Novel Approach for Modeling Very Dynamic and Flexible Real Time Applications

Ismail Ktata, Fakhreddine Ghaffari, Bertrand Granado, Mohamed Abid

Novel Approach for Modeling Very Dynamic and Flexible Real Time Applications

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 F95000 Cergy-Pontoise, France*
[2]*Computer & Embedded Systems Laboratory (CES), National School of Engineers of Sfax, (ENIS), B.P.W. 3038 Sfax, Tunisia*
[1]Email: {firstname.name}@ensea.fr
[2]Email: {firstname.name}@enis.rnu.tn

Abstract

Modeling techniques are used to solve a variety of practical problems related to processing and scheduling in several domains like manufacturing and embedded systems. In such flexible and dynamic environments, there are complex constraints and a variety of unexpected disruptions. Hence, scheduling remains a complex and time-consuming process and the need for accurate models for optimizing this process is increasing. This paper deals with dynamically reconfigurable architectures which are designed to support complex and flexible applications. It focuses on defining a solution approach for modeling such applications where there is significant uncertainty in the duration, resource requirements and number of tasks to be executed.

Keywords: ***Dynamically Reconfigurable Architecture, uncertainty, scheduling, modeling methodologies, DFG.***

I. Introduction

Today, integrated silicon applications are more and more complex. Moreover, in spite of its performance, ASICs development is still long and very expensive, and provides inefficient solutions for many applications which are composed of several heterogeneous tasks with different characteristics. In addition, the growing complexity of real-time applications today presents important challenges, in great part due to their dynamic behavior and uncertainties which could happen at runtime [1]. To overcome these problems, designers tend to use dynamically reconfigurable architectures (DRA). The development of the latter opens new horizons in the field of architecture design. Indeed, the DRAs are well suited to deal with the dynamism of new applications and allow better compromise between cost, flexibility and performance [2]. In particular, fine grained dynamically reconfigurable architectures (FGDRA), as a kind of DRAs, can be adapted to any application more optimally than coarse grain DRAs. This feature makes them today an interesting solution when it comes to handle computational tasks in a highly constrained context. 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 some level in two ways: at design time by providing design tools and at run time by providing an operating system that abstracts the lower level of the system [4]. Moreover, such architecture requires the presence of an appropriate operating system that could manage new tasks at run time and under different constraints. This operating system, and to effectively manage dynamic applications, has to be able to respond rapidly to events. This can be achieved by providing a suitable scheduling approach and dedicated services like hardware preemption that decreases configurations and contexts transfer times. To realize an efficient schedule of an application, this operating system needs to know the behavior of this application, in particular the part where the dynamicity can be exploited on a DRA.

In this paper, we are interested in the modeling of applications that could be executed on dynamically reconfigurable architecture. This kind of applications is characterized, in addition to its real-time constraints, by several types of flexibility. The purpose is to improve the performance of the modeling techniques which facilitates the job to design an efficient scheduling approach.

The remainder of this paper is structured as follows: in Section 2, brief review is given about the context and the related work on modeling techniques used in different domains. Section 3 describes our new technique modeling applications targeted to DRA. The Section 4 reports a description of the proposed modeling method and comparisons with other models, while the last Section draws conclusions.

II. Context and problem definition

Today, embedded systems are more and more used in several domains: automobiles, robots, planes, satellites, boats, industrial control systems, etc. An important feature of these systems is to be reactive. A reactive system is a system that continuously reacts to its environment at a rate imposed by this environment itself. It receives inputs from the environment, responds to these stimuli by making a number of operations and produces the outputs used by the environment, known as reactions. Dynamically reconfigurable architectures are an interesting solution for this type of applications. Due to

that emerging range of applications with dynamic behavior, dynamic scheduling for reconfigurable system-on-chip (RSoC) platforms has become an important field of research [2].

A. Problematics

This paper deals with the constraint-based scheduling for real-time applications executed on FGDRA. In particular, we focus on two major problems:

- The modeling of the application that should exhibit its dynamical aspects and must allow the expression of its constraints, in particular real-time constraints.
- The run-time performance of the scheduling algorithm that must be reasonable in term of overhead for a typical application.

