A Constraint-Based Model for High-Performance Real-Time Computing

Christophe Guettier
AXLOG, 19-21, rue du 8 mai 1945
F-94110 Arcueil, France
Christophe.Guettier@axlog.fr

Jean-François Hermant
INRIA, Projet REFLECS, B.P. 105
F-78153 Le Chesnay Cedex, France
Jean-Francois.Hermant@inria.fr

Abstract

The design of critical and real-time high-performance systems has become a complex matter. To tackle this complexity, the whole problem formulation can be broken down into several models. We present a system modelization that addresses parallel computation. A partial solution, based on the Earliest Deadline First (EDF) algorithm, is emphasized. We show in this paper how a Constraint Logic Programming (CLP) language is able to construct models and the partial solution to efficiently find solutions using a composite search process. In order to fulfill the CLP requirements, we present the necessary refinement steps of an EDF-based solution. We conclude with various examples provided by an efficient CLP implementation.

Keywords

Parallel Processing, Real-Time Scheduling, Computation Models, Constraint Logic Programming, Problem Solving

1 Introduction

Deficiencies in the design or dimensioning of a critical and real-time high-performance system can cause definitive failures. Right now, it is necessary to prove that for the whole system, real-time and architectural size requirements are met. A relevant example of correct system dimension is ensuring that the processing power is sufficient, and efficient. This is a tremendous matter as the increasing complexity of systems involves highly combinatoric design.

The lack of a method for correctly and provably designing and dimensioning complex and/or critical computer-based systems is the main reason why a growing number of major failures are being experienced by the industry [6, 7]. A proof-based system engineering method, such as the TRDF[1] method, involves correctness proof obligations. The TRDF method allows translation of the (incomplete and/or ambiguous) description of an application problem into the specification of a computer science problem. It also can produce the specification of a computer-based system, along with proofs that all the decisions made during the system design and the system dimensioning do satisfy the specification of the computer science problem considered.

Meanwhile, the complexity of target architectures has increased too. For instance, many functional units can be available on a single chip organized in different levels. Static techniques based on code transformations such as tiling [14], loop partitioning [15], or fine-grain scheduling [13] have been improved to tackle the resources or the latency of a program. However, combining multiple resources and latency constraints in order to address the global problem formulation means solving NP-Hard problems with non-linear constraints [13, 16].

In fact, getting a proven global solution is very difficult. The necessary combination of several hard subproblems leads to the concept of modelization. This step extracts and adapts the invariants of system functions, addressing the complexity of the system from coarse to fine grain. So doing, solving the complete problem requires combining different models. In another context, this is similar to automatically solving data-layout for High-Performance FORTRAN, by using $0-1$ modeling associated to branch-and-bound searches [12]. Other relevant results have shown the efficiency of this approach to model and solve mapping problems in the VLSI domain [11, 18]. In fact, to be efficient, it is necessary to distribute the solving over several models using the compositionality property. Thus, it is fundamental to preserve this property during the modelization.

Stemming from logic programming, integer and mathematical programming, Constraint Logic Programming (CLP) languages are recognized as powerful tools to cope with difficult and large combinatorial problems [20]. Underlying models of the approaches presented below could easily be expressed using a CLP

[1] TRDF is the French acronym for Real-Time Distributed Fault-Tolerant Computing.

language based on the simplex algorithm [19]. However the solving method would be restricted to a linear existential part of the constraint language.

Recent results allow us to use less restricted languages defined on finite parts of $\mathbb{N}$ with all its classical operators [23]. The power of this constraint language design enables to tackle non linear constraints. We claim that this language can support the modelization of complex real-time parallel systems [11]. A model is created with complex constraints that represent the invariants of a sub-problem. Relations between models are conjunctions of constraints that maintain the consistency of the global solution. The mathematical composition of the models is transformed into concurrent search processes that can be controlled with powerful logical operators [24].

To design and dimension complex systems, we must solve for crucial parameters representing locality, parallelism, and periodicity. We present a solution based on a block-cyclic computation distribution [15] using on-line Earliest Deadline First (EDF) scheduling. The well-known corresponding algorithm belongs to the class of deadline-driven scheduling algorithms and dominates any fixed-priority scheduling algorithm [3]. In section § 2, we present the global problem specification, the composite calculation is explained in § 3 and we present encouraging results using a CLP implementation in § 4.

