Real-Time Syst (2008) 40: 1–43
DOI 10.1007/s11241-008-9049-2

# Symbolic quality control for multimedia applications

**Jacques Combaz · Jean-Claude Fernandez · Joseph Sifakis · Loïc Strus**

Published online: 12 February 2008


**Abstract** We present a fine grain quality control 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 Controller which computes adequate action schedules and corresponding quality levels, so as to meet QoS requirements for a given platform. These include requirements for safety (action deadlines are met) as well optimality (maximization and smoothness of quality levels).

The Controller consists of a Quality Manager and a Scheduler. For each action, the Controller uses a quality management policy for choosing a schedule and quality levels meeting the QoS requirements. The schedule is selected amongst a set of optimal schedules computed by the Scheduler.

We extend and improve results of previous papers providing a solid theoretical basis for designing and implementing the Controller.

We propose a symbolic quality management method using *speed diagrams*, a representation of the controlled 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 computation steps can be relaxed without degrading quality.

We study techniques for efficient computation of optimal schedules.

We present experimental results including the implementation of the method and benchmarks for an MPEG4 video encoder. The benchmarks show drastic performance improvement for controlled quality with respect to constant quality. They also show that symbolic quality management allows significant reduction of the overhead with respect to numeric quality management. Finally, using optimal schedules can lead to considerable performance gains.

J. Combaz · J.-C. Fernandez · J. Sifakis · L. Strus (✉)
DCS, Verimag, Centre Equation, 2 avenue de Vignate, 38610 Gières, France
e-mail: loic.strus@imag.fr

**Keywords** Real-time · Multimedia · Embedded systems · QoS control · Symbolic techniques

## 1 Introduction

There exist two diverging approaches in systems engineering.

- *Critical systems engineering* based on worst-case analysis using conservative approximations of system dynamics and static resource reservation. This approach is applied whenever a system's correctness means no violation of critical conditions such as missing a deadline or reaching a dangerous state.
- *Best effort engineering* based on average-case analysis and seeking efficient use of resources without addressing critical behavior issues, e.g., optimization of speed, jitter, memory, bandwidth, power. It is applied whenever some degradation or even temporal denial of service is tolerated e.g., telecommunications.

The two approaches are currently disjoint. They correspond to different research communities and different practices. They adopt different computing paradigms, use specific execution platforms, middleware and networks. It is often advocated that such a separation is inevitable, especially for embedded systems with uncertain execution and external environments. Meeting critical properties and making optimal use of available resources seem to be two antagonistic requirements. To ensure critical properties, worst-case estimates must be used and this may lead to inefficient use of resources if they are statically pre-allocated. The existing gap between critical and best effort approaches often leads to costly and unreliable solutions.

To bridge the gap between the two approaches, it is essential to develop design techniques for adaptive systems meeting both critical and best effort properties. Such techniques should allow control of the overall system behavior so as to meet critical properties while making the best possible use of resources, taking into account both average and worst-case behavior.

Adaptivity is a means for bridging the gap between the two approaches. For multimedia embedded software, the fast evolution of market needs, user requirements and platforms requires reliable adaptation of features at minimal costs. Currently, adaptation of application software to target platforms and needs is too costly. To meet given QoS requirements a significant amount of experimentation is needed on virtual or real prototypes involving fine tuning of parameters of the components of the application software. After tuning, the behavior of application software can be modified only by changing user-defined input parameters. Thus, adaptability is coarse grain as it can be achieved only by modifying global parameters. Furthermore, some delay is necessary for adaptation due to limited controllability of the application software over the underlying execution system.

In previous papers (Combaz et al. 2005a, 2005b), we have presented an adaptive method for QoS management in multimedia applications. The method allows adapting the overall system behavior by adequately setting quality level parameters for its actions. The objective is to meet QoS requirements including two types of properties: (1) safety (no deadline missed); (2) optimality, meaning both maximal and smooth quality during a cycle.

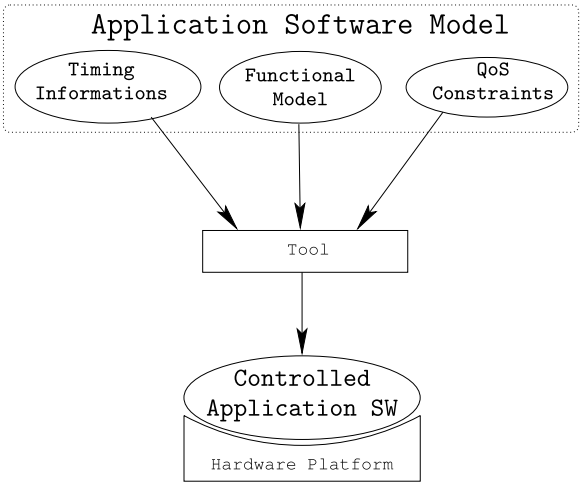



**Fig. 1** Prototype tool implementation

The method takes as input an application software with timing information about its actions (see Fig. 1). 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. The method is characterized by the following:

- The application software cyclically performs input/output transformations of data streams. It is described as a partially ordered set of actions (C-functions). Its execution during a cycle can be controlled by choosing for actions adequate schedules and *quality level parameters*. We assume that the execution times of actions are unknown and are increasing with quality. Thus, quality maximization implies maximal utilization of CPU time.
- 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 *Controller* (see Fig. 2).

- The controller monitors the progress of the computation in a cycle and chooses the next action to run and its quality level. It consists of a *Quality Manager* and a *Scheduler*. These are constructed from a functional model of the application software, a dependency relation between its actions equipped with deadlines and average/worst-case execution times for each action.
- During a cycle, the Controller chooses a schedule and the quality of the next action to be executed guided by a *quality management policy*. This is a constraint of the form $t_p \geq t$ where $t$ is actual time and $t_p$ is a function giving an estimate of the actual time depending on the quality management policy. Safety means that

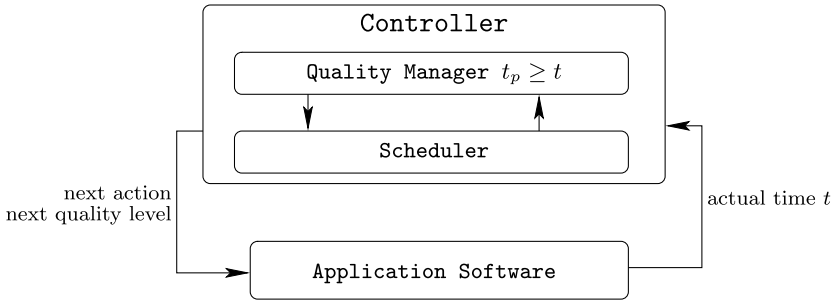



**Fig. 2** Controller architecture

no action deadline is missed during a cycle. It is implied by the condition $t_p \geq t$. Maximization of the utilization of the available time budget is achieved when the difference $t_p - t$ is minimal. It means that the available time for completing an action is used as much as possible to obtain the best quality (without violating safety). The Quality Manager is assisted by a Scheduler, which provides for different qualities, optimal schedules, that is, schedules maximizing $t_p$. It chooses the safe schedule corresponding to the maximal quality.

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 (Isovic et al. 2003; Bril et al. 2001), e.g., home TVs, or where hard deadlines must be respected. Buttazzo et al.'s elastic tasks model (Buttazzo et al. 1998), as well as slack scheduling (Davis et al. 1993; Lehoczky and Thuel 1994) and gain time techniques (Audsley et al. 1994) 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 (Koren and Shasha 1996). Lu et al. (2002) propose a feedback scheduling based on PID controllers, but deadline misses remain possible. Steffens et al. (Wüst et al. 2004; Papalau et al. 2004) minimize deadline misses of an MPEG decoder by applying a Markov decision process and reinforcement learning techniques, combined with structural load analysis. Rajkumar et al. (Rajkumar et al. 1997; Hansen et al. 2001) provide resource allocation algorithms that maximize global QoS for concurrent and independent tasks. The proposed model (Q-RAM) is event-driven and does not encompass uncertainty about resource consumption, that is, actual resource utilization is determined by the resource allocation algorithms.

This paper improves and extends results presented in (Combaz et al. 2005a, 2005b) in two directions. It proposes a *symbolic* quality management technique and studies techniques for computing optimal schedules. Its main contributions regarding symbolic quality management are the following:

- 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 Fig. 7). 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 (Combaz et al. 2005b) is satisfied for a given quality, if and only if the quality chosen (at a state) is such that the 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 a 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 relax control for $r$ consecutive actions and thus, to considerably reduce the overhead due to quality management.

The main contributions regarding computation of optimal schedules are the following:

- We define two functions for selecting schedules that maximize $t_p$. These functions characterize respectively uncertainty (the difference between worst-case and average execution times) and system's fall-back ability (the difference between worst-case execution time for the minimal quality and average execution time).
- We show that for systems with unknown execution times, EDF schedules are not optimal, in general. We use the two functions to compute optimal EDF schedules.

The paper is organized as follows. In Sect. 2 we discuss the Quality Control problem and present a general approach for designing quality controllers. Section 3 discusses possible choices of quality management policies. We propose and compare three policies, one for safety and two others for both safety and optimality. Amongst these policies, the *mixed management policy* better fits the QoS requirements. For this policy, we define speed diagrams and symbolic quality management techniques. Section 4 discusses scheduler design issues for the mixed management policy. Section 5 presents experimental results including the implementation of the method and

benchmarks for a video MPEG4 encoder. The benchmarks show drastical performance improvement for controlled quality with respect to constant quality. They also show that symbolic quality management allows significant reduction of the overhead with respect to numeric quality management. Finally, using optimized schedules can lead to considerable performance gains.

## 2 Quality control problem and general approach

### 2.1 Quality control problem for known execution times

We provide preliminary results about the quality control problem for known execution times.

#### *2.1.1 The problem*

A precedence graph is used to describe functional behavior of an application software. It models dependencies between its actions (C-functions), and from which all the possible schedules can be extracted.

**Definition 2.1** (Precedence graph) A **precedence graph** is a pair $G = (A, \prec)$ where $A$ is a set of *actions* and $\prec \subseteq A \times A$ is a partial order on A.

(Semantics) The precedence graph $G = (A, \prec)$ defines a **transition system** $(S, A, \longrightarrow)$ where $S$ is a set of *states* and $\longrightarrow \subseteq S \times A \times S$ is a *labeled transition relation* defined by:

- a state $s_i \subseteq A$ is a backwards closed set of actions, that is, for all $a_1 \in s_i$, $a_2 \prec a_1 \Rightarrow a_2 \in s_i$
- for two states $s_i$ and $s_j$, we have $s_i \xrightarrow{a_n} s_j$ if $s_i = \{a_1, \ldots, a_{n-1}\}$ and $s_j = \{a_1, \ldots, a_{n-1}, a_n\}$.

A sequence of actions $a_1..a_n$ is a **schedule** of $G$ if $n = |A|$ and there exist states $s_0, \ldots, s_n$ such that $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n$. Notice that as $n = |A|$ we have $s_0 = \emptyset$ and $s_n = A$. We denote by $\Sigma(G)$ the set of the schedules of $G$. Given a state $s_{i-1} = \{a_1, \ldots, a_{i-1}\}$ of $G$, a sequence of actions $a_i..a_n$ is a schedule from state $s_{i-1}$ if there exist states $s_i, \ldots, s_n$ such that $n = |A|$ and $s_{i-1} \xrightarrow{a_i} s_i \xrightarrow{a_{i+1}} \cdots \xrightarrow{a_n} s_n$. We denote by $\Sigma(G, s_{i-1})$ the set of the schedules from state $s_{i-1}$. Notice that $\Sigma(G, s_0) = \Sigma(G)$.

Given a precedence graph $G = (A, \prec)$ and a subset of actions $A' \subseteq A$, we define $G/A'$, the restriction of $G$ to $A'$ by $G/A' = (A', \prec \cap (A' \times A'))$.

*Example 2.1* Consider the precedence graph $G = (A, \prec)$ with five actions $A = \{\texttt{Quant}, \texttt{IQuant}, \texttt{IntraP}, \texttt{IDCT}, \texttt{Coding}\}$, shown in Fig. 3. This precedence graph is a fragment of the model of the video encoder presented in Sect. 5. The relation $\prec$ is the transitive closure of the relation $\rightsquigarrow = \{(\texttt{Quant}, \texttt{IQuant}), (\texttt{Quant}, \texttt{IntraP}), (\texttt{IntraP}, \texttt{Coding}), (\texttt{IQuant}, \texttt{IDCT})\}$ shown in Fig. 3. Since $s = \{\texttt{Quant}, \texttt{IntraP}, \texttt{Coding}\}$ is a backward closed set of actions, it is a state of $G$ and `IQuant IDCT` is the only schedule from this state. The sequence of actions `Quant IntraP Coding IQuant IDCT` is a schedule of $G$.

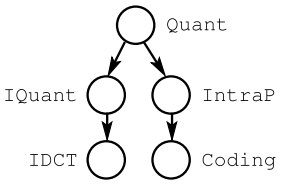



**Fig. 3** Example of precedence graph

A system is an application software running on a platform. It is modeled by the application software and functions associating with each action its execution time and its deadline.

**Definition 2.2** (System) A **system** is a tuple $SY = (G, Q, C, D)$ where:

- $G$ is a precedence graph.
- $Q = [q_{min}, q_{max}]$ is a finite interval of integers corresponding to *quality levels*.
- $C : A \times Q \to \mathbb{R}^+$ ($\mathbb{R}^+$ denotes the set of non-negative reals) is a function giving the *execution time* $C(a, q)$ of action $a$ for quality level $q$. We assume that, for all $a \in A$, $q \mapsto C(a, q)$ is a non-decreasing function.
- $D : A \to \mathbb{R}^+ \cup \{+\infty\}$ is a function giving for any action $a$ its *deadline* $D(a)$.

(Semantics) The system $SY$ defines a **transition system** $(S \times \mathbb{R}^+, A \times Q, \longrightarrow)$ such that:

- states are given by pairs $(s_i, t_i)$ where $s_i$ is a state of $G$ and $t_i \in \mathbb{R}^+$ is a value of time; we take $t_0 = 0$ for $s_0 = \emptyset$;
- for two states $(s_i, t_i)$ and $(s_j, t_j)$, an action $a$ and a quality level $q$, we have $(s_i, t_i) \xrightarrow{a,q} (s_j, t_j)$ if $s_i \xrightarrow{a} s_j$ in $G$ and $t_j - t_i = C(a, q)$.

Controllers are used to adequately restrict the behavior of a system to meet given properties.

**Definition 2.3** (Controller) A **controller** of a system $SY = (G, Q, C, D)$ is a function $\Gamma : S \times \mathbb{R}^+ \to A \times Q$ giving for any state $(s_{i-1}, t_{i-1})$, an action $a_i$ and its quality level $q_i$ such that there exists $(s_i, t_i)$ and $(s_{i-1}, t_{i-1}) \xrightarrow{a_i,q_i} (s_i, t_i)$.

$SY \| \Gamma$ denotes the *controlled system* obtained as the composition of the system $SY$ and the controller $\Gamma$. It has a single execution sequence $\{(s_{i-1}, t_{i-1}) \xrightarrow{a_i,q_i} (s_i, t_i)\}_{1 \le i \le |A|}$ such that $(a_i, q_i) = \Gamma(s_{i-1}, t_{i-1})$.

For a system $SY = (G, Q, C, D)$, a **quality assignment** is a function $\theta : A \to Q$ giving for any action $a$ its quality level $q$. A controller $\Gamma$ of $SY$ computes a schedule $a_1..a_n$ and a quality assignment $\theta$ such that $SY \| \Gamma$ has a single execution sequence: $\{(s_{i-1}, t_{i-1}) \xrightarrow{a_i,\theta(a_i)} (s_i, t_i)\}_{1 \le i \le |A|}$.

