Preprints of WODES 2010
August 30th – September 1st, 2010
Berlin, Germany

Multicriteria optimal reconfiguration of fault-tolerant real-time tasks

Emil Dumitrescu * **Alain Girault** ** **Hervé Marchand** *** **Eric Rutten** ****

* *INSA Lyon, France (e-mail: Emil.Dumitrescu@insa-lyon.fr).*
** *INRIA Grenoble / LIG, France (e-mail: Alain.Girault@inria.fr)*
*** *INRIA Rennes, France, (e-mail: Herve.Marchand@inria.fr)*
**** *INRIA Grenoble / LIG, France (e-mail: Eric.Rutten@inria.fr)*

Abstract: We propose a technique for discrete controller synthesis, with optimal synthesis on bounded paths, in order to model, design, and optimize fault-tolerant distributed systems, taking into account several criteria (e.g., the execution costs of the tasks and their quality of service). Different combinations are explored for multi-criteria optimization.

Keywords: Fault tolerant systems, discrete controller synthesis, multicriteria optimization.

1. MOTIVATIONS AND PROBLEM STATEMENT

An embedded system being intrinsically critical, it is essential to insure that it is tolerant to processor failures. Fault-tolerance is the faculty to *maintain functionality of a system, whatever the failures* under some failure hypothesis. This even motivates distribution itself: the failure of one computing site must not lead to the failure of the whole application. We use formal methods to model systems with fault-tolerance guarantees, in particular we use *discrete controller synthesis* (DCS). The advantages are the correctness of the resulting system and the easy modifiability of the controller (thanks to automatic tools), i.e., the possibility to study and test several fault-tolerance objectives or failure hypotheses on the same system model, without the need to re-design the system. To achieve this, we model our distributed systems, and express formally the fault-tolerance objective, in terms of events and states.

When a solution is found, it can be used either as a guideline for implementation (if the model was an abstract one, see [Dumitrescu et al. (2004)]) or for deployment with dynamic failure recovery (this paper). In our approach, a system consists of a set of real-time periodic tasks placed in a *configuration* onto a set of processors, as in [Dumitrescu et al. (2007)]. Each task has a known worst case execution time (WCET) and is divided into a succession of *phases* separated by *checkpoints*. Upon the occurrence of a failure, one or several processors become unusable, and tasks must be placed anew in another configuration, by migrating them onto other processors, so that execution can proceed. Upon its migration, a task is resumed from its last saved checkpoint (*rollback*). These *reconfigurations* of the system have to be controlled according to a fault-tolerance policy, enforced by a *task manager*. The latter is specified in terms of properties concerning placement constraints, reachability of termination, and optimization of the WCET of the tasks.

The contribution of this paper is twofold: (1) a refined model of the multi-task model, taking into account processor time-sharing, and formulating fault-tolerance guarantee and WCET optimization as a multi-criteria problem (one for each task) and (2) the optimal DCS algorithms exploring different ways of combining the multiple criteria (aggregation, hierarchization, and transformation).

2. RECALLS OF BASIC NOTIONS

We essentially adopt the synchronous framework of [Marchand et al. (2000)] and [Altisen et al. (2003)].

2.1 Labelled transition systems

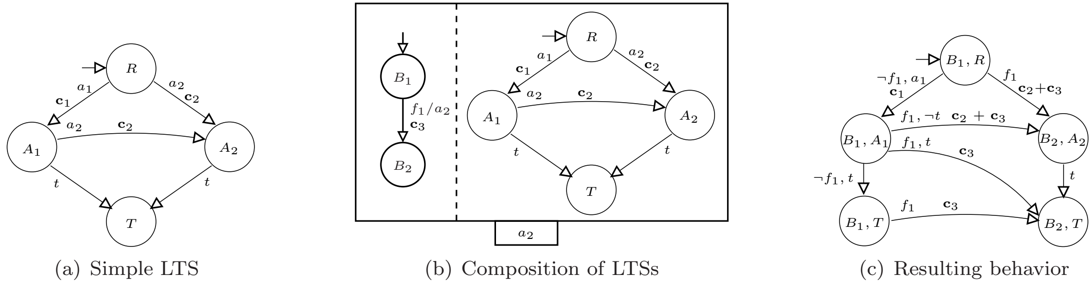
Definition. A LTS is a tuple $S = \langle \mathcal{Q}, q_0, \mathcal{I}, \mathcal{O}, \mathcal{T} \rangle$, where $\mathcal{Q}$ is a finite set of states, q_0 is the initial state of S, $\mathcal{I}$ is a finite set of input events, $\mathcal{O}$ is a finite set of output event, and $\mathcal{T}$ is the transition relation, that is a subset of $\mathcal{Q} \times \mathcal{B}ool(\mathcal{I}) \times \mathcal{O}^* \times \mathcal{Q}$, where $\mathcal{B}ool(\mathcal{I})$ is the set of boolean expressions of $\mathcal{I}$. If we denote by $\mathbb{B}$ the set $\{true, false\}$, then a guard $g \in \mathcal{B}ool(\mathcal{I})$ can be equivalently seen as a function from $2^{\mathcal{I}}$ into $\mathbb{B}$. [1]

Each transition has a *label* of the form g/a, where the *guard* $g \in \mathcal{B}ool(\mathcal{I})$ must be true for the transition to be taken, and where the *action* $a \in \mathcal{O}^*$ is a conjunction of outputs that are emitted when the transition is taken. State q is the *source* of the transition (q, g, a, q'), and state q' is the *destination*; it is noted: $q \xrightarrow{g/a} q'$.

A LTS is *deterministic* (resp. *reactive*) iff, for each state $q \in \mathcal{Q}$ and for each valuation of the inputs, there exists at most (resp. at least) one transition from q and whose guard is true for this inputs valuation.

For optimal discrete control synthesis with respect to quantitative criteria, we assign *weights* to the states and/or transitions of the system. We define $C(q)$ be a cost function mapping each state of a LTS to a positive integer cost value: $C : \mathcal{Q} \to N$. An alternative is to consider a cost function attached to the transition of the system: $\mathcal{T} \to N$.

[1] For any set X, 2^X is the set of all subsets of X.



(a) Simple LTS (b) Composition of LTSs (c) Resulting behavior

Fig. 1. Labelled transition systems.