2 The Problem

In this section we specify the problem under consideration according to the TRDF method [6, 7]. First, we state the problem models (§ 2.1). The system is modeled in a fairly standard fashion, taking into account the time model, the task model, the external event types models, the computational model, and the architectural model.

Second, we state the problem properties (§ 2.2): the timeliness property. Models support parallelism to speed-up task activations and executions. In that way, it is possible to meet hard real-time constraints that would not be satisfied with one processing element.

2.1 Global Relational Model

Later in this paper, demonstrations are performed with the assumptions explained in the following sections. The aim is to break down the global problem into several sub-problems of lower complexity. A sub-problem can be represented as a model due to its own mathematical invariants. Finding a set of feasible solutions to a given sub-problem requires the instantiation of all the variables of the associated model. We distinguish the initially instantiated models from the models to be instantiated.

A set of relations, described in (§ 2.1.2), results from this decomposition, and correlates between model variables. It maintains the consistency of the different local solutions and consequently, of a global feasible solution (Fig. 1).

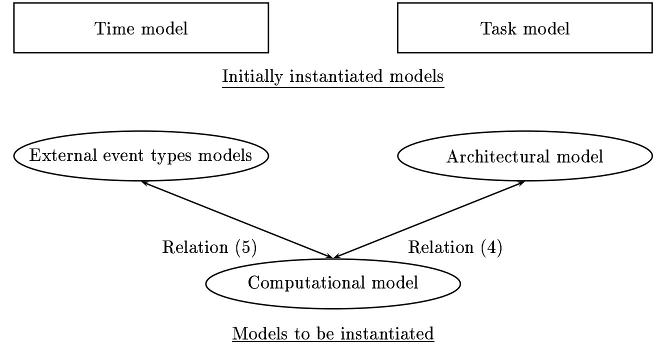


Figure 1: The problem models

2.1.1 Initially Instantiated Models

Time Model Formally, we will use the continuous time model, which is more general than the discrete time model: $t \in \mathbb{R}^+$.

Task Model Consider a level of specification where each task contains an independent local loop that includes an elementary computation (EC). Loop iterations are executed each time the task is activated. This loop could result from a fine-grain parallelization such as tiling [17]. The set of activations may be represented as an infinite loop; so a task can be represented by the following nested loop:

```
DO t_i
   DOALL j_i = 0, U_i - 1
      EC_i(t_i, j_i)
```

where $\forall i,\ t_i \in \mathbb{N}$ is the iterator that represents activations and $\forall i,\ j_i \in \mathbb{N}$ is the local iterator. An elementary duration $\forall i,\ n_i \in \mathbb{R}^+$ is associated to the elementary computation EC_i.

There are no dependencies between the different tasks.

2.1.2 Models to be Instantiated with Relations

These models are instantiated using an arithmetic reasoning involving automatic solving.

External Event Types Models Activation demands of a task are constrained by a periodic or sporadic arrival model [10].

Periodic	**Sporadic**
The $(t_i+1)^{th}$ activation date of a task i is denoted $d(t_i)$:	
$\forall i, \phi_{i,0} \in \mathbb{N}, t_i \in \mathbb{N},$	$\forall i, \phi_{i,j} \in \mathbb{N}, j \in \mathbb{N}, t_i \in \mathbb{N},$
$d(t_i) = \phi_{i,0} + t_i T_i.$	$d(t_i) = \sum_{j=0}^{t_i} \phi_{i,j} + t_i T_i.$
The periodicity or sporadicity interval of a task i is denoted $T_i \in \mathbb{N}$.	
The concrete or non-concrete attributes of a task i.	
Concrete: $\phi_{i,0}$ known.	$\{\phi_{i,j}\}_{j\in\mathbb{N}}$ known.
Non-concrete: $\phi_{i,0}$ unknown.	$\{\phi_{i,j}\}_{j\in\mathbb{N}}$ unknown.

The sporadic model is stronger than or equal to the periodic model [10].

$$Sporadic \supseteq Periodic.$$

We consider in both cases the non-concrete form where periods may be known natural numbers or characterized by a lower and upper bound.

Computational Model Distribution formulations have been widely used so far to represent loop transformations or to express the compilation of mapping directives [15, 17, 14]. For each task $\mathbf{i}$, we use two well-known forms of distribution in order to size the granularity, the parallelism and the periodicity of the task according to the whole problem.

