Using Speed Diagrams for Symbolic Quality Management

Jacques Combaz, Jean-Claude Fernandez, Joseph Sifakis, Loic Strus
Verimag, Centre Equation - 2 avenue de Vignate F38610 Gières, France

Abstract

We present a quality management method for multimedia applications. The method takes as input an application software composed of actions. The execution times of actions are unknown increasing functions of quality level parameters. The method allows the construction of a Quality Manager which computes adequate action quality levels so as to meet QoS requirements for a given platform. These include deadlines for the actions as well as quality maximization and smoothness.

We extend and improve results of a previous paper by focusing on the reduction of overhead due to quality management. We propose a symbolic quality management method using speed diagrams, a representation of the system's dynamics. Instead of numerically computing a quality level for each action, the Quality Manager changes action quality levels based on the knowledge of constraints characterizing control relaxation regions. These are sets of states in which quality management for a given number of steps can be relaxed without degrading quality.

We provide experimental results for quality management of an MPEG encoder, in particular performance benchmarks for both numeric and symbolic quality management.

1 Introduction

Designing systems meeting both hard and soft real-time requirements is a challenging problem. There exist well-established design methodologies for hard real-time systems, that is systems that do not violate critical properties such as deadlines. These methodologies are based on worst-case analysis using conservative approximations of the system dynamics and static resource reservation, which implies a non optimal use of resources.

In contrast, design methodologies for soft real-time are based on average-case analysis and seek more efficient use of resources (e.g. optimization of speed, jitter, memory, bandwidth, power) without addressing critical behavior issues. They are used for applications where some degradation or even temporal denial of service is tolerated e.g., multimedia and telecommunications.

These two classes of design methodologies are currently disjoint. Meeting hard real-time properties and making optimal use of available resources seem to be two antagonistic requirements. The existing gap between hard and soft real-time often leads to costly and unreliable solutions.

Adaptivity is a means for bridging the gap between the two classes of design methodologies. In previous papers [5, 6], we have presented an adaptive method for QoS management allowing optimal use of computing resources without missing deadlines. The method targets multimedia applications. It allows adapting the overall system behavior by adequately setting quality level parameters for its actions. The objective of the quality management policy is to meet QoS requirements including three types of properties: 1) safety (no deadlines are missed); 2) optimality, (maximization of the available time budget); 3) smoothness of quality levels.

The method takes as input an application software with timing information about its actions. This includes deadlines and (platform-dependent) worst-case and average execution times. It produces a controlled application software meeting the QoS requirements for the target platform. This is obtained by applying to the application software a *Controller* consisting of a *Scheduler* and a *Quality Manager*. Depending on the progress of the computation, the Scheduler chooses the next action to be executed and the Quality Manager the associated quality level parameter.

In [6], we provided tools for implementing the controlled software on bare machines. The implementations use a Quality Manager which computes online quality level parameters for each action of the application software. Despite optimizations, the overhead in execution time due to numeric computation of these parameters can be high. In this paper, we investigate a *symbolic* quality management method allowing considerable overhead reduction. It uses a symbolic representation of the system's dynamics and constraints characterizing control relaxation regions in which quality management can be relaxed without degrading quality. We consider the following simplified version of the general problem by assuming that the application software is

1-4244-0910-1/07/$20.00 ©2007 IEEE.

already scheduled (see figure 1):
• The application software cyclically performs input/output transformations of data streams. It is described as a finite sequence of actions (C-functions). Its execution during a cycle can be controlled by choosing *quality level parameters*. We assume that the execution times of actions are unknown and are increasing with quality.
• We consider single-thread implementations of the application software on a platform for which it is possible, by using timing analysis and profiling techniques, to compute estimates of worst-case execution times and average execution times of actions for different quality levels. Action execution is assumed to be atomic. A compiler is used to generate the controlled software from the initial application software, for given deadline requirements and execution times.

The controlled software can be considered as the composition of the initial application software with a *Quality Manager* (see figure 2). The latter monitors the progress of the computation within a cycle of the application software. At any state of the cycle, it chooses the quality level for the next action to be executed, guided by a *quality management policy*. This is a constraint guaranteeing safety and embodying an optimality criterion. The Quality Manager chooses the maximal quality satisfying this constraint.

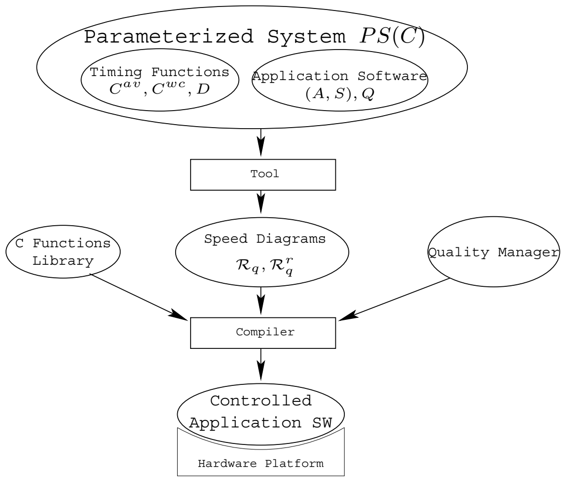

Parameterized System $PS(C)$
Timing Functions C^{av}, C^{wc}, D
Application Software $(A, S), Q$
Tool
C Functions Library
Speed Diagrams $\mathcal{R}_q, \mathcal{R}_q^r$
Quality Manager
Compiler
Controlled Application SW
Hardware Platform