Figure 1(a) shows the graphical syntax that will be used in this paper, with a very simple example of LTS. Its initial state is R. From there it can go, upon occurrence of a_1, to state A_1, with cost $\mathbf{c}_1$, or upon occurrence of a_2, to state A_2, with cost $\mathbf{c}_2$. From A_1 a transition leads, upon occurrence of a_2, to state A_2, with cost $\mathbf{c}_2$. Finally, from either A_1 or A_2, it goes to state T upon occurrence of t.

Parallel composition. The composition operator of two LTSs put in parallel is the *synchronous product*, noted $||$, as defined in [Milner (1989)] and a characteristic feature of the synchronous languages [Benveniste et al. (2003)]. It is commutative and associative. Formally, if $S_i = \langle \mathcal{Q}_i, q_{i,0}, \mathcal{I}_i, \mathcal{O}_i, \mathcal{T}_i \rangle$, $i = 1, 2$, with $\mathcal{O}_1 \cap \mathcal{O}_2 = \emptyset$, then:

$$\langle \mathcal{Q}_1, q_{1,0}, \mathcal{I}_1, \mathcal{O}_1, \mathcal{T}_1 \rangle \, || \, \langle \mathcal{Q}_2, q_{2,0}, \mathcal{I}_2, \mathcal{O}_2, \mathcal{T}_2 \rangle \\ = \langle \mathcal{Q}_1 \times \mathcal{Q}_2, (q_{1,0}, q_{2,0}), \mathcal{I}_1 \cup \mathcal{I}_2, \mathcal{O}_1 \cup \mathcal{O}_2, \mathcal{T} \rangle$$

$$\text{with } \mathcal{T} = \{((q_1, q_2) \xrightarrow{g_1 \wedge g_2 / a_1 \wedge a_2} (q_1', q_2')) \\ \mid q_1 \xrightarrow{g_1/a_1} q_1' \in \mathcal{T}_1, q_2 \xrightarrow{g_2/a_2} q_2' \in \mathcal{T}_2, g_1 \wedge g_2 \wedge a_1 \wedge a_2\}$$

Here, (q_1, q_2) is called a *macro state*, where q_1 and q_2 are its two *component states.*

When we are concerned by how an individual LTS $S_i = \langle \mathcal{Q}_i, q_{i,0}, \mathcal{I}_i, \mathcal{O}_i, \mathcal{T}_i \rangle$ contributes to a global transition of the synchronous product $S = S_1 || \cdots || S_i || \cdots || S_n$, we write the global transition of S as follows: $q_i X \to q_i' X'$, where $q_i \to q_i'$ is a transition of S_i. Accordingly, X (resp. X') denotes the contribution of the other LTSs of the product (that is, all the LTSs except S_i) in the source state of the transition (resp. in the destination state).

When composing LTSs, the value of a cost function is defined on the resulting global state/transition as either the sum of the local costs of the states/transitions, or considering the maximum, minimum of the local costs, etc. Figure 1(b) shows the graphical syntax for composing LTSs, assembling them in a box, separated with a dashed line. It is illustrated on an example featuring the simple LTS of Figure 1(a), composed with another LTS with just two states, intially B_1, and upon occurrence of f_1, with cost $\mathbf{c}_3$, it emits a_2 and goes into B_2. A further box indicates that a_2 is a local variable. The resulting behavior is in Figure 1(c), where composed states are products of the two state spaces, and when the first LTS can take a transition upon occurrence of input f_1, then a corresponding global transition is taken, where local costs are added up into a global cost. The two local transitions are taken synchronously, i.e., in the same global step.

2.2 Discrete controller synthesis

Initially introduced in the 80's by [Ramadge and Wonham (1989)], DCS allows to use constructive methods, that ensure, a priori, required properties on the system behavior. Within our framework, DCS is an operation that applies on a LTS (originally uncontrolled), where $\mathcal{I}$ is partitioned into two subsets, $\mathcal{I}_c$ and $\mathcal{I}_u$, respectively the set of *controllable* and *uncontrollable* inputs. It is applied with a control objective: a property that has to be enforced by control. We here consider invariance or reachability of a sub-set of states.

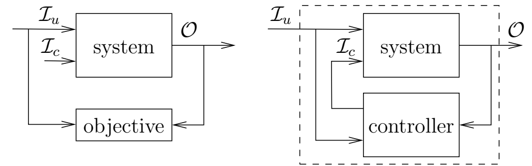


Fig. 2. From uncontrolled system to closed-loop control.

As illustrated in Figure 2, the objective is expressed in terms of the system's outputs. The controller is obtained automatically from a LTS and an *objective* both specified by the user, as computed by appropriate algorithms [Marchand et al. (2000)] which we will use without describing them in detail here. Its purpose is to constrain the values of controllable variables $\mathcal{I}_c$, in function of outputs and of uncontrollable inputs $\mathcal{I}_u$, such that all remaining behaviours satisfy the property given as objective.

A controller is a function $\mathcal{C} : \mathcal{O}^* \times \mathcal{I}_u \to \mathcal{B}ool(\mathcal{I}_c)$. For a given output $a \in \mathcal{O}^*$ and a given valuation of the uncontrollable variables i_u, $\mathcal{C}(a, i_u)$ is a boolean predicate over the variables $\mathcal{I}_c$, such that $\mathcal{C}(a, i_u)(i_c) = true$ means that the controller allows the controllable variables to valuate to i_c. The automaton S controlled by the controller $\mathcal{C}$ is another automaton $(\mathcal{Q}, q_0, \mathcal{I}, \mathcal{O}, \mathcal{T}^{\mathcal{C}})$, noted $(S, \mathcal{C})$, such that $\mathcal{T}^{\mathcal{C}} \subseteq \mathcal{Q} \times \mathcal{B}ool(\mathcal{I}) \times \mathcal{O}^* \times \mathcal{Q}$ is such that:

$$t = (q, g, a, q') \in \mathcal{T} \Leftrightarrow (q, g \wedge \mathcal{C}(a, i_u)(i_c), a, q') \in \mathcal{T}^{\mathcal{C}}$$

and no other transition is allowed.

For invariance, we define: $S' = make_invariant(S, E)$ with $E \subseteq \mathcal{Q}$, the function that synthesizes and returns a controllable system S' such that the controllable transitions leading to states $q_{i+1} \notin E$ are inhibited, *as well as* those leading to states from where a sequence of uncontrollable transitions can lead to such states $q_{i+k} \notin E$.

