input, all the associated data occur at the same time in the corresponding inputs.

In this paper, we consider periodic global input events. According to [16], we characterize them by a release time r , a period p and a jitter j (the maximum deviation of the period).

In this paper, we consider a distributed application on D devices connected by a network : $device_0 ..., device_{D-1}$. we suppose r_i resources in each device $device_i$, $i \in [0,\ D-1]$.

On the other hand, we don't restrict our researches to a particular network. For sake of simplicity, we suppose a bus network connecting the different devices. We denote by M the set of messages to exchange on such network. We apply a non-preemptive policy to schedule them.

Running Example. *For all the continuation, we use the following toy example to explain the proposed approach.*

We consider a distributed control application on several devices connected by a network. We show in the figure 2 a part of this application in a device of the execution support.

This device is composed of two resources containing applications blocks. These blocks interact with physical processes and also with other blocks located in other devices.

We consider ie_{11} and ie_{41} as global input events of the application. In addition, the input events ie_{22} and ie_{32} correspond respectively to oe_{91} and $oe_{10,1}$.

We particularly present the behavior of the function blocks FB_{11} and FB_{12} .

When the ECC of FB_{11} 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}).

When the ECC of FB_{12} selects an ie_{32} occurence, it asks the processor to perform the corresponding algorithm. When the execution ends, the ECC sends oe_{52} and oe_{62}.

We simplify the problem by supposing that $WCET = 3$ and $BCET = 1$ of all algorithms. In addition, we suppose the following temporal characterizations, $ie_{11} : (r = 3, j = 1, p = 50)$ and $ie_{41} : (r = 6, j = 1, p = 50)$.

3 Characterization of an IEC 61499 application

To validate the temporal behavior, we proposed (in [10, 11] to transform a FBs network into a particular actions system with precedence constraints [14]. The purpose is to exploit classical results on the scheduling of real-time tasks.

This system is different from all those proposed in other researches. It allows to specify all the possible execution scenarios of the application.

3.1 Action

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 corresponding to ie.
- $pred(act)$: the action to execute before act. $pred(act)$ belongs to the FB producing the output event corresponding 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.
- (r, j, p, d) : The three first parameters characterize the activation of act [16]. They should be processed while taking into account the execution of $pred(act)$ [17]. The deadline d defines the latest completion date of the execution. To respect $eertb$ constraints, it should be processed while taking into account the deadlines of the successors of act [17].

succ(act_{11})	succ(act_{41})	succ(act_{32})
{act_{21}, act_{31}}	{{act_{51}}, {act_{61}}}	{act_{52}, act_{72}}

Running example. *Based on the blocks behavior presented above, we present the successors of the following actions. At the end of act_{11} execution (resp act_{41}) , the actions act_{21} and act_{31} (resp act_{51} or act_{61}) are activated.*

For all the continuation, we denote by act_i^j the j-th instance of the action act_i. Let Σ be the set of the application actions. We denote by σ a subset of Σ. Note that this subset corresponds to a resource R of a device. We denote by first(σ) (resp last(σ)) the set of actions with no predecessors (resp successors) in σ.

To validate the temporal behavior of the application, we propose to temporally characterize its actions. Nevertheless, the execution of some of them is not statically foreseeable. Indeed, it depends on the execution of their predecessors. In this case, we say that these actions are not principal.

An action act$\in \Sigma$ is principal if it is periodically executed : the set of the successors of all its predecessors contains only one actions set.

$\forall act_j \in pred^*(act_i), cardinality(succ(act_j)) = 1$

Note that the function $pred^*()$ denotes the transitive closure.