**Definition 2.4** (Quality control problem with known execution times) Given a system $SY = (G, Q, C, D)$, find a controller $\Gamma$ which computes a schedule $a_1..a_n$ and a quality assignment $\theta$, such that:

**Table 1** Execution time function $C$ and deadline function $D$

| Action $a$ | $C(a, q_{min})$ | $C(a, q_{max})$ | $D(a)$ |
|---|---|---|---|
| `Quant` | 10 | 20 | $+\infty$ |
| `IQuant` | 25 | 75 | $+\infty$ |
| `IDCT` | 25 | 75 | $D_1$ |
| `IntraP` | 5 | 25 | $+\infty$ |
| `Coding` | 5 | 25 | $D_2$ |

- (safety) the controller is **safe**, that is, action deadlines are met, meaning that for any state $(s_i, t_i)$ of $SY \| \Gamma$ we have $D(a_i) \geq t_i$
- (maximal utilization) the overall execution time is maximal, that is, for any safe controller $\Gamma'$, $t_n \geq t'_n$, where $t_n$ (resp. $t'_n$) is the completion time of the last action in $SY \| \Gamma$ (resp. $SY \| \Gamma'$).

*Example 2.2* Consider the system $SY = (G, Q, C, D)$ given in Fig. 3. We take $Q = \{q_{min}, q_{max}\}$, and an execution time function $C$ and a deadline function $D$ as given in Table 1.

If $D(\texttt{IDCT}) = D_1 = 180$ and $D(\texttt{Coding}) = D_2 = 240$, a controller $\Gamma$ computing the schedule `Quant IntraP Coding IQuant IDCT` and the quality assignment $\theta = q_{max}$ is not safe, as $C(\texttt{Quant IntraP Coding IQuant IDCT}, q_{max}) = 20 + 75 + 75 + 25 + 25 = 220 > D(\texttt{IDCT}) = D_1 = 180$.

A solution to the quality control problem for $SY$ is a controller $\Gamma$ computing the schedule `Quant IQuant IDCT IntraP Coding` with constant quality assignment $\theta = q_{max}$. Since the quality levels are maximal, the overall execution time $t_n = 220$ is maximal. Furthermore, the controller is safe:

$$C(\texttt{Quant IQuant IDCT}, q_{max}) = 170 \quad \leq \quad D(\texttt{IDCT}) = D_1 = 180, \quad \text{and}$$

$$C(\texttt{Quant IQuant IDCT IntraP Coding}, q_{max}) = 220$$

$$\leq \quad D(\texttt{Coding}) = D_2 = 240.$$

As $C : A \times Q \to \mathbb{R}^+$ is a known execution time function, the controller $\Gamma$ can be computed statically. We provide an algorithm for computing a schedule $a_1..a_n$ and a quality assignment $\theta$ which is a solution to the quality control problem.

### *2.1.2 Controller design*

**Definition 2.5** (Policy function $t_p$) Given a system $SY = (G, Q, C, D)$, a schedule $a_1..a_n$ and a quality assignment $\theta$, the **policy function** $t_p$ is defined by:

$$t_p(a_1..a_n, \theta) = \min_{1 \leq k \leq n} D(a_k) - C(a_1..a_k, \theta),$$

where $C(a_1..a_k, \theta)$ denotes the overall execution time of the sequence of actions $a_1..a_k$ at quality level $\theta$, that is:

$$C(a_1..a_k, \theta) = \sum_{1 \leq i \leq k} C(a_i, \theta(a_i)).$$

Notice that $t_p(a_1..a_n, \theta)$ gives the margin of the schedule $a_1..a_n$ with respect to action deadlines and for the quality assignment $\theta$.

**Proposition 2.1** *Let $SY = (G, Q, C, D)$ be a system and $\Gamma$ be a controller of $SY$ computing a schedule $a_1..a_n$ and a quality assignment $\theta$. The controller $\Gamma$ is safe if and only if $t_p(a_1..a_n, \theta) \geq 0$.*

*Proof* We have:

$$\begin{aligned} & t_p(a_1..a_n, \theta) \geq 0 \\ \Leftrightarrow \quad & \min_{1 \leq k \leq n} D(a_k) - C(a_1..a_k, \theta) \geq 0 \\ \Leftrightarrow \quad & \forall i \in \{1, \ldots, n\} \quad D(a_i) \geq C(a_1..a_k, \theta) \\ \Leftrightarrow \quad & \forall i \in \{1, \ldots, n\} \quad D(a_k) \geq t_k. \end{aligned}$$

□

*Example 2.3* For the two schedules of the system given in Example 2.2, `Quant IntraP Coding IQuant IDCT` and `Quant IQuant IDCT IntraP Coding`, we have:

$$\begin{aligned} & t_p(\texttt{Quant IntraP Coding IQuant IDCT}, q_{max}) \\ & \quad = \mathbf{min}\{D_2 - C(\texttt{Quant IntraP Coding}, q_{max}), \\ & \qquad\quad D_1 - C(\texttt{Quant IntraP Coding IQuant IDCT}, q_{max})\} \\ & \quad = \mathbf{min}\{240 - 70, 180 - 220\} = -40, \text{ and} \end{aligned}$$

$$\begin{aligned} & t_p(\texttt{Quant IQuant IDCT IntraP Coding}, q_{max}) \\ & \quad = \mathbf{min}\{D_1 - C(\texttt{Quant IQuant IDCT}, q_{max}), \\ & \qquad\quad D_2 - C(\texttt{Quant IQuant IDCT IntraP Coding}, q_{max})\} \\ & \quad = \mathbf{min}\{180 - 170, 240 - 220\} = 10. \end{aligned}$$

We conclude that the schedule `Quant IntraP Coding IQuant IDCT` misses the deadline $D_1$ of the action `IDCT` for constant quality level $q_{max}$, whereas the schedule `Quant IQuant IDCT IntraP Coding` for the same quality level $q_{max}$, meets all the deadlines.

**Definition 2.6** (Optimal scheduler *Best_Sched*) For a system $SY = (G, Q, C, D)$ an **optimal scheduler** is a function *Best_Sched* giving for any quality assignment $\theta$ a schedule $a_1^\theta..a_n^\theta = Best\_Sched(\theta)$ such that $a_1^\theta..a_n^\theta$ maximizes $t_p(a_1^\theta..a_n^\theta, \theta)$, that is:

$$t_p(a_1^\theta..a_n^\theta, \theta) = \mathbf{max}\{t_p(a_1..a_n, \theta) \mid a_1..a_n \in \Sigma(G)\}.$$

Let $\theta$ be a schedule of $G$. Notice that, for any schedule $a_1..a_n$, the overall execution time $C(a_1..a_n, \theta)$ is independent of $a_1..a_n$. We denote by $\ll$ the binary relation on $\Theta$ such that $\theta \ll \theta' \Leftrightarrow C(a_1..a_n, \theta) < C(a_1..a_n, \theta')$. The relation $\ll$ is a strict total order on classes of quality assignments $\{\theta \mid C(a_1..a_n, \theta) = \text{constant}\}$.

**Proposition 2.2** *For a given system* $SY = (G, Q, C, D)$ *and associated optimal scheduler Best_Sched, the following algorithm provides a solution to the quality control problem with known execution times* (*Definition* 2.4).

$$\textbf{for all } \theta \in \Theta \textbf{ do } a_1^\theta..a_n^\theta := \mathit{Best\_Sched}(\theta) \textbf{ od}$$

$$\theta_M = \underset{\ll}{\boldsymbol{max}}\{\theta : A \to Q \mid t_p(a_1^\theta..a_n^\theta, \theta) \geq 0\}$$

$$\textbf{return } (a_1^{\theta_M}..a_n^{\theta_M}, \theta_M).$$

*Proof* Consider the schedule $a_1^{\theta_M}..a_n^{\theta_M}$ and the quality assignment $\theta_M$ computed by the algorithm given in the proposition. Assume that there exists a schedule $a_1..a_n$ and a quality assignment $\theta$ such that $t_p(a_1..a_n, \theta) \geq 0$ and $\theta_M \ll \theta$. Since $a_1^\theta..a_n^\theta = \mathit{Best\_Sched}(\theta)$ maximizes $t_p$, we have $t_p(a_1^\theta..a_n^\theta, \theta) \geq t_p(a_1..a_n, \theta) \geq 0$, that is, $\theta_M \gg \theta$ or $\theta_M$ and $\theta$ are the same. (Contradiction). □

We show that EDF schedules are optimal. Their computation is based on a backward propagation of critical deadlines in the precedence graph.

**Definition 2.7** (EDF schedule) Given a system $SY = (G, Q, C, D)$, we define the **global** deadline function $D^* : A \to \mathbb{R}^+$ as follows:

$$D^*(a) = \textbf{min}\{D(a') \mid a \prec a'\} \cup \{D(a)\}.$$

A schedule $a_1..a_n$ of $G$ is an **EDF** schedule if for all $i \in \{1, \ldots, n-1\}$ we have $D^*(a_i) \leq D^*(a_{i+1})$. We denote by $EDF(G, D)$ the set of the EDF schedules of $G$ with respect to the deadline function $D$.

*Example 2.4* Consider the system $SY = (G, Q, C, D)$ given in Examples 2.2 and 2.3. The global deadline function $D^*$ is such that $D^*(\texttt{Quant}) = \textbf{min}\{D_1, D_2\}$, $D^*(\texttt{IQuant}) = D(\texttt{IDCT}) = D_1$ and $D^*(\texttt{IntraP}) = D(\texttt{Coding}) = D_2$ (see Fig. 4). For actions `IDCT` and `Coding`, the functions $D^*$ and $D$ are the same.

If $D_1 < D_2$, the only EDF schedule is given by the sequence `Quant IQuant IDCT IntraP Coding`. If $D_1 > D_2$, the only EDF schedule is given by the sequence `Quant IntraP Coding IQuant IDCT`.

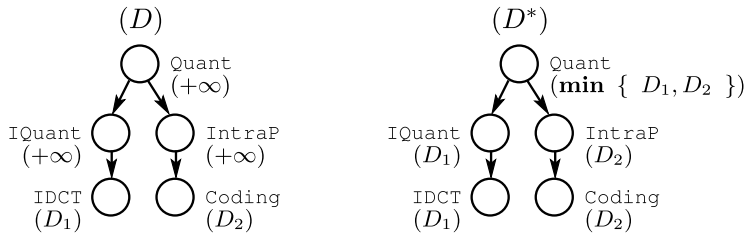



**Fig. 4** Deadline functions $D$ and $D^*$

**Proposition 2.3** *The systems* $SY = (G, Q, C, D)$ *and* $SY^* = (G, Q, C, D^*)$ *have the same policy function*

$$t_p(a_1..a_n, \theta) = \min_{1 \le k \le n} D(a_k) - C(a_1..a_k, \theta) = \min_{1 \le k \le n} D^*(a_k) - C(a_1..a_k, \theta).$$

*Proof* Without loss of generality, we assume that $C > 0$. Assume that $\mathbf{min}_{1 \le k \le n} D^*(a_k) - C(a_1..a_k, \theta) \neq \mathbf{min}_{1 \le k \le n} D(a_k) - C(a_1..a_k, \theta)$. As $D^* \le D$, we obtain $\mathbf{min}_{1 \le k \le n} D^*(a_k) - C(a_1..a_k, \theta) < \mathbf{min}_{1 \le k \le n} D(a_k) - C(a_1..a_k, \theta)$.

Let $i$ be an index such that $D^*(a_i) - C(a_1..a_i, \theta) = \mathbf{min}_{1 \le k \le n} D^*(a_k) - C(a_1..a_k, \theta)$. Then, we have $D^*(a_i) - C(a_1..a_i, \theta) < D(a_i) - C(a_1..a_i, \theta)$, that is, $D^*(a_i) < D(a_i)$. By definition of $D^*$, we conclude that there exists $j > i$ such that $D^*(a_i) = D^*(a_j)$. Then, we obtain $D^*(a_j) - C(a_1..a_j, \theta) < D^*(a_i) - C(a_1..a_i, \theta)$ (Contradiction). □

The following Proposition 2.4 allows the computation of a function *Best_Sched* that returns an EDF schedule $a_1..a_n = Best\_Sched(\theta)$. Notice that this function *Best_Sched* is constant and its computation can be done in polynomial time. We need the following lemma in order to demonstrate the proposition.

**Lemma 2.1** *Let* $a_1..a_n$ *be a schedule such that there exists two consecutive and independent actions* $a_i$ *et* $a_{i+1}$ ($a_i \not\prec a_{i+1}$ *and* $a_{i+1} \not\prec a_i$) *such that* $D(a_i) \ge D(a_{i+1})$. *For any quality assignment* $\theta$ *we have*:

$$t_p(a_1..a_{i-1}a_{i+1}a_ia_{i+2}..a_n, \theta) \ge t_p(a_1..a_n, \theta).$$

*Proof* Let $I_1$, $I_2$ and $I_3$ be subsets of indexes such that $I_1 = \{1, \ldots, i-1\}$, $I_2 = \{i, i+1\}$ and $I_3 = \{i+2, \ldots, n\}$. We have:

$$t_p(a_1..a_n, \theta) = \mathbf{min}\{D(a_k) - C(a_1..a_k, \theta) \mid k \in I_1 \cup I_2 \cup I_3\} \quad \text{and}$$

$$\begin{aligned} &t_p(a_1..a_{i-1}a_{i+1}a_ia_{i+2}..a_n, \theta) \\ &\quad = \mathbf{min}\{D(a_k) - C(a_1..a_k, \theta) \mid k \in I_1\} \cup \{D(a_{i+1}) - C(a_1..a_{i-1}a_{i+1}, \theta)\} \\ &\qquad \cup \{D(a_i) - C(a_1..a_{i-1}a_{i+1}a_i, \theta)\} \\ &\qquad \cup \{D(a_k) - C(a_1..a_{i-1}a_{i+1}a_ia_{i+2}..a_k, \theta) \mid k \in I_3\}. \end{aligned}$$

As $C(a_1..a_{i-1}a_{i+1}a_ia_{i+2}..a_k, \theta) = C(a_1..a_k, \theta)$ for any $k \in I_3$, we have $\{D(a_k) - C(a_1..a_{i-1}a_{i+1}a_ia_{i+2}..a_k, \theta) \mid k \in I_3\} = \{D(a_k) - C(a_1..a_k, \theta) \mid k \in I_3\}$. Thus, the lemma holds if:

$$D(a_{i+1}) - C(a_1..a_{i-1}a_{i+1}, \theta) \ge \mathbf{min}\{D(a_k) - C(a_1..a_k, \theta) \mid k \in I_2\} \quad \text{and} \qquad (1)$$

$$D(a_i) - C(a_1..a_{i-1}a_{i+1}a_i, \theta) \ge \mathbf{min}\{D(a_k) - C(a_1..a_k, \theta) \mid k \in I_2\}. \qquad (2)$$

As $C(a_1..a_{i-1}a_{i+1}, \theta) \le C(a_1..a_{i+1}, \theta)$ we have $D(a_{i+1}) - C(a_1..a_{i-1}a_{i+1}, \theta) \ge D(a_{i+1}) - C(a_1..a_{i+1}, \theta)$. This demonstrates (1). Since $D(a_i) \ge D(a_{i+1})$ and $C(a_1..a_{i-1}a_{i+1}a_i, \theta) = C(a_1..a_{i+1}, \theta)$ we have $D(a_i) - C(a_1..a_{i-1}a_{i+1}a_i, \theta) \ge D(a_{i+1}) - C(a_1..a_{i+1}, \theta)$. This demonstrates (2). □

**Proposition 2.4** *Let $SY = (G, Q, C, D)$ be a system and $a_1..a_n$ be an EDF schedule of $G$. For any quality assignment $\theta$ we have*:

$$t_p(a_1..a_n, \theta) = \mathbf{max}\{t_p(a'_1..a'_n, \theta) \mid a'_1..a'_n \in \Sigma(G)\}.$$

*Proof* We apply Lemma 2.1 as follows. Let $a_1..a_n$ be an EDF schedule and $a'_1..a'_n$ be an arbitrary schedule. We can obtain $a_1..a_n$ from $a'_1..a'_n$ by successively swapping two consecutive independent actions with inverted deadlines $D^*$. □

*Example 2.5* Consider the system $SY = (G, Q, C, D)$ given in Examples 2.2, 2.3 and 2.4, and a deadline function $D$ such that $D(\texttt{IDCT}) = D_1 = 180$ and $D(\texttt{Coding}) = D_2 = 240$. The only EDF schedule is `Quant IQuant IDCT IntraP Coding` and we have $t_p(\texttt{Quant IQuant IDCT IntraP Coding}, q_{max}) = 10$ (see Example 2.3), which is maximal for the quality level $q_{max}$. A proof is given as follows.