There can be several controllers satisfying the control objectives; actually, sometimes forbidding any move is a

control which avoids the states not satisfying the property, but this is less than satisfactory w.r.t. the activity of the control system. The notion of maximally permissive controller captures that we have the controller which insures the properties satisfaction while keeping the greatest subset of behaviors of the original, uncontrolled, system.

2.3 Optimal controller synthesis

One-step optimal control. It is possible to consider *weights* assigned to the states and/or transitions of the system, and to specify that some upper or lower bound must never be reached. *Optimal controller synthesis* [Marchand and Samaan (2000)] can then be used to control transitions so as to minimize/maximize, in one step, some function w.r.t. these weights; i.e., *go only to next states with optimal weight* or *trigger only transitions with optimal weights.* There can be several equally weighted solutions, so optimization does not necessarily lead to determinism. It can be noted that this gives us only a *one step* choice i.e., a local optimal, not a global optimal on all the behaviors (we shall come back to this point further in the paper). With respect to our problem, such weights can model the worst-case execution time (WCET) of a given task onto a given processor, its power consumption, the amount of processor load it requires, or the quality of its results when executed on this particular processor.

Optimal control on paths. Costs can also be assigned to execution paths across an LTS. An execution path of length k starting at state q_1 and ending at state q_k is determined by the sequence of $k-1$ transitions between them. The cost function is the sum of costs $C(t_i), i = 1..k-1$ of each transition between q_1 and q_k. Bellman's algorithm for dynamic programming computes an optimal strategy for reaching a *target* state of an LTS in presence of uncontrollable events, belonging to an adversary environment [Bellman (1957); Marchand et al. (1998)]. By driving the uncontrollable inputs adequately, the adversary tries to prevent the LTS from reaching the target. The optimal strategy, if it exists, is the control solution that drives the LTS towards the target state at a best cost despite the worst moves of the adversary.

Algorithm optimal_DCS $(\mathcal{S}, Q_f, C)$*:* It takes as input the sytem $\mathcal{S} = (\mathcal{Q}, q_o, \mathcal{I}_c, \mathcal{I}_u, \mathcal{O}, \mathcal{T})$, the cost function C, the final states $Q_f \in \mathcal{Q}$, and returns the controlled system $\mathcal{S}^C$:

(1) Computation of the best cost to reach a target state in Q_f: it maps each state q, by taking into account the *worst-case* moves of the adversary. If such an execution path does not exist, then the best cost achievable is equal to $+\infty$. Let $W : \mathcal{Q} \rightarrow N$ be the mapping function. W is defined as the greatest fixed point of the following recurrent equations:

$$W^0(q) = \begin{cases} 0 & \text{iff } q \in Q_f \\ +\infty & \text{otherwise} \end{cases}$$

$$W^i(q) = \min \begin{cases} W^{i-1}(q) \\ \forall i > 0, \max\limits_{\mathcal{I}_u} \min\limits_{\mathcal{I}_c} C(t) + W^{i-1}(q') \\ \qquad \text{s.t. } t = (q, i_u, i_c, a, q') \in \mathcal{T} \end{cases}$$

(2) Use of W to generate the best trajectory reaching Q_f. For any state q, compute the best transition relation set $\mathcal{T}^C$ leading to state q':

$$(q, i_u, i_c, a, q') \in \mathcal{T}^C \text{ iff } (q, i_u, i_c, a, q') \in \mathcal{T} \text{ and}$$

$$W(q') = \min\{W(q'') \text{ s.t. } \forall i_u, \exists i_c, \exists a : (q, i_u, i_c, a, q'') \in \mathcal{T}\}$$

These are the only ones that are allowed in the controlled system.

(3) $\mathcal{S}^C = (\mathcal{Q}, q_o, \mathcal{I}_c, \mathcal{I}_u, \mathcal{O}, \mathcal{T}^C)$.

It has been proved that, for finite systems with positive costs, this algorithm is guaranteed terminate with a complexity polynomial w.r.t the size of the system [Maler (2004); Seidl (1996)]. The obtained greatest fixed point is the optimal solution of the synthesis problem. Its implementation is presented in [Marchand et al. (1998)]. There are previous works on the integration of DCS in a synchronous programming environment. SIGALI [Marchand et al. (2000)] is a tool that manipulates Symbolic LTSs to provide functionalities for DCS (e.g., invariance, reachability, and optimal control on paths). A methodology for property-enforcing layers is proposed in [Altisen et al. (2003)]. A deeper integration is proposed in a language called BZR [Delaval et al. (2010)], with DCS encapsulated in the compilation process.

3. MODEL OF THE MULTI-TASK SYSTEM

This section outlines the different parts of a model, detailed in [Dumitrescu et al. (2007)], with the addition of multi-task aspects in Sections 3.2 and 3.3. Figure 3 summarizes the global model for an example, as compositon of LTSs for (clockwise) environment , tasks (three) and architecture (three). The final reactive model for a complete system can also include a scheduler (an application-specific automaton responsible for emitting sequences of requests r^j), and finally the synthesized controller, obtained by DCS according to the techniques mentioned in Section 2, following the method of Section 4.

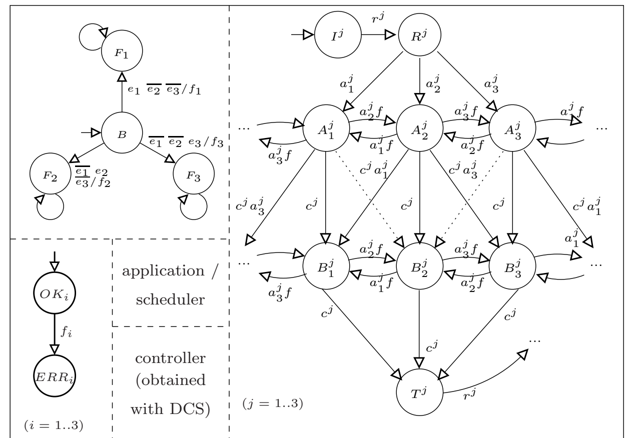


Fig. 3. Global model of multi-tasks system.

3.1 Architecture model

Local processor model. Each is modeled by the lower left LTS of Figure 3: OK_i means that the processor i is running, while ERR_i means that it has crashed. We assume that only the processors can fail, with a fail-silent behavior (which can be achieved at a reasonable cost [Baleani et al. (2003)]). Failures are also permanent, hence a processor cannot go back from the ERR to the OK state. Modelling intermittent failures or degraded modes (e.g. at a slower speed, overloaded) would also be possible [Dumitrescu et al. (2007)]. Processors can be used by tasks in a time-sharing manner, several of them active on the same processor at the same time.