Figure 1. Prototype tool implementation

Our method significantly differs from existing ones. The main difference is fine granularity of quality management, which allows combination of hard and soft real-time techniques. Most existing techniques are applied at system or task level, focus on optimality criteria and are adequate only for soft real-time. The integration of safety criteria is useful in applications where quality should remain above some minimal level [9], [3], e.g., home TVs, or where hard deadlines must be respected. Buttazzo et al.'s elastic tasks model [4], as well as slack scheduling [7], [11] and gain time techniques [1] are based only on worst-case execution times and do not deal with quality smoothness. A common and simple way to treat CPU overload is to skip an instance of a task [10]. Lu et al. [12] propose a feedback scheduling based on PID controllers, but deadline misses remain possible. Steffens et al. [15], [13] minimize deadline misses of an MPEG decoder by applying a Markov decision process and reinforcement learning techniques, combined with structural load analysis.

This paper improves and extends results presented in [5, 6] in several directions.

• It defines and studies *speed diagrams*, a graphical representation of the controlled software's state space for which quality management policies admit a geometric interpretation (see figure 3). A state is defined as a point in a two-dimensional space. One dimension represents the actual (real) time while the other dimension represents a virtual time used by the Quality Manager. The slope of a vector in this space represents (relative) *speed* between virtual time and actual time. In speed diagrams, vectors at 45 degrees slope represent state trajectories where actual and virtual times are equal. Consequently, the locus of optimal states coincides with the bisectrice of the first quadrant. States below the bisectrice, are those where actual time is larger than virtual time and thus the Quality Manager should enforce acceleration of computation by choosing lower quality. In contrast, for states above the bisectrice, optimal use of the available time budget implies the choice of higher qualities.
• It introduces, for a given state of the controlled software and quality q, two kinds of speeds: 1) *ideal speed* characterizes the estimated evolution if all the remaining actions of the application software are run with quality level q; 2) *optimal speed* is the vector characterizing optimal system evolution, that is respecting the deadlines and making the best possible use of the available time budget. We show that the constraint applied by the quality management policy defined in [6] is satisfied for a given quality, if and only if the quality chosen (at a state) is such that the corresponding ideal speed is the least ideal speed exceeding the optimal speed.
• It shows, based on this characterization in terms of speeds, that speed diagrams allow *symbolic* quality management policies. For a given deadline, it is possible to specify the set of the states for which the Quality Manager chooses constant quality q. These states form a *region* defined by a set of inequalities involving actual time, and average and worst-case execution times of actions. Knowledge of these constant quality regions allows a more efficient implementation of the quality management policy. An even more efficient implementation can be achieved by using a symbolic description of the regions of states from which it can be ensured that the Quality Manager will choose quality q for the next r actions. From these regions, it is possible to re-

lax control for r steps and thus, to considerably reduce the overhead due to quality management.

The paper is organized as follows. In section 2 we present the quality management problem and its solution described in [6]. The results about speed diagrams and their use for quality management are presented in section 3. Section 4 presents experimental results for a non trivial video encoder.

2 Quality Management: The Problem and its Solution

2.1 Definition of the Problem

We provide a formalization of the quality management problem by considering that the application software is already scheduled. It is characterized by an execution sequence $\{ s_{i-1} \xrightarrow{a_i} s_i \}_{1 \le i \le n}$, where $S = \{ s_0, \ldots, s_n \}$ is a set of *states* and $A = \{ a_1, \ldots, a_n \}$ is a set of *actions*. Actions correspond to blocks of code. Their execution is atomic.

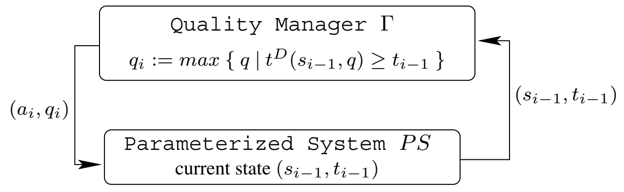
The execution of the application software (A, S) on a platform, is modeled by an *execution time function* $C : A \to \mathbb{R}^+$, associating with an action a_i its execution time $C(a_i)$. The corresponding timed execution sequence is $\{ (s_{i-1}, t_{i-1}) \xrightarrow{a_i} (s_i, t_i) \}_{1 \le i \le n}$ such that $t_0 = 0$ and $t_i - t_{i-1} = C(a_i)$.

Execution times for actions may considerably vary over time as they depend on the contents of data. Furthermore, non predictability of the underlying platform is an additional factor of uncertainty. We consider that they are not known in advance, but are bounded by worst-case estimates. To cope with the inherent uncertainty of execution times, we assume that the actions are parameterized by quality levels. This leads to the following model.