Running example. *In the example, the actions act_{21} and act_{31} are principal because they have to be executed each time act_{11} is executed. The actions act_{51} and act_{61} are not principal because their execution is not foreseeable : it depends on the execution of act_{41}.*

In several cases, we say that application actions act_h and act_k of $first(\sigma)$ are disjoined if they have to be executed in exclusion.

$$\exists act_i \in \Sigma / \forall s \in succ(act_i)$$

$$s \not\subset pred^*(act_h) \cap pred^*(act_k)$$

We note that two disjoined actions are obviously not principal. To optimize the static scheduling of σ , we propose to construct a new virtual action act_{null} as a predecessor of act_h and act_k such as,

- $succ(act_{null}) = \{\{act_h\}, \{act_k\}\}$
- $WCET(act_{null}) = 0$

The new action act_{null} is then principal. It is activated periodically.

Running example. *In the resource R_2, the actions act_{22} and act_{32} are disjoined. Indeed, their execution depends on that of act_{41}. To optimize the static scheduling of R_2 , we construct a new action act_{null} as depicted in figure 3.*

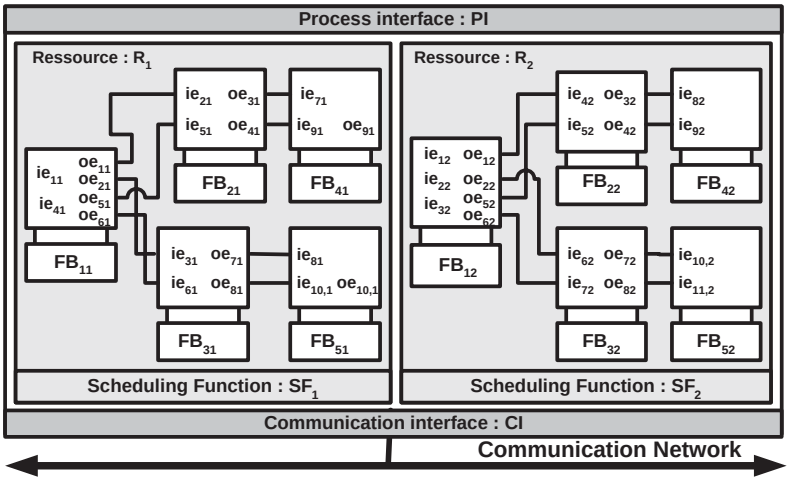

Figure 2. A function Block.

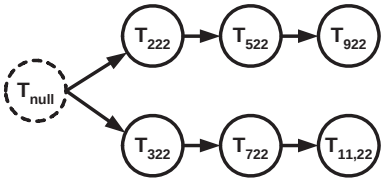

Figure 3. The fusion of two resource operations.

3.2 Actions trace

Considering the precedence constraints between actions, we define a trace tr of σ the following sequence,

$$tr = act_0,\ act_1 ....,\ act_{n-1}$$

such as,

- $\forall act_i \in]1, n-1], act_{i-1} = pred(act_i)$.
- $act_0 \in first(\sigma)$ and $act_{n-1} \in last(\sigma)$.

The trace tr represents a possible execution of σ. It corresponds to an $eertb$ constraint according to specifications [17].

We denote by $first(tr)$ (resp $last(tr)$) the first (resp last) action of tr. In this paper, we suppose non reentry traces [18] : the execution of the k-th instance cannot be done before the execution end of the (k-1)-th one.

Running example. *Let us take as example the resource R_1. We distinguish four traces as follows,*

$$\begin{aligned} tr_1 &= act_{11}, act_{21}, act_{71} \\ tr_2 &= act_{11}, act_{31}, act_{81} \\ tr_3 &= act_{41}, act_{51}, act_{91} \\ tr_4 &= act_{41}, act_{61}, act_{10,1} \end{aligned}$$

4 Hybrid validation of a device

In this section, we are interested to validate the temporal behavior of application blocks in only one device.

According to the standard instructions [4], we have to apply a non-preemptive policy to schedule actions inside resources. Nevertheless, an on-line and preemptive policy is often required by RTOSs of devices [19].

To meet these requirements and to remain also adaptable to the standard, we propose a hybrid approach combining off-line and on-line policies.