The local loop $\mathbf{j}_i$ can be simply partitioned [15]; it gives local blocks and partially expresses the parallelism. Let $0 \leq j_i < U_i$, be the independent iterator, we have the constraints:

$$\forall i,\ \exists \mathcal{B}_i\ \in \mathbb{N} \mid j_i = \mathcal{B}_i b_i + \lambda_i,\ 0 \leq \lambda_i < \mathcal{B}_i,$$

where $\mathcal{B}_i$ is the block partitioning parameter that we must solve. The scalar λ_i represents the set of local iterations and b_i is a part of the processor identifier.

When they are independent, activations $\mathbf{t}_i$ may be parallelized due to a cyclic distribution. Parallelization of activations brings more flexibility by stretching periods $\mathbf{T}_i$. The following formulation gives the periodicity and partially the parallelism:

$$\forall i,\ \exists \mathcal{C}_i\ \in \mathbb{N} \mid t_i = \mathcal{C}_i \kappa_i + c_i,\ 0 \leq c_i < \mathcal{C}_i,$$

where $\mathcal{C}_i$ is the cyclic parameter that we must solve. The scalar c_i represents the set of parallel activations, the second part of the processor identifier. The scalar κ_i is the new activation iterator.

That way, the computational model can be also characterized as:

Synchronous: Upper and lower bounds on computational delays (e.g., time taken by a processor to make a computational step) exist and their values are known. The exact computational delays result from the computation distribution constraints and are expressed with the number of local iterations times the number of cycles to perform an elementary computation. The amount of local iterations is defined by the parameter $\mathcal{B}_i$, so that the duration of the computation can be given by:

$$\forall i,\ \exists n_i\ \in \mathbb{N} \mid C_i = n_i \mathcal{B}_i, \qquad (1)$$

where the constant n_i gives the elementary duration for one iteration.

Parallel: The number of processors to map the entire task is given by combining both cyclic distribution and partitioning parameters. For each task i, the couple (c_i, b_i) completely defines all the activations and local iterations processed on one processor (Fig. 2), so that we must consider the following constraints:

$$\forall i,\ 0 \leq c_i < \mathcal{C}_i,\ 0 \leq b_i < \left\lceil \frac{U_i}{\mathcal{B}_i} \right\rceil. \qquad (2)$$

Architectural Model The architectural model is SPMD "Single Process Multiple Data", which means that the different processors can execute only identical processes in parallel. At coarse grain, it may be seen as a centralized architecture, while at fine grain it is parallel execution. The number of processors $\mathbf{P}_{used}$ is upper-bounded by a constant $\mathbf{P}_{max}$.

$$\exists (P_{used}, P_{max}) \mid P_{used} \leq P_{max}. \qquad (3)$$

2.1.3 Relations

From a system engineering viewpoint, the set of the resulting relations necessitates some crucial and critical trade-offs. The first issue is the parallelism and the

size of the architecture (§ 2.1.3). The second issue is the periodicity / sporadicity and the available parallelism (§ 2.1.3). These are required to efficiently solve global problems such as the system dimensioning.

Relation between Computational and Architectural models

The distribution of the computation for each task i leads to the capacity resource constraints of the total number of processors:

$$max_i\left(\mathcal{C}_i\left\lceil\frac{U_i}{\mathcal{B}_i}\right\rceil\right)=P_{used}. \quad (4)$$

Relation between Computational and External event types models

The new period, resulting from the cyclic partitioning is given by:

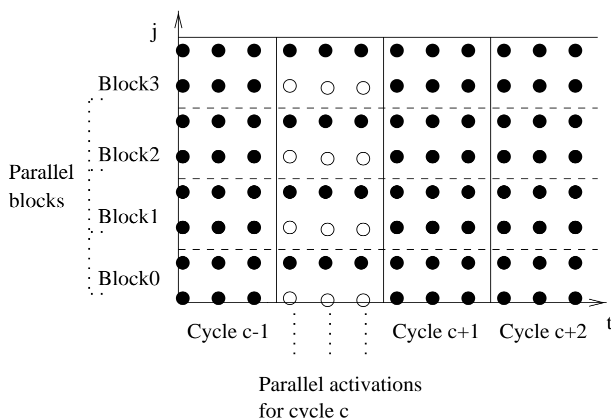
$$\forall i,\ T_i=T_i^0\mathcal{C}_i, \quad (5)$$

where T_i^0 is the original period specified by a user.

