Managing multi-mode tasks with time cost and quality levels using optimal discrete control synthesis

Hervé Marchand
VerTeCs, IRISA / INRIA
35042 RENNES, France
Herve.Marchand@irisa.fr, www.irisa.fr/vertecs

Éric Rutten
BIP, INRIA Rhône-Alpes
38334 SAINT ISMIER, France
Eric.Rutten@inrialpes.fr, www.inrialpes.fr/bip

Abstract

Real-time control systems are complex to design, and automation support is important. We are interested in systems with multiple tasks, each with multiple modes, implementing a functionality with different levels of quality (e.g., computation approximation), and cost (e.g., computation time, energy). It is complex to control the switching of modes in order to insure properties like bounding cost while maximizing quality. We outline a technique for the automatic generation of such controllers involving an automaton-based formal model, and using optimal discrete control synthesis.

1. Motivation

Occurrences in application domains. We are interested in real-time control systems with multiple tasks, each with multiple modes, which can be versions implementing the same functionality with different levels of quality and cost. It is complex to control the switching of modes in order to insure properties like bounding cost while maximizing quality (i.e., limiting degradation). Application domains are:

- control systems (e.g. robotics) where tasks implement the computation of a control law or function, in a cyclic way, reading sensor input, and producing actuator command output. A notion of quality is that the numerical computations involved, e.g., matrix computation for kinematic model update, can be performed with different levels of accuracy, by making approximations e.g., by limiting the development of series, or avoiding the computation of terms that have a negligible value under some conditions [6]. The control laws can also differ by the way energy is consumed by actuators or side-effects on the environment.
- telecommunication systems and portable devices like cellular telephones present many functionalities; e.g., signal processing as in voice recognition or encodings for transmission, can have different implementations according to the way energy is spent.
- automobile embedded equipments, in every of their activation configurations, have a given energy consumption: this latter has to be bounded, because of the battery, so that the autonomy of the car is not jeopardized. At the same time, priority tasks (e.g., related to safety) have to be respected.

All this suggests that in a multi-task system, the control of activations and of mode switching must restrict the system within allowed combinations, with respect to statical or dynamical conditions.

Multi-mode tasks and discrete control synthesis. The notion of modes and their switching is approached in various ways, some related to operating system mechanisms, or language constructs [10]. Some works address the schedulability analysis of the combination of the task schedulings for each mode and for the transition itself [2, 12]. We focus on the case of cyclic real-time systems in the synchronous approach to reactive systems [3]. It offers an automated tool support for controller generation, for which the basis is a formal model upon which algorithms can be defined. Language support is given under different forms, imperative or data-flow, with mixed approaches like Mode Automata [5]. The formal model underlying the approach is labeled transition systems (finite state machines). It is used as a tool for compilation, analysis and verification. In relation with our topics of interest, it can be used for the encoding of the activity state of each task, and the possible switches.

Discrete control synthesis is one of the formal operations applicable on such models. It works on a transition system obtained from the specification of a system, describing its possible behaviors. Some of the events labelling the transitions are declared to be controllable. Synthesis consists of the automated computation of constraints on *controllable events*, such that the transition system is limited to behaviors which are correct with respect to some conditions (properties on the dynamical behaviors, objectives) [9, 7].

Proceedings of the 14 th Euromicro Conference on Real-Time Systems (ECRTS'02)
1068-3070/02 $17.00 © 2002 **IEEE**

Optimal discrete control synthesis [8] allows for the taking into account of weights associated with states or events, and involving maximizing or minimizing functions over them when determining control constraints. Tool support is available in the SIGNAL/SIGALI environment [7], providing for specification, formal computation (synthesis), and execution/simulation of the controlled system.

Our approach. In this context, we outline a technique using optimal discrete control synthesis, based on a model of multi-mode tasks, for the automated obtention of such controllers; SIGNAL/SIGALI is used to implement and validate the approach. The intention is to go towards a design environment where the user should be a driver rather than a mechanic of the formal tools, and an automated, "push-button" use of discrete control synthesis. This paper shows the feasability of this, by describing a particular task pattern and an experiment. The real-time systems considered here are simple (cyclic, sequential implementation of synchronous reactive languages), and the discrete control synthesis technique in itself is not new, but the combination of both in the proposed model is. It can be related to recent work using timed automata as a model, and concerning the automated correct scheduling of real-time tasks [1]. Perspectives of generalization are given along with these first results. Up to now, only feasability experiments have been carried; validation of this concept in terms of evaluation and scalability is in progress; performances of model checking technology make us expect to handle large systems involving millions of states (as we know algorithms are similar).

2. A task pattern with multiple modes

In this section we propose a generic task pattern, representing explicitly the existence of multiple modes, and the possibility to switch between them. This pattern can be instantiated, with parameters describing the set of modes, the switching transitions and the values of weights associated with configurations. This task pattern elaborates on previous work [11], where only single-mode tasks were considered, with properties of exclusivity and of required or forbidden sequences. Here, we exploit the extension of the model to consider properties of bounding and optimization.