The different components of a scheduling problem are the tasks, the potential constraints, the resources and the objective function. Tasks execution must be programmed to optimize a specific objective with the consideration of several criteria. Many resolution strategies have been proposed in literature [5]. Usually these methods assume that processing times can be modeled with deterministic values. They use predictive schedule that gives an explicit idea of what should be done. Unfortunately, in real environments, the probability of a pre-computed schedule to be executed exactly as planned is low [6]. This is because of not only variations, but also because of 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, for an effective resolution, we need to make a significant reformulation of the problem and the solving methods in order to facilitate the incorporation of this uncertainty and imprecision in scheduling [7].

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

- The release time of tasks can be variable, even unexpected.
- New unexpected tasks may occur.
- Cancellation or modification of existing tasks.
- The execution order of tasks on resources can be changed.
- Resources may become unavailable.
- 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.
- 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 job, taking into account whether or not a time of preparation, changing the number of resources needed for a task, etc.

We are considering real cases where some variations could occur and some data may change over the forecast. The model has to be few sensible to data uncertainties and variations, and be flexible to be adaptable to the possible disturbances.

B. Scheduling under uncertainty

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]:

- Proactive scheduling approach aims to build 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 (off-line). 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. Therefore, in this case, a reactive approach may be more appropriate.
- Instead of anticipating future uncertainties, reactive scheduling takes decisions in real-time when some unexpected events occur. A reference deterministic scheduling, determined off-line, is sometimes used and re-optimized. 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 [9] [10]. 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 [11].

Scheduling techniques are quite different depending on the nature of the problem and the type of disturbance considered: resources failure, the duration of the variation and the fact that new tasks can occur, etc. The mainly used methods are dispatching rules, heuristics, metaheuristics and artificial intelligence techniques [12]. In [13] 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 [14] and [15] scheduling problem with uncertain resource availabilities was encountered. Authors used proactive-reactive strategies and heuristic techniques. Another uncertainty case, which is uncertain tasks duration, had been studied in [16] and [17]. Authors discuss properties of robust schedules, and develop exact and heuristic solution approaches.

C. Scheduling model

To study a system we need models to describe it, including significant system characteristics of geometry, information and dynamism. The latter is a crucial system characteristic as it permits to represent how a system behaves and changes states over time. Moreover, dynamic modeling can cover different domains from the very general to the very specific [18]. Model types have different presentations, as shown in figure 1: some are text-based using symbols while others have associated diagrams.

- Graphical models use a diagram technique with named symbols that represent process and lines that connect the symbols and represent relationships and various other graphical notations to represent constraints (figure 1(a), (b) (d)).
- 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 executional behavior of the application.

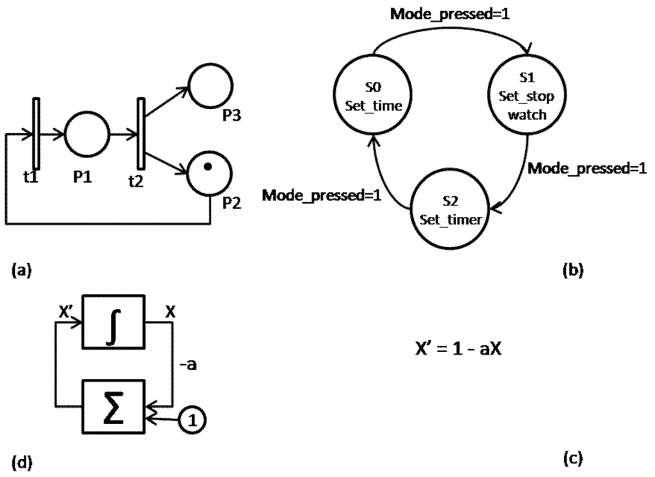

Figure 1. Four types of dynamic system models.
(a) Petri net. (b) Finite state machine. (c) Ordinary differential equation. (d) Functional block model.

In the domain of embedded systems, a large number of modeling languages have been proposed [19], [20], [21], including extensions to finite state machines, data flow graphs, communicating processes, and Petri nets, among others. In this section we present main models of computation for real-time applications reported in the literature.

- Finite State Machines

The classical 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 that have to be explicitly captured in the model as the system complexity increases making the model increasingly difficult to visualize and analyze [22]. For dynamic systems, the FSM representation is not appropriate because the only way to model such kind of systems 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.

- 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 represent the partial order followed by the data. However, the conventional model is inadequate for representing the control unit of systems [23]. It provides no information about the ordering of processes. It is therefore inappropriate to model dynamic applications.