Consider an arbitrary schedule $a_1..a_5$ of $G$, and let $i \in \{1, \ldots, 5\}$ be the index such that $a_i = \texttt{IDCT}$. Due to the precedence constraints of $G$, `Quant` and `IQuant` must be executed before executing `IDCT`. Consequently, $C(a_1..a_i, q_{max}) \geq C(\texttt{Quant}, q_{max}) + C(\texttt{IQuant}, q_{max}) + C(\texttt{IDCT}, q_{max}) = 170$. This demonstrates that $t_p(a_1..a_5, q_{max}) \leq D(a_i) - C(a_1..a_i, q_{max}) \leq 180 - 170 = 10$.

## 2.2 Quality control problem under uncertainty

### *2.2.1 The problem*

Execution times for actions may considerably vary over time as they depend on data contents. Furthermore, non predictability of the underlying platform is an additional factor of uncertainty. We consider the quality control problem for unknown but bounded execution times.

**Definition 2.8** A **parameterized system** is a tuple $PSY(C) = (G, Q, C^{wc}, D, C)$ where:

- $G$ is a precedence graph.
- $Q = [q_{min}, q_{max}]$ is a finite interval of integers corresponding to *quality levels*.
- $C^{wc} : A \times Q \to \mathbb{R}^+$ is a function giving for an action $a$ and quality level $q$ its *worst-case* execution time $C^{wc}(a, q)$. We assume that, for all $a \in A$, $q \mapsto C^{wc}(a, q)$ is a non-decreasing function.
- $D : A \to \mathbb{R}^+ \cup \{+\infty\}$ is a function giving for an action $a$, its *deadline* $D(a)$.
- The parameter $C : A \times Q \to \mathbb{R}^+$ is a function giving for action $a$ and quality level $q$ its *actual* execution time $C(a, q)$. We assume that, for all $a \in A$, $q \mapsto C(a, q)$ is a non-decreasing function such that $C \leq C^{wc}$.

(Semantics) The parameterized system $PSY(C)$ defines a family of transition systems $(S \times \mathbb{R}^+, A \times Q, \longrightarrow)$ depending on the parameter $C$:

- states are given by pairs $(s_i, t_i)$ where $s_i$ is a state of $G$ and $t_i \in \mathbb{R}^+$ is a value of time; we take $t_0 = 0$ for $s_0 = \emptyset$;

- for two states $(s_i, t_i)$ and $(s_j, t_j)$, an action $a$ and a quality level $q$, we have $(s_i, t_i) \xrightarrow{a,q} (s_j, t_j)$, if $s_i \xrightarrow{a} s_j$ in $G$ and $t_j - t_i = C(a, q)$.

The definition of a controller for a parameterized system is similar to the one given for known execution times (Definition 2.3)

**Definition 2.9** Given a parameterized system $PSY(C)$, a **controller** is a function $\Gamma : S \times \mathbb{R}^+ \to A \times Q$ giving, for a state $(s_{i-1}, t_{i-1})$ of $PSY(C)$, an action $a_i$ and its quality level $q_i$ such that there exists $(s_i, t_i)$ and $(s_{i-1}, t_{i-1}) \xrightarrow{a_i,q_i} (s_i, t_i)$.

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

The quality control problem for a given parameterized system $PSY(C) = (G, Q, C^{wc}, D, C)$ consists in finding a controller $\Gamma$ such that the controlled system respects the deadlines while keeping quality maximal and smooth. It is formalized as follows.

**Definition 2.10** (Quality control problem under uncertainty) Given a parameterized system $PSY(C)$ find a controller $\Gamma$ such that for any actual time function $C \le C^{wc}$:

- **(Safety)** $\Gamma$ is *safe* (deadlines are met), that is, for all state $(s_i, t_i)$ of $PS(C)\|\Gamma$ we have $D(a_i) \ge t_i$.
- **(Optimality)** The overall execution time is maximal, that is, for any safe controller $\Gamma'$, $t_n \ge t'_n$, where $t_n$ (resp. $t'_n$) is the completion time of the last action in $PS(C)\|\Gamma$ (resp. $PS(C)\|\Gamma'$).

An additional optimality requirement is *smoothness* for the quality chosen by the controller. Informally, smoothness means low deviation of the quality levels with respect to the average quality. We do not formalize this property which is essential for most multimedia applications (Schuster et al. 1999; Westerink et al. 1999).

### 2.2.2 *Controller design*

Due to uncertainty on execution times, the computation of adequate schedules and their associated quality assignments is made online. We adapt the algorithm proposed in Sect. 2.1.2 (Proposition 2.2). To cope with state explosion problem, the algorithm considers at each state constant quality assignments for the remaining actions.

The proposed algorithm is parameterized by a policy $X$ characterized by a function $C^X$ giving for a sequence of actions $a_1..a_n$ and a quality level $q$ an estimate $C^X(a_1..a_n, q)$ of the execution time of $a_1..a_n$ for the quality level $q$. The algorithm uses appropriate approximations $t_p^X$ of $t_p$, and $Best\_Sched^X$ of $Best\_Sched$, defined from the $X$-execution time function $C^X$ as follows.

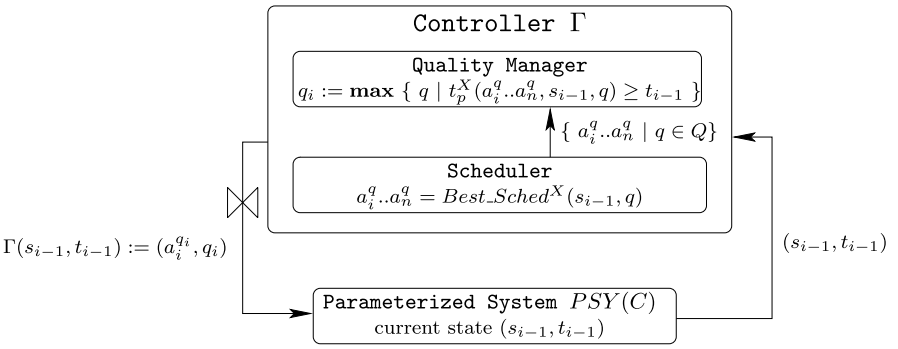



**Fig. 5** Controller architecture

**Definition 2.11** ($X$-policy function $t_p^X$) Let $PSY(C) = (G, Q, C^{wc}, D, C)$ be a parameterized system and $C^X$ be a function giving for a sequence of actions $a_i..a_n$ and a quality level $q$, an estimate $C^X(a_i..a_n, q)$ of the execution time of $a_i..a_n$ at the quality level $q$. Given a state $s_{i-1}$ of $G$, a schedule $a_i..a_n$ from $s_{i-1}$, and a quality level $q$, we define the **$X$-policy function** $t_p^X$ associated to the $X$-execution time function $C^X$ as follows:

$$t_p^X(a_i..a_n, s_{i-1}, q) = \min_{i \leq k \leq n} D(a_k) - C^X(a_i..a_k, q).$$

**Definition 2.12** (Optimal scheduler $Best\_Sched^X$) For a parameterized system $PSY(C) = (G, Q, C^{wc}, D, C)$ and a $X$-policy function $t_p^X$, an **optimal scheduler** $Best\_Sched^X$ is a function giving, for any state $s_{i-1}$ of $G$ and for any quality level $q$, a schedule $a_i^q..a_n^q = Best\_Sched^X(s_{i-1}, q)$ from $s_{i-1}$ such that $a_i^q..a_n^q$ maximizes the $X$-policy function $t_p^X$, that is:

$$t_p^X(a_i^q..a_n^q, s_{i-1}, q) = \mathbf{max}\{t_p^X(a_i..a_n, s_{i-1}, q) \mid a_i..a_n \in \Sigma(G, s_{i-1})\}.$$

Figure 5 shows interaction between the Controller $\Gamma$, and the parameterized system $PSY(C)$ representing an application software running on a platform. The Controller monitors the current state $(s_{i-1}, t_{i-1})$ of $PSY(C)$ and computes the next action $a_i$ and its corresponding quality level $q_i$, as specified by the following algorithm which generalizes the one given in Proposition 2.2.

$\Gamma(s_{i-1}, t_{i-1})$ {
  **for all** $q \in Q$ **do** $a_i^q..a_n^q := Best\_Sched^X(s_{i-1}, q)$ **od**
  $q_i = \mathbf{max}\{q \mid t_p^X(a_i^q..a_n^q, s_{i-1}, q) \geq t_{i-1}\}$
  **return** $\Gamma(s_{i-1}, t_{i-1}) := (a_i^{q_i}, q_i)$.
}

The Controller is composed of an optimal scheduler $Best\_Sched^X$, and of a the Quality Manager, such that for a given state $(s_{i-1}, t_{i-1})$:

- The Scheduler computes for each quality level $q \in Q$, a schedule from state $s_{i-1}$, $a_i^q..a_n^q = Best\_Sched^X(s_{i-1}, q)$.
- The Quality Manager computes the maximal quality level $q_i$ meeting the $X$-quality management policy, that is:

$$q_i = \mathbf{max}\{q \mid t_p^X(a_i^q..a_n^q, s_{i-1}, q) \geq t_{i-1}\}.$$

The function $t_p^X : A \times S \times Q \to \mathbb{R}^+$ characterizes the *X-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 $t_p^X(a_i^q..a_n^q, s_{i-1}, q)$ at state $s_{i-1}$ if the rest of the actions $a_i^q..a_n^q$ is executed with constant quality $q$. If the inequality $t_p^X(a_i^q..a_n^q, s_{i-1}, q) \geq t_{i-1}$ 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 $t_p^X(a_i^q..a_n^q, s_{i-1}, q) \geq t_{i-1}$. The maximization of the quality level is done for the optimality criterion, that is, maximizing the time budget utilization.

Section 3 deals with the definition of policy functions $t_p^X$ ensuring safety and optimality of the chosen quality levels. Finding an optimal scheduler $Best\_Sched^X$ is a non trivial problem discussed in Sect. 4. We propose heuristics for the online computation of the schedules. The interest of the proposed policy function $t_p^X$ and optimal scheduler $Best\_Sched^X$ is shown through both theoretical and experimental results in the rest of the paper.

## 3 Quality manager design

### 3.1 Quality management policies

This section deals with the definition of an adequate $X$-policy function $t_p^X$ that ensures safety and optimality. Safety means that no deadline is violated. Optimality means maximization of the time budget utilization and smoothness of the quality levels chosen by the Quality Manager.

Let $PSY(C) = (G, Q, C^{wc}, D, C)$ be a parameterized system. As schedules are computed and provided to the Quality Manager by the Scheduler, we consider, without loss of generality, quality management policies for a fixed schedule $a_1..a_n$. Let $a_1..a_n$ be the planed schedule for $PSY(C)$. For simplified notation, we will write $t_p^X(s_{i-1}, q)$ instead of $t_p^X(a_i..a_n, s_{i-1}, q)$.

#### *3.1.1 Safe quality management policy*

**Definition 3.1** We introduce the *safe* policy function $t_p^{sf}$ corresponding to the *safe* execution time function $C^{sf} : A^+ \times Q \to \mathbb{R}^+$ defined by:

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

where $C^{wc}(a_{i+1}..a_k, q_{min})$ denotes the total execution time of the sequence $a_{i+1}..a_k$, that is,

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

As the quality level may 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. Thus, $t_p^{sf}(s_{i-1}, q) \ge t_{i-1}$ guarantees that the first action $a_i$ meets its deadline when executed with quality $q$, and the rest of the actions of the schedule meet their deadline when executed with quality $q_{min}$, for the worst-case assumption.

**Proposition 3.1** (Safety) *Given a parameterized system* $PSY(C) = (G, Q, C^{wc}, D, C)$ *such that deadlines are met for the minimal quality level and the worst-case execution times, that is,*

$$\forall k \in \{1, \ldots, n\} \quad D(a_k) \ge C^{wc}(a_1..a_k, q_{min}).$$

*Then, the controller* $\Gamma$ *applying quality management policy* $t_p^{sf} \ge t$ *is safe.*

The above proposition also holds when $t_p^{sf}$ is replaced by any policy function $t_p^X$ such that $t_p^{sf} \ge t_p^X$.

**Lemma 3.1** *For any controller* $\Gamma$ *satisfying the assumptions of Proposition 3.1, we have* $q_{min} \in \{q \mid t_p^{sf}(s_{i-1}, q) \ge t_{i-1}\}$ *at any reachable state* $(s_{i-1}, t_{i-1})$ *of* $PSY(C) \| \Gamma$.

*Proof* The proof is made by induction of $i \in \{0, \ldots, n-1\}$. Let $\mathcal{P}(i)$ be the assertion $q_{min} \in \{q \mid t_p^{sf}(s_i, q) \ge t_i\}$.

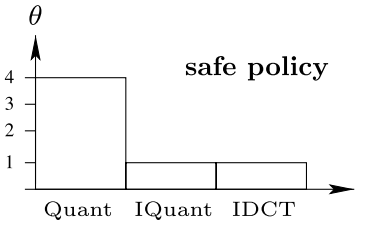
• $\mathcal{P}(0)$ : We have:

$$t_p^{sf}(s_0, q_{min}) = \min_{1 \le k \le n} D(a_k) - C^{wc}(a_1..a_k, q_{min}). \tag{3}$$

As we have for all $k \in \{1, \ldots, n\}$, $D(a_k) \ge C^{wc}(a_1..a_k, q_{min})$, we conclude from (3) that $t_p^{sf}(s_0, q_{min}) \ge 0 = t_0$, that is, $\mathcal{P}(0)$.