2.1. Informal presentation

Cyclic, multi-mode tasks. A reactive system can be constructed in terms of tasks, each with an idle state, from which they can be started, going to an active state. The computation associated with the task is not performed when in the idle state. When having a set of such tasks in parallel, each reaction or cycle will see the performance of the computations associated with each of the active tasks. Hence, the duration of a reaction is the sum of individual durations for active tasks. Indeed, the so-called "zero-time" hypothesis of synchronous languages is a metaphor of the fact that interactions inside a cycle can be compiled away, and hence costless; actual implementation has a worst-case execution time (WCET) to be measured w.r.t. the environment dynamics.

Modes and their characteristics. If one considers tasks with a variety of possible active modes, such as proposed in the Mode Automata [5, 6], then one can consider that these modes are differentiated by some characteristics, such as e.g. time cost, i.e. the duration taken by one cycle of computation (e.g., each mode has a WCET in the reaction) and quality (e.g., precision of numerical computation).

In the present proposal cost and quality are closely related in the sense that the higher the quality delivered by the mode implementing the functionality, the higher the corresponding cost. When considering multiple tasks running in parallel (meaning: sharing processor time within each cycle) time costs naturally combine additively; for quality, we will also consider additive combination in this paper: although it is a bit simplistic, it facilitates explanation of the approach. However this does not in principle exclude other interpretations to be explored in the future (see Section 2.5).

Switching between modes. Once a set of tasks with modes is defined, one has to consider the switching between them. A task is initially idle, and a request is awaited for; it can come from an application written on top of the task set, from a system end-user command, or from other tasks or sensors. Within a multi-task environment, starting a new task can occur in several ways: when enough computing resource is available, it can be started right away; when some computing resource has to be made available, it can be achieved by lowering quality (hence time cost) of other already active tasks, through mode switching, while keeping global quality maximal; when no sufficient resource can be released, then we have to go to a waiting state.

This kind of control, managing mode switches according to criteria of time cost and quality level, is what we want to obtain automatically, through discrete control synthesis.

2.2. The task pattern

A task pattern. Figure 1 gives the automaton for a task, say T_i, for the case with three modes. The pattern formalizes the aspects described above. We have a first level with states: $\texttt{Idle}_i$ (or $\texttt{I}_i$), where the task is deactivated; $\texttt{Wait}_i$ (or $\texttt{W}_i$), where it has been requested, by an event $\texttt{Req}_i$, but is not launched yet, because of lack of authorization $\texttt{Go}_i$ by the controller (due to constraints with the environment or other tasks); $\texttt{Act}_i$ (or $\texttt{A}_i$), where it has been requested, and authorized through $\texttt{Go}_i$, and hence is active.

Within the active state $\texttt{Act}_i$, several modes can be defined. Each task T_i has m_i modes $M_{ij}, 1 \leq j \leq m_i$.

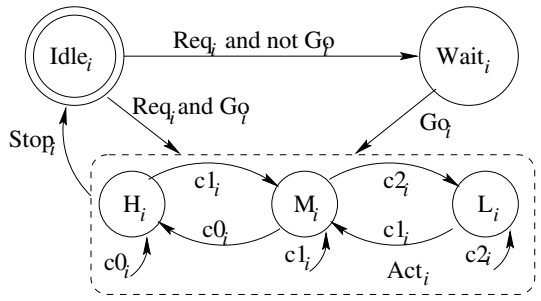

Figure 1. Task pattern with multiple modes.

In order to distinguish the currently active mode we define $\delta_{ij} = 1$ when the mode M_{ij} is active, 0 otherwise, at each instant (i.e., in each system configuration). In this paper, we consider that there is at most one active mode per task, at a given time. This means that, for a tasks T_i:

$$a_i = \sum_{1 \leq j \leq m_i} \delta_{ij} = \begin{cases} 0 & \text{when in state } \texttt{Idle}_i \text{ or } \texttt{Wait}_i \\ 1 & \text{when some mode in } \texttt{Act}_i \text{ is active} \end{cases}$$

For each task T_i, transitions between modes are labelled by conditions c_{k_i}, managed by the controller in order to authorize the switch or not. The mode which becomes active when entering $\texttt{Act}_i$ is chosen according to the one of these conditions which is true. The choice of the mode when starting is made deterministic by the control values, according to the optimal control: it starts in the best possible mode.

In the case of three modes, illustrated in Figure 1, we can think of applications such that we have: $\texttt{H}_i$ (highest-quality and time); $\texttt{M}_i$ (medium); $\texttt{L}_i$ (low). These three modes can be switched between according to the transitions and their conditions c_{k_i}. In particular, you have to go through the $\texttt{M}_i$ mode between $\texttt{H}_i$ and $\texttt{L}_i$. An example is [6] a task computing at each cycle an expression summing three terms: $E = E_1 + E_2 + E_3$, where E_3 and E_2 can be approximated by 0. Each of the modes corresponds to: $\texttt{H}_i$: the full sum, $\texttt{M}_i$: an approximation $E_1 + E_2$, $\texttt{L}_i$: a degraded version E_1. In the following, we will see how valued characteristics can be associated to modes.