- 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 [18]. Petri Net is a 5-tuple PN = (P, T, F, W, Mo) 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 ➔ {1, 2, 3, ... } 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 [24]. 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 [25]. The conventional model suffers from good specification of complex systems like lack of the notion of time which is an essential factor in embedded applications and lack of hierarchical composition [26]. 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 allow 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 [18].

Most of the methodologies mentioned provide no sufficient support for systems which include variable dynamic features. Dynamic creation of tasks for instance is not supported by most of the systems above mentioned [26]. In [26], authors proposed an extension of high-level Petri net model [27] in order to capture dynamically modifiable embedded systems. They coupled that model with graph transformation techniques and used a double pushout 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.

III. Proposed method

Before beginning to describe our modeling method, we define the constraints that typically appear in dynamic systems. In our case, we consider a firm real-time context. In fact, for actually developed applications, especially multimedia and control applications, tardiness or deadline violations results only in degradation of Quality of Service (QoS) without affecting good processing. In this context hardware tasks are characterized by the following parameters:

- Execution time (Ci),
- Deadline (Di),
- Periodicity (Pi),
- Precedence constraints among tasks,
- Tasks could be preempted,
- Used resources of the DRA.

In real world, all characteristics may change: tasks (the release time, deadline and execution time), as well as the availability of resources. There are several types of changes like uncertainty, unexpected changes and values variation. For our study, we will consider three cases of dynamic scheduling problems for dynamic 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 those cases, the goal is to develop a robust scheduling method that is little sensible to data uncertainties and variations between theory and practice. In addition, the

schedule has to be flexible to be adaptable to the possible disturbances. Therefore, we consider a proactive-reactive approach. The proactive technique (a reference schedule computed offline for the static part) is used to facilitate the execution of the reactive strategy online, so that scheduling decisions have a better quality and produced in a shorter time. To represent all those constraints in the same model and to be adequate for the adopted scheduling approach, our graph will be composed of two forms of nodes. The first type of nodes refers to static tasks which are known in advance and whose execution is permanent during the whole application of the lifecycle execution. The second type is for dynamic tasks which could be executed in a cycle and not in others and with a variable number of needed resources. Therefore, the first step is to separate the two parts of the application (static and dynamic). Then, a priority-based schedule is established offline for static part. Such a schedule serves very important functions. The first is to allocate cells of the hardware DRA to the different hardware tasks. The second is to serve as a basis for planning tasks that may occur during execution. The basic idea is to sort static tasks from the graph and schedule them according to a priority scheme, while respecting their precedence constraints (topological order). Tasks are executed at the earliest possible time. If there is equality between some tasks, task which has maximum execution time will have priority to be launched before the others. At runtime, the objective is to generate a new schedule that deviates from the original schedule as little as possible so that the repair operations will be mostly simple. Therefore, the online scheduler must take into account the next tasks to be performed with their dynamic aspects (different duration, more or less instances to execute, more or less number of needed cells for execution). It has to prefetch configurations context of new tasks in the columns that are available for executing and to find the possible way to integrate them in the current schedule without effect on performance. This schedule repair must rely on rapid algorithms based on a simple scheduling technique so that it can perform online execution with no overhead. It consists in finding a suitable partitioning of N tasks, forming the application, to be executed on M target resources of the hardware devices. In addition, tasks execution order has to meet the real time constraints. This scheduling problem is known to be NP-hard [28] [29]. In this context, heuristics are schedule repair methods which offer fast and reasonably good solution but do not guarantee to find an optimal schedule.

We make use of the example shown in figure 3 in order to illustrate the different definitions corresponding to our model. For this example, the set {T1, T2, T3, T4, T5, T6, T7, T8} represents the static tasks which are always executed in each period and {T9, T10, T11} are dynamic tasks which may be executed in some period but not in others and whose number of resource requirements is variable. Each task is represented with time characteristics: Ci for the execution time and Di for the deadline.

The first dynamic feature considered in this model is tasks with variable execution time. To represent that case, we are inspired by the Program Evaluation and Review Technique (PERT) [30], which is a network model that allows randomness in activity execution times. For each task, PERT indicates a start date and end time at the earliest and latest. The chart identifies the critical path which determines the minimum duration of the project. Tasks are represented by arcs with an associated number presenting the tasks duration. Between arcs, we find circles marking events of beginning or end of tasks (figure 2). In this model, we replace the release time feature by the execution time as it would be changed over execution, and we keep the deadline as it will be useful to minimize the makespan (or the length of the schedule). In figure 3, tasks with variable execution time are {T1, T7, T8}. This will be noticed by the use of asterisk.