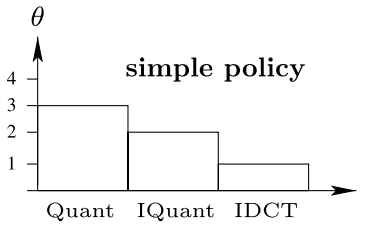
• $\mathcal{P}(i) \Rightarrow \mathcal{P}(i+1)$ : Assume $\mathcal{P}(i)$, that is, $t_p^{sf}(s_i, q_{min}) \ge t_i$. Consider $(a, q) = (a_{i+1}, q_{i+1}) = \Gamma(s_i, t_i)$. As $\Gamma$ apply the quality management policy $t_p^{sf} \ge t$ and $t_p^{sf}(s_i, q_{min}) \ge t_i$, we have $t^{sf}(s_i, q) \ge t_i$. We have:

$$\begin{aligned}
&\min_{i+1 \le k \le n} D(a_k) - C^{sf}(a_{i+1}..a_k, q) \ge t_i \\
&\quad \Rightarrow \quad \min_{i+2 \le k \le n} D(a_k) - C^{wc}(a_{i+1}, q) - C^{wc}(a_{i+2}..a_k, q_{min}) \ge t_i \\
&\quad \Rightarrow \quad \min_{i+2 \le k \le n} D(a_k) - C^{wc}(a_{i+2}..a_k, q_{min}) \ge t_i + C^{wc}(a, q).
\end{aligned}$$




Execution times

| $q$ | $C^{wc}(a,q)$ | $C(a,q) = C^{av}(a,q)$ |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 5 | 2 |
| 3 | 6 | 3 |
| 4 | 7 | 5 |



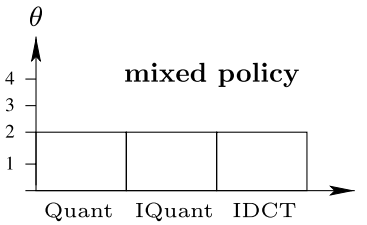




**Fig. 6** Comparison between different policies

As $q$ is the chosen quality level for the action $a$, we have $t_i + C^{wc}(a,q) \geq t_{i+1}$. We obtain $\mathbf{min}_{i+2\leq k\leq n} D(a_k) - C^{wc}(a_{i+2}..a_k, q_{min}) \geq t_{i+1}$, that is, $t_p^{sf}(s_{i+1}, q_{min}) \geq t_{i+1}$. This demonstrates $\mathcal{P}(i+1)$. □

*Proof of Proposition 3.1* By the Lemma 3.1, we have $q_{min} \in \{q \mid t_p^{sf}(s_{i-1}, q) \geq t_{i-1}\}$ at any state of $PSY(C)\|\Gamma$, that is, $\{q \mid t_p^{sf}(s_{i-1}, q) \geq t_{i-1}\}$ is a non-empty set. Let $(a,q)$ be $(a,q) = (a_i, q_i) = \Gamma(s_{i-1}, t_{i-1})$. Then we have $t_p^{sf}(s_{i-1}, q) \geq t_{i-1}$ and the action $a$ meets its deadline:

$$t_p^{sf}(s_{i-1}, q) = \min_{i\leq k\leq n} D(a_k) - C^{sf}(a_i..a_k, q) \geq t_{i-1}$$
$$\Rightarrow \quad D(a_i) - C^{wc}(a,q) \geq t_{i-1}$$
$$\Rightarrow \quad D(a_i) \geq t_{i-1} + C^{wc}(a,q) \geq t_i.$$

This demonstrates that any action $a$ meets its deadline. □

The *safe* quality management policy ensures that all action deadlines are met. Nonetheless, by considering the minimal quality for the last actions using the safe quality management policy can lead to variation of the quality levels before a critical deadline (see Example 3.1 and Fig. 6).

We propose two other quality management policies leading to smoother quality levels. Their influence is shown through an example and confirmed by experimental results in Sect. 5.3.

### *3.1.2 Simple quality management policy*

The following example illustrates non smoothness of the quality levels by using the safe quality management policy.

*Example 3.1* Consider a $PSY(C)$ with this three actions {`Quant`, `IQuant`, `IDCT`}, $Q = \{1, \ldots, 4\}$, and a single deadline $D = 9$ (for all $i$, $D(a_i) = 9$). Assume that $PSY(C)$ has a schedule `Quant IQuant IDCT` such that the actual and the worst-case execution times are given in the table of Fig. 6, where $a \in$ {`Quant`, `IQuant`, `IDCT`}. The computed quality assignment for the schedule by using the safe policy $t_p^{sf} \geq t$ is not smooth (Fig. 6).

We can improve smoothness by combining worst-case and average behavior. Simple quality management policy defined bellow can be used to improve the smoothness of the computed quality assignment. It uses *average* execution time function $C^{av} : A \times Q \to \mathbb{R}^+$. These execution times can be estimated by static analysis and/or profiling techniques. We denote by $t_p^{av}$ the corresponding policy function.

**Definition 3.2** Given a parameterized system $PSY(C) = (G, Q, C^{wc}, D, C)$ and an *average* execution time function $C^{av} : A \times Q \to \mathbb{R}^+$, the *simple* policy function $t_p^{sp}$ corresponds to the *simple* execution time function $C^{sp} : A^+ \times Q \to \mathbb{R}^+$ defined by:

$$C^{sp} = \mathbf{max}\{C^{sf}, C^{av}\}.$$

**Proposition 3.2** *The* simple *policy function* $t_p^{sp}$ *satisfies* $t_p^{sp} = \mathbf{min}\{t_p^{sf}, t_p^{av}\}$.

*Proof of Proposition 3.1* For all $i \in \{0, \ldots, n-1\}$ and $q \in Q$, we have:

$$\begin{aligned}
t_p^{sp}(s_{i-1}, q) &= \min_{i \leq k \leq n} D(a_k) - C^{sp}(a_i..a_k, q) \\
&= \min_{i \leq k \leq n} D(a_k) - \mathbf{max}\{C^{av}(a_i..a_k, q), C^{sf}(a_i..a_k, q)\} \\
&= \min_{i \leq k \leq n} \mathbf{min}\{D(a_k) - C^{av}(a_i..a_k, q), D(a_k) - C^{sf}(a_i..a_k, q)\} \\
&= \mathbf{min}\Big\{ \min_{i \leq k \leq n} (D(a_k) - C^{av}(a_k, q)), \Big( \min_{i \leq k \leq n} D(a_k) - C^{sf}(a_i..a_k, q) \Big) \Big\} \\
t_p^{sp}(s_{i-1}, q) &= \mathbf{min}\{t_p^{av}(s_{i-1}, q), t_p^{sf}(s_{i-1}, q)\}.
\end{aligned}$$

□

Notice that using $t_p^{sp}$ also leads to feasible schedules. For the previous example, the schedule computed by using $t_p^{sp} \geq t$ is smoother than the one computed by using $t_p^{sf} \geq t$ (see Fig. 6).

### *3.1.3 Mixed quality management policy*

The simple quality management policy may also lead to non smoothness of the quality before a critical deadline. Even if actual time follows exactly average time (i.e. $C = C^{av}$), the quality may need to be decreased along a sequence of actions where $C^{sf} > C^{av}$ (see Example 3.1 and Fig. 6). We propose the *mixed* quality management policy which is robust with respect to this phenomena. The mixed execution function $C^{mx}$ combines the use of two execution functions $C^{sf}$ and $C^{av}$. The second is used to enhance smoothness of quality levels.

**Definition 3.3** We introduce $\delta^{max}$ as the maximum difference between the worst-case and the average behavior, that is,

$$\delta^{max}(a_i..a_k, q) = \max_{i \le j \le k} \delta(a_j..a_k, q),$$

where $\delta(a_j..a_k, q) = C^{sf}(a_j..a_k, q) - C^{av}(a_j..a_k, q)$.

Notice that, for a sequence of actions $a_i..a_k$ and a quality level $q$, $\delta^{max}(a_i..a_k, q)$ is a kind of safety margin we need to keep with respect to the average behavior in order to meet the deadlines. It is due to uncertainty on execution times.

**Definition 3.4** The *mixed* policy function $t_p^{mx}$ corresponds to the *mixed* execution time function $C^{mx} : A^+ \times Q \to \mathbb{R}^+$ defined by:

$$C^{mx} = C^{av} + \delta^{max}.$$

The mixed execution time jointly takes into account average and worst-case behavior. For the example given in Fig. 6, the schedule computed by using the mixed quality management policy $t_p^{mx} \ge t$ is the smoothest one ($\theta$ is constant). In the rest of the paper, we consider a Quality Manager applying the mixed quality management policy, that is, $C^X = C^{mx}$.

## 3.2 Speed diagrams and their use for quality management

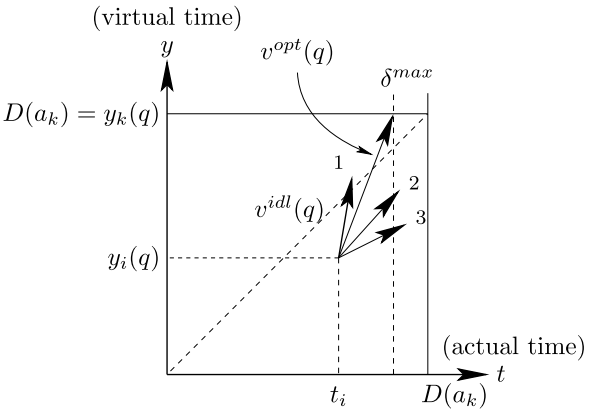
Speed diagrams are a graphical representation of system's states, in which quality management policies have a geometric interpretation in terms of relative speed between a notion of virtual time and actual time. They allow a better understanding of the impact of worst-case execution times on achieving 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.

### *3.2.1 Definition*

Speed diagrams represent in a two-dimensional space the evolution of a parameterized system $PSY(C) = (G, Q, C^{wc}, D, C)$ and its Quality Manager applying the mixed quality management policy defined in Sect. 3.1.3 (Fig. 7). One dimension represents virtual time computed from average execution times and their deadlines, while the other 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.

**Fig. 7** Speed diagram




*System state representation* Let $(s_i, t_i)$ be a state of a parameterized system $PSY(C) = (G, Q, C^{wc}, D, C)$ 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 the time distance from the deadline $D(a_k)$ after the execution of the action $a_i$ if the sequence of the actions $a_1, \ldots, a_k$ 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 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 Fig. 7).

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.

*Ideal and optimal speeds* Let $(s_i, t_i)$ and $(s_j, t_j)$ be two states of $PSY(C) = (G, Q, C^{wc}, D, C)$ 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 $q_{i+1} = \cdots = q_j = q$, 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 3.3** *Given a parameterized system $PSY(C) = (G, Q, C^{wc}, D, C)$, a state $(s_i, t_i)$ of $PSY(C)$, a quality level $q$ and a target deadline $D(a_k)$, $k > i$, we have*:

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

*Proof*

$$\begin{aligned}
& v^{idl}(q) \geq v^{opt}(q) \\
\iff\ & \frac{D(a_k)}{C^{av}(a_1..a_k, q)} \geq \frac{D(a_k)}{C^{av}(a_1..a_k, q)} \cdot \frac{C^{av}(a_{i+1}..a_k, q)}{D(a_k) - \delta^{max}(a_{i+1}..a_k, q) - t_i} \\
\iff\ & 1 \geq \frac{C^{av}(a_{i+1}..a_k, q)}{D(a_k) - \delta^{max}(a_{i+1}..a_k, q) - t_i} \\
\iff\ & D(a_k) - \delta^{max}(a_{i+1}..a_k, q) - t_i \geq C^{av}(a_{i+1}..a_k, q) \\
\iff\ & D(a_k) - C^{mx}(a_{i+1}..a_k, q) \geq t_i.
\end{aligned}$$

□

The above proposition 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 ideal 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.

### 3.2.2 Quality regions

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 this section, 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.

Consider a parameterized system $PSY(C) = (G, Q, C^{wc}, D, C)$ and a Quality Manager 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 3.5** Given a parameterized system $PSY(C) = (G, Q, C^{wc}, D, C)$ and a Controller $\Gamma$, a **quality region** $\mathcal{R}_q$ for the quality level $q$ is a subset of states $(s_i, t_i)$ of $PSY(C)$ defined by:

$$\mathcal{R}_q = \big\{(s_i, t_i) \mid \Gamma(s_i, t_i) = (a_{i+1}, q)\big\}.$$

Let $(s_i, t_i)$ be a state of a parameterized system $PSY(C) = (G, Q, C^{wc}, D, C)$, 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 $t_p^{mx}$ is a non-increasing function of $q$. This implies that

- for $q < q_{max} = \mathbf{max}\, Q$, $\Gamma(s_i, t_i) = (a_{i+1}, q)$ iff $t_p^{mx}(s_i, q) \geq t_i$ and $t_i > t_p^{mx}(s_i, q+1)$.
- for $q = q_{max}$, $\Gamma(s_i, t_i) = (a_{i+1}, q)$ iff $t_p^{mx}(s_i, q) \geq t_i$. This leads to the following proposition.

**Proposition 3.4** *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*

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

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 Fig. 8).

### 3.2.3 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 $PSY(C) = (G, Q, C^{wc}, D, C)$. Assume that the Quality Manager $\Gamma$ 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 $q$.

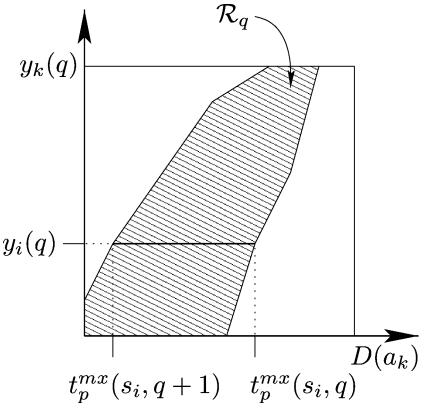



**Fig. 8** Quality region for a quality level $q$

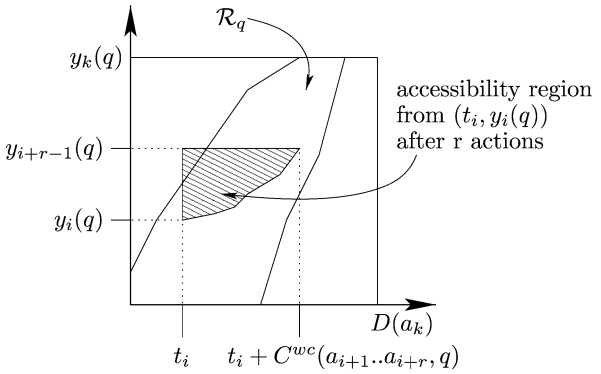



**Fig. 9** Control relaxation: the principle

**Definition 3.6** Given a parameterized system $PSY(C) = (G, Q, C^{wc}, D, C)$ and a Quality Manager $\Gamma$, a **control relaxation region** $\mathcal{R}_q^r$ for the quality level $q$ and an integer $r \geq 1$ is defined by:

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

We consider the states $(s_j, t_j)$, $j \in \{i, i+1, \ldots, i+r-1\}$ of $PSY(C) = (G, Q, C^{wc}, D, C) \| \Gamma$, and find conditions for these states to be in $\mathcal{R}_q$ (see Fig. 9). For instance, Fig. 9 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 $t_j$:

$$t_i + C^{wc}(a_{i+1}..a_j, q) \geq t_j \geq t_i. \tag{4}$$

By Proposition 3.4 and (4), $(s_j, t_j) \in \mathcal{R}_q$ if the following equations are satisfied for all $j \in \{i, i+1, \ldots, i+r-1\}$,

$$t_p^{mx}(s_j, q) - C^{wc}(a_{i+1}..a_j, q) \geq t_i, \tag{5}$$

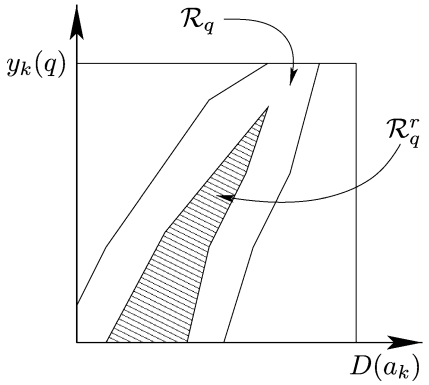



**Fig. 10** Control relaxation region

$$t_i > t_p^{mx}(s_j, q+1). \tag{6}$$

then, we can relax the Quality Manager for $r$ steps. As $t_p^{mx}(s_j, q+1)$ is increasing with $j$, (6) is satisfied for all $j$ if and only if $t_i > t_p^{mx}(s_{i+r-1}, q+1)$. This leads to the following proposition.

**Proposition 3.5** *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*

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

*where* $t_p^{mx,r}(s_i, q) = \mathbf{min}_{i \leq j \leq i+r-1}\, t_p^{mx}(s_j, q) - C^{wc}(a_{i+1}..a_j, q)$.