proof

The activation date of a sequential task i is given by the function $d(t_i)=T_i^0 t_i$. With the parallel form, the dates of the $\mathcal{C}_i$ activations are the same: $\forall c_i \mid 0\le c_i<\mathcal{C}_i,\ T_i\kappa_i=T_i^0 t_i-T_i^0 c_i$. By using the cyclic partitioning formulation (2.1.2), we have: $T_i\kappa_i=T_i^0(\mathcal{C}_i\kappa_i+c_i)-T_i^0 c_i \Leftrightarrow T_i\kappa_i=T_i^0\mathcal{C}_i\kappa_i \Leftrightarrow T_i=T_i^0\mathcal{C}_i$.

end proof



For a given task with $U=8$, we have $\mathcal{B}=2$ and $\mathcal{C}=3$. Finally, as shown by the unfilled points, 12 elementary computations will be performed in parallel.

Figure 2: Cyclic activations composed with block execution

2.2 Properties

Timeliness Tasks are assigned timeliness constraints: latest termination deadline. For every possible system run, every timeliness constraint is met. On one hand, the value of deadlines does not depend on the parallelism and may be known as a natural number or characterized by a lower and an upper bound: $\forall i,\ D_i\in\mathbb{N}$.

sketch of the proof

Let $T_i^0 t_i+e_i$ be the end of the task execution in the sequential form. The deadline property states: $T_i^0 t_i+e_i\le T_i^0 t_i+D_i$. According to previous formulations (2.1.2,5, § 2.1.2), the parallel execution of the tasks satisfies:

$$T_i^0 t_i+e_i-T_i^0 c_i\le T_i^0 t_i+D_i-T_i^0 c_i \Leftrightarrow T_i\kappa_i+e_i\le T_i\kappa_i+D_i$$

end of the sketch of the proof

On the other hand, the deadline satisfaction would also depend on the parallelism (§ 4).

3 Solving the Problem

Once all the invariants have been expressed through the modelization, it is possible to express the solution to the problem. Following our approach, the calculation is done in two steps. The first one is based on the Earliest Deadline First scheduling solution. Feasibility is expressed due to necessary and sufficient conditions that are out of the scope of CLP capabilities. However, it is possible to extract sufficient conditions that can be directly translated in a CLP Language. The section § 3.1 explains how we refine the solution in order to extract those sufficient conditions.

The second step (§ 3.2) deals with the expression and the solving in CLP of the global problem. It takes into account the formulation of the EDF's invariants, in conjunction with the expression of the system specification. Then, a model-based technique can be applied using CLP features.

3.1 The Earliest Deadline First Partial Solution

The Earliest Deadline First (EDF) scheduling algorithm belongs to the class of on-line real-time scheduling algorithms. There are at least two subclasses: deadline-driven scheduling algorithms and fixed-priority scheduling algorithms. EDF belongs to the former subclass and dominates any algorithm belonging to the latter subclass, such as, Highest Priority First/Rate Monotonic (HPF/RM) or Highest Priority First/Deadline Monotonic (HPF/DM) [3].

There are two main reasons why only EDF is covered in this paper. First, from a theoretical viewpoint, we have the dominance property of EDF. Second, from a practical viewpoint, the implementation of the sufficient feasibility condition for EDF is fairly simple (see further).

EDF works as follows [8]. At any time $t \in \mathbb{R}^+$, if there are pending tasks (i.e., tasks which have been previously activated but which have not been fully completed yet), EDF runs the task which has the earliest absolute deadline. The processor is then said to be busy. To decide between tasks having the same absolute deadline, EDF makes use of a tie-breaking rule (e.g., a lexicographical order). If there are no pending tasks, EDF runs no task. The processor is then said to be idle (Fig. 3).

