HAL
open science

# Dynamic Application Model for Scheduling with Uncertainty on Reconfigurable Architectures

Ismail Ktata, Fakhreddine Ghaffari, Bertrand Granado, Mohamed Abi

## To cite this version:

Ismail Ktata, Fakhreddine Ghaffari, Bertrand Granado, Mohamed Abi. Dynamic Application Model for Scheduling with Uncertainty on Reconfigurable Architectures. International Journal of Reconfigurable Computing, 2011, Volume 2011, pp.ID 156946. 10.1155/2011/156946 . hal-00666410

**HAL Id: hal-00666410**

**https://hal.science/hal-00666410**

Submitted on 4 Feb 2012

**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.

Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2011, Article ID 156946, 15 pages
doi:10.1155/2011/156946

*Research Article*

# Dynamic Application Model for Scheduling with Uncertainty on Reconfigurable Architectures

**Ismail Ktata,[1,2] Fakhreddine Ghaffari,[1] Bertrand Granado,[1] and Mohamed Abid[2]**

[1] *ETIS Laboratory, CNRS UMR8051, University of Cergy-Pontoise, ENSEA, 6 avenue du Ponceau 95014 Cergy-Pontoise, France*
[2] *Computer & Embedded Systems Laboratory (CES), University of Sfax, ENIS, 3038 Sfax, Tunisia*

Correspondence should be addressed to Ismail Ktata, ismail.ktata@ensea.fr

Received 26 August 2010; Revised 16 December 2010; Accepted 10 February 2011

Academic Editor: Michael Hübner

Copyright © 2011 Ismail Ktata et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Applications executed on embedded systems require dynamicity and flexibility according to user and environment needs. Dynamically reconfigurable architecture could satisfy these requirements but needs efficient mechanisms to be managed efficiently. In this paper, we propose a dedicated application modeling technique that helps to establish a predictive scheduling approach to manage a dynamically reconfigurable architecture named OLLAF. OLLAF is designed to support an operating system that deals with complex embedded applications. This model will be used for a predictive scheduling based on an early estimation of our application dynamicity. A vision system of a mobile robot application has been used to validate the presented model and scheduling approach. We have demonstrated that with our modeling we can realize an efficient predictive scheduling on a robot vision application with a mean error of 6.5%.

## 1. Introduction

Embedded systems are everywhere in our automobiles, robots, planes, satellites, boats, industrial control systems, and so forth. An important feature of such systems is to be reactive, as they continuously react with their environment at a rate imposed by this later itself. They receive inputs, process these stimuli, and produce the outputs, known as reactions. In general, the inputs are not known until they appear, and the systems must react dynamically to these stimuli. DRAs (dynamically reconfigurable architectures) are good candidate to implement an embedded application as they could be in perfect adequacy with its dynamic behavior.

In this paper, we are interested in the modeling and scheduling of soft real-time applications that could be embedded on DRA. Soft real time constraint means that the application may tolerate lateness in its deadline and could respond with decreased quality of service (loss of frames in video, e.g.). One of the particularities of the considered applications is their uncertainty: we cannot have complete information on the characteristics of the processing. These characteristics, latency, functionality, resources need, and so forth, depend on the environment inputs and are known as the application is processed (e.g., in computer vision, object recognition process depends on the number of objects present in the image or video sequence, lighting or color, viewing direction, size/shape, etc.). One of the solutions to minimize the effect of the dynamicity of the application is to realize a predictive scheduling that takes into account the characteristics of the application. In order to realize this predictive scheduling we need to exhibit the dynamicity of the application. In this paper, we propose an original dynamic application model that could be used to do predictive scheduling. We present also a complete and original predictive scheduling based on our dynamic applications model.

The remainder of our paper is structured as follows: Section 2 introduces the context and the problematic. Section 3 describes the related works on modeling techniques as well as the scheduling approaches for dynamic systems. Our new proposed method of dynamic application modeling is presented in Section 4 and compared to other models. Section 5 outlines the predictive scheduling technique. Section 6 presents a real case example of a vision system on

a mobile robot and the validation of our model and our predictive scheduling. The last section concludes the paper with a summary of achievements.

## 2. Context and Problem Definition

Today, the growing complexity of soft real-time applications represents important challenges due to their dynamic behavior and uncertainties which could happen at runtime [1]. To overcome these problems, designers tend to use DRAs that are well suited to deal with the dynamism of applications and allow better compromise between cost, flexibility and performance [2]. In particular, fine grained dynamically reconfigurable architectures (FGDRAs), as a kind of DRAs, can be adapted to any application with a great optimality. However, this type of architecture makes the applications design very complex [3], especially with the lack of suitable and efficient tools. This complexity could be abstracted at runtime by providing an operating system which abstracts the lower level of the system [4]. This operating system has to be able to respond rapidly to events. In the case of soft real-time application with dynamic behavior, the goal is to meet some quality of service requirements. This goal can be achieved by an operating system with a suitable predictive scheduling approach.

In order to realize an efficient predictive scheduling of an application, an operating system needs to know the behavior of this application, in particular the part where the dynamicity can be efficiently exploited on a DRA.

In this paper, we focus on two major problems to realize an efficient scheduling on an FGDRA.

(a) The modeling of the application that should exhibit its dynamical aspects and must allow the expression of its constraints, in particular real-time constraints.

(b) The run-time performance of the predictive scheduling algorithm.

The different items of a scheduling problem are the tasks, the constraints, the resources, and the objective function. Many resolution strategies have been proposed in literature [5]. These methods usually assume that execution times can be modeled with deterministic values. They use an off-line schedule that gives an explicit idea of what should be done. Unfortunately, in real environments, the probability of a precomputed schedule to be executed exactly as planned is low [6]. This is due to not only variations, but also to a lot of data that are only previsions or estimations. It is then necessary to deal with uncertainty or flexibility in the process data. Hence, a significant on-line reformulation of the problem and the solving methods are needed in order to facilitate the incorporation of this uncertainty and imprecision in scheduling [7].

Uncertainty in scheduling may arise from many sources [8]:

(a) the release time of tasks can be variable, even unexpected;

(b) new unexpected tasks may occur. We named such a task a hazardous task;

(c) cancellation or modification of existing tasks;

(d) resources may become unavailable;

(e) tasks assignments: if a task could be done on different resources (identical or not), the choice of this resource can be changed. This flexibility is necessary if such a resource becomes unusable or less usable than others;

(f) the ability to change execution mode: this mode includes the approval or disapproval of preemption, whenever a task could be resumed or not, the overlap between tasks, changing the range of a task, changing the number of resources needed for a task, and so forth.

In order to describe these features, a new model description is needed. It should be a little sensitive to data uncertainties and variations and adaptable to the possible disturbances.

## 3. Related Works

*3.1. Modeling Methods.* In order to efficiently implement an application, it should be described by a suitable model showing significant system characteristics of geometry, information, and dynamism. The latter is a crucial application characteristic as it permits to represent how an application behaves and changes states over time. Moreover, dynamic modeling can cover different application domains from the very general to the very specific [9]. Model types have different presentations, as shown in Figure 1; some are text-based using symbols while others have associated diagrams.

(a) Graphical models use a diagram technique with named symbols that represent processes, lines that connect the symbols and show relationships, in addition to various other graphical notations representing constraints (Figures 1(a), 1(b), and 1(d)).

(b) Textual models typically use standardized keywords accompanied by parameters (Figure 1(c)).

In addition, some models have static form, whereas others have natural dynamics during model execution as in Figure 1(a). The solid circle (a token) moves through the network and represents the execution behavior of the application.

In the domain of embedded systems, a large number of modeling languages have been proposed [10–12], including extensions to finite state machines, data flow graphs, communicating processes, and Petri nets. In this section, we present main models of computation for real-time applications reported in the literature.