The off-line policy (i.e non-preemptive) allows to validate the temporal behavior in each resource. If it is unfeasible, then we conclude as soon as possible that the application is also unfeasible. Otherwise, we generate a pre-scheduling [12] encoding all possible execution scenarios.

Considering its conditional structure, we transform each pre-scheduling into an OS task [19]. The application is considered then as a set of OS tasks in the device. We exploit an existing schedulability condition [13] checking their on-line feasibility.

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 considering their restriction in the number of tasks to schedule [19].

4.1 Pre-scheduling step

To validate the temporal behavior of σ (a set corresponding to a resource R), we apply a schedulability analysis based on the EDF policy [10, 11]. This analysis constructs an accessibility graph in a hyper period hp.

Let lcm be the least common multiple of the periods of actions in $first(\sigma)$. Let $act_{max} = \{r_{max},\ j_{max},\ p_{max}\}$ and $act_{min} = \{r_{min},\ j_{min},\ p_{min}\}$ be two actions of $first(\sigma)$ such as,

$$\forall act_i \in first(\sigma)$$

$$r_{min} + j_{min} \leq r_i + j_i \leq r_{max} + j_{max}$$

Based on the work proposed in [20], the analysis is done in $hp = [r_{min} + j_{min}, r_{max} + j_{max} + 2.lcm]$. In this hyper period, we distinguish two behavioral modes : the $stationary$ mode and the $non - stationary$ one.

The $non - stationary$ mode corresponds to the resource behavior during $[r_{min} + j_{min}; r_{max} + j_{max}]$. The stationary one corresponds to the behavior during $[r_{max} + j_{max}; r_{max} + j_{max} + 2.lcm]$. This behavior is performed periodically.

If the system σ is feasible, then we generate a static scheduling S_R to use by a sequencer at run time.

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 $O(n^2)$.

The static scheduling is a direct acyclic graph (DAG) where each trajectory specifies a possible execution scenario. A state of the graph specifies the execution start time of an action instance [10, 11].

We denote by $stat\_succ(act_i^j)$ (resp $stat\_pred(act_i^j)$) the set of instances following (resp preceding) the instance act_i^j in S_R.

In this paper, we consider a static scheduling as a pre-scheduling [12]. Indeed, the execution of a pre-scheduling may be preempted to execute a pre-scheduling of another resource.

Running example. *We take as example the resource* R_1.

Applying the algorithm proposed in [10], we check the feasibility of the corresponding set σ_1*. We generate a pre-scheduling depicted in the figure 4. This pre-scheduling is a DAG where each state represents an instance of an action to execute.*

4.2 Transformation into OS Tasks

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

considering the conditional structure of these pre-schedulings, we exploit the recurring task model to represent them [13]. This model was introduced to represent conditional real time codes. The application is considered then as a set of OS tasks distributed on different devices.

A recurring task Γ is characterized 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 (τ , τ') is characterized by a real number $p(\tau, \tau')$ denoting the minimum amount of time that must elapse after vertex τ is triggered ($t(\tau)$) and before vertex τ' can be triggered ($t(\tau')$).

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

- $pred(\tau)$: a set of subtasks in Γ such as only one subtask has to be executed before τ.
- $succ(\tau)$: a set of subtasks in Γ such as only one subtask has to be executed once the τ execution finishes.

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. considering that the stationary behavior is periodic, the corresponding recurring task Γ is also periodic with the same period.

A possible transformation 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 [13]. Therefore, we propose to merge a sequence of instances of actions into a unique subtask.

To verify all bounds during the feasibility analysis of these OS tasks, an instance $act_m^n \in S_R$ such as $act_m \in last(\sigma)$ 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 implement a subtask τ of Γ as follows,

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

- $\forall\, i \in [0, k-2]$, $stat\_succ(act_i^h) = \{act_{i+1}^l\}$.
- act_{k-1} is as follows,

$$act_{k-1} \in last(\sigma) \text{ or}$$
$$cardinality(stat\_succ(act_{k-1}^f)) > 1$$

We denote by $first(\tau)$ (resp $last(\tau)$) the first (resp last) instance of the subtask τ.