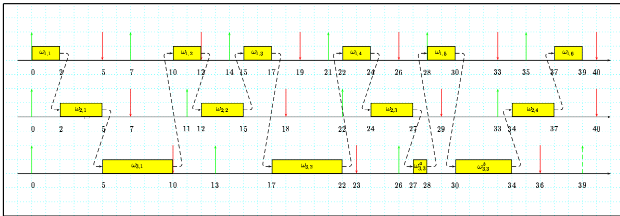


Figure 3: Example of a preemptive EDF schedule

The EDF schedule of the synchronous concrete traffic $\omega \in \tau$, where τ is the traffic made up of the three periodic or sporadic non-concrete tasks $\tau_1 : (C_1, T_1, D_1) = (2, 7, 5)$, $\tau_2 : (C_2, T_2, D_2) = (3, 11, 7)$, and $\tau_3 : (C_3, T_3, D_3) = (5, 13, 10)$.

Later in this paper, we consider a periodic or sporadic non-concrete traffic τ, which is a set of n periodic or sporadic non-concrete tasks τ_j. We state feasibility conditions for EDF with τ.

3.1.1 Basic Concepts

The workload $W(t;\tau)$

$$W(t;\tau) = \sum_{j=1}^{n} W(t;\tau_j) = \sum_{j=1}^{n} \left\lceil \frac{t}{T_j} \right\rceil C_j. \qquad (6)$$

By definition, the workload $W(t;\tau)$ is the amount of time that is needed to run all the tasks whose activation times are in $[0, t)$ [2]. To give the expression of $W(t;\tau)$, we consider the synchronous concrete traffic $\omega \in \tau$.

The processor demand $h(t;\tau)$

$$h(t;\tau) = \sum_{j=1}^{n} h(t;\tau_j) = \sum_{j=1}^{n} Max \left\{ 0, 1 + \left\lfloor \frac{t - D_j}{T_j} \right\rfloor \right\} C_j. \qquad (7)$$

By definition, the processor demand $h(t;\tau)$ is the amount of time that is needed to run all the tasks whose activation times and absolute deadlines are in $[0, t]$ [2]. To give the expression of $h(t;\tau)$, we consider the synchronous concrete traffic $\omega \in \tau$.

3.1.2 Feasibility Conditions for EDF

A necessary feasibility condition (NC)

$$\tau \text{ is feasible by EDF} \Rightarrow \sum_{j=1}^{n} \frac{C_j}{T_j} \leq 1. \qquad (8)$$

sketch of the proof

We derive the utilization factor $U(\tau)$ from the workload $W(t;\tau)$:

$$U(\tau) = Lim_{t \to \infty} \left\{ \frac{W(t;\tau)}{t} \right\} = \sum_{j=1}^{n} \frac{C_j}{T_j}.$$

By definition, the utilization factor $U(\tau)$ is the fraction of time that is needed to run all the tasks over $[0, \infty)$, i.e., the limit of $W(t;\tau)/t$ as t tends to infinity. If τ is feasible by EDF, then $U(\tau) \leq 100\%$.

end of the sketch of the proof

A necessary and sufficient feasibility condition (NSC)

$$\tau \text{ is feasible by EDF} \Leftrightarrow \forall t \in \mathbb{R}^+,\ h(t;\tau) \leq t; \qquad (9)$$

$$\tau \text{ is feasible by EDF} \Leftrightarrow Sup_{t \in \mathbb{R}^{+*}} \left\{ \frac{h(t;\tau)}{t} \right\} \leq 1. \qquad (10)$$

sketch of the proof

By definition, the processor demand $h(t;\tau)$ is the amount of time that is needed to run all the tasks whose activation times and absolute deadlines are in $[0, t]$. τ is feasible by EDF, if and only if, $\forall t \in \mathbb{R}^+$, $h(t;\tau) \leq t$, i.e., if and only if, $Sup_{t \in \mathbb{R}^{+*}} \{h(t;\tau)/t\} \leq 100\%$.

end of the sketch of the proof

A sufficient feasibility condition (SC)

$$\sum_{j=1}^{n} \frac{C_j}{Min\{T_j, D_j\}} \leq 1 \Rightarrow \tau \text{ is feasible by EDF.} \quad (11)$$

sketch of the proof

Since $Sup\{f+g\} \leq Sup\{f\} + Sup\{g\}$, we have:

$$Sup_{t\in\mathbb{R}^{+*}}\left\{\frac{1}{t}\sum_{j=1}^{n} h(t;\tau_j)\right\} \leq \sum_{j=1}^{n} Sup_{t\in\mathbb{R}^{+*}}\left\{\frac{1}{t}h(t;\tau_j)\right\};$$

$$Sup_{t\in\mathbb{R}^{+*}}\left\{\frac{h(t;\tau)}{t}\right\} \leq \sum_{j=1}^{n} \frac{C_j}{Min\{T_j, D_j\}}.$$

If $\sum_{j=1}^{n} C_j/Min\{T_j, D_j\} \leq 100\%$, then τ is feasible by EDF.

end of the sketch of the proof

Later in this paper, we only consider a sufficient feasibility condition for EDF. Eq. 11 can be easily implemented since its complexity is in $O(n)$.