Definition 1 *A* **parameterized system** *PS is an application software (A, S) with*
• *a finite set of integer* quality levels *Q*
• *a* worst-case *execution time function $C^{wc} : A \times Q \to \mathbb{R}^+$ non-decreasing with quality levels that is, for all actions a, the function $q \mapsto C^{wc}(a, q)$ is a non-decreasing function*
• *a parameter C, called* actual *execution time function, $C : A \times Q \to \mathbb{R}^+$ non-decreasing with quality levels and such that $C(a, q) \le C^{wc}(a, q)$ for any action a and quality level q.*

The execution of a parameterized system is characterized by the family of sequences $\{ (s_{i-1}, t_{i-1}) \xrightarrow{a_i, q_i} (s_i, t_i) \}_{1 \le i \le n, q_i \in Q}$ such that $t_0 = 0$ and $t_i - t_{i-1} = C(a_i, q_i)$.

Quality Managers are used to adequately restrict the behavior of a parameterized system so as to meet given properties.



Quality Manager Γ
$q_i := max \{ q \mid t^D(s_{i-1}, q) \ge t_{i-1} \}$
(a_i, q_i)
(s_{i-1}, t_{i-1})
Parameterized System PS
current state (s_{i-1}, t_{i-1})

Figure 2. Quality Manager

Definition 2 *Given a parameterized system PS a* **Quality Manager** *is a function $\Gamma : S \times \mathbb{R}^+ \to Q$ giving, for a state (s_{i-1}, t_{i-1}) of PS, the quality level q_i for executing the next action a_i.*

$PS||\Gamma$ denotes a controlled system *obtained as the composition of the parameterized system PS and the Quality Manager Γ. For a given actual execution time function C, it has a single execution sequence $\{ (s_{i-1}, t_{i-1}) \xrightarrow{a_i, q_i} (s_i, t_i) \}_{1 \le i \le n}$ such that $q_i = \Gamma(s_{i-1}, t_{i-1})$.*

The quality management problem for a given parameterized system PS consists in finding a Quality Manager Γ meeting the QoS requirements. That is,there are no deadline misses and the overall quality is maximal. It is formalized as follows.

Definition 3 (quality management problem) *Given a parameterized system PS and a* deadline function *$D : A \to \mathbb{R}^+$ associating with each action a its deadline $D(a)$, find a Quality Manager Γ such that for any actual time function $C \le C^{wc}$:*
• *Γ is* safe *(deadlines are met), that is for any state (s_i, t_i) of $PS||\Gamma$ we have $D(a_i) \ge t_i$ where $t_i = C(a_1, q_1) + \ldots + C(a_i, q_i)$.*
• *The overall execution time is maximal, that is for any safe Quality Manager Γ', $t_n \ge t'_n$, where t_n (resp. t'_n) is the completion time of the last action in $PS||\Gamma$ (resp. $PS||\Gamma'$).*

In [6], we require in addition to feasibility and optimality, *smoothness* for the quality levels chosen by the Quality Manager. Informally, smoothness means low fluctuation of quality levels. Due to lack of space, we do not study this property which is essential for most multimedia applications [14].

2.2 Quality Manager Design

We summarize the method for the design of Quality Managers, given in [6].

2.2.1 Quality Manager Design Principles

Figure 2 shows interaction between the Quality Manager Γ, applying a *quality management policy*, and the application software, i.e. the parameterized system PS. The Quality Manager observes the current state (s_{i-1}, t_{i-1}) of PS and computes the next quality level q_i for the next action a_i. The Quality Manager is defined by:

$$\Gamma(s_{i-1}, t_{i-1}) = q_i = \mathbf{max}\ \{\ q \mid t^D(s_{i-1}, q) \geq t_{i-1}\ \}.$$

The function $t^D : A \times Q \rightarrow \mathbb{R}^+$ defines the *quality management policy* of the Quality Manager. It gives for a state of the application software s_{i-1} and a quality level q, the estimated elapsed time $t^D(s_{i-1}, q)$ at next state s_i if the rest of the actions is executed with constant quality q. If the inequality $t^D(s_{i-1}, q) \geq t_{i-1}$ is satisfied, then it is possible to complete execution without missing the deadlines specified by D. The chosen quality level q_i at state (s_{i-1}, t_{i-1}) is maximal amongst the quality levels q meeting the inequality $t^D(s_{i-1}, q) \geq t_{i-1}$.

The function t^D is defined as follows:

$$t^D(s_{i-1}, q) = \mathbf{min}_{i \leq k \leq n}\ D(a_k) - C^D(a_i..a_k, q),$$

where $C^D(a_i..a_k, q)$ denotes an estimation of the total execution time for the sequence of actions $a_i, a_{i+1}, \ldots, a_k$.

Choosing an adequate quality management policy, i.e., that ensures safety and an optimal use of resources, is a non trivial problem discussed in [5] and [6]. In [6] we proposed the *mixed* quality management policy based on the *mixed* execution time function C^D defined below. Its interest has been shown through both theoretical and experimental results.