Heterogeneous architecture model. The processors are embedded inside a fully connected network of point-to-point communication links. We note $\mathcal{S}$ the set of all these processors. We assume that the communication links cannot fail and that we have a stable memory. One processor is dedicated to executing the controller, P_0, and only the other processors are available for executing the system's tasks. In our running example, we have three, one for each of the processors P_1, P_2, and P_3, with capacity bounds as in [Dumitrescu et al. (2007)].

Environment or fault models specify what failures can occur in the system, for instance, how many, in what order or known sequences or pattern. In terms of our processor, the question is how can the f_i events occur? It seems natural that all the f_i events be uncontrollable (i.e., $\in \mathcal{I}_u$), since a failure is an event intrinsically uncontrollable. But there would be no constraints whatsoever on them. In particular, *all* events f_i could occur, meaning that all processors could fail at the same time. This would result in a total failure of the system, with no possibility at all to ensure the fault-tolerance of the system. No one expects a system to tolerate a failure of all its processors. Therefore, we need to specify the failure patterns that we consider.

To model this, we choose to have one LTS modeling the environment. Its purpose is to issue the signals f_i from signals e_i from the environment. These signals e_i will be uncontrollable (i.e., $\in \mathcal{I}_u$), reflecting the fact that a failure can occur at any time, while the signals f_i will be local, i.e., neither in $\mathcal{I}_u$ nor in $\mathcal{I}_c$, and will be used only for computing the synchronous product of all the LTSs. The simple environment model of the upper left LTS in Figure 3. It allows only one failure to occur in the system. According to the available knowledge about the system, one can directly specify, as part of the design work, the *failure patterns* by giving directly the LTS producing the local signals f_i from the input signals e_i. For convenience, we introduce a failure event that signals the occurrence of at least one failure: $f \stackrel{def}{=} \bigvee_i f_i$.

3.2 Task model

Basic control structure pattern. Each task t^j is formally modeled by a LTS, which describes how the control of the activity of the task is done in reaction to events. For example, we assume that the task can be executed on three processors, as in the right of Figure 3 (some transitions have been omitted or dotted for readability, and transitions going to "..." are meant to go to the mentioned state at the other side of the Figure). It features an initial *idle* state I^j, a *ready* state R^j after reception of the *request* signal r^j, a *terminal* state T^j, and several *active* states A_i^j, representing task *configurations*, one for each processor in the system. By convention, subscripts/superscripts refer to processors/tasks. In the state A_i^j, task j is executed on processor i, until the occurrence of the progress event c^j. A transition from state A_i^j to state A_k^j represents the *reconfiguration* of the system, by *stopping* task j on processor i and *migrating* and *restarting* it onto processor k. We consider reconfigurations only upon progress (e.g., $c^j a_3^j$) or failure (e.g., $a_2^j f$).

We introduce a notion of phases and checkpoint. Going from one phase A_2^j to the next in sequence B_2^j is acknowledged with an uncontrollable checkpoint event c^j. The last checkpoint is actually the termination. When a task is migrated, it is restarted from the beginning of the *current phase* and not from the very start of the task. In that sense, each phase transition is a control checkpoint, and when a task is migrated it rolls-back to the latest checkpoint.

The signals r^j, c^j, and e_i will be uncontrollable (i.e., $\in \mathcal{I}_u$), while the signals a_i^j will be controllable (i.e., $\in \mathcal{I}_c$).

Local costs. We define a weight function $\mathcal{W}$ characterizing worst case execution time (WCET), for each phase of t^j on each processor P_i, e.g., $\mathcal{W}(A_i^j)$. We consider that the full cost is associated to the transition entering each phase: indeed, the worst case is that the whole WCET is consumed before either quitting upon termination of the phase, or migrating just before terminating. In addition, we represent costs of the checkpoint and of the rollback operations, respectively writing and reading to the stable memory, by $\mathcal{W}_c$ and $\mathcal{W}_r$. Particularly:

- (a) when starting a task from R^j, $\mathcal{W}_r$ corresponds to loading the initial data in the working memory;
- (b) when terminating to T^j, $\mathcal{W}_c$ corresponds to saving the computation result to the stable memory.

On top of this, we represent costs of the checkpointing and rollback, according to the transition entering into a phase ϕ_i^j, as illustrated in Figure 4:

- (c) when proceeding in sequence from same processor: first save current checkpoint, then start new computation: $\mathcal{W}_c + \mathcal{W}(\phi_i^j)$;
- (d) when migrating in same phase: first load last checkpoint on other processor, then start new computation: $\mathcal{W}_r + \mathcal{W}(\phi_i^j)$;
- (e) when doing both: first save current checkpoint, then load last checkpoint on other processor, then start new computation: $\mathcal{W}_c + \mathcal{W}_r + \mathcal{W}(\phi_i^j)$.

3.3 Multi-tasks system

Behavior and local costs. The multi-task system is specified modularly as the synchronous product of LTSs as above. It represents all possible configurations and reconfiguration paths. The local costs presented above correspond to a task running in isolation on a processor. In a multi-tasks system, global transitions are labeled with

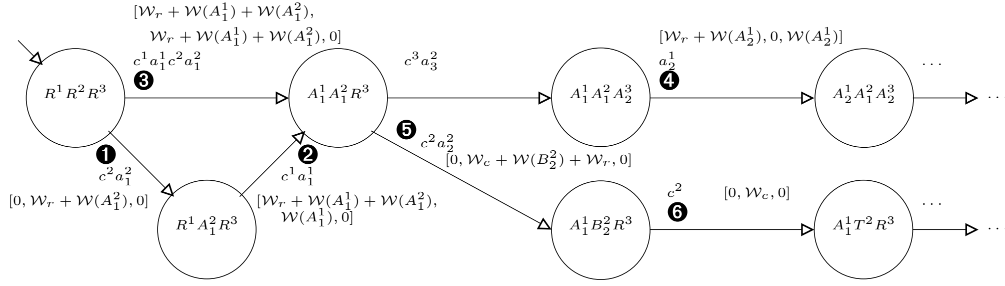


Fig. 5. Extract of the model for three tasks and three processors.