*3.1.1. Finite State Machines.* The finite state machine (FSM) representation is probably the most well-known model used for describing control systems. However, one of the disadvantages of FSMs is the exponential growth of the number of states to be explicitly captured as the system complexity rises, making the model increasingly difficult to

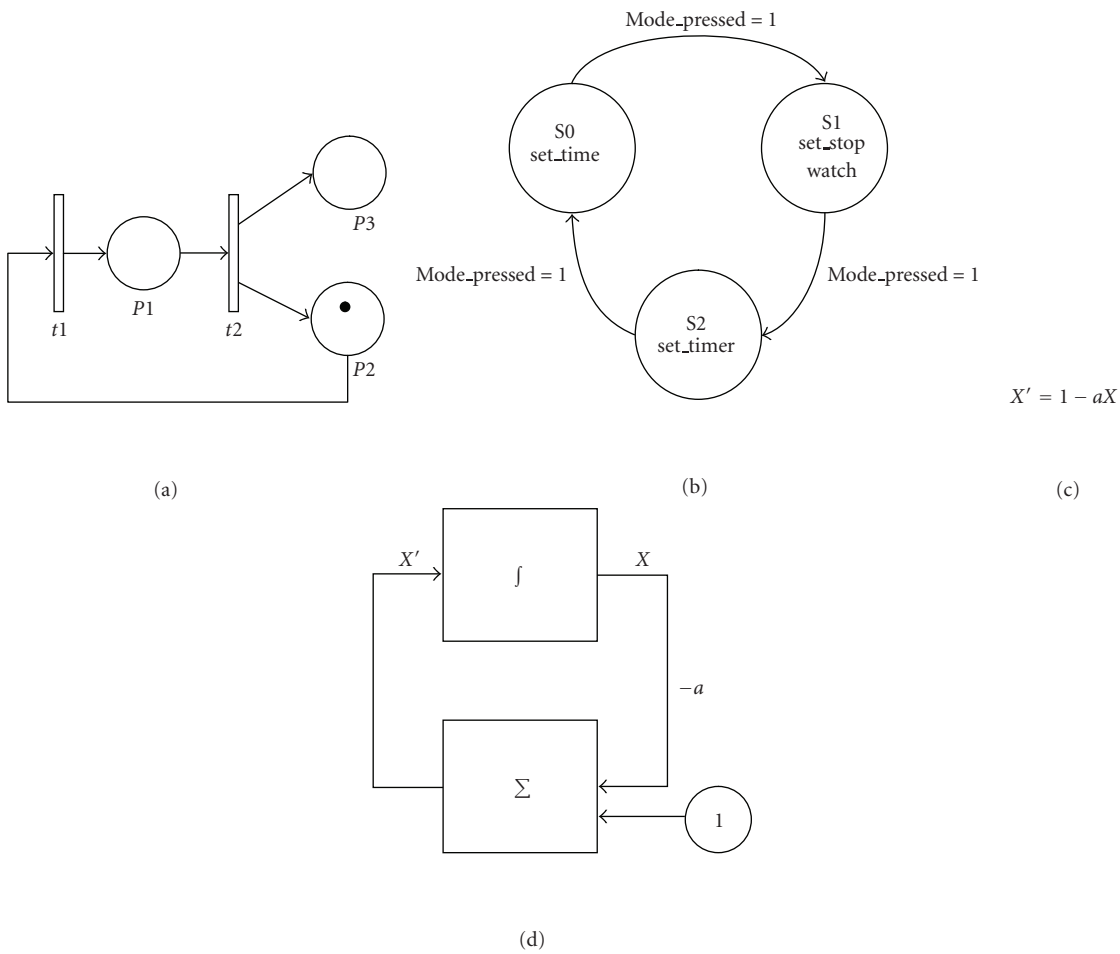


Figure 1: Four types of dynamic system models. (a) Petri net. (b) Finite state machine. (c) Ordinary differential equation. (d) Functional block model.

visualize and analyze [13]. For dynamic systems, the FSM representation is not appropriate because the only way to model it is to create all the states that represent the dynamic behavior of the application. It is then unthinkable to use it as the number of states could be prohibitive.

*3.1.2. Data-Flow Graph.* A data-flow graph (DFG) is a set of compute nodes connected by directed links representing the flow of data. It is very popular for modeling data-dominated systems. It is represented by a directed graph whose nodes describe the processing and the arcs show the partial order followed by the data. However, the conventional model is inadequate for the systems control unit representation [14]. It does not provide information about the ordering of processes for the scheduler. It is therefore inappropriate to model dynamic applications.

*3.1.3. Program Evaluation and Review Technique (PERT).* PERT is a model that was originally introduced in 1958 by the US Navy for its polaris weapon system. Since then, PERT has spread rapidly throughout almost all industries. Tasks are represented by arcs with an associated number presenting their duration. Between arcs, there are circles marking events of beginning or end of tasks (Figure 2). Each node ($i$) contains three characteristics: event's number (event number $i$), an earliest start time ($R_i$) on which an event can be expected to take place, and a latest finish time ($D_i$) on which an event can take place without extending the completion date of the application. The difference between the latest finish time and the earliest start time is called slack time. A task, from node A to node B, is considered critical if the difference between the latest finish time of B and the earliest start time of A is equal to the execution time of the task. All critical tasks form the critical path, which is the path on which no task should be delayed, so that the whole application would not be delayed.

An interesting feature of PERT model is that it allows predicting the minimum and maximum duration of the application based on a known tasks execution time [15]. For each event, PERT indicates earliest time when a task can start/finish and latest time when a task can start/finish. The earliest and latest times are considered as random variables. To estimate them, the scheduler refers to the schedule of event occurrences that were established at the beginning of the application. The scheduler would have at

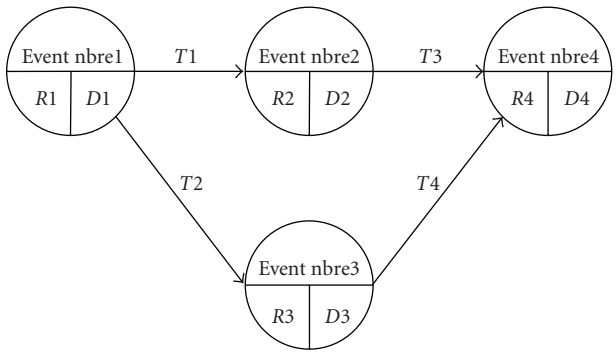


Figure 2: PERT graph.

his disposal a large volume of historical data from which to make its estimates. Obviously, the more historical data is available, the more reliable the estimate is. The calculations of critical paths and slack times were based on these best estimates. This model seems interesting as it can describe a dynamic duration of a task, but it cannot model other dynamic features, for instance, the variable number of tasks or resources.

*3.1.4. Petri Net.* Petri net (PN) is a modeling formalism which combines a well-defined mathematical theory with a graphical representation of the dynamic behavior of systems [9]. Petri net is a 5-tuple PN $= (P, T, F, W, M_0)$ where $P$ is a finite set of places which represent the status of the system before or after the execution of a transition. $T$ is a finite set of transitions which represent tasks. $F$ is a set of arcs (flow relation). $W : F \rightarrow \{1, 2, 3, \ldots\}$ is a weight function. $M_0$ is the initial marking. However, though Petri net is well established for the design of static systems, it lacks support for dynamically modifiable systems [16]. In fact, the PN structure presents only the static properties of a system while the dynamic one results from PN execution which requires the use of tokens or markings (denoted by dots) associated with places [17]. The conventional model suffers from good specification of complex systems such as lack of the notion of time which is an essential factor in embedded applications and lack of hierarchical composition [18]. Therefore, several formalisms have independently been proposed in different contexts in order to overcome the problems cited above, such as introducing the concepts of hierarchy, time, and valued tokens. Timed PNs are those with places or transitions that have time durations in their activities. Stochastic PNs include the ability to model randomness in a situation and also allow for time as an element in the PN. Colored PNs enable the user and designer to witness the changes in places and transitions through the application of color-specific tokens, and movement through the system can be represented through the changes in colors [9].

None of the methodologies mentioned provides sufficient support for systems which include dynamic features. Dynamic creation of tasks, for instance, is not supported. In [18], authors proposed an extension of high-level Petri net model [19] in order to capture dynamically modifiable embedded systems. They coupled that model with graph transformation techniques and used a double push-out approach which consists of the replacement of a Petri net by another Petri net after firing of transitions. This approach allows modeling dynamic tasks creation but not variable execution time nor variable number of needed resources. As we can see there is no model to exhibit dynamic features of an application. Next section will present related works about scheduling under uncertainty.

*3.2. Scheduling under Uncertainty.* In literature, most of the works have been focused on finding optimal or near-optimal predictive schedules for simple scheduling models with respect to various criteria assuming that all problem characteristics are deterministic. However, many embedded systems operate in dynamic environments, frequently subject to various real-time events and several sorts of perturbations, such as random task releases, resources failure, task cancellation, and execution time changes. Therefore, dynamic scheduling is of great importance for the successful implementation of real-world scheduling applications. In general, there are two main approaches dealing with uncertainty in a scheduling environment according to phases in which uncertainties are taken into account [8].

(a) Proactive scheduling approach aims at building a robust baseline schedule, that is, protected as much as possible against disruptions during schedule execution. It takes into account uncertainties only in design phase (offline). Hence, it constructs predictive schedule based on statistical and estimated values for all parameters, thus implicitly assuming that this schedule will be executed exactly as planned. However, this could become infeasible during the execution due to the dynamic environment, where unexpected events continually occur. Moreover, overestimations may lead to a schedulability test failure and overutilization of resources. Therefore, in this case, a reactive approach may be more appropriate [8].

(b) Instead of anticipating future uncertainties, reactive scheduling takes decisions in real time when some unexpected events occur. A reference deterministic scheduling, determined offline, is sometimes used and reoptimized. In general, reactive methods may be more appropriate for high degrees of uncertainty, or when information about the uncertainty is not available.