2.2.2 Mixed Quality Management Policy

The execution time function C^D combines the use of two execution time functions C^{sf} and C^{av}. The first allows respecting the safety requirement, that is no deadline is missed. The second is used to enhance smoothness of quality levels.

The *safe* execution time function C^{sf} gives a worst-case estimation of the total execution time of $a_i..a_k$:

$$C^{sf}(a_i..a_k, q) = C^{wc}(a_i, q) + C^{wc}(a_{i+1}..a_k, q_{min})$$

where $C^{wc}(a_{i+1}..a_k, q_{min})$ denotes the total worst-case execution time for the sequence of actions $a_{i+1}, \ldots, a_k$ with the minimal quality level $q_{min} = \mathbf{min}\ Q$. That is,

$$C^{wc}(a_{i+1}..a_k, q_{min}) = \sum_{i+1 \leq j \leq k} C^{wc}(a_j, q_{min}).$$

As the quality level can be changed by the Quality Manager after the execution of the first action a_i, we take the quality level q for the first action, and q_{min} for the remaining actions. The application of *safe* quality management policy ensures safety of the Quality Manager. Nevertheless, it may lead to considerable variations of quality levels in a sequence e.g., by starting with high quality levels and terminating with low quality levels.

To improve smoothness of the quality levels, we introduce an *average* execution time function $C^{av} : A \times Q \rightarrow \mathbb{R}^+$, non-decreasing with quality. Average execution times can be estimated by static analysis and/or profiling techniques. We define δ^{max} as the maximum difference between the worst-case and the average behavior:

$$\delta^{max}(a_i..a_k, q) = \mathbf{max}_{i \leq j \leq k}\ \delta(a_j..a_k, q),$$

where $\delta(a_j..a_k, q) = C^{sf}(a_j..a_k, q) - C^{av}(a_j..a_k, q)$. That is, for a sequence of actions $a_i..a_k$ and quality level q, $\delta^{max}(a_i..a_k, q)$ is a kind of safety margin with respect to the average behavior. It measures uncertainty on execution times in order to meet the deadlines.

The mixed execution time function C^D is defined by $C^D = C^{av} + \delta^{max}$. It combines average and worst-case behavior. It is possible to take into account execution time needed for quality management by adequately overestimate average and worst-case execution times.

In the rest of the paper, we consider a Quality Manager applying the mixed quality management policy.

3 Speed Diagrams and their Use for Quality Management

Speed diagrams are a graphical representation of system's states, in which quality management policies have a geometric interpretation in terms of relative speed between virtual time (average execution times) and actual time. They allow a better understanding of the impact of worst-case execution times on achieving optimality. They also allow a symbolic approach for the definition and implementation of the Quality Manager. We show that a quality management policy can be expressed by a partition of the state space into regions specified by constraints involving deadlines and worst-case and average execution times.

3.1 Definition

Speed diagrams represent in a two dimensional space the evolution of a parameterized system PS and its Quality Manager applying the mixed quality management policy defined in section 2.2.2 for a deadline function D (figure 3). The vertical axis of the speed diagram corresponds to virtual time computed from average execution times and their deadlines. The horizontal axis represents actual time.

The following definitions provide a formalization of speed diagrams, as well as results about the interpretation of the mixed quality management policy in terms of speed vectors.

3.1.1 System State Representation

Let (s_i, t_i) be a state of a parameterized system PS and $D(a_k)$ the deadline of an action in the remaining sequence of actions $a_{i+1}, \ldots, a_k, \ldots, a_n$. The virtual time variable $y_i(q)$ is used to estimate at some point of the execution, the time distance from the deadline $D(a_k)$ if the sequence of the actions $a_1, \ldots, a_n$ is run with uniform quality q. It is defined by:

$$y_i(q) = \frac{C^{av}(a_1..a_i, q)}{C^{av}(a_1..a_k, q)} \cdot D(a_k).$$

Intuitively, $y_i(q)$ is the percentage of the consumed virtual time at state s_i with respect to the available time budget $D(a_k)$. Notice that normalization with respect to the deadline implies that $y_k(q) = D(a_k)$ (see figure 3).

As a result of the normalization, points on the diagonal (45 degree slope) correspond to optimal behavior. Points $(t_i, y_i(q))$ below the diagonal correspond to states where the actual computation is late with respect to virtual time. Conversely, for points above the diagonal, the computation goes faster than estimated.

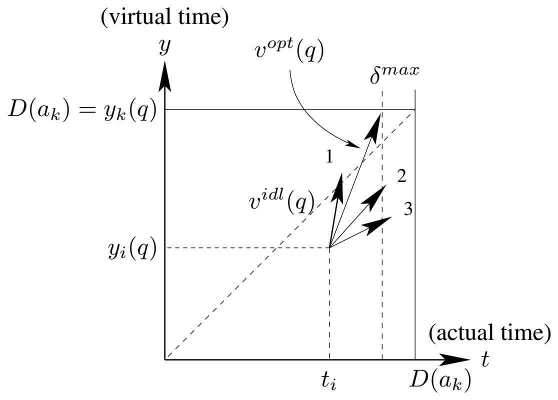


Figure 3. Speed diagram

3.1.2 Ideal and Optimal Speeds

Let (s_i, t_i) and (s_j, t_j) be two states of PS such that $j > i$, Consider their corresponding positions $(t_i, y_i(q))$ and $(t_j, y_j(q))$ in the speed diagram for a quality level q and a deadline $D(a_k)$, $k \geq j$.