Applications. Complete systems involving multiple tasks can be constructed by adding an application program. In this paper, we consider just the parallel composition of the n tasks, as in synchronous languages [3]. Hence, a global state or configuration S is described by a vector of state values, one for each task, giving the current active mode. For an example with two instances T_1 and T_2 of the three-mode pattern, we can be in a mode where T_1 is in state $\texttt{Idle}_1$ while T_2 is in state $\texttt{Act}_2$, in mode $\texttt{H}_2$: the configuration can be noted in short S_1 = ($\texttt{I}_1$, $\texttt{H}_2$). It will be useful in the following to define the number m of active modes in a configuration S. One can observe that $m \leq n$, as at most one mode is active per task. Hence we have, for each configuration: $m = \sum_{1 \leq i \leq n} a_i$. A global step is taken when one or several mode changes or a task start and/or stop event occurs, going from one global configuration to the next. Such a transition is labeled with a vector of conditions/events of the local transitions, or ϵ on the side where no transition is taken. For example, going from S_1 to S_2 = ($\texttt{H}_1$, $\texttt{M}_2$) would be through a transition labeled with a combination of the two involved local transitions: ($\texttt{Req}_1$ and $\texttt{Go}_1$ and $\texttt{c0}_1$), $\texttt{c1}_2$).

Assume that initially, each task is in the $\texttt{Idle}_i$ state. Hence the initial configuration in the example is ($\texttt{I}_1$, $\texttt{I}_2$). The set of the configurations reachable from the initial one by applying global steps describes the state-space defined by the *a priori*, uncontrolled, behaviors of the system. We can define, for a configuration S, the set of its successor configurations, reachable in one step following one transition: $Succ(S)$. Our goal in this paper is to obtain controllers of the tasks that manage mode switching by *restricting it* so that cost and quality objectives are satisfied over the resulting reachable state-space: the corresponding properties are defined in the next section. In the example, when in the configuration S_1, upon the occurrence of ($\texttt{Req}_1$ and $\texttt{Go}_1$) and in the absence of $\texttt{Stop}_2$, the following configurations can be reached in one step. They are reached through transitions which are labeled by, for T_1: either one of $\texttt{c0}_1$, $\texttt{c1}_1$, $\texttt{c2}_1$, and for T_2: either $\texttt{c1}_2$ or ϵ (empty label, no movement of T_2):

label	new configuration
($\texttt{c0}_1$, $\texttt{c1}_2$)	S_2 = ($\texttt{H}_1$, $\texttt{M}_2$)
($\texttt{c1}_1$, $\texttt{c1}_2$)	S_3 = ($\texttt{M}_1$, $\texttt{M}_2$)
($\texttt{c2}_1$, $\texttt{c1}_2$)	S_4 = ($\texttt{L}_1$, $\texttt{M}_2$)
($\texttt{c0}_1$, ϵ)	S_5 = ($\texttt{H}_1$, $\texttt{H}_2$)
($\texttt{c1}_1$, ϵ)	S_6 = ($\texttt{M}_1$, $\texttt{H}_2$)
($\texttt{c2}_1$, ϵ)	S_7 = ($\texttt{L}_1$, $\texttt{H}_2$)

2.3. Modes, tasks and applications characteristics

We will here assign quantitative characteristics to modes, and define how to deduce them for tasks and applications. We define a cost function representing quality C_q, and another one representing computing time C_t (i.e., the time to perform computation within one step of the reactive system). Such cost function must be manually defined, on the basis of e.g., WCET analysis. We propose simple and basic characteristics; they could be made more elaborate, and more realistic in terms of real complex systems: the point here is rather to show how this can be used by optimal discrete control synthesis. In particular, other, logical, properties can be meaningful for such systems, like mutual exclusion of given modes of different tasks, and required or forbidden sequences [11], but in this paper we focus on optimization aspects.

Modes. The functions are defined for each mode of each task: $q_{ij} = C_q(M_{ij})$ and $c_{ij} = C_t(M_{ij})$.One can also think

to associate costs to the events in order to take into account the time necessary for mode changes. In this paper, we consider *a priori* that a task in `Idle` or `Wait` state has a null cost and quality. A non-null, δ cost of a waiting task could be used to account for the possible operating system overhead, or an initialization computation performed (in this latter case, that cost might very well be non-δ in fact, but then the model must be built differently). We also consider that the cost and quality of a mode are related in such a way that: $\forall j,k : q_{ij} \geq q_{ik} \Rightarrow c_{ij} \geq c_{ik}$. They need not be considered proportional, though (e.g., computation of inertia in movement control involves a big time cost for a small precision gain, when acceleration is small). In the example of the three-mode task pattern, we can define costs as follows (with $\delta = 0$) for two instances T_1 and T_2:

		$\mathtt{I}_i$	$\mathtt{W}_i$	$\mathtt{H}_i$	$\mathtt{M}_i$	$\mathtt{L}_i$
T_1	C_t	0	ε	7	5	2
	C_q	0	0	3	2	1
T_2	C_t	0	ε	6	3	1
	C_q	0	0	6	4	3

Tasks. For a task T_i, we can define its current time cost: $t_i = \sum_j C_t(M_{ij}) * \delta_{ij}$ and, accordingly, its current quality: $q_i = \sum_j C_q(M_{ij}) * \delta_{ij}$. Let us recall that, in our framework, only one mode is active at each time. So the current task cost is the cost of the currently active mode.

Applications. For a composition of the n tasks, the current global time cost (within a cycle) is: $T = \sum_i t_i$. The current global quality of an application is: $Q = \sum_i q_i$. In the example of the two instances of the three-mode task pattern, the configurations are characterized as follows:

	S_1	S_2	S_3	S_4	S_5	S_6	S_7
T	5	10	8	5	13	11	8
Q	6	7	6	5	9	8	7

Alternately, and depending on the notion of quality adopted, we could take it to be the average of local qualities, as in: $Q_{alt} = \left(\frac{\sum_i q_i}{n}\right)$ but it would be quite like the sum if you consider that you want to use it for bounding or maximization purposes, as we will see next. Also, in our approach, it allows for the inactive tasks not to interfere in the value of quality. Other notions of quality would require another model, like the minimum, for cases where the global quality is that of the weakest component.

2.4. Properties and objectives

Bounding the sum of costs Sharing the processor means that at each cycle, the time needed to compute each of the tasks (one step of each) must be contained within a global period T_{max}, i.e.: *the sum of local costs should always be less than a given maximum* $T < T_{max}$. The control of mode switching must *go only to global configurations where this property is true.* In the example, we take $T_{max} = 11$. Then, we have to exclude configurations S_5 and S_6, i.e., the controller has to *forbid* the transitions from S_1 to S_5 and S_6.

Maximizing quality We want to deliver the functionality at the best possible global quality (i.e., least degradation): maximal, and evenly distributed. From a configuration S with successors $S' \in Succ(S)$, we want to keep only transitions going to a configuration such that the property is satisfied.

First objective: maximizing global quality. *Amongst the remaining possible next global configurations S', go only to those where the sum of local qualities is maximal* i.e., $Q = Q_{max} = max_{S' \in Succ(S)}(Q(S'))$. In the example, on the remaining successor states, we have $Q_{max} = 7$: we keep only configurations S_2 and S_7. There can be several successors with equal, maximal, global quality, but with different local distributions amongst modes.

Second objective: minimize time. For an equivalent quality, we want to pay the least cost: *Amongst the remaining possible next global configurations S', go only to those where the time cost T is minimum, i.e.,* $T = T_{min} = min_{S' \in Succ(S)}(T(S))$. In the example, for the same quality $Q_{max} = 7$, S_2 has a time cost of 10, S_7 has a time cost of 8. Hence, we keep only configuration S_7. That is to say, the controller must authorize only ($\mathtt{c2}_1$,ϵ) from S_1.

Alternate objective: having a homogeneous quality. Another objective could concern homogeneous quality, defined as: amongst configurations with equal maximal quality, choose those where the values are closest to the average, That is to say, the property we want to be satisfied is: *amongst the remaining possible next global configurations S', go only to those where the difference D between local qualities and the average is minimum, i.e.,* $D = D_{min} = min_{S' \in Succ(S)}(D(S))$. The average quality of modes *of active tasks* (which are m in number) is: $Q_{avg} = \frac{Q}{m}$ We define the difference $D(S) = \sum_i (|q_i - Q_{avg}| \times a_i)$. More elaborate notions of distance could be meaningful here, like variance (the average of distances to the average) or standard deviation. In the example, on the states remaining after the first objective, we have $Q_{avg} = 3.5$, and $D(S_2) = 1, D(S_7) = 5$. Hence, we keep only configuration S_2. That is to say, the controller must authorize only ($\mathtt{c0}_1$ `and` $\mathtt{c1}_2$) from S_1.

2.5. Conclusion, and other possible patterns

As a conclusion, we have described in this section patterns of tasks and properties that can be used by a user like predefined entities, and be assembled into a complex system model. We will see in the next sections how this model is sufficient to perform optimal discrete controller synthesis, in particular benefiting from the theoretical and tool support provided by SIGNAL/SIGALI. This means that a user can be offered this functionality of discrete control synthesis without having to acquire more technicalities about it.

One could imagine more complex models of tasks with multiple modes, e.g., other top-level activity control patterns (e.g., tasks that can be started by the controller, with no request [11]), other mode structures (like parallel composition, hierarchy as in Mode Automata [5]) involving interaction and/or synchronization (communication) between tasks or with sensors.