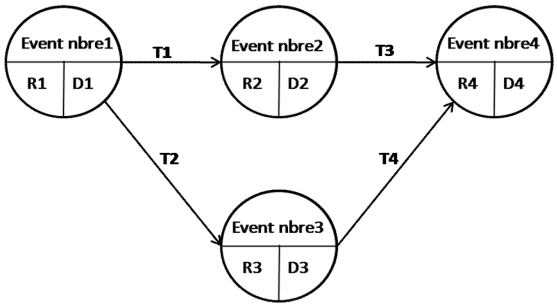

Figure 2. PERT graph

To be executed, hardware tasks need a minimum number of resources. The percentage of this minimal number is indicated in labels over tasks nodes. This case is frequently found on some computer vision applications where a first task is detecting objects and then a particular processing is applied on each detected object. To execute multiple instances of this process, the minimum needed number of resources will be multiplied by the number of instances. In figure 3, task T9 will be executed *n* times. T9 is represented with circled node and the minimum needed number of resources indicated in the label will be multiplied by *n*. If the integer number *n*=0 then task will not occur. The instance number is unexpected from the static phase and decision will be taken online. The number *n* will depend on the previous executions and the actual input data to be processed (such as keypoints in the robotic vision application). Thus, *n* will be recalculated, after each period, based on its previous values. For more details, we take an example of execution. In each iteration the scheduler will have two values of *n*: n_p which is a predicted value of *n* to be used by the scheduler for the next period, and n_r which is the real value of *n* for the executed task. At t=0, let $n= n_p =0$ (no prediction to execute T9). In the first execution, the application needs $n= n_r =10$ instances of T9. So, for the second iteration, the scheduler will predict to execute $n= n_p =10$ instances of T9, while, in the real execution, $n= n_r =6$. For the next execution, n_p could be: the maximum of precedent values, the highest values of *n*, the average of real last values ($n= n_p =8$), or a Gaussian like probability which is a typical realistic distribution, etc. The choice will depend on the application. For the example of a moving robot which need to predict the direction and the presence or absence of obstacles, it will be preferred to take the last real values with different weights. In our model, the number *n* is represented above the arcs. The arcs represent the dependencies between tasks. For static (permanent) tasks, we represent arcs with solid lines, while unpredictable dependencies are represented by dashed lines.