A combination of the advantages of both precedent approaches is called proactive-reactive scheduling. This hybrid method implies a combination of a proactive strategy for generating a protected baseline schedule with a reactive strategy to resolve the schedule infeasibilities caused by the disturbances that occur during schedule execution. Hence, this scheduling/rescheduling method permits to take into account uncertainties all over the execution process and ensures better performance [20, 21]. For rescheduling, the literature provided two main strategies: schedule repair and

complete rescheduling. The first strategy is most used as it takes less time and preserves the system stability [22].

Dynamic scheduling techniques are widely studied in information and industrial systems and are quite different depending on the nature of the problem and the type of disturbance considered: resources failure, the variation of the tasks duration and the fact that new tasks can occur, and so forth. The mainly used methods are dispatching rules, heuristics, metaheuristics, and artificial intelligence techniques [23]. In [24], authors considered a scheduling problem where some tasks (called "uncertain tasks") may need to be repeated several times to satisfy the design criteria. They used an optimization methodology based on stochastic dynamic programming. In [25, 26], scheduling problem with uncertain resource availabilities was encountered. Authors used proactive-reactive strategies and metaheuristic algorithm that combines genetic algorithms. Another uncertainty case, which is uncertain tasks duration, had been studied in [27, 28]. Since their complexity in implementation and calculation time, metaheuristic techniques are more easily to be applied offline. On the other hand, when it is about an online context, priority-based scheduling (dispatching rules and list scheduling) is more rapid to have reasonable solutions. However, priority-based scheduling is inefficient and not appropriate for systems where the required scheduling behavior changes during runtime [29]. Therefore, it is necessary, in our case, to combine different techniques together to endow the scheduling approach with the required flexibility and robustness. For permanent tasks with deterministic features, priority-based algorithms will be practical, while for hazardous tasks, the scheduler should occur to a prediction service based on heuristic techniques. Taking the case of tasks with uncertain execution time, the predictive algorithm must take advantage of dynamic characteristics of the application and attribute, for each task, a dynamic online estimated value. This will permit to schedule the different hardware tasks ($T_i$) and allocate its needed resources of the DRA based on online estimated values.

There are basically two approaches for execution time prediction [30].

(a) Static approaches: these methods do not need a code execution on real hardware or on a simulator but rather rely on code structure and possible control-flow paths analysis to compute upper bounds of execution times [31]. A given code analysis technique is typically limited to a specific code type or a limited class of architectures. Thus, these methods are not very applicable to DRA. In addition, these offline analysis methods do not take into account changes in the processed data on each tasks' activation. Therefore, online changes in execution time will not be considered to avoid schedulability test failure, or too much resources uses.

(b) Measurement-based approaches: these methods execute the task on the real target hardware or hardware emulator, for some set of inputs. They then take the measured execution times and derive the maximal and minimal observed execution times on their distribution. These methods have complexity and safety problems [32]. These measurement-based methods require maintaining a history of execution time values of all tasks forming the application. During tasks execution, execution time is measured, and this measurement is subsequently added to the set of previous observations in order to improve the precision. Thus, as the number of observations increases, the estimates produced by a statistical algorithm will be improved [33]. These methods have the advantages that they are able to compensate for data input parameters (such as the problem size) and do not need any direct knowledge of the data characteristics, internal design of the code, or the considered architecture.

In measurement approaches, generally the execution time estimation problem is considered as a regression problem. They use regression algorithms to compute estimates from the set of previous observations. In literature, there are two classes of regression techniques: parametric techniques and nonparametric techniques. In general, parametric techniques require the definition of a form that describes the execution time ($Y$) of a task as a function (e.g., polynomial regression model) of a parameter $X$ (e.g,, $X$ is the problem size, or number of objects). A popular parametric technique for solving this type of problem is the least squares method. These techniques need offline computation of the function coefficients. However, for flexible and dynamic applications, it is difficult to make any assumptions on the functional form. Parametric techniques are then not well suited to this problem. Nonparametric techniques are a better choice. Nonparametric regression techniques (also called nonparametric estimators or smoothing techniques) are considered to be data driven, since the estimate depends only upon the set of previous observations, and not on any assumptions about $Y(X)$ function. All nonparametric regression techniques compute $Y(X)$ using a variation of the equation

$$Y(X) = \frac{1}{n}\sum_{i=1}^{n} W_i(X) \cdot C_i, \tag{1}$$

where $W_i(X)$ is a weighting function. $Y(X)$ is a weighted average of the execution time values $C_i$, of the $n$ previous observations. The weight function $W_i(X)$ typically assigns higher weights to observations close to the parameter $X$, and lower weights to observations farther away from $X$. This is illustrated in Figure 3. A popular nonparametric technique is k-nearest neighbor (k-NN) algorithm [34].

For nonparametric methods (like k-NN), the main problem is that, especially in embedded systems, they are more complex in implementation (need historic observation, distance calculation).

However, in contrast with a static property of a task, the estimated parameters are not only dependent on the code to be executed by the task (and its possible set of parameters) but also on the system's state and the environment at the

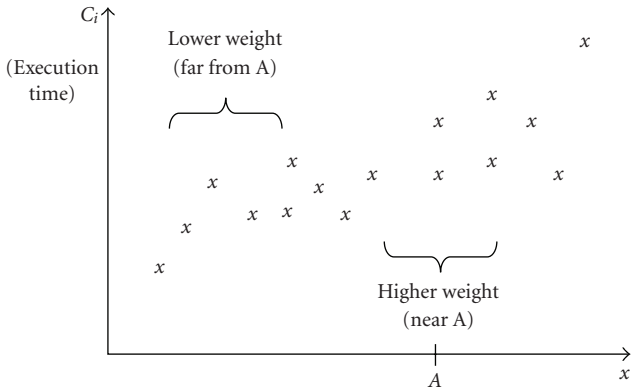


Figure 3: Weights assigning to previous observations.

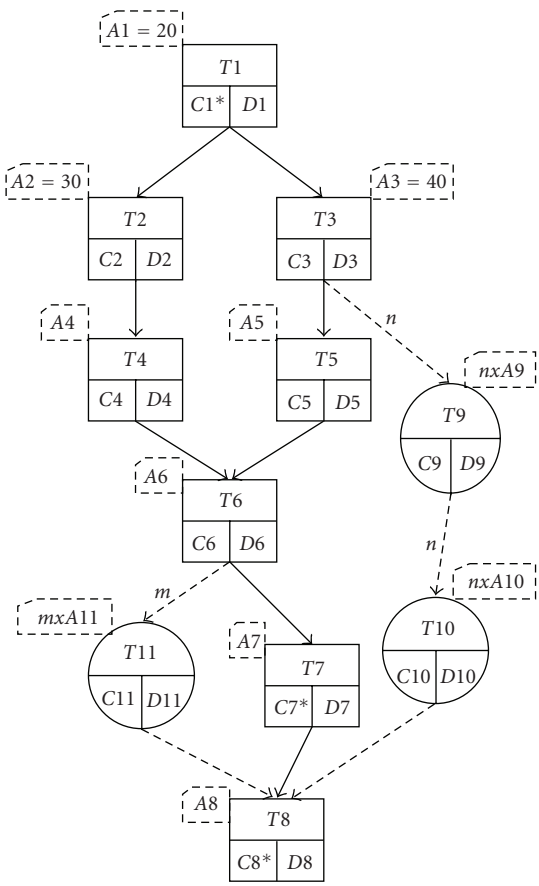


Figure 4: New model for dynamic application.

time ($t$) when is executed. This means that the estimated parameters are probabilistic values, and they may change over the lifetime of the system. In order to predict those parameters for the next instance ($t$) of a task $T$, it should be a good approximation to look at the most recent executions of instances of $T$. This is the typical case of some real time applications such as image and video processing, where tasks features variations may be correlated with some parameters like keypoints for edge extracting or interesting points for particular processing.

The next section deals with our new modeling method for dynamic applications. The model will permit to the scheduler to take into account the dynamic features and so to help in its predictions.

## 4. Dynamic Application Modeling

*4.1. Proposed Method.* As we mentioned in Section 3, to realize a predictive scheduling, we must exhibit the dynamic characteristics of an application. We consider three cases of dynamicity in applications.

(a) The number of tasks is not fixed. It may change from iteration to another.

(b) The tasks execution time may change too.

(c) The number of needed resources for tasks execution is variable. In addition, the number of available resources may decrease after a failure occurs.