We can also have other interpretations of weights, like: degradation of quality implies a longer time needed for the same result (we can then apply optimization along trajectories, among several ones joining same start and goal). A lower cost in energy can require more time for the functionality to be fulfilled (this goes well with our previous scheme if it is interpreted as a higher quality). We could also consider algorithms to which an *a priori* cost (e.g., search depth) can be associated, according to the time available (as with Taylor function developments, "anytime algorithms").

3. Discrete control synthesis, SIGNAL/SIGALI

This section briefly presents the essential concepts underlying discrete control synthesis, introducing just the necessary notions for optimal discrete control synthesis to be understood in Section 4. It also mentions how the SIGNAL/SIGALI environment provides for a practical software tool for specifying systems, synthesizing controllers, and graphically simulating the controlled system.

The SIGNAL language and programming environment. Among tools for compilation, analysis, and code generation (See web site: `www.irisa.fr/espresso`) [4], it features the verification and synthesis tool SIGALI. So, the controller synthesis methodology is integrated from specification to simulation of the synthesized controller [7]. Rather than presenting SIGNAL syntax, available elsewhere, let us consider that it is a structured reactive language, with which it is easy to specify reactive systems like those of previous section, and that it can be compiled into the form of transition system mentioned just below. We here restrict ourselves to SIGNAL processes involving only booleans and events. Such SIGNAL programs are finite state machines. Controller synthesis is performed by translating SIGNAL into dynamical systems over $\mathbb{Z}/_{3\mathbb{Z}} = \{-1, 0, +1\}$. The following coding is used: *true* $\mapsto +1$, *false* $\mapsto -1$, *absent* $\mapsto 0$.

Transition system. Any SIGNAL specification can then be translated into a set of equations called polynomial dynamical system (PDS) of the form :

$$S \quad = \quad \left\{ \begin{array}{rcl} X' & = & P(X,Y,U) \\ 0 & = & Q(X,Y,U) \\ 0 & = & Q_0(X) \end{array} \right. \qquad (1)$$

where X, Y, U, X' are vectors of variables in $\mathbb{Z}/_{3\mathbb{Z}}$ and $dim(X) = dim(X') = n$. The components of the vectors X and X' represent the current and next states of the system and are called *state variables*. Y is a vector of variables in $\mathbb{Z}/_{3\mathbb{Z}}$, called *uncontrollable event variables*, whereas U is a vector of *controllable event variables.* The first equation is the *state transition equation*; the second equation is called the *constraint equation* and specifies which events may occur in a given state, i.e., Y, U which are solutions of this equation for a state X are labels of the transitions going out from X; the last equation gives the *initial states*. The behavior of such a PDS is the following: at each instant t, given a state x_t and an admissible y_t, we can choose some u_t which is admissible, i.e. such that $Q(x_t, y_t, u_t) = 0$, and the system evolves into state $x_{t+1} = P(x_t, y_t, u_t)$.

The events labelling the transitions are partitioned into those, Y that can not be controlled (typically inputs received from sensors, exceptions events, events with priority, failures, tick of a clock) and those, U, of which the value can be determined or constrained, typically by a discrete controller (typically the starting of some task). The former are called *uncontrollable*, and the latter *controllable*. Hence, we consider a transition systems, in which events can occur simultaneously. A transition between two consecutive states can be labelled by a vector of events (some controllable, some others uncontrollable). This constitutes one of the main differences with [9]. In our case, transitions are partially controllable, whereas in the Ramadge & Wonham formulation, they are either controllable or uncontrollable.

These notions can be used in control synthesis, where a transition system can be modified by constraining events declared controllable, *making* it satisfy a property [7]. A transition system can be submitted to a series of such operations, in a process of incremental synthesis. Given a PDS S, as defined by (1) a controller is defined by a system of two equations $C(X, Y, U) = 0$ and $C_0(X) = 0$, where the latter equation $C_0(X) = 0$ determines initial states satisfying the control objectives and the former describes how to choose the instantaneous controls; when the controlled system is in state x, and an event y occurs, any value u such that $Q(x, y, u) = 0$ and $C(x, y, u) = 0$ can be chosen. The behavior of the system S composed with the controller is:

$$S_c \; = \left\{ \begin{array}{rcl} X' & = & P(X,Y,U) \\ 0 & = & Q(X,Y,U) \; = \; C(X,Y,U) \\ 0 & = & Q_0(X) \; = \; C_0(X) \end{array} \right. \qquad (2)$$

Using algebraic methods, avoiding state space enumeration, we can compute automatically, that is to say *synthesize* controllers (C, C_0) which ensure:

- the *invariance* of a set of states, the *reachability* of a set of states from the initial states of the system, the *attractivity* of a set of states E from a set of states F,
- the *minimally restrictive control*, choice of a control such that the system evolves, at the next instant, into a state where the maximum number of uncontrollable

events is admissible, as well as the *stabilization of a system*.