## 4 Scheduler design for mixed quality management policy

This section provides results for computing the Scheduler of a Controller based on the mixed quality management policy $t_p^{mx} \geq t$. The problem is to find an optimal scheduler $Best\_Sched^{mx}$, that is, which maximizes the policy function $t_p^{mx}$. Given a parameterized system $PSY(C) = (G, Q, C^{wc}, D, C)$, a state $s_{i-1}$ of $G$ and a quality level $q$, we seek for a schedule $a_i^q..a_n^q = Best\_Sched^{mx}(s_{i-1}, q)$ of the remaining actions that satisfies:

$$t_p^{mx}(a_i^q..a_n^q, s_{i-1}, q) = \mathbf{max}\{t_p^{mx}(a_i..a_n, s_{i-1}, q) \mid a_i..a_n \in \Sigma(G, s_{i-1})\}.$$

Without loss of generality, we study the problem for the initial state $s_0 = \emptyset$ and for a given quality level $q$. To simplify notation we will write $t_p^{mx}(a_1..a_n, q)$ for $t_p^{mx}(a_1..a_n, s_0, q)$.

### 4.1 Computing $Best\_Sched^{mx}$ for a single deadline

We first consider the case where the function $D$ is constant (single deadline). Let $a_1..a_n$ be a schedule, we have:

$$t_p^{mx}(a_1..a_n, q) = \max_{1\le k\le n} D(a_k) - C^{av}(a_1..a_k, q) - \delta^{max}(a_1..a_k, q).$$

Since the average execution time $C^{av}(a_1..a_n, q)$ is the constant $\sum_{a\in A} C^{av}(a, q)$, $t_p^{mx}(a_1..a_n, q)$ is maximal if and only if $\delta^{max}(a_1..a_n, q)$ is minimal. Then, the problem is to find a schedule $a_1..a_n$ such that for all schedule $a'_1..a'_n$ we have $\delta^{max}(a_1..a_n, q) \le \delta^{max}(a'_1..a'_n, q)$.

**Definition 4.1** We define the functions $\eta : A \times Q \to \mathbb{R}^+$ and $\beta : A \times Q \to \mathbb{R}$ as follows:

$$\eta(a, q) = C^{wc}(a, q) - C^{av}(a, q),$$
$$\beta(a, q) = C^{wc}(a, q_{min}) - C^{av}(a, q).$$

For an action $a$ and a quality level $q$, $\eta(a, q)$ is the difference between the worst-case and the average execution time. The value $\eta$ can be considered as the uncertainty for the execution time of the action $a$ for quality $q$. The value $\beta(a, q)$ is the difference between the worst-case execution time for the action $a$ at the minimal quality level $q_{min}$, and the average execution time for the actions at the quality level $q$. It is related to the "fall-back" capability of $a$ for quality $q$: for small values of $\beta$, in particular negative values, the controller can speed up the application by selecting the minimal quality level, even if we consider the worst-case assumption (i.e. $C = C^{wc}$). Then, we write $\delta$ as follows:

$$\delta(a_1..a_n, q) = \eta(a_1, q) + \beta(a_2..a_n, q)$$

where $\beta(a_2..a_n, q) = \beta(a_2, q) + \cdots + \beta(a_n, q)$.

**Proposition 4.1** (Minimizing $\delta^{max}$) *Given a schedule $a_1..a_n$ and a quality level $q$, consider two consecutive and independent actions $a_i$ and $a_{i+1}$ ($a_i \not\prec a_{i+1}$ and $a_{i+1} \not\prec a_i$). Let $a'_1..a'_n$ be the schedule in which $a_i$ and $a_{i+1}$ are swapped, that is, $a'_1..a'_n = a_1..a_{i-1}a_{i+1}a_i a_{i+2}..a_n$. Then, we have $\delta^{max}(a'_1..a'_n, q) \le \delta^{max}(a_1..a_n, q)$ in the following situations*:

R1: $\eta(a_i, q) \le \eta(a_{i+1}, q)$ *and* $\beta(a_i, q) \le 0$
R2: $\beta(a_i, q) \le 0$ *and* $\beta(a_{i+1}, q) \ge 0$.
R3: $(\eta - \beta)(a_i, q) \ge (\eta - \beta)(a_{i+1}, q)$, $\beta(a_i, q) \ge 0$, *and* $\beta(a_{i+1}, q) \ge 0$.

*Proof* By definition we have $\delta^{max}(a_1..a_n, q) = \mathbf{max}_{1\le j\le n}\, \delta(a_j..a_n, q)$ and $\delta^{max}(a'_1..a'_n, q) = \mathbf{max}_{1\le j\le n}\, \delta(a'_j..a'_n, q)$. We compare values $\delta(a_j..a_n, q)$ and $\delta(a'_j..a'_n, q)$. Let $I_1 = \{i, i+1\}$ and $I_2 = \{1, \ldots, i-1, i+2, \ldots, n\}$.

• For all $j \in I_2$, we have:

$$\begin{aligned}\delta(a_j..a_n, q) &= \eta(a_j, q) + \beta(a_{j+1}..a_n, q) \\ &= \eta(a_j, q) + \sum_{k=j+1}^{n} \beta(a_k, q).\end{aligned}$$

Since $\{a_{j+1}, \ldots, a_n\} = \{a'_{j+1}, \ldots, a'_n\}$, and $a_j = a'_j$ we obtain $\delta(a_j..a_n, q) = \delta(a'_j..a'_n, q)$.

• For $j \in I_1$, we have:

$$\delta(a_i..a_n, q) = \eta(a_i, q) + \beta(a_{i+1}, q) + \beta(a_{i+2}..a_n, q), \tag{7}$$

$$\delta(a_{i+1}..a_n, q) = \eta(a_{i+1}, q) + \beta(a_{i+2}..a_n, q). \tag{8}$$

Since $\{a_{i+2}, \ldots, a_n\} = \{a'_{i+2}, \ldots, a'_n)\}$,we have:

$$\delta(a'_i..a'_n, q) = \eta(a_{i+1}, q) + \beta(a_i, q) + \beta(a_{i+2}..a_n, q), \tag{9}$$

$$\delta(a'_{i+1}..a'_n, q) = \eta(a_i, q) + \beta(a_{i+2}..a_n, q). \tag{10}$$

Since $\delta^{max}(a_1..a_n, q) = \mathbf{max}_{j\in I_1\cup I_2}\, \delta(a_j..a_n, q)$ and $\delta^{max}(a'_1..a'_n, q) = \mathbf{max}_{j\in I_1\cup I_2}\, \delta(a'_j..a'_n, q)$, it is sufficient to show that $\mathbf{max}_{j\in I_1}\, \delta(a'_j..a'_n, q) \leq \mathbf{max}_{j\in I_1}\, \delta(a_j..a_n, q)$ in order to ensure that $\delta^{max}(a'_1..a'_n, q) \leq \delta^{max}(a_1..a_n, q)$. The following results come from the definitions:

$$\text{from (8) and (9)}, \quad \beta(a_i, q) \leq 0 \quad \Rightarrow \quad \delta(a'_i..a'_n, q) \leq \delta(a_{i+1}..a_n, q), \tag{11}$$

$$\text{from (7) and (10)}, \quad \beta(a_{i+1}, q) \geq 0 \quad \Rightarrow \quad \delta(a'_{i+1}..a'_n, q) \leq \delta(a_i..a_n, q), \tag{12}$$

$$\text{from (8) and (10)}, \quad \eta(a_i, q) \leq \eta(a_{i+1}, q) \quad \Rightarrow \quad \delta(a'_{i+1}..a'_n, q) \leq \delta(a_{i+1}..a_n, q) \tag{13}$$

R1: Suppose that $a_i$ and $a_{i+1}$ are such that $\beta(a_i, q) \leq 0$ and $\eta(a_i, q) \leq \eta(a_{i+1}, q)$. Since $\beta(a_i, q) \leq 0$, we have $\delta(a'_i..a'_n, q) \leq \delta(a_{i+1}..a_n, q)$ (implication (11)) and thus $\delta(a'_i..a'_n, q) \leq \mathbf{max}_{j\in I_1}\, \delta(a_j..a_n, q)$.

Since $\eta(a_i, q) \leq \eta(a_{i+1}, q)$, we have $\delta(a'_{i+1}..a'_n, q) \leq \delta(a_{i+1}..a_n, q)$ (implication (13)) and thus $\delta(a'_{i+1}..a'_n, q) \leq \mathbf{max}_{j\in I_1}\, \delta(a_j..a_n, q)$. We can conclude that $\delta^{max}(a'_1..a'_n, q) \leq \delta^{max}(a_1..a_n, q)$.

R2: Since $\beta(a_i, q) \leq 0$, we have $\delta(a'_i..a_n, q) \leq \delta(a_{i+1}..a_n, q) \leq \mathbf{max}_{j\in I_1}\, \delta(a_j..a_n, q)$ (implication (11)). Since $\beta(a_{i+1}, q) \geq 0$, we have $\delta(a'_{i+1}..a_n, q) \leq \delta(a_i..a_n, q) \leq \mathbf{max}_{j\in I_1}\, \delta(a_j..a_n, q)$ (implication (12)). We obtain $\mathbf{max}_{j\in I_1}\, \delta(a'_j..a_n, q) \leq \mathbf{max}_{j\in I_1}\, \delta(a_j..a_n, q)$. Thus we conclude that, in this case, we have $\delta^{max}(a'_1..a'_n, q) \leq \delta^{max}(a_1..a_n, q)$.

R3: Suppose that $a_i$ and $a_{i+1}$ are such that $\beta(a_i, q) \geq 0$, $\beta(a_{i+1}, q) \geq 0$ and $(\eta - \beta)(a_i, q) \geq (\eta - \beta)(a_{i+1}, q)$. Since $\beta(a_{i+1}, q) \geq 0$, we have $\delta(a'_{i+1}..a_n, q) \leq$

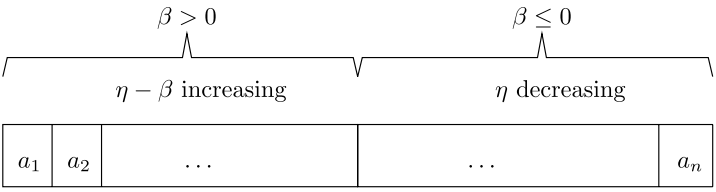



**Fig. 11** Minimization of $\delta^{max}$

$\delta(a_i..a_n, q) \leq \mathbf{max}_{j \in I_1} \delta(a_j..a_n, q)$ (implication (12)). Since $(\eta - \beta)(a_i, q) \geq (\eta - \beta)(a_{i+1}, q)$ we have:

$$\begin{aligned}\delta(a'_i..a_n, q) - \delta(a_i..a_n, q) &= \eta(a_{i+1}, q) + \beta(a_i, q) - (\eta(a_i, q) + \beta(a_{i+1}, q)) \\ &= (\eta - \beta)(a_{i+1}, q) - (\eta - \beta)(a_i, q), \\ \delta(a'_i..a_n, q) - \delta(a_i..a_n, q) &\leq 0.\end{aligned}$$

Thus, we obtain $\delta(a'_i..a_n, q) \leq \delta(a_i..a_n, q) \leq \mathbf{max}_{j \in I_1} \delta(a_j..a_n, q)$. We obtain also $\mathbf{max}_{j \in I_1} \delta(a'_j..a_n, q) \leq \mathbf{max}_{j \in I_1} \delta(a_j..a_n, q)$. We can conclude that, in this case, we have $\delta^{max}(a'_1..a'_n, q) \leq \delta^{max}(a_1..a_n, q)$. □

As we consider a constant deadline function $D$, the conditions R1, R2, R3 define rules for getting, from a schedule $a_1..a_n$, a schedule $a'_1..a'_n$ such that $t_p^{mx}(a'_1..a'_n, q) \geq t_p^{mx}(a_1..a_n, q)$. The following proposition allows the computation of an optimal schedule $a_1..a_n$ for a given quality level $q$ when there is no precedence constraint, that is, $G = (A, \emptyset)$. This schedule is obtained from a partition of the actions according to the value of the function $\beta$:

- actions for which $\beta > 0$ are scheduled first and by $(\eta - \beta)$ increasing
- actions for which $\beta \leq 0$ are scheduled at the end of the schedule and by $\eta$ decreasing. This is summarized in Fig. 11.

**Proposition 4.2** (No precedence constraint) *Let $PSY(C) = (G, Q, C^{wc}, D, C)$ be a parameterized system such that $G = (A, \emptyset)$. If there exists a schedule $a_1..a_n$ and an index $i$ such that*:

1. *For all $j \in \{1, \ldots, i\}$ we have $\beta(a_j, q) > 0$, and for all $j \in \{i+1, \ldots, n\}$ we have $\beta(a_j, q) \leq 0$.*
2. *For all $j_1, j_2 \in \{1, \ldots, i\}$, $j_1 < j_2 \Rightarrow (\eta - \beta)(a_{j_1}, q) \leq (\eta - \beta)(a_{j_2}, q)$.*
3. *For all $j_1, j_2 \in \{i+1, \ldots, n\}$, $j_1 < j_2 \Rightarrow \eta(a_{j_1}, q) \geq \eta(a_{j_2}, q)$.*

*Then the schedule $a_1..a_n$ minimizes $\delta^{max}$, that is, for all schedule $a'_1..a'_n$ we have $\delta^{max}(a_1..a_n, q) \leq \delta^{max}(a'_1..a'_n, q)$.*

*Proof* Let $a'_1..a'_n$ a schedule. Since there is no precedence constraint, we can obtain $a_1..a_n$ from $a'_1..a'_n$ by using the rules R1, R2, R3 of Proposition 4.1. Starting from $a'_1..a'_n$, applying the rule R2, we first obtain the partition $\beta > 0$ and $\beta \leq 0$. Then we apply successively the rule R1 to obtain $\eta$ decreasing for actions with $\beta \leq 0$ and

the rule R3 to obtain $\eta - \beta$ increasing for actions with $\beta > 0$. Since the value $\delta^{max}$ decreases when applying the rules R1, R2, R3, we conclude that $\delta^{max}(a_1..a_n, q) \leq \delta^{max}(a'_1..a'_n, q)$. □

The following propositions allow the computation of an optimal schedule $a_1..a_n$ for a given quality level $q$ when the function $\beta : A \times Q \to \mathbb{R}$ has a constant sign over the set of actions $A$.

**Proposition 4.3** (Scheduling when $\beta \leq 0$) *Let $PSY(C) = (G, Q, C^{wc}, D, C)$ be a parameterized system, and $q$ be a quality level such that for each action $a \in A$, $\beta(a, q) \leq 0$. We define the function $\eta^*$ as follows*:

$$\eta^*(a, q) = \mathop{\mathbf{max}}_{a'=a \vee a \prec a'} \eta(a', q).$$

*Let $a_1..a_n$ be a schedule such that for all indexes $i$ and $j$, we have $i < j \Rightarrow \eta^*(a_i, q) \geq \eta^*(a_j, q)$. Then the schedule $a_1..a_n$ minimizes $\delta^{max}$, that is, for all schedule $a'_1..a'_n$ of $A$ we have $\delta^{max}(a_1..a_n, q) \leq \delta^{max}(a'_1..a'_n, q)$.*

The proof of the proposition is based on the following lemma.