For these cases, the goal is to develop a robust scheduling method, that is, little sensible to data uncertainties and variations between theory and practice. To represent all those constraints in the same model, we have developed a graphical model. In this model, the graph is composed of two forms of nodes. We make use of the example shown in Figure 4 in order to illustrate the different definitions corresponding to our model. The first type of nodes refers to permanent tasks which are known in advance and which are always executed during the whole lifecycle execution of the application. This is the case of the set $\{T1, T2, T3, T4, T5, T6, T7, T8\}$. The second type, corresponding to the set $\{T9, T10, T11\}$, is for hazardous tasks which may be executed in some period but not in others. Each task of the proposed model is characterized by the following four parameters:

(a) $T_i$ for the task's number,

(b) $C_i$ for the execution time, and $C_i^*$ for random execution time,

(c) $D_i$ for the deadline,

(d) $(A)$ is the number of needed resources. It may be multiplied by a variable resource factor $(n)$ that represents $n$ hazardous tasks,

(e) plain arc between two nodes represents the precedence between two tasks,

(f) dashed arc between two nodes represents the precedence between several nodes indicated by $n$. If $n = 0$, then the initial node has no successor.

In our model, to represent tasks with variable execution time, we have been inspired by PERT model presented in Section 3.1. We replace the earliest start time ($R_i$) in PERT by the execution time ($C_i$) as it would be changed over execution. We kept the deadline $D_i$ as it will be useful to calculate the makespan (i.e., the length of the schedule). In the example of Figure 4, tasks with variable execution time are $\{T1, T7, T8\}$. This will be noticed by the use of asterisk.

To represent the uncertainty of the number of tasks instances, for example in Figure 4, task $T11$ will be executed

$m$ times, we introduce a valued arc by the number of instances. In order to show that in some cases the task will not be executed (there is no need for that processing), we present the task with circled node (e.g., $T9$, $T10$, and $T11$). Hence, the number of instances is defined as integer, and if the number $m = 0$, then task $T11$ will not occur. The number $m$ will depend on the previous executions and the actual input data to be processed. Thus, $m$ will be recalculated, after each period, based on its previous values.

To be executed, hardware tasks need resources like certain amount of silicon area. This feature is indicated in the labels over each tasks node. For permanent tasks (e.g., $T1$ in Figure 4), the label contains one parameter, that is, $A_1 = 20$ indicating the volume of needed resources for $T1$. For hazardous tasks, and to execute multiple instances of its processes, the volume of resources will be multiplied by the number of instances. For $T9$, the volume of needed resources is $n \times A_9$ of the whole hardware architecture.

The arcs represent the dependencies between tasks. For permanent tasks, arcs are represented with solid lines, while uncertain dependencies are represented by dashed lines.

*4.2. Comparison of Models.* Compared with other models (Section 3.1), our proposed technique presents several advantages. For uncertain number of occurring tasks, the data flow graph (DFG model) does not contain information about the number of instances. During execution of the application, every task represented by the nodes of DFG is executed once in each iteration [35]. Only when all nodes have finished their executions, a new iteration can start. So to model this, we need to represent $n$ nodes of the same task, which increases the size of the model (see Figure 5(b)). In our model, information about uncertain number of instances of a same task to be executed is noted by the circle form of the task and the number above its arc. For PN model, (see Figure 5(a)), arcs could be labeled with their weights where a $k$-weighted arc can be interpreted as the set of $k$ parallel arcs [36]. But, from its definition (Section 3.1), weights are positive integers, so PN cannot present a fictive arc with nonfiring transition representing a task that may not be executed in some iterations. In Figure 5(a), if $T9$ and $T10$ are not executed, then $n$ should be null, which is impossible from PN definition. In addition, to fire $T8$, all input places should have at least one token, which will be not possible if $T10$ or $T11$ was not executed (fired).

Petri net does not provide any timing information, that is, mandatory for determining minimum application completion time, latest starting time for an activity which will not delay the system, and so on. The only important aspect of time is the partial ordering of transitions. For example, it presents variable tasks duration with a set of consequent transitions for each task which will complicate the model (see Figure 6). The addition of timing information might provide a powerful new feature for Petri nets but may be difficult to implement in a manner consistent with the basic philosophy of Petri nets research [37]. For resource representation, PN represents this feature by an added place with a fixed number of tokens. To begin execution, a task removes a token from the resource place and puts it back in the end of its execution. However, this model is inadequate in our case since the number of available resources may change over the execution.

Therefore, the use of conventional modeling methods is not effective for dynamic applications. With PN and DFG models (Figure 5), there is no distinction between permanent tasks and hazardous ones (that may not be executed), nor an explicit notion of time (as variable execution time of some tasks). PERT technique enables to present tasks temporal features in order to identify the minimum time needed to complete the total project. However, it lacks functional properties in estimation and does not assume resources constraints (considered as unlimited) [15].

The main advantage of our method is the possibility to represent several dynamic features of real-time applications with the minimum of nodes and thus in a simple formalism. We can bring out three main characteristics of this model:

(a) the distinction between the static execution and the dynamic execution;

(b) the tasks whose execution time is variable (presented by the asterisk);

(c) the volume, for each task, of needed resources for its execution. In each iteration, and depending on the available resources, schedule is able to decide which ready tasks could be executed on the device.

## 5. Predictive Scheduling

*5.1. OLLAF Architecture.* Our goal is an efficient management of dynamically reconfigurable architectures; more precisely, as case study, we target the OLLAF architecture. OLLAF, as presented in [4], is an original FGDRA specifically designed to enhance the efficiency of OS (operating system) services necessary to manage such architecture. In particular, OLLAF can efficiency support the context switching service of an OS. From the global view (Figure 7), OLLAF has a reconfigurable logic core organized in columns. Each column can be reconfigured separately and offer the same set of services. A task uses an integer number of columns and can be moved from one column to another without any change on the configuration data [4]. Each column provides a hardware configuration manager (HCM) and a local cache memory (LCM). The reconfigurable logic core uses a double memory plan. With this topology, the context of a task can be shifted in while the previous task is still running and shifted out while the next one is already running. The effective task switching context overhead is then taken down to one clock cycle. To obtain such a rapid context switching service, the configurations of the tasks have to be placed in advance in the LCM of the corresponding columns. In the first version of OLLAF, those memories can store 3 configurations and 3 task contexts.

In OLLAF, as an OS is purely a control process it is implemented on a microprocessor denoted by (HW Sup + HW RTK + CCR) in Figure 7. The OS must manage the context switching and then needs to take into account

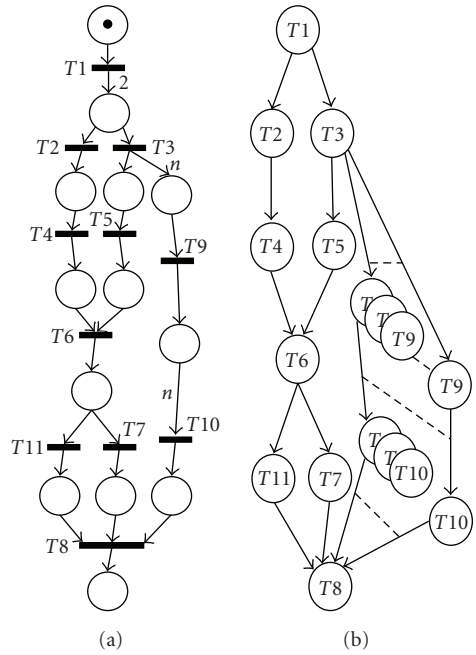


Figure 5: (a) A Petri net representation of the example of Figure 4. (b) A DFG representation of the example of Figure 4.

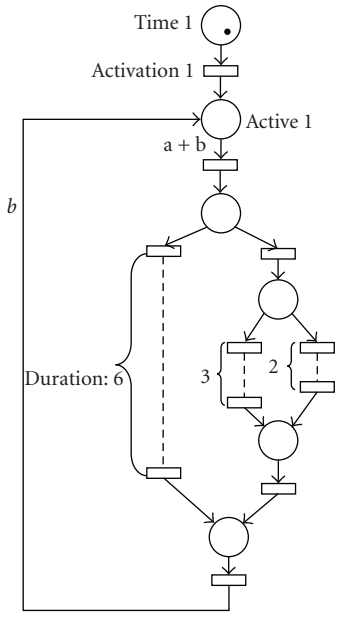


Figure 6: PN model for variable tasks duration.

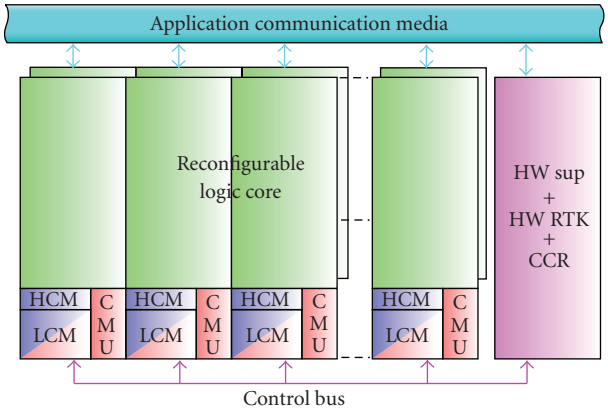


Figure 7: Global view of OLLAF architecture.

a prefetch task configuration on the LCM of the columns where it might most probably be placed. It is a conditional process because OLLAF is targeted to execute dynamic application. In [4], authors showed some case studies which demonstrate that the OLLAF architecture can perform a greater efficiency than the one performed using a traditional commercial FPGA. Our goal is then to make a dynamic and predictable scheduling to better manage a dynamic real-time application executed on such architecture. To realize this, we propose to use as the input of the scheduler an application represented by our dynamic graph modeling described in Section 4.