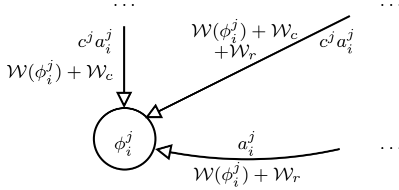


Fig. 4. Costs local to a phase ϕ_i^j of single task t^j.

vectors of costs, whose components c_j are the individual costs of each task t^j e.g., in a system with three tasks each global transition cost is $C = [c_1, c_2, c_3]$. An extract of the model is illustrated in Figure 5 for the case of three tasks, with initial state $R^1R^2R^3$. Costs for checkpointing and rollback are also represented, according to Figure 4. Transition ❶ represents the starting of phase A of task t^2 on processor P_1, upon events c^2 and a_1^2. The cost for that task is that of loading the initial data: $\mathcal{W}_r$ (as in (a) of Section 3.2.1), and computing: $\mathcal{W}(A_1^2)$; the effect on costs for other tasks remains null. Transition ❹ illustrates migration (as in (d)): from a state where the three tasks are active, upon a_2^1 task t^1 is migrated from processor P_1 to processor P_2. The first term of the cost vector shows the cost of data upload, and the WCET of the phase on the new processor: $\mathcal{W}(A_2^1)+\mathcal{W}_r$. Transition ❺ illustrates phase sequence, checkpointing and migration: task t^2 terminates phase A on processor P_1, hence writes the checkpoint with cost $\mathcal{W}_c$, and starts phase B on processor P_2, with cost $\mathcal{W}(B_2^2)$, requiring to upload the results of the previous phase with cost $\mathcal{W}_r$ (as in (c) or (e)). In transition ❻, task t^2 terminates,which has no effect on computation costs in the vector, but costs $\mathcal{W}_c$ (as in (b)). We consider WCET, so actual execution time can be anything lower or equal to it, hence all interleavings of c^j events are in our model.

Contextual costs. When several tasks share a processor, the WCET of each task depends upon the computation context, i.e., the set of tasks active on that processor. Each task has to share processor time with the others, and this is handled by the operating system, which is not represented here. The global computation to be achieved costs the sum of local costs. Therefore in the worst case, each task can be the last one to perform its termination; hence *each* of the tasks has a WCET, in this particular context, of: $\sum_{\{j|t^j \text{ runs phase } \phi \text{ on } P_i\}} \mathcal{W}(\phi_i^j)$. For example, in Figure 5, transition ❸ shows simultaneous starting of t^1 and t^2 on P_1: in the vector both costs take sharing into account: both have the same computing cost $\mathcal{W}(A_1^1)+\mathcal{W}(A_1^2)$.

Cost functions are cumulated along the paths, representing the vector, for all tasks, of WCETs following this sequence of activations, migrations, and changes of context. When a new task is activated on a processor, where other tasks were already running, then its cost is the sum of all costs, and all other tasks have their cost added with that of the new task. For example, in Figure 5, in transition ❷, task t^1 is activated on processor P_1 where t^2 is already active: hence its worst case has to take into account that of the new incoming task, hence in the vector we have $\mathcal{W}(A_1^2)$ which will add up on the previous costs. For the newly started task t^1, the cost also has to take into account sharing with t^2, hence we have $\mathcal{W}_r + \mathcal{W}(A_1^1) + \mathcal{W}(A_1^2)$.

When a task migrates or terminates, in the worst case it does so at its full cost, and it can have consumed the resource while the others waited, hence the WCETs of other tasks are not diminished. Accordingly we have:

(1) When task t^j starts phase ϕ on processor P_i: The global transition is $X^jX \rightarrow \phi_i^jX'$, and in the jth component $C[j]$ of the global cost C, we have the computing cost of activating t^j plus the costs of all the other tasks also running on the same processor P_i: $\mathcal{W}(\phi_i^j) + \sum_{\{\phi_i^k \in X'\}} \mathcal{W}(\phi_i^k)$. In addition: when $X^j = R^j$ (i.e., (a)) we also have $\mathcal{W}_r$; this is illustrated by transitions ❶ for t^2, ❷ for t^1 and ❸ for both; and when going to a next phase in sequence $X^j = \phi_i'^j \neq \phi_i^j$ (i.e., (c) or (e)) we also have $\mathcal{W}_c$, as illustrated by ❺ for t^2.

(2) When the task t^j migrates, in phase ϕ, from processor $P_{i'}$ to P_i: $\phi_{i'}^jX \rightarrow \phi_i^jX'$, the computation cost is as previously: $\mathcal{W}(\phi_i^j) + \sum_{\{\phi_i^k \in X'\}} \mathcal{W}(\phi_i^k)$. As in (d) above, we also have $\mathcal{W}_r$, which is illustrated by ❹ for t^1. When also proceeding in next phase (i.e., (e)), we also have $\mathcal{W}_c + \mathcal{W}_r$, as illustrated by ❺ for t^2.

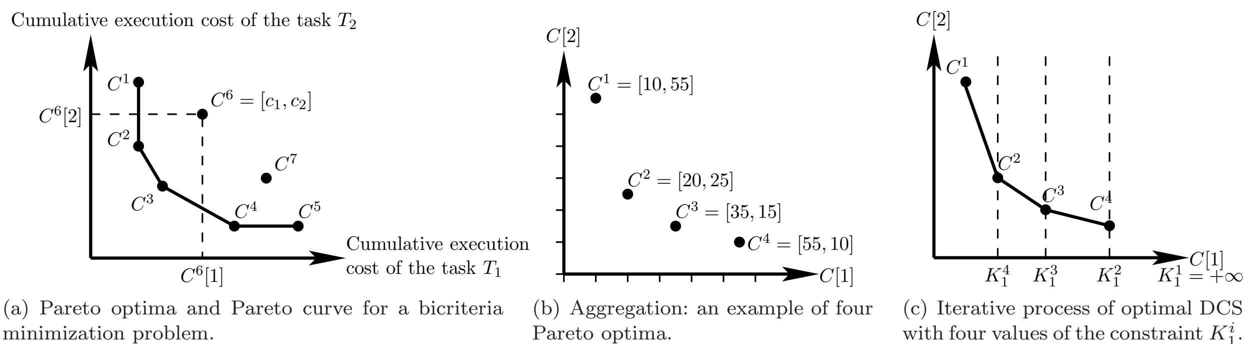
(3) When the task t^j continues its execution on the same processor P_i: $\phi_i^jX \rightarrow \phi_i^jX'$, and the global cost for t^j is the costs of all the other tasks activated on or migrated to the same processor P_i: $C[j] = \sum_{\{\phi_i^k \in X' \wedge \phi_i^k \notin X\}} \mathcal{W}(\phi_i^k)$, as in ❷ and ❹ for t^2.