3.2 Automatic Solving Using CLP Language

In our approach, the resulting problem formulation – EDF constraint-based sufficient conditions in conjunction with the general system modelization – perfectly matches the expressiveness of CLP and its solving capabilities. The distinction between the problem formulation and the solving facilities allows one to find solutions for various goals automatically.

3.2.1 Problem Formulation

A Constraint Logic Programming language can be viewed as an extension of Logic Programming where unification is replaced with constraint satisfaction. Logical predicates can be constraints interpreted in a mathematical algebra [21, 24] which is over the finite domains in our context : $\{\mathcal{P}(\mathbb{N}), +, -, >, =, *\}$. Such a language enables the composition of predicates through logical operators and quantifiers. This leads to a more understandable, compositional and modular problem representation. Model based computing is a practical way to take advantage of those CLP properties [25, 22]. Problem variables and constrained predicates are developed in order to express independently complex models' invariants. The consistency between models is also maintained due to constrained predicates over model variables. Those relations propagate local solutions between models and trigger a backtrack event whenever a constraint cannot be satisfied.

The EDF-partial solution is also expressed such that its constraints are added to those of the global modelization. The sufficient condition certifies that the final solution belongs to the EDF-feasibility domain and thus must be stated. In so doing, the necessary condition becomes redundant. However, this condition is constraining the whole system in a different way. Thus, stating the necessary condition enables one to partition the search space more efficiently.

3.2.2 Concurrent Solving Over Models

We use generic constraint solving algorithms which can handle constraint propagation and arithmetic reasoning. Each model is associated with a solving process that searches for a solution to the corresponding sub-problem. All solving processes are running simultaneously in order to find a global solution that satisfies all the constraints of the problem. To insure the consistency between model solutions and to reinforce the concurrency between the solving processes, CLP offers the powerful control operators *Ask & Tell*. The satisfaction operator *Tell* states a constraint to the solver and the entailment operator *Ask* checks if a constraint is already satisfied [24].

The *Ask & Tell* paradigm has been widely used to compose complex constraints. In our system modelization, we make use of the basic non linear product constraint $Y \leq \prod_i X_i$ and the maximal constraint $Y = max_i(X_i)$.

At the search process level, the *Tell* operator is utilized to exchange partial solutions between models through relations. When two partial solutions are not consistent, the system generates a backtrack event. The *Ask* operator is used for the synchronization of the global search. For example, associated to a variable of a model, it triggers an associated search process when a given property is known. Finally, using those operators, global resolution strategies including domain heuristics are designed to control and assist the composite global search over all models.

4 Preliminary Results

The implementation, based on CLP, reflects exactly the whole formulation and the resolution. According to our system modelization (Fig. 1) and the EDF's invariants, we can use the prototype to solve for different goals such as optimizing the number of processors or finding the appropriate set of deadlines according to the processing power. Figure (4) describes the task models of an application, where we can note that some deadlines are unfixed and correspond to application parameters.

Task	*Elementary Duration (ms)*	*Initial Period (ms)*	*Deadline (ms)*	*Parallel Iterations*
`Reading`	12	42	22	1
`Cursor`	3	24	64	12
`Slow Motion`	8	164	D_{SM}?	32
`Sizing`	1	42	D_S?	28

The task `Reading` performs a sequential read of a video from a compact disk. The task `Sizing` adapts the video according to the display context and task `Cursor` screens a pointer. A selected part of the image can be displayed in `Slow Motion`.

Figure 4: Task Set Specification

As an example, we exhibit in Figure (5) the optimization of a deadline which represents the `Slow Motion` fluidity. The system took three steps to find this optimal solution. Thanks to constraint propagation over models, the solver has detected that the task `Cursor` is critical for the optimization. Consequently, the task's period is automatically stretched from 24 ms to 72 ms, using parallelism in order to favor the `Slow Motion` deadline.