*5.2. Scheduling Flow.* We can represent our scheduler flow as in Figure 8. It takes as inputs on one hand a dynamic graph modeling of an application and on the other hand a description of a DRA, here a model of OLLAF FGDRA as we focus on it. For validate purpose of our scheduling, OLLAF is modeled as a set of columns. Based on these descriptions, our scheduler makes an online dynamic scheduling based on prediction process. The parts which are outlined with dashed lines, visible in Figure 8, present the prediction process which will make online decisions based on precedent observations of variable parameters and taking into account the application constraints.

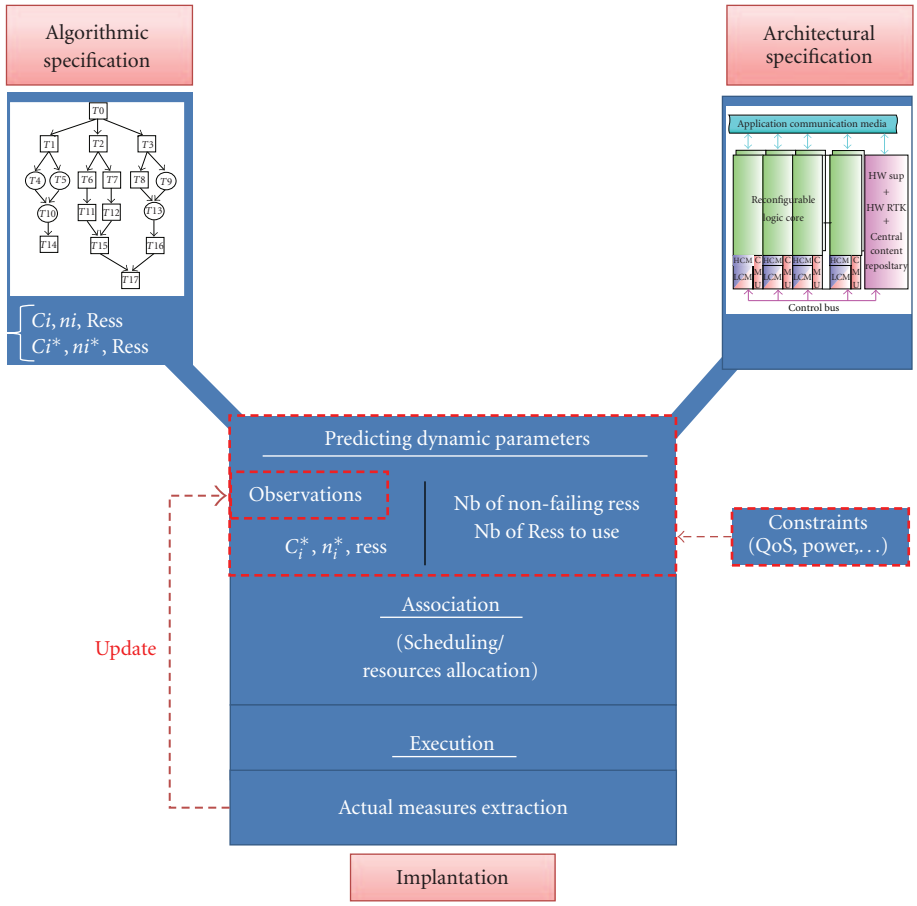


Figure 8: Scheduling flow on OLLAF architecture.

With our model the scheduler is able to distinct between two parts of the application: one containing all the permanent tasks and other containing hazardous tasks, as well as, in permanent tasks, those that have a variable execution time. Permanent tasks will be scheduled respecting their precedence constraints (topological order). Their configuration data will be prefetched, as much as possible, on the columns making a maximum use of the whole of the architecture. This ensures minimizing the number of configuration data transfers and tasks relocation, and thus the time of saving and restoring process. The principle of our scheduler is to realize an initial scheduling where all dynamic features are not taken into account (all dynamic parameters are equal to zero). Then an execution is done and dynamic parameters are updated. Based on these new parameters, a rescheduling is done that takes into account dynamic features. The update of dynamic parameters is computed with a prediction technique. We use a least laxity first (LLF) policy [38] because it permits a dynamic priority assumption depending on the laxity which varies according to the execution time. If there is equality between some tasks, task which has maximum execution time will have priority to be launched before the others.

The scheduler will proceed the rescheduling with a minimum effect on performance. This reschedule must rely on rapid algorithms based on a simple scheduling technique, so that it can perform online execution with no overhead. From the ready list, LLF algorithm determines the tasks that can be executed on the reconfigurable device. Tasks with the higher priorities will be placed first until the area of device is fully occupied. If there are not enough hardware resources available, the last recently used (LRU) strategy is used to select which tasks' configuration data will be removed. During runtime, effective values of dynamic parameters are measured and stored. These values are used to update dynamic parameters with an approximation function. Such function can be, for a parameter, a mean of all its measured values, or a maximum value of all the measured values of a more complex function. In next section, we present four used functions for predicting dynamic parameter values in a robotic application.

## 6. Experimental Results

*6.1. Robotic Application Benchmark.* As an illustration of the modeling approach outlined in previous section, consider an image-processing application of a visual system embedded in a mobile robot. This application illustrates the modeling of systems using our proposed model. In this application, robot learns its environment to identify keypoints in the

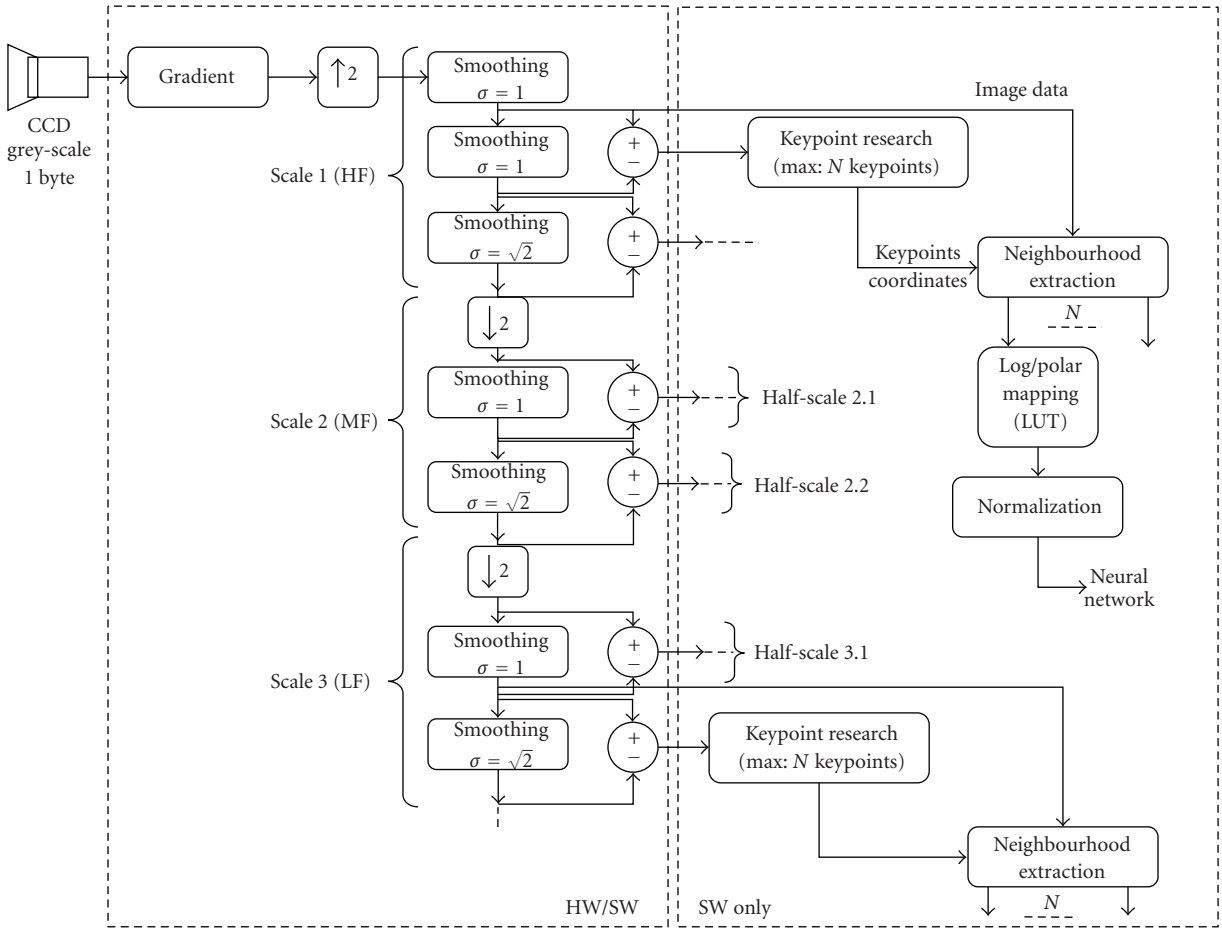


Figure 9: Global architecture of the robotic application.

landscape [39]. The keypoints correspond to the image filtered by difference of Gaussians (DoGs). This application is dynamic in the sense that the number of keypoints depends on the visual scene and is not known a priori. In addition, there are three application modes corresponding to the behavior of the robot, and each mode executes different processing under different rates (Figure 9).