Let $first\_sub$ (Γ) be the set of instances in 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 $cardinality(first\_sub(\Gamma)) = 1$

Then $\{ \tau_0 \} = first\_sub(\Gamma)$

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

- $WCET(\tau_0) = 0$.
- For each state $act_i^j \in first(\Gamma)$, we construct a subtask τ_k such as (τ_0, τ_k)$\in G(\Gamma)$ and $p(\tau_0, \tau_k) = 0$.

We note that $t(\tau_0) = min\{r(act_i^j), act_i^j \in first\_sub(\Gamma)\}$

Rule 1. Subtasks construction.

Let τ_i be a subtask of Γ such as $act_0^q \in stat\_succ(last(\tau_i))$.

We denote by $\tau_j \in succ(\tau_i)$ a new subtask as follows,

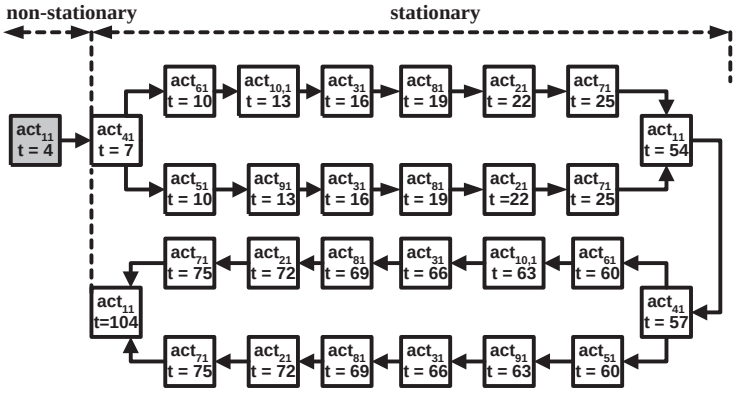

Figure 4. A pre-scheduling

$$\tau_j = act_0^q, ..., act_k^p.$$

This new subtask is temporally characterized as follows,

- **The ready time** $t(\tau_j)$ is characterized as follows,

$$t(\tau_j) = max\{r(act_0^q); max_{\tau_i \in pred(\tau_j)}\{t(\tau_i) + \sum_{act_p^q \in \tau_i} BCET(act_p)\}\}$$

- **The minimum amount of time** $p(\tau_i, \tau_j)$ is equal to the difference between the triggering times of τ_j and $\tau_i : p_j = t(\tau_j) - t(\tau_i)$.
- **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 WCETs of the actions implementing τ_j.

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

Running example. *In the example, we transform the pre-scheduling of R_1 into a recurring task Γ_1 (figure 5).*

This task contains trajectories of subtasks specifying the possible execution scenarios in R_1. Each subtask contains instances of S_R.

The actions (act_{611}, $act_{10,11}$); (act_{311}, act_{811}) and (act_{211}, act_{711}) are respectively transformed into the subtasks τ_1,τ_2 and τ_3 of Γ_1. We note that,

$$t(\tau_0) = r(act_{41}^1) = 6$$
$$t(\tau_0) = r(act_{11}^2) = 54$$

4.3 Feasibility analysis

The application is considered then as set of OS tasks in the device. We propose to analyze their preemptive online feasibility. We exploit a schedulability condition to check such feasibility [13].

According to the last transformation, the scheduling problem in the device $device_i$, $i \in [0, D-1]$ is the scheduling problem of at most $2 * r_i$ OS tasks.

Exploiting the theorem proposed in [13], we check the feasibility of these tasks in a fixed hyper-period,

$$hp = [0, \frac{\sum_{\Gamma \in S} 2.E(\Gamma)}{1 - \sum_{\Gamma \in S} \rho_{ave}(\Gamma)}] \text{ where,}$$

- S : the set of all recurring tasks in the device,.
- $E(\Gamma)$: the maximum possible cumulative execution requirement on any path from the source node to the sink node of the task graph $G(\Gamma)$.
- $\rho_{ave}(\Gamma)$: the quantity $E(\Gamma)/P(\Gamma)$.