**Lemma 4.1** *Let $PSY(C) = (G, Q, C^{wc}, D, C)$ be a parameterized system, and $q$ be a quality level such that for each action $a \in A$, $\beta(a, q) \leq 0$. For each schedule $a_1..a_n$ of $G$, we have*:

$$\delta^{max*}(a_1..a_n, q) = \delta^{max}(a_1..a_n, q),$$

*where $\delta^{max*}$ is computed as $\delta^{max}$ by replacing $\eta$ with $\eta^*$, that is*:

$$\delta^{max*}(a_1..a_n, q) = \mathop{\mathbf{max}}_{1 \leq i \leq n} \delta^*(a_i..a_n, q) = \mathop{\mathbf{max}}_{1 \leq i \leq n} \eta^*(a_i, q) + \beta(a_{i+1}..a_n, q).$$

*Proof* First, we show that $\delta^{max}(a_1..a_n, q) \geq \delta^{max*}(a_1..a_n, q)$. Since $\delta^{max*}(a_1..a_n, q) = \mathbf{max}_{1 \leq j \leq n} \delta^*(a_j..a_n, q)$, there exists $i \in \{1, \ldots, n\}$ such that $\delta^{max*}(a_1..a_n, q) = \delta^*(a_i..a_n, q)$. By definition, we have $\delta^*(a_i..a_n, q) = \eta^*(a_i, q) + \beta(a_{i+1}..a_n, q)$ and $\eta^*(a, q) = \mathbf{max}_{a'=a \vee a \prec a'} \eta(a', q)$. That is, there exists $j \geq i$ such that $\eta^*(a_i, q) = \eta(a_j, q)$. Thus:

$$\begin{aligned}
\delta^{max*}(a_1..a_n, q) &= \delta^*(a_i..a_n, q) \\
&= \eta^*(a_i, q) + \beta(a_{i+1}..a_n, q) \\
&= \eta(a_j, q) + \beta(a_{i+1}..a_n, q) \\
&= \eta(a_j, q) + \beta(a_{i+1}..a_j, q) + \beta(a_{j+1}..a_n, q) \\
&\leq \eta(a_j, q) + \beta(a_{j+1}..a_n, q) \text{ since } \beta \leq 0 \\
&\leq \delta(a_j..a_n, q) \\
&\leq \delta^{max}(a_1..a_n, q).
\end{aligned}$$

We show that $\delta^{max*}(a_1..a_n, q) \geq \delta^{max}(a_1..a_n, q)$. According to the definition of $\eta^*$, we have for each action $a$ and each quality level $q$, $\eta^*(a, q) \geq \eta(a, q)$. Thereby, for all $i \in \{1, \ldots, n\}$, we have $\delta^*(a_i..a_n, q) \geq \delta(a_i..a_n, q)$. This implies that $\delta^{max*}(a_1..a_n, q) \geq \delta^{max}(a_1..a_n, q)$. □

*Proof of Proposition 4.3* By applying the rule R1 of the Proposition 4.1 to the functions $\delta^{max*}$ and $\eta^*$, we show that the schedule $a_1..a_n$ in which actions are scheduled by $\eta^*$ decreasing minimizes $\delta^{max*}$. Since $\delta^{max} = \delta^{max*}$ (Lemma 4.1), we conclude that $a_1..a_n$ minimizes $\delta^{max}$. □

**Proposition 4.4** (Scheduling when $\beta \geq 0$) *Let $PSY(C) = (G, Q, C^{wc}, D, C)$ be a parameterized system, let $q$ a quality level such that for each action $a \in A$, $\beta(a, q) \geq 0$. We define the function $(\eta - \beta)^*$ as follows* :

$$(\eta - \beta)^*(a, q) = \mathop{\mathbf{max}}_{a'=a \vee a' \prec a} (\eta - \beta)(a', q).$$

*Let schedule $a_1..a_n$ be a schedule such that for all indexes $i$ and $j$, we have $i < j \Rightarrow (\eta - \beta)^*(a_i, q) \leq (\eta - \beta)^*(a_j, q)$. Then the schedule $a_1..a_n$ minimizes $\delta^{max}$, that is, for all schedule $a'_1..a'_n$ of $A$ we have $\delta^{max}(a_1..a_n, q) \leq \delta^{max}(a'_1..a'_n, q)$.*

The proof of the proposition is based on the following lemma.

**Lemma 4.2** *Let $PSY(C) = (G, Q, C^{wc}, D, C)$ be a parameterized system, let $q$ a quality level such that for each action $a \in A$, $\beta(a, q) \geq 0$. For each schedule of $G$, we have*:

$$\delta^{max*}(a_1..a_n, q) = \delta^{max}(a_1..a_n, q),$$

*where $\delta^{max*}$ is computed as $\delta^{max}$ by replacing $\eta$ by $(\eta - \beta)^* + \beta$, that is*:

$$\begin{aligned}\delta^{max*}(a_1..a_n, q) &= \mathop{\mathbf{max}}_{1 \leq i \leq n} \delta^*(a_i..a_n, q) \\ &= \mathop{\mathbf{max}}_{1 \leq i \leq n} ((\eta - \beta)^* + \beta)(a_i) + \beta(a_{i+1}..a_n, q).\end{aligned}$$

*Proof* First we show that $\delta^{max}(a_1..a_n, q) \geq \delta^{max*}(a_1..a_n, q)$. Since $\delta^{max*}(a_1..a_n, q) = \mathbf{max}_{1 \leq j \leq n} \delta^*(a_j..a_n, q)$, there exists $i \in \{1, \ldots, n\}$ such that $\delta^{max*}(a_1..a_n, q) = \delta^*(a_i..a_n, q)$. We have :

$$\delta^*(a_i..a_n, q) = ((\eta - \beta)^* + \beta)(a_i, q) + \beta(a_{i+1}..a_n, q).$$

As $(\eta - \beta)^*(a, q) = \mathbf{max}_{a'=a \vee a' \prec a}(\eta - \beta)(a', q)$, there exists $j \leq i$ such that $(\eta - \beta)(a_j, q) = (\eta - \beta)^*(a_i, q)$. Thus,

$$\begin{aligned}\delta^{max*}(a_1..a_n, q) &= \delta^*(a_i..a_n, q) \\ &= ((\eta - \beta)^* + \beta)(a_i, q) + \beta(a_{i+1}..a_n, q) \\ &= (\eta - \beta)(a_j, q) + \beta(a_i, q) + \beta(a_{i+1}..a_n, q)\end{aligned}$$

$$
\begin{aligned}
&\leq (\eta-\beta)(a_j,q)+\beta(a_{j+1}..a_{i-1},q)+\beta(a_i,q)\\
&\quad +\beta(a_{i+1}..a_n,q) \text{ since } \beta\geq 0\\
&\leq (\eta-\beta)(a_j,q)+\beta(a_{j+1}..a_n,q)\\
&\leq \delta(a_j..a_n,q)\\
&\leq \delta^{max}(a_1..a_n,q).
\end{aligned}
$$

We conclude that $\delta^{max}(a_1..a_n,q)\geq\delta^{max*}(a_1..a_n,q)$.

Then we show that $\delta^{max*}(a_1..a_n,q)\geq\delta^{max}(a_1..a_n,q)$. The result comes directly from the definitions. We have:

$$
\begin{aligned}
&(\eta-\beta)^*(a_i,q)\geq(\eta-\beta)(a_i,q)\\
&\Rightarrow\quad (\eta-\beta)^*(a_i,q)+\beta(a_i,q)+\beta(a_{i+1}..a_n,q)\\
&\qquad\geq(\eta-\beta)(a_i,q)+\beta(a_i,q)+\beta(a_{i+1}..a_n,q)\\
&\Rightarrow\quad \delta^*(a_i..a_n,q)\geq\delta(a_i..a_n,q)\\
&\Rightarrow\quad \delta^{max*}(a_1..a_n,q)\geq\delta^{max}(a_1..a_n,q).
\end{aligned}
$$

□

*Proof of Proposition 4.4* By applying the rule R3 of the Proposition 4.1 at the functions $\delta^{max*}$ and $(\eta-\beta)^*$, we show that a schedule $a_1..a_n$ in which actions are scheduled by $(\eta-\beta)^*$ decreasing minimizes $\delta^{max*}$. Since $\delta^{max*}=\delta^{max}$ (Lemma 4.2), we conclude that $a_1..a_n$ minimizes $\delta^{max}$. □

### 4.2 Computing *Best_Sched* for the general case

Computing an optimal schedule for the mixed quality management policy is a non-trivial problem when the deadline function $D$ is not constant. As EDF schedules are optimal for the quality control problem with known execution times, an idea is to restrict the exploration to the EDF schedules. The proposed heuristic is based on results of the previous section.

**Definition 4.2** Let $PSY(C)=(G,Q,C^{wc},D,C)$ be a parameterized system. We say that a schedule $a_1..a_n$ of $G$ is **EDF-optimal** with respect to the quality level $q$ if $a_1..a_n$ is an EDF schedule and:

$$t_p^{mx}(a_1..a_n,q)=\mathbf{max}\{t_p^{mx}(a'_1..a'_n,q)\mid a'_1..a'_n\in EDF(G,D)\}.$$

The following proposition allows the computation of EDF-optimal schedules. This is achieved by a local minimization of $\delta^{max}$.

**Proposition 4.5** *Let $a_1..a_n$ be an EDF schedule of $G=(A,\prec)$ and $q$ be a quality level. The global deadline function $D^*$ defined in Sect. 2.1.2 (Definition 2.7) induces a partition $A_1\ldots A_L$ of $A$ such that $D^*(A_1)<\cdots<D^*(A_L)$. We have*:

- *$a_1..a_n=\alpha_1..\alpha_L$ where for all $l\in\{1,\ldots,L\}$, $\alpha_l$ is the sequence of actions of $A_l$.*

- *If $\delta^{max}(\alpha_l, q)$ is minimal, then $\alpha_l$ is an EDF-optimal schedule of $G/A_l$ with respect to the quality level $q$.*
- *If for all $l \in \{1, \ldots, L\}$, $\alpha_l$ is EDF-optimal with respect to the quality level $q$, then $a_1..a_n$ is EDF-optimal with respect to the quality level $q$.*

**Lemma 4.3** *Let $a_1..a_n$ be a schedule of $G$ such that $a_1..a_n = \alpha_1..\alpha_L$, where for all $l \in \{1, \ldots, L\}$ $\alpha_l$ is a sequence of actions. Then:*

$$\delta^{max}(a_1..a_n, q) = \delta^{max}(\alpha_1..\alpha_L, q) = \max_{1 \le l \le L} \delta^{max}(\alpha_l, q) + \sum_{l < i \le L} \beta(\alpha_i, q).$$

*Proof* We write $b(l)$ for the index of the first action of $\alpha_l$, that is, $\alpha_l = a_{b(l)}..a_{b(l+1)-1}$, and $b(L+1) = n+1$.

$$\begin{aligned}
&\delta^{max}(\alpha_1..\alpha_L, q) \\
&\quad = \max_{1 \le i \le n} \eta(a_i, q) + \beta(a_{i+1}..a_n, q) \\
&\quad = \max_{1 \le l \le L} \Big( \max_{b(l) \le i \le b(l+1)-1} \big(\eta(a_i, q) + \beta(a_{i+1}..a_n, q)\big) \Big) \\
&\quad = \max_{1 \le l \le L} \Big( \max_{b(l) \le i \le b(l+1)-1} \big(\eta(a_i, q) + \beta(a_{i+1}..a_{b(l+1)-1}, q) + \beta(a_{b(l+1)}..a_n, q)\big) \Big).
\end{aligned}$$

As $\beta(a_{b(l+1)}..a_n, q)$ does not depend of $i$, we have:

$$\begin{aligned}
&\delta^{max}(\alpha_1..\alpha_L, q) \\
&\quad = \max_{1 \le l \le L} \Big( \max_{b(l) \le i \le b(l+1)-1} \big(\eta(a_i, q) + \beta(a_{i+1}..a_{b(l+1)-1}, q)\big) + \beta(a_{b(l+1)}..a_n, q) \Big) \\
&\quad = \max_{1 \le l \le L} \delta^{max}(\alpha_l, q) + \beta(a_{b(l+1)}..a_n, q) \\
&\quad = \max_{1 \le l \le L} \delta^{max}(\alpha_l, q) + \beta(\alpha_{l+1}, q) + \ldots + \beta(\alpha_L, q).
\end{aligned}$$

□

*Proof of Proposition 4.5*

- The first point of the proposition is left as an exercise to the reader.
- As $D$ is constant over the subset of actions $A_l$, we can apply results of Sect. 4.1, that is, a schedule $\alpha_l$ of $G/A_l$ is optimal (i.e. maximizes $t_p^{mx}(\alpha_l, q)$) if and only if it minimizes $\delta^{max}(\alpha_l, q)$. As any schedule of $G/A_l$ is an EDF schedule since $D$ is constant over $A_l$, we can conclude that if $\delta^{max}(\alpha_l, q)$ is minimal, then $\alpha_l$ is an EDF-optimal schedule of $G/A_l$.
- Let $D_1, \ldots, D_L$ be the value of $D$ over the subset of actions $A_1, \ldots, A_L$. For any EDF schedule $a_1..a_n = \alpha_1..\alpha_L$ of $G$, we have:

$$\begin{aligned}
t_p^{mx}(a_1..a_n, q) &= \max_{1 \le i \le n} D(a_i) - C^{av}(a_1..a_i, q) - \delta^{max}(a_1..a_i, q) \\
&= \max_{1 \le l \le L} D_l - \big(C^{av}(\alpha_1) + \cdots + C^{av}(\alpha_l, q)\big) - \delta^{max}(\alpha_1..\alpha_l, q).
\end{aligned}$$

As $\alpha_l$ is a schedule of $A_l$, the value $(C^{av}(\alpha_1) + \cdots + C^{av}(\alpha_l, q))$ does not depend of the EDF schedule $a_1..a_n$. By Lemma 4.3, we have:

$$\delta^{max}(\alpha_1..\alpha_l, q) = \max_{1 \le i \le l} \delta^{max}(\alpha_i, q) + \beta(\alpha_{i+1}..\alpha_L, q).$$

As $\beta(\alpha_{i+1}..\alpha_L, q)$ does not depend of the EDF schedule $a_1..a_n$, we can conclude that if for all $i \in \{1, \ldots, L\}$, $\delta^{max}(\alpha_i, q)$ is minimal, then $t_p^{mx}(a_1..a_n, q)$ is maximal amongst the EDF schedule, that is, $a_1..a_n$ is an EDF-optimal schedule. □

## 5 Experimental results

This section provides experimental results which confirm the interest of theory developed in previous sections. We present the experimental framework as well as a description of the target application (an MPEG4 video encoder) and platform. We compare the application running with a controller generated by our method, to the same application running with constant quality, which corresponds to the standard industrial practice. Then, we show how optimizations of the controller (quality managements policies, symbolic approach, scheduling policy) impact the application.

### 5.1 Experimental framework

We designed controllers for an MPEG4 encoder written in C (more than 15000 loc). The encoder treats frames cyclically. Each frame is split into $N$ macroblocks of 256 pixels. The precedence graph corresponding to the treatment of a frame is given in Fig. 12. It is composed of the first action – `Grab_Picture` – followed by $N$ iterations of the same precedence graph. We have a precedence constraint between two consecutive iterations of the same node of this precedence graph. For instance, the $i^{\text{th}}$ iteration of `DCT` must be scheduled before its $i + 1^{\text{th}}$ iteration. However, all iterations of `Motion_Estimate` can be scheduled before the first iteration of `DCT`.

The video encoder architecture is shown in Fig. 13. The considered application corresponds to a videophone application. It captures a sequence of frames with a camera, transmits the sequence, and then displays the frames on a screen. From a captured frame, the video encoder produces a corresponding bitstream. The latter is transmitted to a video decoder which decodes the bitstream and displays decoded frame on a screen. This architecture uses input and output buffers of the same size $K$, to cope with changes of load and avoid as much as possible frame skips. These may happen when the input buffer is full.

We developed a prototype tool (Fig. 14) that allows the generation of controlled application software. The inputs of the tool are a parameterized system $PSY(C) = (G, Q, C^{wc}, D, C)$ and an average execution time function $C^{av}$, that is,

- the precedence graph $G = (A, \prec)$ modeling actions (C functions) and data dependencies between the actions, and the corresponding set of quality level parameters $Q$,
- average execution times $C^{av}$ and worst-case execution times $C^{wc}$,
- action deadlines $D$.