(a) Fast mode (Scale 3): the robot moves around for a coarse description of the landscape but at a high frame rate. Thus, only the lower scales are processed. It is the reason why in this first mode the maximum number of extracted keypoints is fixed to $N = 10$, and the number of frames processed per second (fps) to 20.

(b) Intermediate mode (Scales 2 and 3): the robot moves slowly, for example, when the passage is blocked (obstacle avoidance, door passage) and needs more precision on its environment. In this mode, the middle scale and the lower scale are processed. In this mode, $N$ is fixed to 30 and the system works at a rate of 5 fps.

(c) High-detail mode (Scales 1, 2, and 3): the robot is stopped in a recognition phase (object tracking, new place exploration). All scales are fully processed and full information on the visual environment is provided. The number of processes depends on the number of keypoints in the video frames. For this mode, $N = 120$, and the rate of the system is fixed to 1 fps.

As a consequence, application tasks could be divided in three groups:

(i) intensive data-flow computation tasks that execute in a constant time,

(ii) tasks whose execution number is correlated with the number of interest points,

(iii) tasks with unpredictable execution time (depending on the images features).

Figure 10 shows our proposed model for the robotic vision application. We can notice the presence of permanent branch (squared nodes) which represents permanent tasks that will be executed in all cases or modes (e.g., $T1$, $T14$, $T19$) and hazardous tasks that may occur during execution. Table 1

Table 1: Identification of the robotic application tasks.

| | Tasks | Permanent tasks | Hazardous tasks |
|---|---|---|---|
| Gradient | $T1$ | $T1$, $T2$, $T13$, $T14$, $T15$, $T16$, $T17$, $T18$, $T19$, $T20$, $T21$ | $T1$, $T3$, $T4$, $T5$, $T6$, $T7$, $T8$, $T9$, $T10$, $T11$, $T12$, $T22$, $T23$, $T24$, $T25$, $T26$, $T27$, $T28$, $T29$, $T30$ |
| Subsampling | $T2$ | | |
| Oversampling | $T4$ | | |
| Gauss1 | $T3$, $T5$, $T13$, $T14$, $T22$, $T24$ | | |
| Gauss2 | $T15$, $T6$, $T25$ | | |
| DoG | $T7$, $T8$, $T16$, $T17$, $T23$, $T26$ | | |
| Search | $T9$, $T11$, $T18$, $T20$, $T27$, $T29$ | | |
| Extract | $T10$, $T12$, $T19$, $T21$, $T28$, $T30$ | | |

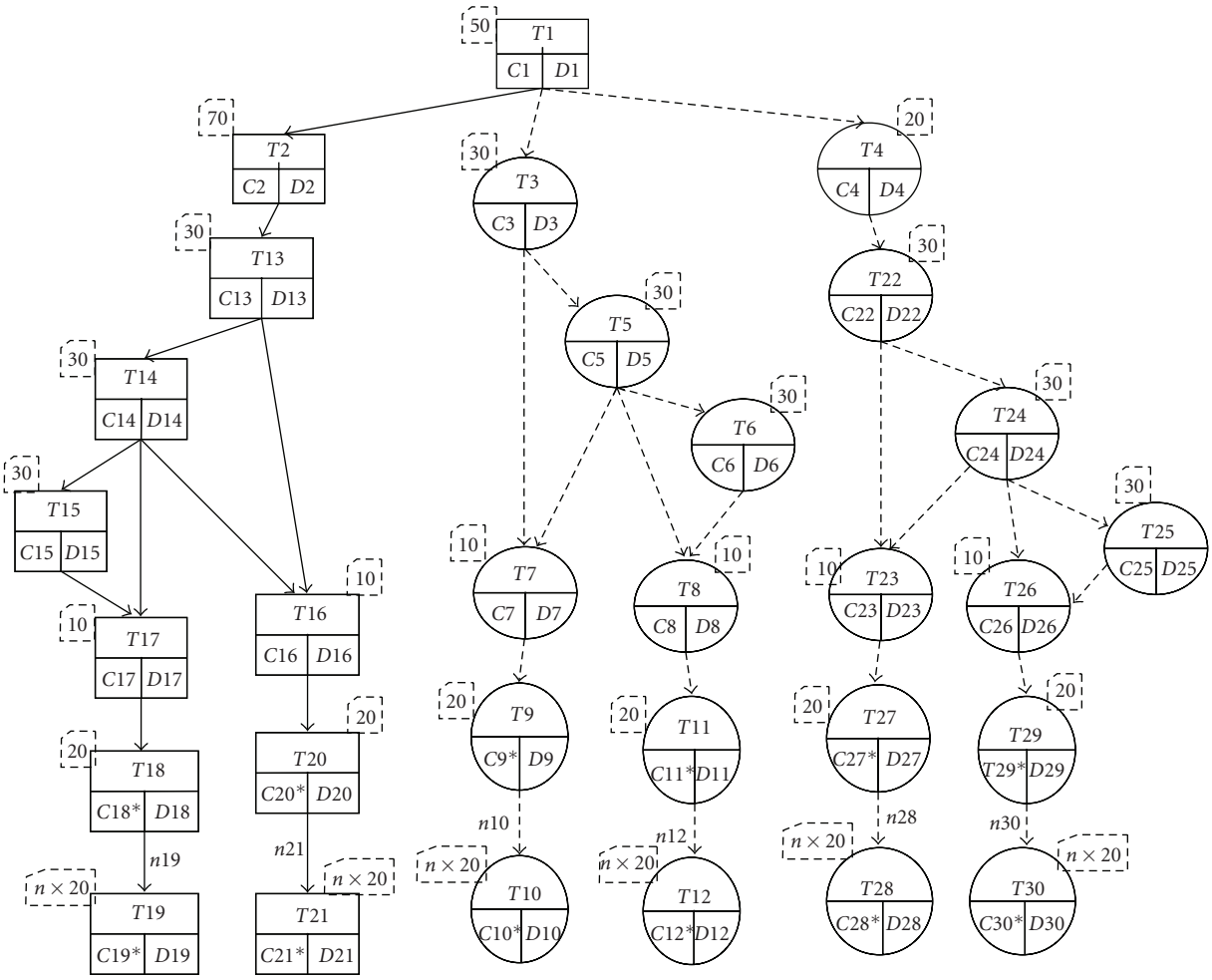


Figure 10: Dynamic model for the robotic vision application.

indicates tasks functions and classifies them to permanent and hazardous. Tasks with unpredictable execution time are indicated by the asterisk (e.g., $T9$, $T11$, $T20$). Another dynamic feature of tasks whose execution number depends on the number of keypoints is indicated by the use of resource factor $n$ (e.g., $T9$, $T10$, $T30$).

For the prediction techniques of uncertain tasks features, we have studied the task of keypoints search. The provided measured execution times on representative samples are presented in Figure 11. As we can see, the values distribution is random especially in the first scale. The elapsed time of keypoints search in an image depends on the number of keypoints that this image contains. The execution time is correlated with the number of keypoints found; this number is also random as shown in Figure 12. Figure 12 shows the number of keypoints found in the corresponding image over

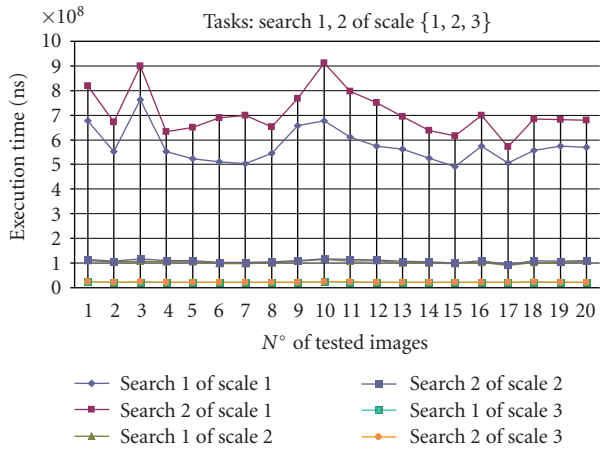


Figure 11: Execution time measurement of search task.