A tool is available, called SIGALI, which implements this with decision diagrams techniques typical of model-checking. Some instructions used in the remainder are: `B_True` (resp. `B_False`) which designates the set of states where a predicate is true (resp. false), and `S_Security`, resp. `S_Reachable`, the synthesis operations for objectives of invariance, resp. reachability. The result of synthesis operations is a decision diagram, characterizing the constraints on controllable events necessary for the property to be satisfied (if possible).

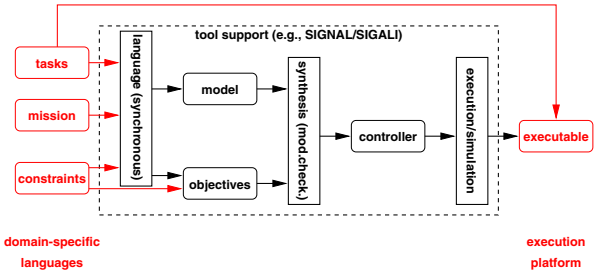

Figure 2. The SIGNAL/SIGALI environment.

Simulation. Specification, synthesis, and simulation in the SIGNAL/SIGALI environment goes as shown within the dashed line in Figure 2 [7]. It involves the modeling of the system in all its possible behaviors and the specification of properties (desirable and undesirable) and objectives (invariance, reachability, attractivity). The properties and objectives can be expressed in SIGNAL (actually SIGNAL+, where calls to SIGALI functionalities can be inserted), which eases their specification: they can be stated in terms of the variables and events used in the system model. The SIGNAL compiler is then used to produce a transition system which is given as input to SIGALI, upon which discrete control synthesis is performed automatically. The resulting controller is produced in a form that is recognized by a generic evaluator, which can be integrated with an application-specific graphical simulation environment. The SIGNAL compiler is used once again for the production of an interactive graphical simulator integrating model and controller.

Modifying the specification and obtaining a new controller can be done automatically, by running the same operations, without having to re-examine the whole controller manually. The result of final synthesis is the maximally permissive controller which, when running in parallel with the system, yields the desired behavior. This means there there can remain a level of indetermination w.r.t. the action to take in response to an input, given an internal state. One answer is to take an arbitrary solution in the set of correct controls. A less arbitrary one is to give optimization criteria (e.g., costs of states traversed, or transitions taken ...) to be minimized. Another one is to offer an interactive determination execution scheme, where the choice between all the admissible controls is given to a user [7].

4. Optimal discrete control synthesis

On the bases of the notions introduced on Section 3, we will see how to answer to our problem of Section 2. For this we introduce optimal discrete control synthesis.

Cost functions. In order to take into account the notion of levels of e.g., quality, time or energy consumption, in the control objectives, let us first explain the notion of state/event cost function. Let $X = (X_1, \ldots, X_n)$ be the state variables of the system. Then, a cost function is a map from $(\mathbb{Z}/3\mathbb{Z})^n$ to $\mathbb{N}$, which associates to each x of $(\mathbb{Z}/3\mathbb{Z})^n$ some integer k. Within our framework, such a function can be defined by first attaching values to each state variable (according to its status) and then by composing these costs according to a given function f (e.g. +, $\times$, min, max). More formally, the values attached to each state variable are:

$$c(X_i) = \begin{cases} a & \text{if } X_i = 0 \ (i.e.\ absent) \\ b & \text{if } X_i = 1 \ (i.e.\ present\ and\ true) \\ c & \text{if } X_i = 1 \ (i.e.\ present\ and\ false) \end{cases}$$

and the cost function is defined by :

$$C(X) = C(X_1, \ldots, X_n) = f(c(X_1), \ldots, c(X_n)) = k$$

Within our framework, a cost function can for example encode the energy consumption (C_e), the time consumption (C_t), or even the quality levels (C_q) of a system.

Bounding costs function. Let Max_C be a bound the system should not exceed. Let $I = \{x \ / \ C(x) > Max_C\}$ be the set of states that have a cost higher than this bound. In order to avoid these states, we want to compute a controller that ensures the non-reachability of this set of states. This is achieved using the following SIGALI command:

```
S_C : S_Invariant(B_False(I));
```

Once this controller is computed, we obtain a system, where all the reachable states x satisfy: $C_e(x) \leq Max\_C$.

Optimal control and order relations. When dealing with quality (resp. energy), it also seems interesting to maximize (resp. minimize) it. Intuitively speaking, the cost function is used to express priority between the different states that a system can reach in one transition. Let us suppose that the system evolves into a state x, and that y is an admissible event at x. As the PDS is in general not deterministic, it may have several controls u such that $Q(x, y, u) = 0$. Let u_1 and u_2 be two controls compatible with y in x. The system can evolve into either $x_1 = P(x, y, u_1)$ or $x_2 = P(x, y, u_2)$. We synthesize a controller that will choose between u_1 and u_2, in such a

way that the system evolves into either x_1 or x_2 according to a given choice criterion (i.e. with the greater cost for the quality and the lower one for the energy).