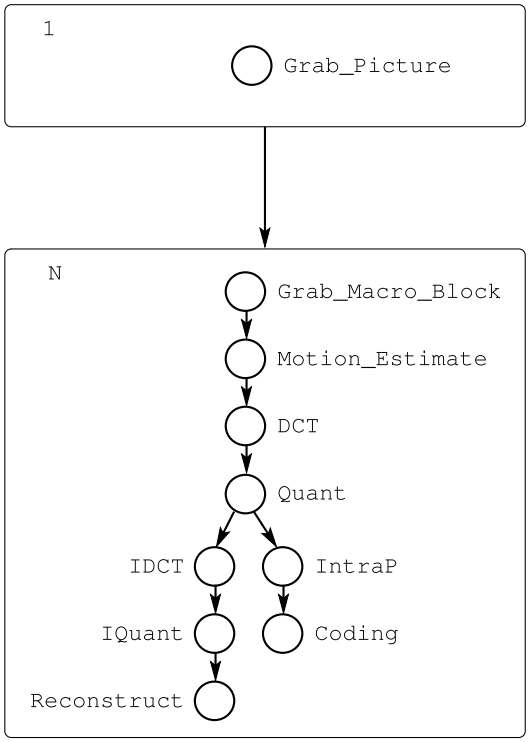



**Fig. 12** Precedence graph of the video encoder

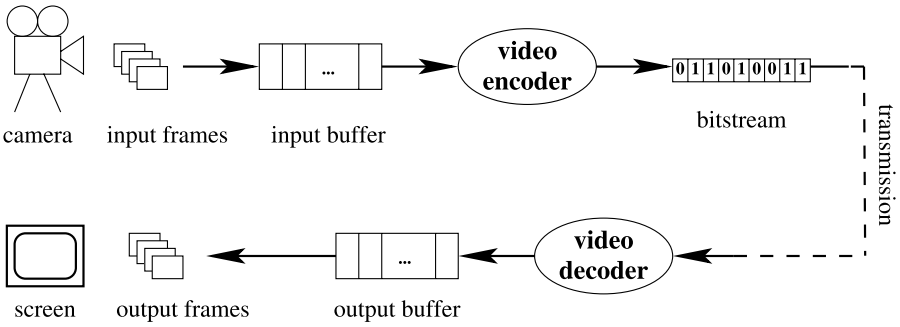



**Fig. 13** Video encoder architecture

From these inputs the tool computes:

- C code corresponding to an optimized EDF schedule,
- tables containing pre-computed values used by the numeric implementations, and Speed Diagrams used by the symbolic implementation.

A compiler is used to link the following items and generate the controlled application software from:

- the schedule, the tables and Speed Diagrams generated by the tool,

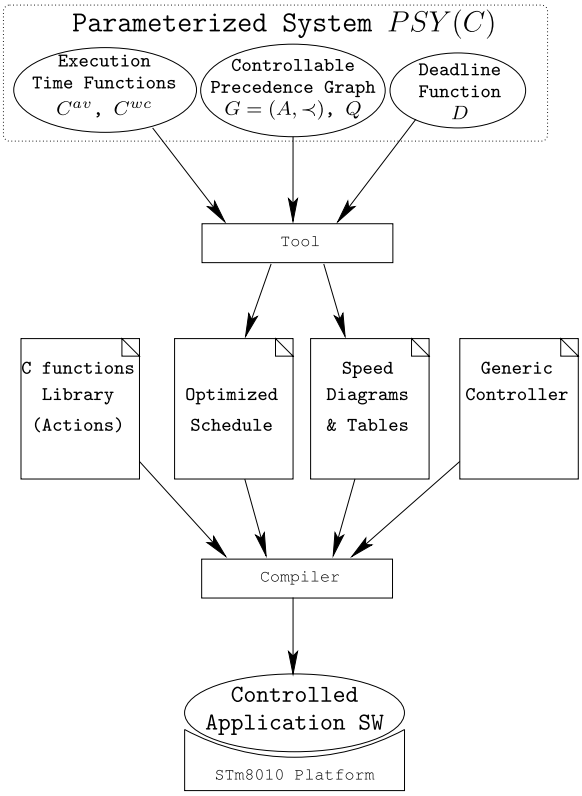



**Fig. 14** Prototype tool for the generation of the controlled application software

- the application code for the actions of the schedule,
- a generic Controller mainly consisting of a Quality Manager.

For the experimental results, the target platform is an STm8010 board from STMicroelectronics. It includes three ST231 processors running at 400 MHz, and it is used in set-top box products. As our approach targets mono-processor platforms, we use only one of the three ST231. A register that counts the number of processor cycles elapsed provides a real-time clock with minimal access overhead.

For the considered example, the execution times of the actions `Motion_Estimate`, `DCT`, `Quant` and `Coding` depend on the quality level as specified in Table 2. The execution times of all the other actions are constant, and are given in same Table 2. We also consider a constant deadline function $D$ corresponding to the time budget allowed for encoding a frame.

The only action which has significant fluctuation of execution time with quality level is `Motion_Estimate`. To reduce the number of Controller calls, we only control quality level parameter before the execution of this action, that is, one time per macroblock. Thus the number of Controller calls per frame is equal to $N$.

We evaluated four implementations:

**Table 2** Average and worst-case execution times

| Action | Average | Worst case |
|---|---|---|
| `Grab_Picture` | 11000 | 20000 |
| `Grab_Macro_Block` | 9000 | 20000 |
| `IntraP` | 8000 | 20000 |
| `IQuant` | 10000 | 15000 |
| `IDCT` | 8000 | 15000 |
| `Reconstruct` | 11000 | 20000 |

| | `Motion_Estimate` | |
|---|---|---|
| Quality | Average | Worst case |
| 0 | 3000 | 10000 |
| 1 | 12000 | 40000 |
| 2 | 20000 | 50000 |
| 3 | 30000 | 100000 |
| 4 | 40000 | 120000 |
| 5 | 50000 | 150000 |
| 6 | 70000 | 200000 |
| 7 | 90000 | 300000 |

| Action | Average $q > 0$ | Worst case $q > 0$ | Average $q = 0$ | Worst case $q = 0$ |
|---|---|---|---|---|
| `DCT` | 11000 | 15000 | 150 | 400 |
| `Quant` | 12000 | 20000 | 1500 | 4000 |
| `Coding` | 10000 | 25000 | 1500 | 3000 |

Constant quality. In this implementation, quality level parameters are constant and defined statically before the execution of the application. This corresponds to the standard industrial practice.

Controlled quality for numeric quality manager. This is a straightforward implementation of the mixed quality management policy given in Sect. 3.1.3. We consider a non-optimized version and an optimized version using pre-computed values in order to speed up online computation of quality levels.

Controlled quality for quality manager using quality regions. We used the prototype tool for pre-computing quality regions $\mathcal{R}_q$ defined in Sect. 3.2.2. These are used by the Quality Manager to compute online action quality levels.

Controlled quality for quality manager using control relaxation regions. We used the prototype tool for pre-computing control relaxation regions $\mathcal{R}_q^r$ defined in Sect. 3.2.3 for $r \in \rho = \{1, 3, 6, 9, 16, 32\}$. These regions are used by the Quality Manager to relax the granularity of control.

## 5.2 Controlled quality vs constant quality

The first experiment consists on a comparison between the standard industrial practice, which is based on constant quality assignment, and the controlled quality method. We measure PSNR (Peak Signal to Noise Ratio) between the input frames and output frames. We also plot the utilization of the time budget which is the ratio between the time for encoding a frame and $D$, as a function of the number of treated frames. PSNR characterizes single frame quality and is used to measure the effect on video quality of the encoding process. We compare the performances of the controlled encoder using mixed quality management policy generated by our prototype and the same encoder for constant quality level.

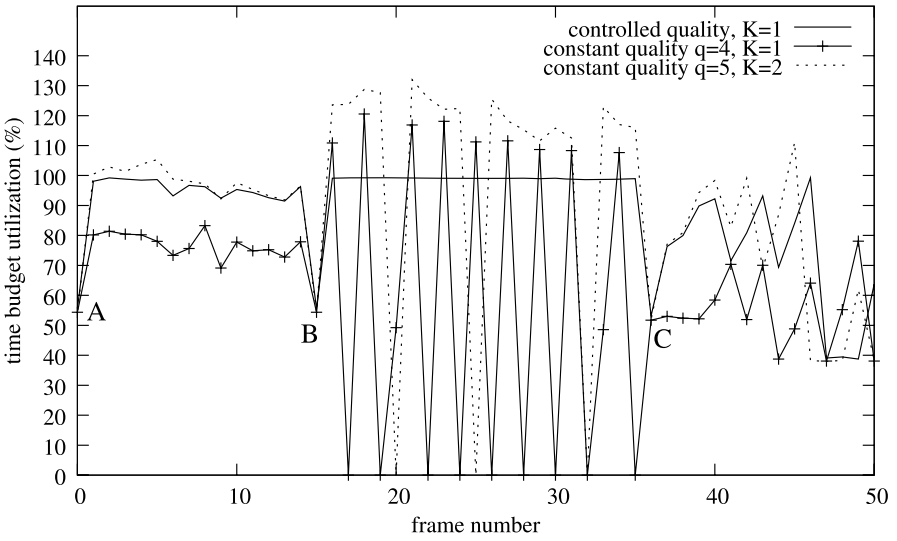



**Fig. 15** Time budget utilization

We consider a test case of 50 frames, consisting of 3 sequences produced by a camera every $D = 100$ ms (i.e. constant framerate of 10 frame/s).

The buffers of size $K$ allow a maximal latency of $D \cdot K$. The time budget allocated to the encoder for the treatment of a frame depends on the buffer occupancy, and is on average $D$. As our method guarantees safety, we can take $K = 1$ for the controlled encoder without deadline miss.

Time budget utilization is shown in Fig. 15, for controlled quality by using mixed quality management policy and for constant quality with $q = 4$, $K = 1$, and with $q = 5$, $K = 2$. Notice the presence of two kinds of jumps:

- three jumps corresponding to changes of video sequences (encoding of I-frames) occurring at frames number 0, 15, and 35 for controlled and constant quality (points A, B, and C in Fig. 15);
- bursts of jumps corresponding to frame skips due to buffer overflow occurring for constant quality only.

PSNR for the same test case is given in Fig. 16. Notice again the two types of jumps due to changes of video sequences and frame skips. When a frame is skipped, the immediately previous frame is displayed by the decoder, and the comparison to the input frame gives a low PSNR value (e.g. lower than 25). PSNR is higher for controlled quality than for constant quality $q = 4$, except for regions where frames are skipped. For these regions, the bits corresponding to skipped frames are used to achieve better quality. Although the PSNR is higher in these regions for constant quality, the video quality is affected as the frame rate is divided by two. Using buffers of size $K = 2$ allows activation of constant quality $q = 5$, but frame skips remain.

Experimental results show that for constant quality levels load fluctuation can lead to poor video quality in absence of sufficiently large buffers. Poor video quality means

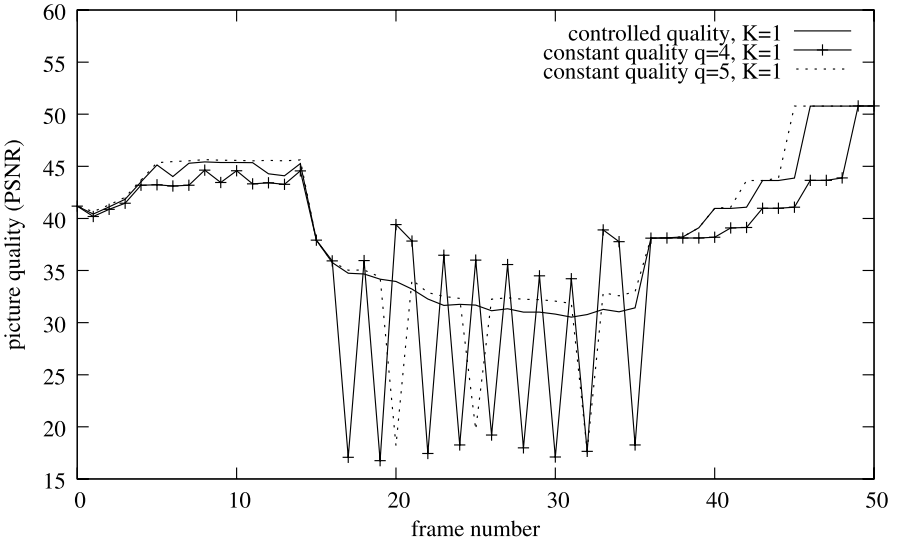



**Fig. 16** PSNR between input and output

low PSNR or frame skips (or both). For controlled quality, there are no frame skips. Thus, overloads result in low PSNR. Furthermore, using buffers may not completely eliminate frame skips. It implies additional cost and increased latency. The comparison between constant and controlled quality shows that for controlled quality we get better video quality even for buffer size 2. Controlled quality completely avoids frame skips; overloads lead to smooth reduction of PSNR.

### 5.3 The impact of quality management policy

This part shows the importance of the choice of the quality management policy. We have compared safe, simple and mixed quality management policies. We provide results for a static schedule. We build the speed diagram (see Sect. 3.2) for a particular input frame. As results obtained with a controller using the safe quality management policy are similar to the ones obtained using simple quality management policy, we only plot results for simple and mixed policies (see Fig. 17).

For simple quality management policy speed discontinuities are observed at points A and B. The speed of the system from point A to point B corresponds to a choice of minimal quality. This drastically reduces the video quality. On the contrary, for mixed policy we get almost uniformly constant speed which leads to significantly improved video quality.

### 5.4 Performance of symbolic quality managers

These experiments provide results the symbolic approach developed Sect. 3.2. We have compared the performances of numeric and symbolic implementations of the Quality Manager for an input sequence of 29 frames of $320 \times 144$ pixels ($N = 180$ macroblocks).

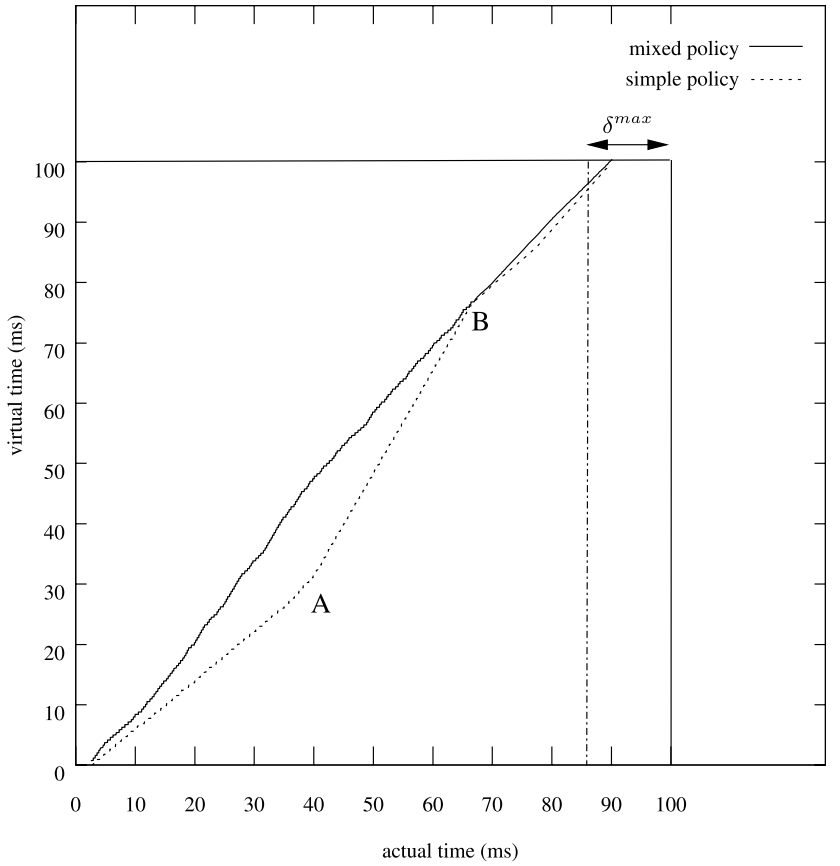



**Fig. 17** Speed diagram for simple and mixed quality management policies

Overhead in memory allocation for numeric implementations is almost zero. For the symbolic implementations, we have the following overhead in memory allocation.