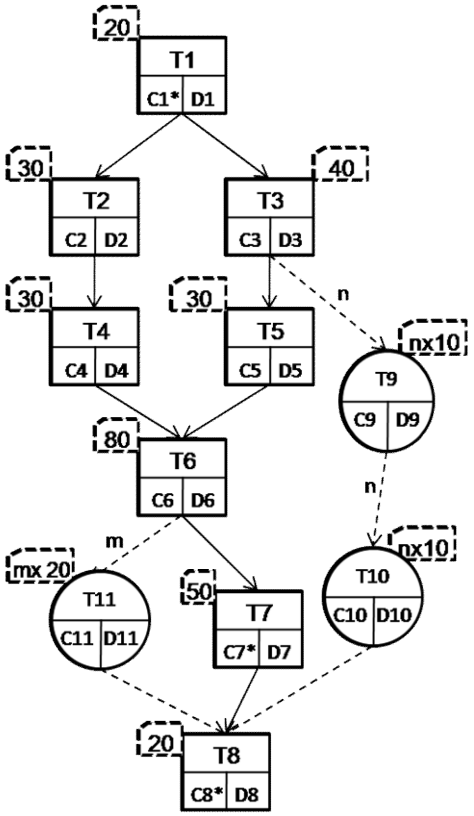

Figure 3. New model for dynamic application

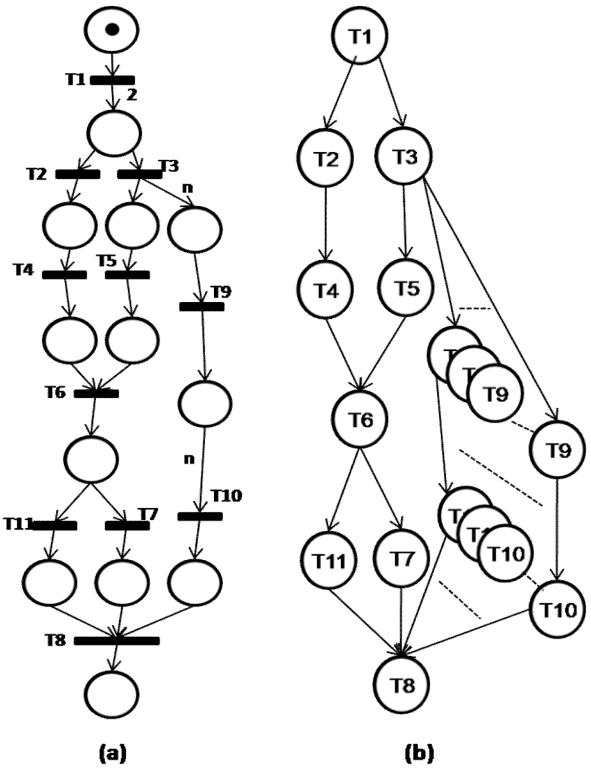

Figure 4. (a) A Petri net representation of the example of figure 3. (b) A DFG representation of the example of figure 3.

On the reconfigurable device, resource failure may occur which will affect the resource availability. Thus, execution may occur or not depending on the available resources compared with needed ones. A variable, which is initialized as the total number of resources, will indicate the amount of remaining resources. From the ready list, 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.

IV. Comparison of models

When we compare with other models (section 2-C), the proposed technique presents several advantages. For unpredicted number of occurring tasks, the data flow graph 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 [31]. 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 4(b)). In our model, information about number of instances of a same task to be executed is noted by the circle form of the task and the indicated number above its arc. For PN model, (see figure 4(a)), arcs could be labeled with their weights where a k-weighted arc can be interpreted as the set of k parallel arcs [32]. But, from its definition (section 2-C), weights are positive integers, so it cannot present a fictive arc with non firing transition representing a task that may not be executed in some iterations. In figure 4(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).

Timing information is useful for determining minimum application completion time, latest starting time for an activity which will not delay the system, and so on. We have inspired from the PERT chart to explicitly represent useful time features that are, in our case, execution time and deadline. However, Petri net does not provide any of this type of information. 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 5). The addition of timing information might provide a powerful new feature for Petri nets but may not be possible in a manner consistent with the basic philosophy of Petri nets Research [33].

For resource representation, PN models 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 gives 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 in our case. In fact, with PN and DFG model (figure 4), there is no distinction between static tasks and dynamic ones (that may not be executed), nor an explicit notion of time (as variable execution time of some tasks).

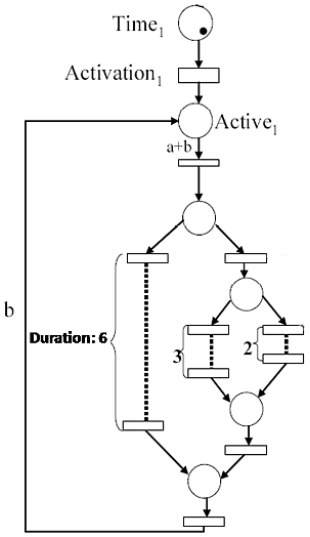

Figure 5. PN model for variable tasks duration

The main advantage of the proposed method is the possibility to present several dynamic features of real time applications with the minimum of nodes and thus in a simple formalism. Indeed, from the first sight of the model, we can bring out three main characteristics of this model:

- The distinction between the static execution (which is presented by the squared nodes) and the dynamic execution i.e. the tasks whose execution and number of instances is uncertain (presented by the circled nodes),
- The tasks whose execution time is variable (presented by the asterisk),
- The percentage, 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.

Those informations will be useful further for the scheduler. We consider a 1D area model as it is a commonly used reconfigurable resource model, but even 2D area model could be considered. In that considered 1D model, tasks can be allocated anywhere along the horizontal device dimension; the vertical dimension is fixed and spans the total height of the hardware task area (see figure 6).