(4) When the task t^j terminates its execution on processor P_i: $\phi_i^jX \rightarrow T^jX'$ (i.e., (b)) , $C[j] = \mathcal{W}_c$.

4. MULTICRITERIA OPTIMAL DCS

4.1 Multicriteria optimizations

We want to consider transitions labelled with vectors of costs instead of scalar costs, because the systems we model



(a) Pareto optima and Pareto curve for a bicriteria minimization problem.

(b) Aggregation: an example of four Pareto optima.

(c) Iterative process of optimal DCS with four values of the constraint K_1^i.

Fig. 6. Multicriteria optimizations

have different types of weights, and several tasks. Hence we are faced with a *multicriteria optimization problem*, where the ith criterion is the cumulative execution cost of the task t^i. We therefore propose a variant of the classical optimal DCS algorithm of Bellman [Bellman (1957)]. We must address two issues: Bellman's algorithm does not deal with path having infinite loops, as it occurs with reactive systems. We have already addressed this issue in [Dumitrescu et al. (2007)]. Also, it can only minimize a single criterion and not a vector of criteria. This issue is related to the notion of Pareto optima, presented below.

Let us consider the particular case of two tasks t^1 and t^2, with execution costs c_1 and c_2; the costs on the transitions of the global LTS are therefore vectors $C = [c_1, c_2]$. In Figure 6(a), each point C^1 to C^7 represents a solution of our bicriteria minimization problem, that is, a vector of two cumulative execution costs, one for each task t^1 and t^2. The points C^1, C^2, C^3, C^4, and C^5 are *Pareto optima* [T'kindt and Billaut (2006)]; the points C^1 and C^5 are *weak optima* while the points C^2, C^3, and C^4 are *strong optima*. The set of all Pareto optima is called the *Pareto curve*. Then, several approaches exist to tackle bicriteria (or multicriteria) minimization problems.

Aggregation of the two criteria into a single one transforms the problem into a classical single criterion minimization one. This can be obtained by a linear combination function ϕ of the costs, e.g., $\phi = \gamma_1 C[1] + \gamma_2 C[2]$, which is then minimized as in [Dumitrescu et al. (2007)]. The coefficients γ_1 and γ_2 can reflect the fact that some tasks are more urgent than others: the higher the coefficient, the more urgent the task. E.g., in Figure 6(b), with $\phi = 1 \cdot C[1] + 1 \cdot C[2]$, the Pareto optimum C^2 is selected (since $\phi(C^2) = 45$), while with $\phi' = 5 \cdot C[1] + 1 \cdot C[2]$, the Pareto optimum C^1 is selected (since $\phi'(C^1) = 105$). Other functions $\phi(C[1], C[2])$ than linear combinations can be used, provided that ϕ is convex; indeed, this condition guarantees that the solution that minimizes ϕ is a Pareto optimum.

Hierarchization of the criteria allows their total ordering, and then the solving by minimizing one criterion at a time, e.g., $C[1]$ and then $C[2]$. In our DCS context, this amounts to choosing, as our first synthesis objective, the minimization of the cumulative execution cost $C[1]$, and then, as our second synthesis objective, the minimization of the cumulative execution cost $C[2]$. Again, the order of the criteria can reflect the fact that some tasks are more urgent than others. E.g., in Figure 6(a), mimizing w.r.t. $C[1]$ and then with $C[2]$ gives the Pareto optimum C^2 (since only C^1 and C^2 are optima for $C[1]$), while mimizing w.r.t. $C[2]$ and then with $C[1]$ gives the Pareto optimum C^4 (since only C^4 and C^5 are optima for $C[2]$).

Transformation of one criterion into a constraint, allows the solving of the problem by minimizing the other criterion under the constraint of the first one. E.g., if $C[1]$ is taken as a constraint, this means that we perform the optimal DCS to minimize $C[2]$ with the additional DCS objective $C[1] < K_1^i$, where K_1^i is the relative deadline of the task t^1. By iteratively applying this process with decreasing values of K_1^i, starting with $+\infty$, we are able to obtain the Pareto curve of the given instance. Figure 6(c) illustrates this process: by performing an optimal DCS with four successive values of K_1^i, four points of the Pareto curve are obtained (each C_i is obtained with the value K_1^i): The epsilon-constraint method [Haimes et al. (1971)] was designed to obtain, thanks to an optimal single-criterion optimization algorithm, the exact Pareto curve of a bicriteria problem, when the number of Pareto-optimal solutions is finite. This method was further extended to more than two criteria, and an efficient algorithm was proposed in [Laumanns et al. (2006)].

Interaction with the user, in order to guide the search for a Pareto optimum by combining the above-mentioned methods. For instance, with 5 criteria $C[1]$ to $C[5]$, the user might choose to take the constraint K_2 on $C[2]$, then aggregate $C[1]$ and $C[4]$ into $\gamma_1 C[1] + \gamma_4 C[4]$, and hierarchize by optimizing first $C[3]$, then $\gamma_1 C[1] + \gamma_4 C[4]$, and finally $C[5]$, all under the constraint $C[2] < K_2$.

4.2 Basic fault tolerance control objectives

Before handling multi-criteria optimization, we briefly recall the basic fault tolerance control objectives applied on the global model of multi-tasks system [Dumitrescu et al. (2007)]. *Insuring consistent execution* is formulated as the fact that *no task is active on a failed processor*: $\neg \bigvee_j \bigvee_i (A_i^j \wedge Err_i)$. Also, it is required that *tasks active are within processor capacity*: $\forall i, C_i \leq b_i$, according to the simpler cost function of [Dumitrescu et al. (2007)]. The synthesis objective for the conjunction of these two properties is to make it *invariantly true*. *Insuring functionality* is not just a state property: it involves paths. We want to avoid that DCS inhibits indefinitely the start of a task: tasks are activated only when *"the path is clear and wide*

enough all the way down" to termination, even in case of failures. *The functionality is fulfilled* iff from all reachable states, the terminal state T of the program is *reachable*. In this paper, the improvement is multicriteria optimal DCS.