Quality manager using quality regions. By Proposition 3.4, quality regions are characterized by the set of the values $t_p^{mx}(s_i, q)$ for all quality levels $q$ and for all states $s_i$. Thus, as $i$ ranges from 0 to $N-1$ this set is specified by $N \times |Q| = 1,440$ integers. For the video encoder application, we have measured an overhead in memory allocation of 20 KB.

Quality manager using control relaxation regions. By Definition 3.5, control relaxation regions are characterized by the set of the values $t_p^{mx}(s_{i+r-1}, q+1)$ and $t_p^{mx,r}(s_i, q)$ for all the quality levels $q$, indices $i \in \{1, \ldots, N-1\}$ and relaxation steps $r \in \rho$, that is, a set of $2N \times |Q||\rho| = 17,280$ integers. We observed an overhead in memory allocation of 350 KB.

Execution time overhead due to quality management is on average 8.5% for the numeric implementation, 4% for optimized numeric implementation, 1.2% for the

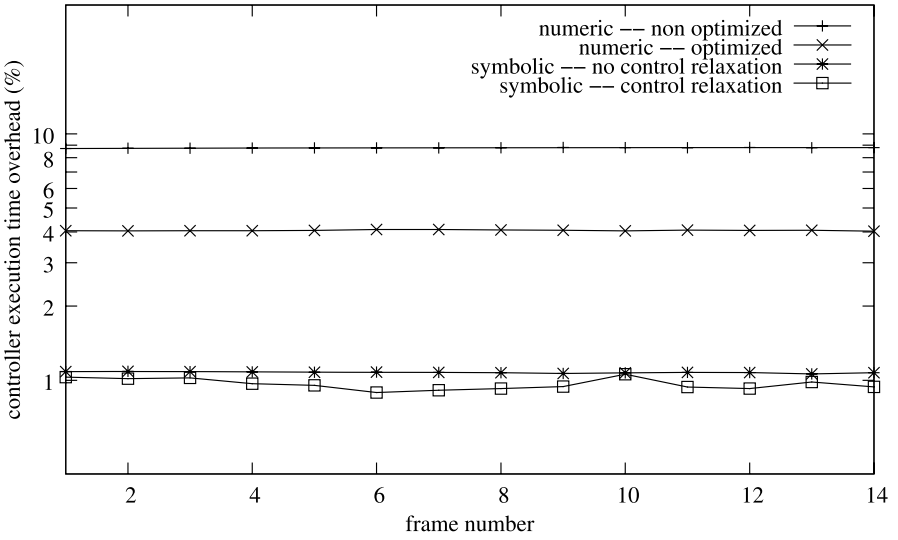



**Fig. 18** Execution time overhead due to quality management

symbolic implementation using quality regions (no control relaxation) and less than 1% for the implementation using control relaxation regions (see Fig. 18). In a previous paper (Combaz et al. 2007), we obtained more significant reduction of execution time overhead by using control relaxation for the same application running on an iPod video.

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 Fig. 19). This leads to a significant improvement of the overall video quality.

Figure 20 compares for a sequence of 180 actions encoding a frame, overheads in execution time with and without control relaxation. The control relaxation technique developed in Sect. 3.2.3 is used to reduce controller execution time overhead. Given a state of the system, the controller can be relaxed for $r$ steps if we can ensure that the quality level chosen by the controller remains the same for the next $r$ actions. Notice that the number of relaxation steps $r$ is dynamically adapted during the execution of the application: $r = 9$ for $a_0$, then $r = 6$ for $a_9$, $r = 3$ for $a_{15}$ to $a_{120}$, and the $r = 1$ for the remaining actions.

## 5.5 Using optimized schedules

The last experiments are based on results of Sect. 4. We have compared the controlled application running with a non-optimized function $Best\_Sched^{mx}$ (high values of $\delta^{max}$), and the same controlled application running with a function $Best\_Sched^{mx}$ obtained by applying the three optimization rules R1, R2, R3 given in Proposition 4.1.

Figure 21 shows that the utilization of time budget is close to 100% when $Best\_Sched^{mx}$ is optimized, whereas approximately 15% are lost on average when

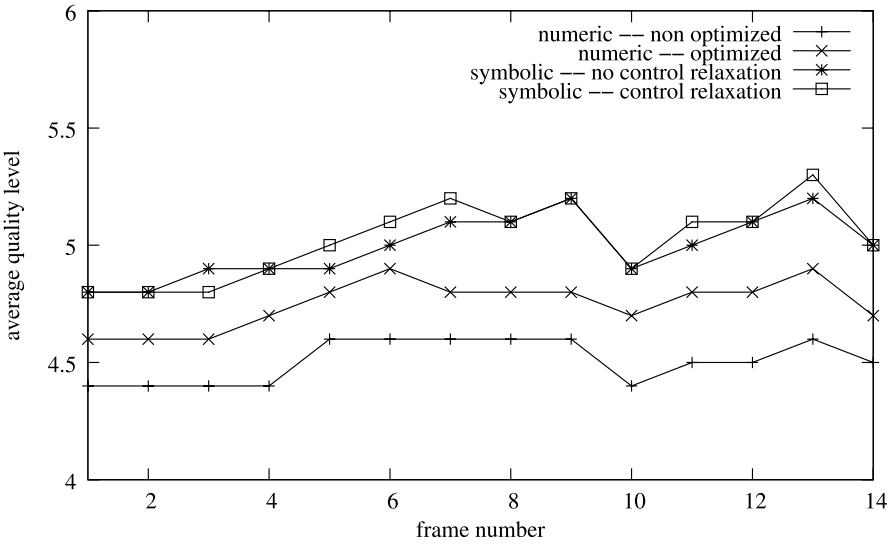



**Fig. 19** Average quality level

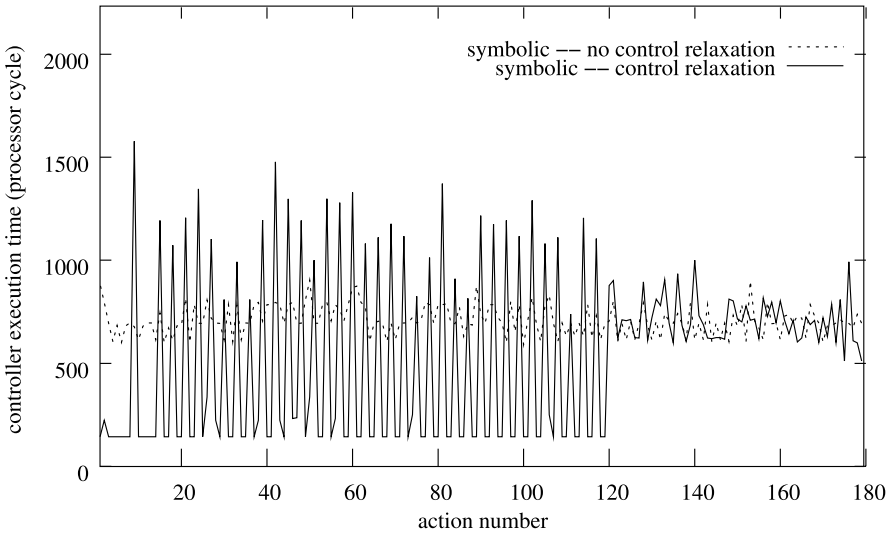



**Fig. 20** Overhead in execution time

*Best_Sched*$^{mx}$ is non-optimized. This corresponds to the difference of the values $\delta^{max}$ encountered during the execution, between the optimized and the non-optimized case.

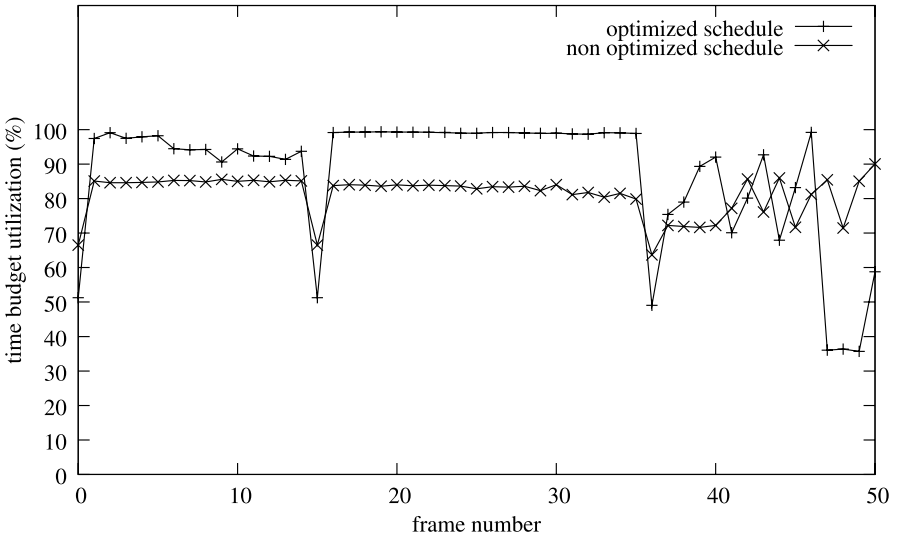



**Fig. 21** Optimization of $Best\_Sched^{mx}$

## 6 Conclusion

The presented method uses fine grain control to meet safety and best effort requirements. It overcomes limitations of hard real-time approaches where strict respect of deadlines implies poor time budget utilization. This is possible through fine grain control, which allows adaptation to load changes during a cycle instead of using a priori known global execution time estimates.

The method is founded on a solid theoretical basis. The controller computes feasible schedules under reasonable assumptions about the controlled system—the existence of statically computable feasible schedules for minimal quality. The behavior of the controller is characterized by its quality management policy and the associated *Best_Sched* function. Experimental results show that the mixed quality management policy significantly improves quality smoothness with respect to the two other policies.

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 results show that even in the ideal case where actual execution times agree with average execution times, meeting safety requirements inherently limits the achievement of optimality. The actual execution time may not fill the entire available time budget. The amount of the available time which is lost must be lower than a constant, which depends on the difference between average and worst-case behavior as well as granularity of control.

The symbolic quality management method improves and extends our previous results (Combaz et al. 2005a, 2005b).

- The use of constant quality and control relaxation regions which can be precomputed 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. Experimental results confirm the interest of symbolic quality management because of its low overhead.

An important finding is that under uncertainty, all EDF schedules are not equivalent with respect to the considered quality management policies. The rules provided for computing the schedules which maximize the corresponding schedule functions, define strategies for performance improvement.

Experimental results confirm the interest of the method and its low overhead. Quality control allows considerable performance gains with respect to constant quality. Symbolic quality management techniques allow a further improvement with respect numeric techniques.

We currently work in several directions to improve the method and the supporting tools: 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.

## References

Audsley NC, Davis RI, Burns A (1994) Mechanisms for enhancing the flexibility and utility of hard real-time systems. In: Real-time systems symposium. IEEE, New York, pp 12–21

Bril RJ, Gabrani M, Hentschel C, van Loo GC, Steffens EFM (2001) Qos for consumer terminals and its support for product families. In: Proceedings of the international conference on media futures

Buttazzo GC, Lipari G, Abeni L (1998) Elastic task model for adaptive rate control. In: RTSS, pp 286–295

Combaz J, Fernandez J, Lepley T, Sifakis J (2005a) Fine grain qos control for multimedia application software. In: Design, automation and test in Europe (DATE'05), vol 2, pp 1038–1043

Combaz J, Fernandez J-C, Lepley T, Sifakis J (2005b) QoS control for optimality and safety. In: Proceedings of the 5th conference on embedded software, September 2005

Combaz J, Fernandez J-C, Sifakis J, Strus L (2007) Using speed diagrams for symbolic quality management. In: IPDPS. IEEE, New York, pp 1–8

Davis RI, Tindell KW, Burns A (1993) Scheduling slack time in fixed priority preemptive systems. In: Proceeding of the IEEE real-time systems symposium, pp 222–231

Hansen JP, Lehoczky JP, Rajkumar R (2001) Optimization of quality of service in dynamic systems. In: IPDPS '01: proceedings of the 15th international parallel & distributed processing symposium. IEEE Computer Society, Washington, p 95

Isovic D, Fohler G, Steffens L (2003) Timing constraints of mpeg-2 decoding for high quality video: misconceptions and realistic assumptions

Koren G, Shasha D (1996) Skip-over: algorithms and complexity for overloaded systems that allow skips. Technical Report TR1996-715

Lehoczky J, Thuel S (1994) Algorithms for scheduling hard aperiodic tasks in fixed-priority systems using slack stealing. In: Proceedings of the IEEE real-time system symposium

Lu C, Stankovic J, Tao G, Son S (2002) Feedback control real-time scheduling: framework, modeling and algorithm. Real-Time Syst J 23(1–2):85–88. Special issue of control-theoretic approach to real-time computing

Papalau L, Pérez CMO, Steffens L (2004) In: Goddard, S (ed) Work-in-progress session of the 16th Euromicro conference on real-time systems, pp 33–36

Rajkumar R, Lee C, Lehoczky J, Siewiorek D (1997) A resource allocation model for QoS management. In: IEEE real-time systems symposium, December 1997, pp 298–307

Schuster GM, Melnikov G, Katsaggelos AK (1999) A review of the minimum maximum criterion for optimal bit allocation among dependent quantizers. IEEE Trans Multimedia 1(1):3–17

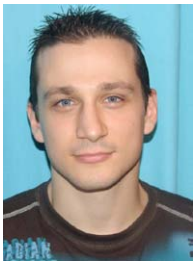
Westerink P, Rajogopalan R, Gonzales C (1999) Two-pass MPEG-2 variable bit-rate encoding. IBM J Res Dev 43(4):471–488

Wüst CC, Steffens L, Bril RJ, Verhaegh WF (2004) Qos control strategies for high-quality video processing. In: Euromicro conference on real-time systems. IEEE, New York, pp 3–12

**Jacques Combaz** received the Ph.D. degree in mathematics and computer science from the University of Joseph Fourier, in Grenoble, France, in 2006.
He is currently post-doc in Verimag laboratory at Grenoble. His research interests include the design of adaptive applications for real-time systems, with special emphasis on multimedia applications.

**Jean-Claude Fernandez** is professor at the University Joseph Fourier, in Grenoble, France. He received the Ph.D. degree in computer science from the University Joseph Fourier.

**Joseph Sifakis** is CNRS researcher and the Founder of Verimag laboratory, in Grenoble, France. He studied Electrical Engineering at the Technical University of Athens and Computer Science at the University of Grenoble.
Joseph Sifakis is recognized for his pioneering work on both theoretical and practical aspects of concurrent systems specification and verification. He contributed to emergence of the area of model-checking. His current research activities include component-based construction of real-time systems with focus on correct-by-construction techniques.
Joseph Sifakis is the Scientific Coordinator of the European Network of Excellence ARTIST2 on Embedded Systems Design. He is a member of the editorial board of several journals, a co-founder of the International Conference on Computer Aided Verification (CAV) and a member of the Steering Committee Board of EMSOFT.



**Loïc Strus** is a Ph.D. student in the DCS team at the laboratory Verima, Joseph Fourier University, Grenoble, France. He focuses his work on the symbolic quality control method.