Goal specification : Maximal number of processors: 128, Deadlines: $D_{Sizing} \leq 40$, Optimize the fluidity of the `Slow Motion`
Result :

Task	Activation and Local Parallelism $\left(\mathcal{C}, \left\lceil \frac{U}{\mathcal{B}} \right\rceil\right)$	Final Period T	Final Duration C	Final Deadline D
`Reading`	(1,1)	42	12	22
`Cursor`	(*3*,12)	*72*	3	64
`Slow Motion`	(1,32)	164	8	**34**
`Sizing`	(1,14)	42	2	20

Total processors: 36, Load: $\sum_{j=1}^{n} \frac{C_j}{T_j} = .56$

Figure 5: A minimal value for the `Slow Motion` deadline

Using our implementation, other relevant optimizations have been performed and give interesting effects. For example, in one hand the load minimization leads obviously to the maximization of the parallelism (Fig. 6-b), while on the other hand the number of processors can be efficiently minimized (Fig. 6-a). In both cases, the sufficient condition is always satisfied (third column) and the final solution is optimal.

Initial Constraints : Maximal number of processors: 128, Deadlines: $D_{Sizing} \leq 40$, $D_{SlowMotion} \leq 164$

a - architecture minimization

Processors	$\sum_{j=1}^{n} \frac{C_j}{T_j}$	$\sum_{j=1}^{n} \frac{C_j}{Min\{T_j, D_j\}}$
32	.50	.98
24	.49	.97
18	.52	1
16	.49	.99

b - load minimization

Processors	$\sum_{j=1}^{n} \frac{C_j}{T_j}$	$\sum_{j=1}^{n} \frac{C_j}{Min\{T_j, D_j\}}$
32	.50	.98
36	.47	.95
60	.44	.95
36	.38	.98
60	.35	.98
36	.32	1
60	.29	1
60	.26	.99
60	.25	.99
84	**.24**	.99

Figure 6: Optimization of system parameters

5 Conclusions and Further Work

As cost functions and execution constraints grow complex, simple resolution schemes will no longer suffice [13, 14]. To achieve global optimizations, we have shown on a real-time and parallel system that dividing the resolution from the modelization is a necessity. This can be done thanks to a methodology such as TRDF.

Under those considerations, Constraint Logic Programming holds the appropriate level of expressiveness to compose models and partial solutions such as the EDF-scheduling policy. This way, the problem can be solved globally, using concurrent constraint programming mechanisms over models. However, this requires preserving the compositionality property by finding sufficient conditions, approximations or by refining the model formulations.

As an example, we have considered the sufficient feasibility condition for EDF. This is less precise than the necessary and sufficient feasibility condition, but it can be implemented more easily in CLP languages. It defines a convex feasibility domain. The vertices of this polyhedron are computable for free. In some cases, it is possible to consider the necessary and sufficient feasibility condition. It also defines a convex feasibility domain. However, the vertices of this polyhedron are not computable for free. This work will be presented in a forthcoming paper.

Future works will also introduce dependencies at

the instruction level or between coarser tasks, and thus communication requirements. The scope of architectures will also be extended by considering multi-level partitioning schema. We are currently developing a framework based on Constraint Logic Programming where components are formal models extracted from the state of the art of parallel and real-time computing. We expect such a framework to be a helpful tool for designing and sizing complex high-performance systems.

Acknowledgements

We thank Corinne Ancourt and François Irigoin for their constructive comments and suggestions in improving this paper. We are deeply grateful to Karen and Tom Conroy for their great help in revising the paper. Finally, we would like to thank Thierry Billoir and Gérard Le Lann for their continuous support.

References

[1] S. Baruah, R.R. Howell, L. Rosier, *Algorithms and complexity concerning the preemptive scheduling of periodic real-time tasks on one processor*, Real-Time Systems, 2, pp. 301-324, 1990.

[2] S. Baruah, A. Mok, L. Rosier, *Preemptively scheduling hard real-time sporadic tasks on one processor*, 11th Real-Time Systems Symposium, pp. 182-190, 1990.

[3] M. Dertouzos, *Control Robotics: the procedural control of physical processors*, Proceedings of the IFIP congress, pp. 807-813, 1974.

[4] J.-F. Hermant, *Some Real-Time Scheduling Problems and Solutions for Distributed Systems*, PhD Thesis, University of Paris VI, 1999.