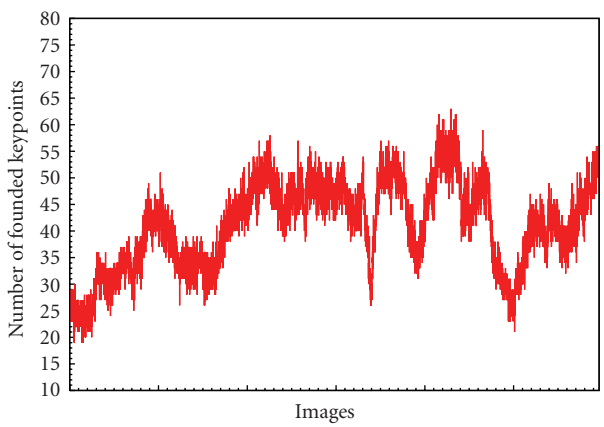


Figure 12: Number of detected keypoints in image sequence.

a sequence of more than 5000 images. It corresponds to the search task $T18$ of scale 1, and it has almost the same look as other search tasks $\{T20, T9, T11, T27, T29\}$ of different scales.

*6.2. Prediction Results.* We have compared several techniques to predict the execution time of the search keypoints task. Comparison is based on error estimation defined as follows:

$$E = \sum_i \frac{\text{Real execution time}\,(t_i) - \text{Estimated execution time}\,(t_i)}{\text{Real execution time}\,(t_i)} \times 100, \tag{2}$$

where $t_i$ is a runtime instant and $i = \{0, \ldots, n\}$.

We will focus on the prediction function corresponding to the search 1 task of scale 1 as it presents very random distributions (Figure 11). We have tested four methods based on statistical measurements obtained by on-line monitoring (Figure 13):

(1) first method (a) uses the mean value of the three last real execution times;

(2) Second method (b) uses the mean of the three last values multiplied by different weights. Weights are determined experimentally on the basis of tested data to minimize estimation error. As in nonparametric regression techniques, the higher weights are assigned to the values closed to the actual parameter and lower weights to those more distant;

(3) the third method (c) takes the maximum of the last three real values;

(4) the fourth method (d) takes the last value increased by 5%.

We notice that all methods lead, in the twenty tested images, to an overestimation of the execution time. As we can see, for the first method after some periods the predicted values become almost constant. In the case of second method, there is a delayed correspondence between the real execution graph and the predicted one. The third methods realize more pessimistic prediction and do overestimations. And finally, the fourth method is closer than the second one with more pessimistic prediction. As seen from Figure 14, the higher error rate is for the first (a) and the third (c) methods. For the fourth method (d), and even with a 1% increase of the last measure value, the error is still greater than that of the second method (b). Hence, the least obtained error percentage is the one related to the method using weighted average of last values (Figure 13(b)). The mean error between estimated and simulated results is about 0.4% for low- and medium-frequency scales and about 2.45% for high-frequency scale. Those results show that the technique based on a weighted average of the execution time values works with an accuracy of almost 98%. Even for keypoints number estimation (Figure 12), the mean estimation error of the same technique is 0.406% for the whole 5000 images. However, a null error rate does not guarantee a good quality of service (QoS). As we can see from the shape of the graphs of Figure 13, some methods (like first one (a)) present a great overestimation leading to overuse of resources, while others have relative great underestimation leading to a decreasing QoS. With keypoints number estimation (Figure 12), we have calculated estimation error of over 1000 successive images for the second method (b) and third one (c). The first technique, based on weighted average prediction, gives less overestimation than the second one (so less useless resources allocation), but it has an almost null mean error leading to a considerable underestimated values which has a great impact on the quality of service than second one. Indeed, the second technique presents more advantage with a relative acceptable mean overestimation of 6.5%, and less than 21% of the whole predicted values were underestimated. The purpose is that prediction be almost near real values and guarantees better compromise between requested QoS and efficient resources management. For this visual system application, and particularly for the two first modes of its process, the latter estimation can assist the prediction engine for better scheduling analysis of real-time tasks and so better

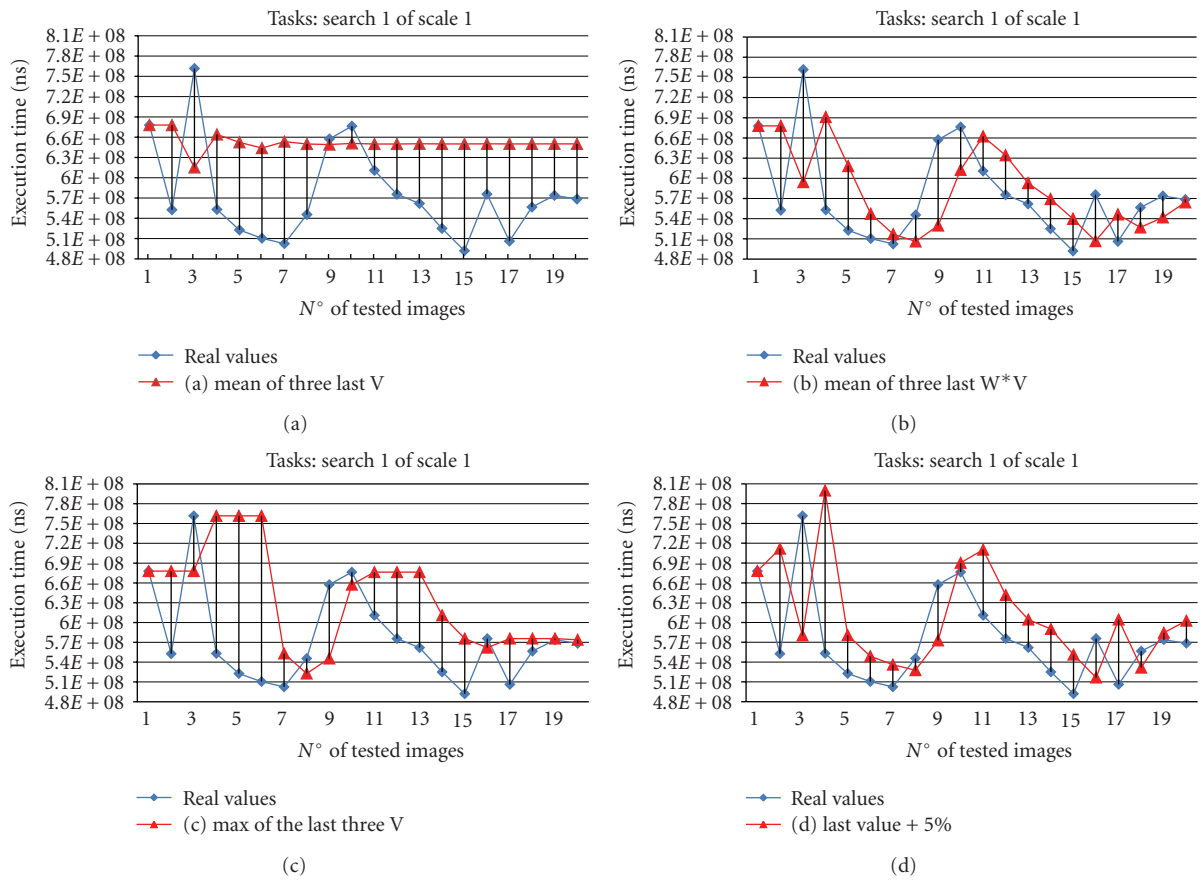


Figure 13: (a) Comparison between real measured values of the execution time of task search 1 of scale 1 and predicted values using method (a). (b) Comparison between real measured values of the execution time of task search 1 of scale 1 and predicted values using method (b). (c) Comparison between real measured values of the execution time of task search 1 of scale 1 and predicted values using method (c). (d) Comparison between real measured values of the execution time of task search 1 of scale 1 and predicted values using method (d).

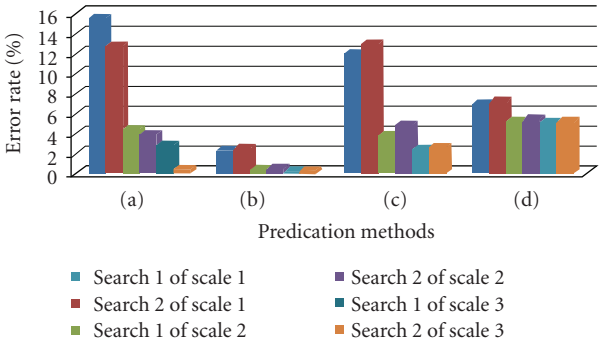


Figure 14: Prediction error rate of different methods for the execution time presented in Figure 11.

exploitation of the DRA like OLLAF. More investigations will be performed for the high-detail mode where all the treatments are realized on high frequencies.