The *speed* $v_{i,j}(q)$ between $(t_i, y_i(q))$ and $(t_j, y_j(q))$ is given by the ratio

$$v_{i,j}(q) = \frac{y_j(q) - y_i(q)}{t_j - t_i}.$$

We introduce two notions of speed to explain the mixed quality management policy.

• The **ideal speed** $v^{idl}(q)$ is the speed for constant quality level q when the actual time is equal to the average time. As $C = C^{av}$ and $q_{i+1} = \ldots = q_j = q$, we have $t_j - t_i = C(a_{i+1}..a_j, q) = C^{av}(a_{i+1}..a_j, q) = C^{av}(a_1..a_j, q) - C^{av}(a_1..a_i, q)$. Then, the ideal speed $v^{idl}(q)$ between $(t_i, y_i(q))$ and $(t_j, y_j(q))$ is equal to

$$v^{idl}(q) = \frac{y_j(q) - y_i(q)}{t_j - t_i}$$

$$= \frac{D(a_k)}{C^{av}(a_1..a_k, q)} \cdot \frac{C^{av}(a_1..a_j, q) - C^{av}(a_1..a_i, q)}{C^{av}(a_1..a_j, q) - C^{av}(a_1..a_i, q)}$$

$$= \frac{D(a_k)}{C^{av}(a_1..a_k, q)}.$$

Notice that the ideal speed $v^{idl}(q)$ is independent of the choice of i and j, and only depends on the target deadline $D(a_k)$ and the quality level q. This means that for constant quality assignments the trajectory of the system in the diagram is linear in the ideal case $C = C^{av}$.

• The **optimal speed** $v^{opt}(q)$ is the speed between the current position $(t_i, y_i(q))$ and the position $D(a_k) - \delta^{max}(a_{i+1}..a_k, q), D(a_k))$. It can easily be shown that $v^{opt}(q)$ is equal to

$$\frac{D(a_k)}{C^{av}(a_1..a_k, q)} \cdot \frac{C^{av}(a_{i+1}..a_k, q)}{D(a_k) - \delta^{max}(a_{i+1}..a_k, q) - t_i}.$$

By targeting point $(D(a_k) - \delta^{max}(a_{i+1}..a_k, q), D(a_k))$ instead of $(D(a_k), D(a_k))$ the quality manager respects a safety margin $\delta^{max}(a_{i+1}..a_k, q)$ which is sufficient to ensure termination before the deadline $D(a_k)$. The value $\delta^{max}(a_{i+1}..a_k, q)$ is a safety margin characterizing the tradeoff between feasibility and optimality for the mixed quality management policy.

Proposition 1 *Given a parameterized system PS, a state (s_i, t_i) of PS, a quality level q and a target deadline $D(a_k)$, $k > i$, we have:*

$$v^{idl}(q) \geq v^{opt}(q) \iff D(a_k) - C^D(a_{i+1}..a_k, q) \geq t_i.$$

The above proposition demonstrated in [6], allows a geometric interpretation of the mixed quality management policy in terms of relative speeds between average execution time and actual time. The Quality Manager makes a conservative approximation of the optimal speed v^{opt} by choosing the optimal speed exceeding v^{opt} with maximal quality. Intuitively, the chosen speed corresponds to an optimal behavior for constant quality assignment (uniform speed) and maximal time budget utilization, in which a safety margin is integrated in order to meet the deadline.

Our quality management technique assumes that the Quality Manager is called before executing each action of the application software. Since the Quality Manager and the application are composed together, there is an overhead for computing the Quality Manager. An important issue is reducing this overhead. In the rest of the paper, we explain how to safely relax the granularity of control that is, reducing the number of Quality Manager calls, whereas choosing the same quality levels.

3.2 Quality Regions

Consider a parameterized system PS and a Quality Manager Γ applying the mixed quality management policy. For a better understanding of the choices of the Quality Manager, we study *quality regions*, sets of system states where the chosen quality level is constant.

Definition 4 *Given a parameterized system PS and a Quality Manager Γ, a* **quality region** *$\mathcal{R}_q$ for the quality level q is defined by:*

$$\mathcal{R}_q = \{ (s_i, t_i) \mid \Gamma(s_i, t_i) = q \}.$$

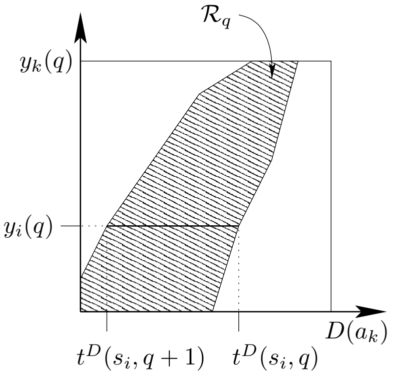


Figure 4. Quality region for a quality level q

Let (s_i, t_i) be a state of a parameterized system PS, and $(t_i, y_i(q))$ the corresponding position in the speed diagrams for a deadline $D(a_k)$, $k \geq i$. It can be shown that t^D is a non-increasing function of q. This implies that

• for $q < q_{max} = \mathbf{max}\ Q$, $\Gamma(s_i, t_i) = q$ iff $t^D(s_i, q) \geq t_i$ and $t_i > t^D(s_i, q+1)$.