We are re-using the optimal DCS algorithm of Section 2.3 in order to handle multiple optimization criteria. We take as input a set of n tasks $\{t^i\}_{i=1..n}$, each being specified by the task LTS of Figure 3. Our multicriteria optimization problem is to minimize the cumulated WCET $C[i]$ of each task t^i. According to Section 4.1, we shall consider three different multicriteria optimization methods: *aggregation*, *hierarchization* and *transformation*. The basic mono-criterion optimization function using DCS is used for constructing optimal solutions according to the three mentioned methods. This algorithm has been introduced in section 2.3: *optimal_DCS*$(\mathcal{S}, Q_f, C)$, where S denotes the global system computed from the synchronous product of a set of LTSs, one for each task, processor and failure model. The set Q_f of target states to be reached following an optimized path is also required, together with the *environment* cost function C.

4.3 Aggregation for multicriteria DCS algorithm

Given a system $\mathcal{S}$ modeling a set of n tasks $\{t^i\}_{i=1\cdots n}$ and n optimization criteria $(C[i])_{i=1..n}$, the aggregation method involves performing a single optimal DCS with the cost function $\sum_{i=1}^{n} \gamma_i C[i]$, where the n coefficients $\{\gamma_i\}_{i=1..n}$ are given by the user. The algorithm computes $\mathcal{S}^C$, the optimal controlled system with respect to the aggregation of the n criteria.

Algorithm aggreg_ODCS$(\mathcal{S},\ (C_{[i]})_{i=1\cdots n},\ (\gamma_i)_{i=1\cdots n})$:

1. $C_{aggreg} \leftarrow \sum_{i=1}^{n} \gamma_i C_{[i]}$
2. $\mathcal{S}^C \leftarrow$ optimal_DCS$(\mathcal{S}, Q_f, C_{aggreg})$

The mono-criterion optimization is performed with respect to the aggregated cost function C_{aggreg} using *optimal_DCS*.

Example Figure 7 illustrates this algorithm on two tasks running on three processors under the optimal constraint computed above. The fault model used is the one presented in Figure 3. Static costs are represented as integer numbers next to their corresponding transitions. The local costs assigned to the transitions of each task only depend on the destination state. For instance, all transitions entering state B_1^1 have their local cost equal to 2. For readability reasons we have not displayed graphically the whole transition set of a task. The simulation scenario starts both tasks at the same moment: events r_1 and r_2 are received simultaneously. Consider the global state $A_3^1 A_2^2$. Possible outgoing transitions are $A_3^1 A_2^2 \rightarrow B_1^1 B_3^2$ or $A_3^1 A_2^2 \rightarrow B_3^1 B_3^2$. The other combinations starting at $A_3^1 A_2^2$ are too expensive. However, transition $A_3^1 A_2^2 \rightarrow B_3^1 B_3^2$ costs $10 \times 3 + 1 \times 3 = 33$ as the contextual costs of t^1 and t^2 due to their activation on processor P_3 is the sum $2+1=3$ of their local activation costs on P_3, as stated in rule (1). On the other hand, transition $A_3^1 A_2^2 \rightarrow B_1^1 B_3^2$ represents an independent activation on processors P_1 and P_3. Its aggregated cost equals $10 \times 2 + 1 = 21$. This transition has a better cost (21) than the former (33). At this point, $e3$ is received, meaning that processor P_3 has failed. The

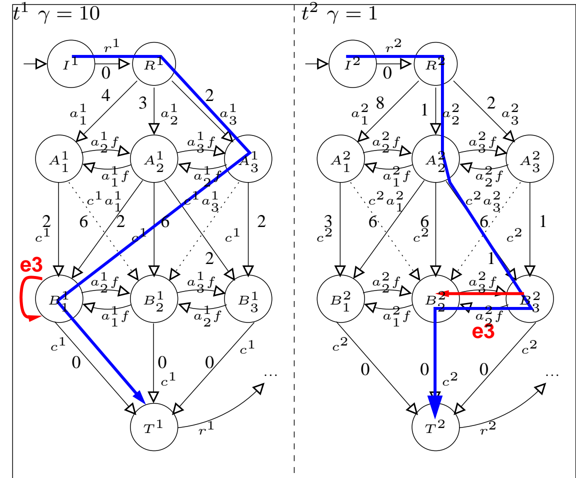


Fig. 7. Cost aggregation method : Simulation of the optimized run of two tasks on three processors.

supervisor migrates task t^2 to state B_2^2, as the migration cost to B_1^2 would be too expensive, due to the presence of t^1 on the same processor.

4.4 Hierarchization for multicriteria DCS algorithm

Here, we apply our optimal DCS algorithm in sequence to each task $i = 1..n$, according to the pre-established ordering, and by taking into account at each step i the optimal solution found at step $i-1$. Let $\tau : \{1..n\} \rightarrow \{1..n\}$ be a permutation which specifies the total ordering of the set of tasks to be scheduled. The hierarchization is achieved according to the order given by τ. The algorithm executes n iterations, one for each task to be scheduled.

Algorithm hier_ODCS$(\mathcal{S},\ (C_{[i]})_{i=1\cdot n},\ \tau)$:

for i = 1..n do $\mathcal{S}^c_{[i]} \leftarrow$ optimal_DCS$(\mathcal{S}_{[i-1]}, Q_f, C_{[\tau(i)]})$
end for

The algorithm starts with the global LTS $\mathcal{S}_0 = \mathcal{S}$, as well as the vector of $(C_{[i]})_{i=1..n}$ optimization criteria and the user-defined permutation τ. $\mathcal{S}_{[i]}$ is the controlled system achieving optimal scheduling for task $t^{\tau(i)}$. It takes into account the optimal solutions computed for tasks $\tau(1)\ldots\tau(i)$: it contains all the transitions accepted by the system $\mathcal{S}_{[i-1]}$ and leading to successor states having minimal $W_{[i]}$. The global optimal scheduler produced by the above algorithm is $\mathcal{S}_{[n]}$. Due to lack of space, we can not illustrate the *hier_ODCS* nor the *transform_ODCS* algorithms.