The theorem indicates that the system S is feasible if and only if,

$$\forall t \in hp\,, (\sum_{\Gamma \in S} \Gamma.dbf(t) \leq t))$$

where, $\Gamma.dbf(t)$ is a function accepting as argument a non negative real number t. This function processes the maximum cumulative execution requirement by jobs of Γ having both ready times and deadlines within any time interval of duration t.

Note that in practice, [13] proposes an interesting technique to compute this function in the hyper period hp.

5 Validation of a devices network

In this section, we treat the general case. We suppose a distributed application on D devices : $device_0$....,$device_{D-1}$.

To guarantee the temporal correctness, we have to check the feasibility inside each device and also on the network.

We apply the hybrid approach for each device $device_i$, $i \in [0, D-1]$ to check the feasibility of the internal application blocks.

To validate the temporal behavior on the network, we have to check the feasibility of the exchanged messages between devices.

In [15], the authors propose a method checking the feasibility of exchanged messages on a CAN network. They suppose a deadline for each message to exchange on such bus. A non-preemptive EDF policy is applied to check such feasibility. We precise that they propose

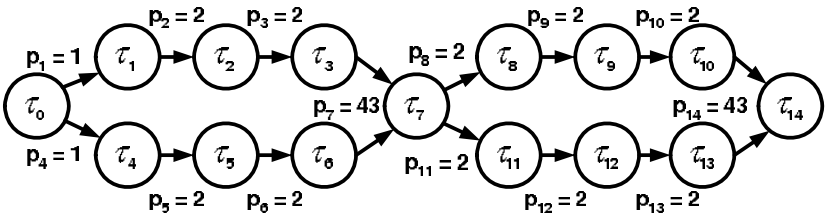

Figure 5. The recurring task Γ_1.

a method processing messages ID from the corresponding deadlines.

In this paper, we don't restrict our researches to a particular network. In addition, we suppose this network as a virtual device.

The exchanged messages between devices have to respect $eertb$ constraints according to specifications. To guarantee the temporal correctness of the application, we propose to assign for each message a deadline. This deadline is the latest time to reach the target device.

If a message deadline is not respected, then an $eertb$ constraint cannot be respected. To just check their feasibility, we propose a temporal characterization of these messages. This characterization is based on our previous works [17].

Let act_k be an action of $device_k$ with successors in other devices,

$$\exists act_h \in \Sigma / act_k = pred(act_h) \wedge act_h \notin \sigma(device_k)$$

We denote by m the message sent from act_k to act_h. According to our previous researches in [17], we characterize m as follows,

$$m = \{r,\, j,\, d,\, p,\, WCTT\}$$

where,

- $r = r_k + BCET_k$: the earliest sending date occurs when the execution of act_k finishes as soon as possible.
- $j = d_k - BCET_k$: the latest sending date occurs when the execution of act_k finishes in time. The difference between the earliest and the latest date corresponds to the jitter.
- The deadline d has to take into account the time for executing all the successors of act_k before their deadlines.

$$d = min_{s \in succ(act_k)}\{min_{act_i \in s \wedge act_i \notin \sigma(device_k)}\{d(act_i) - \sum_{act_j \in s \wedge act_j \notin \sigma(device_k)}^{d(act_j) \leq d(act_i)} WCET(act_j)\}\}$$

- $p = p(act_k)$: As we treat non-reentry traces, then the message m has the period of act_k.
- $WCTT$: the worst case transmission time on the network. This time depends on the network performance.

By considering this characterization, all the exchanged messages can be supposed as independant tasks. We can use then an existing schedulability condition to check their feasibility [14, 15].