[5] J.-F. Hermant, L. Leboucher, N. Rivierre, *Real-time fixed and dynamic priority driven scheduling algorithms: theory and experience*, INRIA Research Report 3081, 142 p., 1996.

[6] G. Le Lann, *Proof-Based System Engineering for Computing Systems*, 1st Joint ESA/INCOSE Conference on Systems Engineering - The Future, IEE/ESA Pub., Vol. WPP-130, 5a.4.1-5a.4.8., Nov. 11-13, 1997.

[7] G. Le Lann, *Proof-Based System Engineering and Embedded Systems*, School on Embedded Systems, Veldhoven (NL), Nov. 1996, Lecture Notes in Computer Science, Springer Verlag Pub., 40 p., to appear in 1998.

[8] C.L. Liu, J.W. Layland, *Scheduling algorithms for multiprogramming in a hard real-time environment*, Journal of the Association for Computing Machinery, 20(1), pp. 40-61, 1973.

[9] J.Y.T. Leung, M.L. Merril, *A note on preemptive scheduling of periodic, real-time tasks*, Information processing Letters, 11(3), pp. 115-118, 1980.

[10] A.K. Mok, *Fundamental design problems for the hard real-time environments*, PhD, MIT/LCS/TR-297, 1983.

[11] C. Ancourt, D. Barthou, C. Guettier, F. Irigoin, B. Jeannet, J. Jourdan, and J. Mattioli, *Automatic mapping of signal processing applications onto parallel computers*, In Proc. ASAP 97, Zurich, July, 1997.

[12] R. Bixby, K. Kennedy, and U. Kremer, *Automatic Data Layout Using 0-1 Integer Programming*, In Proc. PACT94, Montreal, Canada, August, 1994.

[13] P. Feautrier, *Fine-Grain Scheduling under Ressource Constraints*, In 7th Workshop on Language and Compiler for Parallel Computer, Ithaca, New-York, August, 1994.

[14] K. Hogstedt, L. Carter and J. Ferrante, *Calculating the idle time of a tiling*, In Proc. ACM Symposium on Principles Of Programming Languages, Paris, January, 1997.

[15] F. Irigoin and R. Triolet, *Supernode Partitionning*, In Proc. 15th POPL, pages 319-328, San Diego, California, January, 1988.

[16] U. Kremer, Optimal and Near-Optimal *Solutions For Hard Compilation Problems*, Parallel Processing Letters 7(4), World Scientific Publishing Company, 1997.

[17] A. W. Lim and M. S. Lam, *Maximizing Parallelism and Minimizing Synchronisation with Affine Transforms* , In Proc. ACM Symposium on Principles Of Programming Languages, Paris, January, 1997.

[18] Jürgen Teich and Lothar Thiele, *Partioning Processor Arrays under Ressource Constraints*, In Journal of VLSI Signal Processing Systems, 15, pp. 5-21, 1997, Klüwer Academic Publishers, Boston.

[19] A. Colmerauer, *Opening the Prolog III universe*, , Bytes, August 1987.

[20] M. Dincbas, P. Van Hentenryck and H. Simonis, *Solving Large Combinatorial Problems in Logic Programming*, Journal of Logic Programming, Vol. 8, p.75-93, 1990.

[21] J. Jaffar and J-L Lassez, *Constraint Logic Programming*, In Proc. of the 14th ACM Symposium on Principles of Programming Languages, Munich, January 1987.

[22] Jean Jourdan, *Concurrence et coopération de modèles multiples dans les langages de contraintes CLP et CC : Vers une méthodologie de programmation par modélisation.* PhD thesis, Université Denis Diderot, Paris VII, 1995.

[23] P. Van Hentenryck, V. Saraswat, and Y. Deville, *Design, Implementation and Evaluation of the Constraint Language CC(FD)*, In Constraint Programming: Basics and Trends, A. Podelski Ed., Chatillon-sur-Seine, Springer-Verlag LNCCS 910, pp. 68-90, 1995.

[24] V. Saraswat, *The Concurrent Logic Programming Language CP: Denotational and Operational Semantics*, In Proc. of the 14th ACM Symposium on Principles of Programming Languages, Munich, January, 1987.

[25] V. Saraswat, D. Bobrow and D. Kleer, *Infrastructure for Model-based computing* TR, Xerox PARC, Palo Alto Ca. March 93