• for $q = q_{max}$, $\Gamma(s_i, t_i) = q$ iff $t^D(s_i, q) \geq t_i$.

Proposition 2 *For a given quality level q and a state (s_i, q_i), $(s_i, t_i) \in \mathcal{R}_q$ if and only if*

$$t_i \in \left]\ t^D(s_i, q+1)\ ,\ t^D(s_i, q)\ \right] \text{ for } q < q_{max}$$
$$t_i \in \left]\ -\infty, t^D(s_i, q)\right] \text{ for } q = q_{max}.$$

This proposition allows computing quality regions $\mathcal{R}_q$. A region is defined by the set of the $y_i(q)$ for all i and the corresponding interval bounds characterizing its borders (see figure 4).

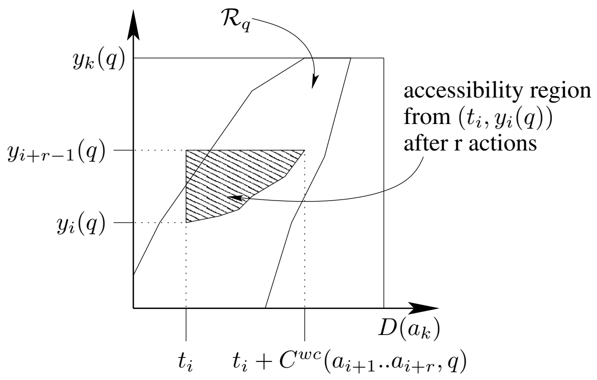
3.3 Control Relaxation Regions

We propose a *control relaxation* method allowing to reduce the number of Quality Manager calls. We define *control relaxation regions*, sets of system states in which the Quality Manager can be relaxed without degrading the quality of control.

Let (s_i, t_i) be a state of a parameterized system PS. Assume that the Quality Manager Γ chooses the quality level q at state (s_i, t_i) that is, $(s_i, t_i) \in \mathcal{R}_q$. We consider a conservative control relaxation: the Quality Manager can be relaxed for $r \geq 1$ steps if we ensure that the quality level chosen for all the next r actions $a_{i+1}, a_{i+2}, \ldots, a_{i+r}$ is q.

Definition 5 *Given a parameterized system PS and a Quality Manager Γ, a* **control relaxation region** *$\mathcal{R}_q^r$ for the quality level q and an integer $r \geq 1$ is defined by:*

$$\begin{cases} \mathcal{R}_q^1 = \mathcal{R}_q \\ (s_i, t_i) \in \mathcal{R}_q^r \ \Leftrightarrow\ (s_i, t_i) \in \mathcal{R}_q \ \wedge (s_{i+1}, t_{i+1}) \in \mathcal{R}_q^{r-1}. \end{cases}$$



$\mathcal{R}_q$
$y_k(q)$
accessibility region from $(t_i, y_i(q))$ after r actions
$y_{i+r-1}(q)$
$y_i(q)$
$D(a_k)$
t_i
$t_i + C^{wc}(a_{i+1}..a_{i+r}, q)$

Figure 5. Control relaxation: the principle

We consider the states (s_j, t_j), $j \in \{i, i+1, \ldots, i+r-1\}$ of $PS||\Gamma$, and find conditions for these states to be in $\mathcal{R}_q$ (see figure 5). For instance, figure 5 shows a case where this property is not satisfied and the Quality Manager cannot be relaxed from state (s_i, t_i) for r steps.

Due to uncertainty, actual execution times can range from 0 to C^{wc}. So we can only give upper and lower bounds for t_j:

$$t_i + C^{wc}(a_{i+1}..a_j, q) \geq t_j \geq t_i. \qquad (1)$$

By proposition 2 and equation (1), $(s_j, t_j) \in \mathcal{R}_q$ if the following equations are satisfied for all $j \in \{i, i+1, \ldots, i+r-1\}$,

$$t^D(s_j, q) - C^{wc}(a_{i+1}..a_j, q) \geq t_i \qquad (2)$$
$$t_i > t^D(s_j, q+1). \qquad (3)$$

then, we can relax the Quality Manager for r steps. As $t^D(s_j, q+1)$ is increasing with j, (3) is satisfied for all j if and only if $t_i > t^D(s_{i+r-1}, q+1)$. This leads to the following proposition.

Proposition 3 *For a given quality level q, an integer $r \geq 1$ and a state (s_i, q_i), $(s_i, t_i) \in \mathcal{R}_q^r$ if and only if*

$$t_i \in \left]\ t^D(s_{i+r-1}, q+1)\ ,\ t^{D,r}(s_i, q)\ \right] \text{ for } q < q_{max}$$
$$t_i \in \left]\ -\infty\ ,\ t^{D,r}(s_i, q)\ \right] \text{ for } q = q_{max},$$

where $t^{D,r}(s_i, q) = \mathbf{min}_{i \le j \le i+r-1} \; t^D(s_j, q) - C^{wc}(a_{i+1}..a_j, q)$.