Definition 1 *Given a PDS S and a cost function C, a state x_1 is said to be C-better than a state x_2 (denoted $x_1 \succeq_C x_2$), if and only if, $C(x_2) \geq C(x_1)$ (or $\leq$).* •

As we deal with a non strict order relation, from $\succeq_C$, we construct a strict order relation, named $\succ_C$ defined as: $x \succ_C x' \Leftrightarrow \{x \succeq_C x' \wedge \neg(x' \succeq_C x)\}$. We now are interested in the direct control policy we want to be adopted by the system; i.e., how to choose the right control when the system S has evolved into a state x and an uncontrollable event y has occurred.

Definition 2 *For a given state x and admissible event y, a control u_1 is said to be* better *compared to a control u_2, iff $x_1 = P(x, y, u_1) \succ_C x_2 = P(x, y, u_2)$.* •

In other words, the controller has to choose, for a pair (x, y), a control compatible with y in x, that allows the system to evolve into one of the states that are maximal for the relation $\succ_C$. Hence, if the system is in state x and the uncontrollable event y is received, the set of suitable controls is

$$\{u \;/\; Q(x,y,u) = 0 \wedge \forall u' \text{such that } Q(x,y,u') = 0, \\ C(P(x,y,u)) = \max_{u'}(C(P(x,y,u')))\}$$

Based on this policy, we can derive a controller from the cost function C without state space enumeration.

Cost function composition. When considering two criteria, e.g., quality and energy, you may want to first maximize quality, and then minimize energy. For the first objective, you follow the previous methodology, and obtain a set I_d of controls $(u_i)_{i \in I}$, for which for all i the cost of the reached states $C_q(P(x, y, u_i))$ are equal to the maximum:

$$\{u \;/\; Q(x,y,u) = 0 \wedge \forall u' \text{such that } Q(x,y,u') = 0, \\ C_q(P(x,y,u)) = \max_{u'}(C_q(P(x,y,u')))\}$$

Then, the set of suitable controls are given by:

$$\{u \in I_d \;/\; \forall u' \in I_d, \\ C_d(P(x,y,u)) = \min_{u'}(C_d(P(x,y,u')))\}$$

i.e. among the remaining controls, we only keep the ones that minimize C_d. The same kind of techniques can be used if one want to mix event and state costs.

These optimal synthesis functionalities are also implemented efficiently in SIGALI, and can be used for experiment, as outlined in the next section.

5. Experiments

The PDS model and operations introduced in Sections 3 and 4 can be used to encode our task patterns and objectives of Section 2, and solve the problems of control synthesis that interest us. The SIGNAL-SIGALI environment gives concrete tool support for implementing a complete design process from specification in the SIGNAL language to simulation and execution. Experiments have been performed on a three tasks system (See Fig. 3), where each task follows the pattern of section 2, and variants. Specification of the behaviors of the (uncontrolled) system has been done using SIGNAL: task patterns are defined as process models, and tasks themselves are parameterized instances of these process models. The tasks are composed simply using parallel composition. Note that, in the considered system, the pattern of Task 3 is slightly different. Indeed, robots or control systems often require to be always under control, even for rest configurations, because of gravity or other external forces. This motivates the introduction of a pattern for *default tasks*. It is similar to the standard task except that it is not necessary to have a request in order for a default task to become active. Also, when the event `stop` occurs, the controller decides upon the termination of the task, by triggering `not go`.

In order to perform controller synthesis on this system, events `Go`$_i$ and `C`$_k$ are declared to be controllable. Objectives are specified using the SIGALI command language: weights have to be associated with local state variables according to their status (*absent*,*true*,*false*), e.g., time cost:

```
C_Time_H_1 :  a_var(H_1, 0, 7, 0);
C_Time_M_1 :  a_var(H_1, 0, 5, 0);
C_Time_L_1 :  a_var(H_1, 0, 2, 0);
```

The way they are composed is defined there too, e.g.,

```
C_Time=C_Time_H_1+C_Time_H_2+C_Time_H_3
```

Synthesis of the controller is performed by SIGALI. The subset of states with the intended property, e.g.,

```
Good_States_Time :  a_inf(C_Time,10)
```

is computed, and the control synthesis *makes it invariant*, i.e. produces the constraints so that the resulting transition system is invariant: e.g.,

```
S_Security(Good_States_Time)
```

Performances vary according to model size and to objectives, of course. It is our experience that for an example of a few tasks, computation takes little time (the order of magnitude of a compilation). An extensive study of complexity should be done, but on these bases one can already say that the functionality does work for non-trivial models, and hence gives practical help. Scalability is of the same order as for model-checking based verification techniques.

The controller is stored in a file, in a format which can be used by a resolver, in order to build a graphical interactive simulator, using the generic tool-box in SIGNAL [7]. Figure 3 shows the graphical display of the configuration of a three tasks system in the simulator obtained with SIGNAL/SIGALI.