4.5 Transformation for multicriteria DCS algorithm

For the sake of simplicity, we shall present the transformation algorithm in the case of only two criteria (i.e., only two tasks t^1 and t^2). Without loss of generality, let us minimize the cost of t^1 under the constraint that the cost of t^2 be less than K_2. The algorithm proceeds as follows, where the superscript (i) indicates the i-th iteration:

Algorithm transform_ODCS($\mathcal{S}$, $C_{[1]}, C_{[2]}$)

1. $K_2^{(0)} \leftarrow +\infty$, $\mathcal{S}_c^{(0)} \leftarrow \mathcal{S}$ and $i \leftarrow 1$
2. $\mathcal{S}_t^{(0)} \leftarrow$ optimal_DCS$(\mathcal{S}_c^{(0)}, Q_f, C_{[1]})$
3. $\mathcal{S}_o^{(0)} \leftarrow$ optimal_DCS$(\mathcal{S}_t^{(0)}, Q_f, C_{[2]})$
4. repeat
5. $\quad K_2^{(i)} \leftarrow W_{[2]}(q_f)$ for any $q_f \in Q_f$
6. $\quad \mathcal{S}_c^{(i)} \leftarrow$ make_invariant$(\mathcal{S}_o^{(i-1)}, K_2^{(i)})$
7. $\quad \mathcal{S}_t^{(i)} \leftarrow$ optimal_DCS$(\mathcal{S}_c^{(i)}, Q_f, C_{[1]})$
8. $\quad \mathcal{S}_o^{(i)} \leftarrow$ optimal_DCS$(\mathcal{S}_t^{(i)}, Q_f, C_{[2]})$
9. $\quad i \leftarrow i + 1$
10. until $\mathcal{S}_o^{(i)} \neq \mathcal{S}_o^{(i-1)}$
11. return $\mathcal{S}_o^{(i)}$

In accordance to the epsilon-constraint method, we initially set the constraint K_2 to $+\infty$. We then use the *make_invariant* procedure on $\mathcal{S}$ to select the paths that satisfy this constraint (except during the first step where we just take $\mathcal{S}$, as all paths of $\mathcal{S}$ satisfy trivially the initial constraint $+\infty$). We then apply *optimal_DCS* on the obtained result to select the paths that lead to the optimal cumulated cost for t^1. Among those paths, we further select those who are also optimal for t^2, by applying a second time the *optimal_DCS* procedure. We then set the constraint K_2 for the next iteration to the cumulated cost for t^2 on any path ending in a terminal state. The process is repeated until no more improvement is obtained.

Algorithm *transform_ODCS* terminates because the first value computed for K_2 in line 7 is finite (since the *optimal_DCS* procedures operates on finite paths only), and because each iteration of the *repeat until* loop strictly decreases this value of K_2 (since all the costs are integers and the constraint on K_2 is strict).

4.6 Complexity issues

The *aggreg_ODCS* and *hier_OCDS* algorithms have the same complexity as ODCS, i.e., polynomial in the size of the transition graph representing the global system $\mathcal{S}$. However, the size of the transition graph representation is exponential in the number of system variables used for the modeling.

As for the *transform_ODCS* algorithm, the complexity depends on the number of iterations required to reach the final solution $\mathcal{S}_o^{(i)}$. This number is bound by the cost $W_{[2]}(q_f)$, because K_2 starts with this value, and then decreases by arbitrary values, and at worse by steps of 1 while remaining positive.

The multicriteria algorithms achieving aggregation, hierarchization and transformation have been implemented inside the Sigali [Marchand et al. (2000)] DCS tool.

5. CONCLUSION

This paper extends previous work [Dumitrescu et al. (2007)] with multi-criteria optimization. We propose a refined model of the multi-task model, taking into account processor time-sharing, and formulating fault-tolerance guarantee and WCET optimization as a multi–criteria problem (one for each task), and the optimal DCS algorithms exploring different ways of combining the criteria (aggregation, hierarchization, and transformation). Perspectives are in the integration of the synthesis operation in a design tool, application to real-size case-studies.

REFERENCES

Altisen, K., Clodic, A., Maraninchi, F., and Rutten, E. (2003). Using controller-synthesis techniques to build property-enforcing layers. In *Proc. of the European Symp. on Programming, ESOP'03.*

Baleani, M., Ferrari, A., Mangeruca, L., Peri, M., Pezzini, S., and Sangiovanni-Vincentelli, A. (2003). Fault-tolerant platforms for automotive safety-critical applications. In *Conf. on Compilers, Architectures and Synthesis for Embedded Systems, CASES'03.*

Bellman, R. (1957). *Dynamic Programming.* Princeton University Press.

Benveniste, A., Caspi, P., Edwards, S., Halbwachs, N., Le Guernic, P., and de Simone, R. (2003). The synchronous languages twelve years later. *Proc. IEEE*, 91(1).

Delaval, G., Marchand, H., and Rutten, E. (2010). Contracts for modular discrete controller synthesis. In *ACM Conf. on Languages, Compilers and Tools for Embedded Systems, LCTES 2010.*

Dumitrescu, E., Girault, A., Marchand, H., and Rutten, E. (2007). Optimal discrete controller synthesis for modeling fault-tolerant distributed systems. In *Workshop on Dependable Control of Discrete Systems, DCDS'07.* See also: `http://hal.inria.fr/inria-00134550`.

Dumitrescu, E., Girault, A., and Rutten, E. (2004). Validating fault-tolerant behaviors of synchronous system specifications by discrete controller synthesis. In *Workshop on Discrete Event Systems, WODES'04.*

Haimes, Y., Lasdon, L., and Wismer, D. (1971). On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE Trans. Systems, Man, and Cybernetics*, 1, 296–297.

Laumanns, M., Thiele, L., and Zitzler, E. (2006). An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method. *European Journal of Operational Research*, 169(3), 932–942.

Maler, O. (2004). On optimal and sub-optimal control in the presence of adversaries. In *Workshop on Discrete Event Systems, WODES'04.*

Marchand, H., Bournai, P., Le Borgne, M., and Le Guernic, P. (2000). Synthesis of discrete-event controllers based on the Signal environment. *Discrete Event Dynamic System: Theory and Applications*, 10(4), 325–346.

Marchand, H. and Le Borgne, M. (1998). On the optimal control of polynomial dynamical systems over Z/pZ. In *Workshop on Discrete Event Systems, WODES'98.*

Marchand, H. and Samaan, M. (2000). Incremental design of a power transformer station controller using a controller synthesis methodology. *IEEE Trans. Software Engin.*, 26(8), 729–741.

Milner, R. (1989). *Communication and Concurrency.* International Series in Computer Science. Prentice-Hall.

Ramadge, P. and Wonham, W. (1989). On the supervisory control of discrete event systems. *Proc. IEEE*, 77(1).

Seidl, H. (1996). Least and greatest solutions of equations over N. *Nordic Journal of Computing*, 3, 41–62.

T'kindt, V. and Billaut, J.C. (2006). *Multicriteria Scheduling: Theory, Models and Algorithms.* Springer-Verlag.