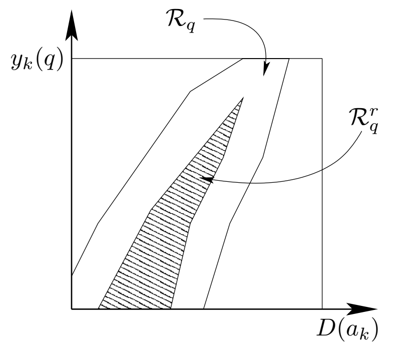


Figure 6. Control relaxation region

4 Experimental Results

4.1 Experimental Framework

We applied our results to an MPEG video encoder written in C (more than 7,000 lines). The encoder cyclically treats frames. Each frame is split into $N(396 \le N \le 1,620)$ macroblocks of 256 pixels.

The parameterized system PS describing the video encoder consists of

• the scheduled video encoder, a sequence of $1,189$ actions, that is $A = \{ a_0, \ldots, a_{1,188} \}$

• for each action, a set of 7 quality levels $Q = \{ 0, \ldots, 6 \}$.

We used the BIP/Think tool chain [2, 8] for the implementation of the controlled software on a bare Apple iPod Video (5G). Unfortunately, this machine is too slow for video applications but this was the only possible target for the implementation tools, in particular because it has a reliable real-time clock needed by the Quality Manager. Thus, the given benchmarks are indicative and useful only for estimating relative values. The generated implementations include binary code for the encoder and its Quality Manager as well as minimal OS features. For the iPod, we estimated worst-case and average execution times by profiling.

We have generated three Quality Managers for a single global deadline $D = 30$ s:

Numeric Quality Manager. This is a straightforward implementation of the mixed quality management policy given in section 2.2.1.

Quality Manager using quality regions. We developed a prototype tool in Matlab/Simulink for pre-computing quality regions $\mathcal{R}_q$ defined in 3.2. These are used by the Quality Manager to compute online action quality levels. By proposition 2, quality regions are characterized by the set of the values $t^D(s_i, q)$ for all quality levels q and for all states s_i. Thus, as i ranges from 0 to $|A| - 1$ this set is specified by $|A||Q| = 8,323$ integers. For the video encoder application, we have measured an overhead in memory allocation of 300 KB.

Quality Manager using control relaxation regions. This is an optimization of the previous implementation in which the control relaxation regions $\mathcal{R}_q^r$ (defined in 3.3) are used to determine the set $\rho = \{ 1, 10, 20, 30, 40, 50 \}$ of the numbers of steps for which quality management can be relaxed. By definition 3, control relaxation regions are characterized by the set of the values $t^D(s_{i+r-1}, q+1)$ and $t^{D,r}(s_i, q)$ for all the quality levels q, indices $i \in \{ 1, \ldots, |A| - 1 \}$ and relaxation steps $r \in \rho$, that is a set of $2|A||Q||\rho| = 99,876$ integers. We observed an overhead in memory allocation of 800 KB.

4.2 Performance of Quality Managers

We have compared the performance of the three Quality Managers for an input sequence of 29 frames of 352×288 pixels (396 macroblocks). Execution time overhead due to quality management is in average 5.7 % for the numeric implementation, 1.9 % for the symbolic implementation using quality regions (no control relaxation) and less than 1.1 % for the implementation using control relaxation regions. Thus, symbolic quality management allows significant overhead reduction with respect to numeric quality management. Consequently, symbolic Quality Managers choose higher quality levels than the numeric Quality Manager (see figure 7). This leads to a significant improvement of the overall video quality.

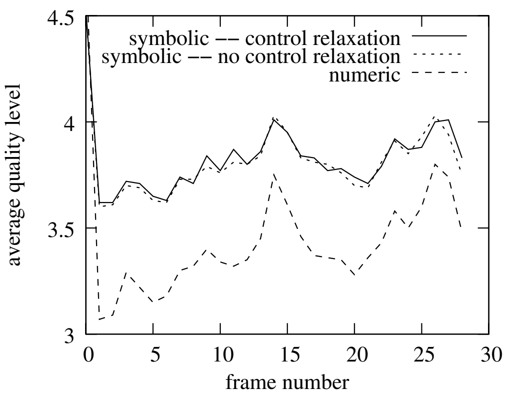


Figure 7. Average quality level

Figure 8 compares for the sequence $a_{200}..a_{700}$ of 501 actions in a frame, overheads in execution time with and without control relaxation. Notice that the number of relaxation steps r is dynamically adapted during the execution of the application : $r = 40$ from a_{200} to a_{421}, $r = 1$ from a_{422} to a_{564}, and $r = 10$ from a_{565} to a_{700}.