6 Conclusion

This paper proposes a temporal validation of a distributed IEC 61499 control application on a network of devices. These devices are multi-resources. We consider each resource as an OS task.

To validate the temporal behavior, we have to check the feasibility in each device and also on the network connecting them.

To check the feasibility in a device, we propose a hybrid approach combining off-line and on-line policies. This approach has first to transform blocks of each resource into a pre-scheduling. We propose to exploit the recurring real-time task model to encode this pre-scheduling [13].

Once all the device resources are correctly transfomed into recurring tasks, we have to check their on-line feasibility using a schedulability condition [13].

To validate the network behavior, we check the feasibility of the exchanged messages between devices. We propose a temporal characterization of these messages based on the FBs execution. This characterization allows to apply an existing schedulability condition checking their on-line feasibility [14, 15].

In our future works, we plan to consider a not deployed application on devices. Therefore, we have to propose a method deploying its blocks in resources of devices.

Based on "placement" heuristics, the deployment has to take into account functional and temporal constraints according to specifications. Moreover, the number of resources to construct in a device should not exceed the number of allowed OS tasks. Finally, we plan to propose a method configuring these resources to provide better services for local blocks.

References

[1] O. Defour, J.-M. Jzquel, and N. Plouzeau, "Extra-functional contract support in components", in *International Symposium on CBSE7*, 2004.

[2] I. Crnkovic and M. Larsson, *Building reliable component-based software systems*, Artech House. London, 2002.

[3] B.-P. Douglass, "Real Time UML : Advances in the UML for Real-Time Systems", in *3rd Edition, ISBN : 0321160762, Addision : Wesley Professional*.

[4] IEC61499, *International Standard IEC TC65 WG6*, IEC, 2004.

[5] K. Thramboulidis, "IEC61499 in Factory Automaton", in *International Conference on Industrial Electronics, Technology and Automaton*, 2005.

[6] R. Lewis, *Modelling control systems using IEC 61499*, The institution of Electrical Engineers, 2001.

[7] R. Gerber, M. Saksena, and H. Hong, "Guaranteeing End-to-End Timing Constraints by Calibrating IntermediateProcesses", in *IEEE Real-Time Systems Symposium*, 2004.

[8] M. Stanica and H. Guguen, "A Timed Automata Model of IEC 61499 Basic Function Blocks Semantic", in *ECRTS'03*, 2003.

[9] V. Vyatkin and H.-M. Hanish, "Modeling of IEC61499 Function Blocks A Clue To Their Verification", in *XI Workshop on supervising and diagnostics of Machining Systems. Wroclaw*, 2000.

[10] M. Khalgui, X. Rebeuf, and F. Simonot-Lion, "A schedulability analysis of an IEC 61499 control application", in *Fet. Mexico*, 2005.

[11] M. Khalgui, X. Rebeuf, and F. Simonot-Lion, "A degraded scheduling generation of a component based application", in *Incom. France*, 2006.

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

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

[14] J. Stankovic, M. Spuri, and K. Ramamritham, *Deadline Scheduling for Real-Time Systems*, Kluwer Academic Publishers. ISBN : 0792382692.

[15] K.-M. Zuberi and K.-G. Shin, "Non-preemptive scheduling of messages on controller area network for real-time control applications", in *Real-Time Technology and Applications Symposium*, 1995.

[16] C.-L. Liu and L. Layland, *Scheduling Algorithms for Multiprogramming in Hard Real-Time Environment*, Journal of the ACM 20, 46-61, 1973.

[17] M. Khalgui, X. Rebeuf, and F. Simonot-Lion, "A schedulability condition of an IEC 61499 control application with limited buffers", in *omer 05. Germany*, 2006.

[18] J. Liu, "Real-Time Systems", in *Prentice Hall*, 2000.

[19] H. Takada and K. Sakamura, *ITRON for Small-Scale Embedded Systems*, IEEE MICRO, vol.15, no.6, pp.46-54, 1995.

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