Such a framework provides us with an experimental basis, where we can study the task patterns and models, as well as the properties and objectives, in order to test their

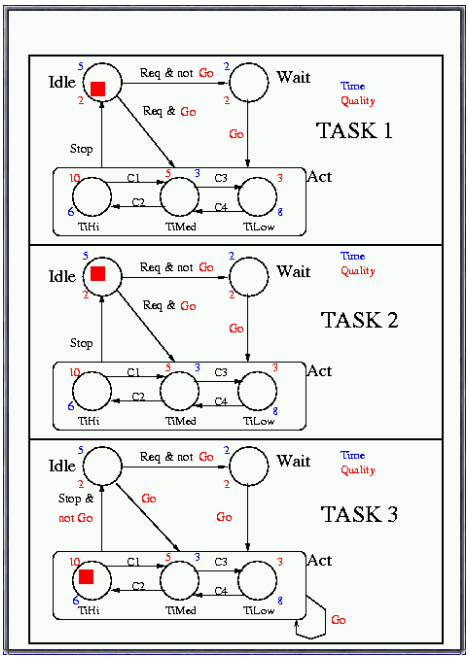

Figure 3. Three multi-mode tasks.

significance, and to determine the synthesis operations to be applied. Up to now, only feasability experiments have been carried; we have onging work on validation the approach w.r.t. scalability (size of modeled applications, complexity of interactions, ...)

6. Conclusion and perspectives

Results described in this paper consist in the proposal of a multi-mode task pattern, equipped with weights describing quality and time. This corresponds to application domains like control systems where functionalities (e.g., control laws) can have different implementations distinguished by cost (e.g., computational, energetic) and quality (e.g., accurateness) [6]. Our task pattern is built in such a way as to enable the use of optimal discrete control synthesis techniques [8]. This provides us with a method where we can obtain, automatically, correct controllers for the switching of modes, satisfying properties on time cost and quality.

This paper is showing first results on the way to a design process where a user would not need to go into technicalities of discrete control synthesis, but would use it on the base of these predefined task patterns, without having to do himself all the encoding of the transition system or formulation of the objectives. The advantages of such an approach are the correctness (by construction) of the controller obtained, the easy modifiability (change the specification and re-run the synthesis), and the precision of a control based on an explicit model of dynamic behaviors.

Perspectives are in several directions. Task patterns and objectives can be enriched, as mentioned earlier, regarding e.g. mode changes cost and objectives on sequences, or control structure: e.g., waiting tasks and release policies involving priorities and order relation (and optimization). Also, it would be interesting to consider enlarging the use of discrete control synthesis techniques: optimization can be applied to trajectories and sequences, rather than to states, and methods dealing with hierarchy and partial observation have been defined. We have plans for considering different application domains: robotics control systems, telecommunications, transportation (automobile equipments).

References

[1] K. Altisen, G. Gößler, and J. Sifakis. Scheduler modelling based on the controller synthesis paradigm. *Journal of Real-Time Systems*, 23(1), 2002. (*to appear*).

[2] G. Fohler. Changing operational modes in the context of pre run-time scheduling. *IEICE Transactions on Information and Systems*, E76-D(11):1333–1340, Nov. 1993.

[3] N. Halbwachs. *Synchronous programming of reactive systems*. Kluwer, 1993.

[4] P. Le Guernic, T. Gautier, M. Le Borgne, and C. Le Maire. Programming real-time applications with signal. *Proceedings of the IEEE*, 79(9):1321–1336, September 1991.

[5] F. Maraninchi and Y. Rémond. Running-modes of real-time systems: A case-study with mode-automata. In *Proceedings of the* 12th Euromicro Conference on Real-Time Systems, ECRTS'00*, Stockholm, Sweden, June 19th - 21th*, 2000.

[6] F. Maraninchi, Y. Rémond, and E. Rutten. Effective programming language support for discrete-continuous mode-switching control systems. In *Proceedings of the 40th IEEE Conference on Decision and Control, CDC'01,* december 4-7, 2001, Orlando, Florida, 2001.

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

[8] H. Marchand and M. Le Borgne. Partial order control of discrete event systems modeled as polynomial dynamical systems. In *1998 IEEE International Conference On Control Applications*, Trieste, Italia, eptember 1998.

[9] P. J. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, 1989.

[10] J. Real and A. Wellings. Implementing mode changes with shared resources in ada. In *Proceedings of the* 11th Euromicro Conference on Real-Time Systems, ECRTS'99*, York, England, UK, June 9th - 11th*, 1999.

[11] E. Rutten. A framework for using discrete control synthesis in safe robotic programming and teleoperation. In *Proc. IEEE Int. Conf. on Robotics and Automation, ICRA'2001,* may 21–26, Seoul, Korea, 2001.

[12] K. Tindell, A. Burns, and A. Wellings. Mode changes in priority pre-emptive scheduled systems. In *Proceedings of the IEEE Real Time Systems Symposium, Phoenix, Arizona*, pages 100–109, Dec. 1992.