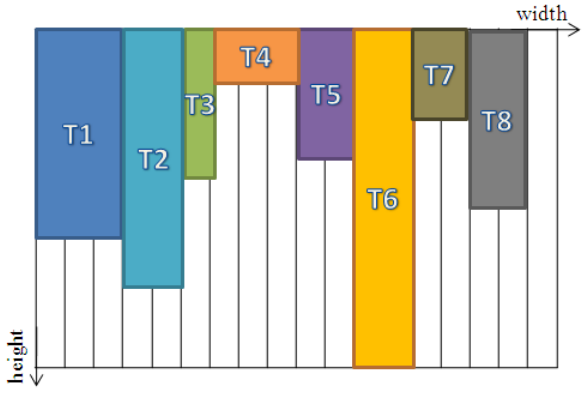

Figure 6. 1D area model of a reconfigurable device

Based on the proposed tasks model, statically defined tasks represented by squared nodes, will be scheduled and placed on the reconfigurable device during the proactive phase. Whereas dynamically defined tasks, which are represented by circled nodes, will be scheduled online in a reactive phase. This online decision will take into account the variable parameters and try to fit into the set of already guaranteed tasks, to delay or to reject these new dynamic tasks.

V. Conclusion

This paper presented a particular modeling problem dealing with the implementation of dynamic and flexible applications on dynamically reconfigurable architecture. The purpose is to consider most of the dynamic features supported by the architecture and to present them in an easy and efficient method. For that point we have proposed a model, based on some features of existing modeling techniques, and which is more dedicated to dynamic real time applications. The main advantage of our specification model is the possibility to obtain more exact scheduling characteristics from the representation. Those characteristics include the distinction between static and dynamic occurring tasks, bring out tasks whose execution time may change over the time and determine the number of resources needed for executing hardware tasks. So, we will be able either to take decisions or not. As a reconfigurable architecture, we target OLLAF [4] (Operating system enabled Low LAtency Fgdra), an original FGDRA specifically designed to enhance the efficiency of an RTOS services necessary to manage such architecture. Future works will consist in integrating our scheduling approach among the services of an RTOS taking into account the new possibilities offered by OLLAF.

VI. References

[1] C.Steiger, H.Walder, M.Platzner, "Operating systems for reconfigurable embedded platforms: online scheduling of real-time tasks", Computers, IEEE Transactions on Volume 53, Issue 11, Nov. 2004 Page(s): 1393 - 1407.

[2] J. Noguera, R.M. Badia, "Multitasking on reconfigurable architectures: Microarchitecture support and dynamic scheduling", ACM Transactions on Embedded Computing Systems, Volume 3, Issue 2 (May 2004) pp. 385-406.

[3] A. Mtibaa, B. Ouni and M. Abid, "An efficient list scheduling algorithm for time placement problem", Computers and Electrical Engineering 33 (2007) 285–298.

[4] S. Garcia, B. Granado, "OLLAF: a Fine Grained Dynamically Reconfigurable Architecture for OS Support", EURASIP Journal on Embedded Systems, October 2009.

[5] P. Brucker & S. Knust, "Complex Scheduling", Springer Berlin Heidelberg, 2006.

[6] V. T'kindt and J.-C. Billaut, "Multicriteria Scheduling: Theory, Models and Algorithms", Springer-Verlag (Heidelberg), second edition (2006).

[7] N. González, R. Vela Camino, I. González Rodríguez, "Comparative Study of Meta-heuristics for Solving Flow Shop Scheduling Problem Under Fuzziness", Second International Work-Conference on the Interplay Between Natural and Artificial Computation, IWINAC 2007, Spain, June 18-21, 2007, Proceedings Part I : 548-557.

[8] A. J. Davenport and J. C. Beck, "A Survey of Techniques for Scheduling with Uncertainty", accessible on-line at http://tidel.mie.utoronto.ca/publications.php on February 2006, 2000.

[9] H.Aytug, M.A.Lawley, K.McKay, S.Mohan, R.Uzsoy, "Executing production schedules in the face of uncertainties: A review and some future directions", European Journal of Operational Research 161, 2005, p86-110.

[10] W.Herroelen and R.Leus, “Project scheduling under uncertainty: Survey and research potentials”, European Journal of Operational Research, Vol. 165(2) (2005) 289--306.