## 7. Conclusion

In this work, we have proposed a new model to represent application with dynamic features. This model overcomes standard model like dataflow graph or Petri network, where it is not possible to represent dynamic characteristics of an application in a static fashion. We have demonstrated that this model is suitable to the aim of performing predictive scheduling on reconfigurable hardware, in particular on OLLAF FGDRA. The proposed model authorizes a scheduler to take into account uncertain characteristics of hardware tasks, the available resources in the target device, and the quality of service of the application. This model enables to present three considered types of dynamicity, which are uncertain tasks execution time, hazardous tasks that may occur, and variable resources needs. We have presented the scheduling method with the use of a prediction method enabling better adaptation of the architecture to the environment variations. As a validation purpose, we have used a case of an image-processing application of a visual system embedded in a mobile robot. Such application shows dynamically variable characteristics that are uncertain. We have demonstrated that with our modeling we can realize an efficient predictive scheduling on a robot vision application with a mean error of 6.5%.

Future works will consist in integrating our scheduling approach among the services of an RTOS taking into account the new possibilities offered by OLLAF. We also plan to

employ preemptive scheduling policies by using the mechanisms of migration and reallocation of tasks configuration and context data proposed by OLLAF.

## References


[1] C. Steiger, H. Walder, and M. Platzner, "Operating systems for reconfigurable embedded platforms: online scheduling of real-time tasks," *IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1393–1407, 2004.

[2] J. Noguera and R. M. Badia, "Multitasking on reconfigurable architectures: microarchitecture support and dynamic scheduling," *ACM Transactions on Embedded Computing Systems*, vol. 3, no. 2, pp. 385–406, 2004.

[3] A. Mtibaa, B. Ouni, and M. Abid, "An efficient list scheduling algorithm for time placement problem," *Computers and Electrical Engineering*, vol. 33, no. 4, pp. 285–298, 2007.

[4] S. Garcia and B. Granado, "OLLAF: a fine grained dynamically reconfigurable architecture for os support," *Eurasip Journal on Embedded Systems*, vol. 2009, Article ID 574716, 2009.

[5] P. Brucker and S. Knust, *Complex Scheduling (GOR-Publications)*, Springer, New York, NY, USA, 2006.

[6] V. T'kindt, J.-C. Billaut, and H. Scott, *Multicriteria Scheduling: Theory, Models and Algorithms*, Springer, New York, NY, USA, 2006.

[7] N. González, C. R. Vela, and I. González-Rodrguez, "Comparative study of meta-heuristics for solving flow shop scheduling problem under fuzziness," in *Proceedings of the 2nd International Work-Conference on the Interplay between Natural and Artificial Computation—part I: Bio-Inspired Modeling of Cognitive Tasks*, pp. 548–557, June 2007.

[8] A. J. Davenport and J. C. Beck, "A survey of techniques for scheduling with uncertainty," http://www.eil.utoronto.ca/profiles/chris/chris.papers.html.

[9] P. A. Fishwick, *Handbook of Dynamic System Modeling*, Cpaman & Hall/Crc Computer and Information Science, Chapman & Hall/CRC, 2007.

[10] S. Edwards, L. Lavagno, E. A. Lee, and A. Sangiovanni-Vincentelli, "Design of embedded systems: formal models, validation, and synthesis," *Proceedings of the IEEE*, vol. 85, no. 3, pp. 366–390, 1999.

[11] L. Lavagno, A. Sangiovanni-Vincentelli, E. Sentovich, A. A. Jerraya, and J. Mermet, "Models of computation for embedded system design," in *System-Level Synthesis*, Kluwer Academic Publishers, 1999.

[12] A. Jantsch, *Modeling Embedded Systems and SoC's: Concurrency and Time in Models of Computation*, Morgan Kaufmann, 2003.

[13] L. A. Cortes, *Verification and scheduling techniques for real-time embedded systems*, Ph.D. thesis, Department of Computer and Information Science, Linköping University, 2005.

[14] L. Alej, R. Cortés, P. Eles, and Z. Peng, "A survey on hardware/software codesign representation models," SAVE Project Report, Department of Computer and Information Science, Linköping University, 1999.

[15] H. Kerzner, *Project Management: A Systems Approach to Planning, Scheduling, and Controlling*, John Wiley & Sons, 2003.

[16] C. Rust and F. J. Rammig, "A petri net based approach for the design of dynamically modifiable embedded systems," in *Design Methods and Applications for Distributed Embedded Systems (DIPES)*, pp. 257–266, Kluwer Academic Publishers, 2004.

[17] M. Tavana, "Dynamic process modelling using Petri nets with applications to nuclear power plant emergency management," *International Journal of Simulation and Process Modelling*, vol. 4, no. 2, pp. 130–138, 2008.

[18] F. Rammig and C. Rust, "Modeling of dynamically modifiable embedded real-time systems," in *Proceedings of the IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, vol. 2004, pp. 28–34, 2003.

[19] E. Badouel and J. Oliver, "Reconfigurable nets, a class of high level Petri nets supporting dynamic changes within workflow systems," in *Proceedings of the Workshop on Workflow Management: Net-based Concepts, Models, Techniques, and Tools (WFM '98)*, pp. 129–145, 1998.

[20] H. Aytug, M. A. Lawley, K. McKay, S. Mohan, and R. Uzsoy, "Executing production schedules in the face of uncertainties: a review and some future directions," *European Journal of Operational Research*, vol. 161, no. 1, pp. 86–110, 2005.

[21] W. Herroelen and R. Leus, "Project scheduling under uncertainty: survey and research potentials," *European Journal of Operational Research*, vol. 165, no. 2, pp. 289–306, 2005.

[22] G. E. Vieira, J. W. Herrmann, and E. Lin, "Rescheduling manufacturing systems: a framework of strategies, policies, and methods," *Journal of Scheduling*, vol. 6, no. 1, pp. 39–62, 2003.

[23] D. Ouelhadj and S. Petrovic, "A survey of dynamic scheduling in manufacturing systems," *Journal of Scheduling*, vol. 12, no. 4, pp. 417–431, 2009.

[24] P. B. Luh, F. Liu, and B. Moser, "Scheduling of design projects with uncertain number of iterations," *European Journal of Operational Research*, vol. 113, no. 3, pp. 575–592, 1999.

[25] O. Lambrechts, E. Demeulemeester, and W. Herroelen, "Proactive and reactive strategies for resource-constrained project scheduling with uncertain resource availabilities," *Journal of Scheduling*, vol. 11, no. 2, pp. 121–136, 2008.

[26] S. Liu, K. L. Yung, and W. H. Ip, "Genetic local search for resource-constrained project scheduling under uncertainty," *International Journal of Information and Management Sciences*, vol. 18, no. 4, pp. 347–363, 2007.

[27] M. A. Turnquist and L. K. Nozick, "Allocating time and resources in project management under uncertaint," in *Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS '03)*, 2003.

[28] J. C. Beck and N. Wilson, "Proactive algorithms for job shop scheduling with probabilistic durations," *Journal of Artificial Intelligence Research*, vol. 28, pp. 183–232, 2007.

[29] F. Ghaffari, B. Miramond, and F. Verdier, "Run-time HW/SW scheduling of data flow applications on reconfigurable architectures," *EURASIP Journal on Embedded Systems*, vol. 2009, Article ID 976296, 13 pages, 2009.

[30] R. Wilhelm, J. Engblom, A. Ermedahl et al., "The worst-case execution-time problem-overview of methods and survey of tools," *ACM Transactions on Embedded Computing Systems*, vol. 7, no. 3, pp. 36:1–36:53, 2008.

[31] "Determining bounds on execution times," in *Handbook on Embedded Systems 2005*, R. Wilhelm and R. Zurawski, Eds., CRC Press, 2006.

[32] J. Engblom, A. Ermedahl, M. Sjödin, J. Gustafsson, and H. Hansson, "Worst-case execution-time analysis for embedded real-time systems," *International Journal on Software Tools for Technology Transfer*, vol. 4, pp. 437–455, 2003.

[33] S. Krishnaswamy, S. W. Loke, and A. Zaslavsky, "Estimating computation times of data-intensive applications," *IEEE Distributed Systems Online*, vol. 5, no. 4, 2004.

[34] M. A. Iverson, F. Ozguner, and G. J. Follen, "Run-time statistical estimation of task execution times for heterogeneous distributed computing," in *Proceedings of the 5th IEEE International Symposium on High Performance Distributed Computing*, pp. 263–270, August 1996.

[35] O. Sinnen, *Task Scheduling for Parallel Systems*, Wiley Series on Parallel and Distributed Computing, Wiley-Interscience, 2007.

[36] T. Murata, "Petri nets: properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[37] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice Hall PTR, 1981.

[38] F. Cottet, J. Delacroix, C. Kaiser, and Z. Mammeri, *Scheduling in Real-Time Systems*, John Wiley & Sons, 2002.

[39] F. Verdier, B. Miramond, M. Maillard, E. Huck, and T. Lefebvre, "Using high-level RTOS models for HW/SW embedded architecture exploration: case study on mobile robotic vision," *EURASIP Journal on Embedded Systems*, vol. 2008, no. 1, Article ID 349465, 2008.