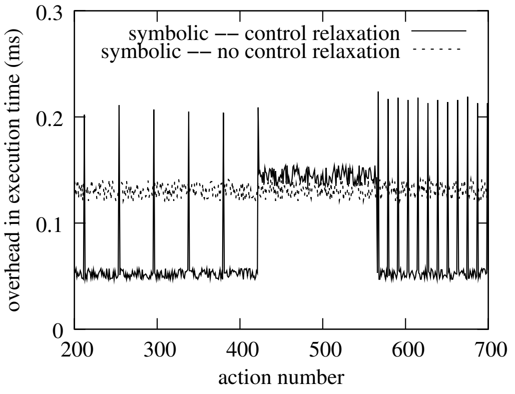


Figure 8. Overhead in execution time

5 Conclusion

We presented a new *symbolic* quality management method for hard real-time and optimal use of resources. The method improves and extends our previous results [5, 6].

• Speed diagrams provide a general and abstract framework for studying the dynamics of the controlled software, determined as the interplay between the execution of the application software and the Quality Manager. The geometric interpretation of system's evolution allows performance analysis and a deeper understanding of control management policies in terms of relations between ideal and optimal speeds.

• The use of quality and control relaxation regions which can be pre-computed from their symbolic representation, allows a more efficient implementation of Quality Managers. Safe control relaxation proves to be a very interesting idea as it allows keeping Quality Manager's intervention minimal and thus reduce the corresponding execution time overhead.

• The implementation technique can be fully automated for platforms providing access to accurate real-time clocks at low overhead as illustrated by using the BIP/Think tool chain.

Experimental results confirm the interest of symbolic quality management because of its low overhead. We work in several directions to improve the method and the supporting tools: adaption to multiple tasks, using linear constraints to approximate control relaxation regions, study of properties of control relaxation regions for classes of programs e.g., iterations, and modular use of speed diagrams. We also explore possible applications of the technique to power management where quality level is replaced by frequency and the objective is to minimize energy consumption without missing the deadlines.

Acknowledgements — We thank the reviewers for their useful and valuable comments.

References

[1] N. C. Audsley, R. I. Davis, and A. Burns. Mechanisms for enhancing the flexibility and utility of hard real-time systems. In *Real-Time Systems Symposium*, pages 12–21. IEEE, 1994.

[2] A. Basu, M. Bozga, and J. Sifakis. Modeling Heterogeneous Real-Time Components in BIP. In *4th IEEE International Conference International Conference on Software Engineering and Formal Methods*, September 2006.

[3] R. J. Bril, M. Gabrani, C. Hentschel, G. C. van Loo, and E. F. M. Steffens. QoS for consumer terminals and its support for product families. In *Proceedings of the International Conference on Media Futures*, 2001.

[4] G. C. Buttazzo, G. Lipari, and L. Abeni. Elastic task model for adaptive rate control. In *RTSS*, pages 286–295, 1998.

[5] J. Combaz, J. Fernandez, T. Lepley, and J. Sifakis. Fine grain QoS control for multimedia application software. In *Design, Automation and Test in Europe (DATE'05) Volume 2*, pages 1038–1043, 2005.

[6] J. Combaz, J.-C. Fernandez, T. Lepley, and J. Sifakis. QoS Control for Optimality and Safety. In *Proceedings of the 5th Conference on Embedded Software*, September 2005.

[7] R. I. Davis, K. W. Tindell, and A. Burns. Scheduling slack time in fixed priority preemptive systems. In *Proceeding of the IEEE Real-Time Systems Symposium*, pages 222–231.

[8] J.-P. Fassino, J.-B. Stefani, J. Lawall, and G. Muller. THINK: A Software Framework for Component-based Operating System Kernels. In *Proceedings of the Usenix Annual Technical Conference*, June 2002.

[9] D. Isovic, G. Fohler, and L. Steffens. Timing constraints of MPEG-2 decoding for high quality video: misconceptions and realistic assumptions.

[10] G. Koren and D. Shasha. Skip-over: Algorithms and complexity for overloaded systems that allow skips. Technical Report TR1996-715, , 1996.

[11] J. Lehoczky and S.Thuel. Algorithms for scheduling hard aperiodic tasks in fixed-priority systems using slack stealing. In *Proceedings of the IEEE Real-Time System Symposium*.

[12] C. Lu, J. Stankovic, G. Tao, and S. Son. Feedback control real-time scheduling: Framework, modeling and algorithm. *special issue of RT Systems Journal on Control-Theoretic Approach To Real-TIme Computing*, 23(1/2):85–88, 2002.

[13] L. Papalau, C. M. O. Pérez, and L. Steffens. In S. Goddard, editor, *Work-In-Progress Session of the 16th Euromicro Conference on Real-Time Systems*, pages 33–36, 2004.

[14] G. M. Schuster, G. Melnikov, and A. K. Katsaggelos. A review of the minimum maximum criterion for optimal bit allocation among dependent quantizers. *IEEE Transactions on Multimedia*, 1(1):3–17, 1999.

[15] C. C. Wüst, L. Steffens, R. J. Bril, and W. F. Verhaegh. QoS control strategies for high-quality video processing. In *Euromicro Conference on Real-Time Systems*, pages 3–12. IEEE, 2004.