[11] G.E.Vieira, J.W.Hermann, and E.Lin, "Rescheduling manufacturing systems: a framework of strategies, policies and methods", Journal of Scheduling, 6 (1), 36-92 (2003).
[12] D. Ouelhadj, and S. Petrovic, "A survey of dynamic scheduling in manufacturing systems", Journal of Scheduling, 2008.
[13] Peter B. Luh, Feng Liu and Bryan Moser, "Scheduling of design projects with uncertain number of iterations", European Journal of Operational Research, 1999, vol. 113, issue 3, pages 575-592.
[14] O.Lambrechts, E.Demeulemeester, W.Herroelen, "Proactive and reactive strategies for resource-constrained project scheduling with uncertain resource availabilities", Journal of scheduling 2008, vol.11, no.2, pp. 121-136.
[15] S.Liu, K.L.Yung, W.H.Ip, "Genetic Local Search for Resource-Constrained Project Scheduling under Uncertainty", International Journal of Information and Management Sciences 2007, VOL 18; NUMB 4, pages 347-364.
[16] M. Turnquist and L. Nozick, "Allocating time and resources in project management under uncertainty", Proceedings of the 36th Annual Hawaii International Conference on System Sciences, Island of Hawaii, January 2003.
[17] J. Christopher Beck, Nic Wilson, "Proactive Algorithms for Job Shop Scheduling with Probabilistic Durations", Journal of Artificial Intelligence Research 28 (2007) 183–232.
[18] P.A Fishwick, "Handbook of dynamic system modeling", Chapman & Hall/CRC Computer and Information Science Series 2007.
[19] S. Edwards, L. Lavagno, E. A. Lee, and A. Sangiovanni-Vicentelli, "Design of Embedded Systems: Formal Models, Validation, and Synthesis", Proc. IEEE, 85(3):366–390, March 1997.
[20] L. Lavagno, A. Sangiovanni-Vincentelli, and E. Sentovich, "Models of computation for embedded system design". In A. A. Jerraya and J. Mermet, editors, System-Level Synthesis, pages 45–102, Dordrecht, 1999. Kluwer
[21] A. Jantsch. "Modeling Embedded Systems and SoC's: Concurrency and Time in Models of Computation". Morgan Kaufmann, San Francisco, CA, 2003.
[22] L. Alejandro Cortes, "Verification and Scheduling Techniques for Real-Time Embedded Systems", Ph. D. Thesis No. 920, Dept. of Computer and Information Science, Linköping University, March 2005.
[23] L. Alejandro Cortés, P. Eles and Z. Peng, "A Survey on Hardware/Software Codesign Representation Models", SAVE Project, Dept. of Computer and Information Science, Linköping University, Linköping, June 1999.
[24] Carsten Rust, Franz J. Rammig: "A Petri Net Based Approach for the Design of Dynamically Modifiable Embedded Systems". DIPES 2004: 257-266.
[25] M. Tavana, "Dynamic process modelling using Petri nets with applications to nuclear power plant emergency management", Int. J. Simulation and Process Modelling (2008), Vol. 4, No. 2, pp.130–138.
[26] Franz-Josef Rammig, Carsten Rust, "Modeling of Dynamically Modifiable Embedded Real-Time Systems", WORDS Fall 2003: 28-34.
[27] E. Badouel and J. Oliver. "Reconfigurable Nets, a Class of High Level Petri Nets Supporting Dynamic Changes". In Proc. of a workshop within the 19th Int'l Conf. on Applications and Theory of Petri Nets, 1998.
[28] Michael Garey and David Johnson, "Computers and Intractability: A Guide to the Theory of NP-completeness", Freeman, 1979.
[29] Z.A. Mann, A. Orbán, "Optimization problems in system-level synthesis". Proceedings of the 3rd Hungarian-Japanese Symposium on Discrete Mathematics and Its Applications, Tokyo-Japan (2003).
[30] F. Chauvet, J.-M. Proth, "The PERT Problem with Alternatives: Modelisation and Optimisation", Report N° RR-3651 (1999) SAGEP (INRIA Lorraine) France.
[31] Oliver Sinnen. "Task Scheduling for Parallel Systems" (Wiley Series on Parallel and Distributed Computing). Wiley-Interscience, 2007.
[32] Tadao Murata, "Petri nets: Properties, Analysis and Applications", Proceedings of the IEEE, 77(4):541-574, April 1989.
[33] James L. Peterson, "Petri net theory and the modeling of systems", Prentice Hall PTR, 1981.