A Dynamic Scheduling Approach to Designing Flexible Safety-Critical Systems

Luis Almeida
DETI / IEETA-LSE
University of Aveiro
3810-193 Aveiro, Portugal
lda@ua.pt

Sebastian Fischmeister
Computer Science Dep.
University of Pennsylvania
Philadelphia, USA
sfischme@seas.upenn.edu

Madhukar Anand
Computer Science Dep.
University of Pennsylvania
Philadelphia, USA
anandm@seas.upenn.edu

Insup Lee
Computer Science Dep.
University of Pennsylvania
Philadelphia, USA
lee@cis.upenn.edu

ABSTRACT

The design of safety-critical systems has typically adopted static techniques to simplify error detection and fault tolerance. However, economic pressure to reduce costs is exposing the limitations of those techniques in terms of efficiency in the use of system resources. In some industrial domains, such as the automotive, this pressure is too high, and other approaches to safety must be found, e.g., capable of providing some kind of fault tolerance but with graceful degradation to lower costs, or also capable of adapting to instantaneous requirements to better use the computational/communication resources.

This paper analyses the development of systems that exhibit such level of flexibility, allowing the system configuration to evolve within a well-defined space. Two options are possible, one starting from the typical static approach but introducing choice points that are evaluated only at runtime, and another one starting from an open systems approach but delimiting the space of possible adaptations. The paper follows the latter and presents a specific contribution, namely, the concept of local utilization bound, which supports a fast and efficient schedulability analysis for on-line resource management that assures continued safe operation. Such local bound is derived off-line for the specific set of possible configurations, and can be significantly higher than any generic non-necessary utilization bound such as the well known Liu and Layland's bound for Rate-Monotonic scheduling.

Categories and Subject Descriptors

D.4.7 [**Operating Systems**]: Organization and Design—*Real-time systems and embedded systems*

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
EMSOFT'07, September 30–October 3, 2007, Salzburg, Austria.
Copyright 2007 ACM 978-1-59593-825-1/07/0009 ...$5.00.

General Terms

Algorithms design theory

Keywords

Real-time systems, embedded systems, flexible scheduling, dynamic QoS management, utilization bounds

1. INTRODUCTION

There is currently a growing importance of flexibility in the design of distributed embedded systems, and particularly automotive systems, e.g., to conceive software architectures that are as independent as possible from the underlying hardware architectures upon which they will execute, to allow different choices of technologies at the hardware level, to allow easy customization of different product models, to improve efficiency in using the hardware resources namely ECUs and networks, etc. This is, for example, the flexibility pursued by the AUTOSAR consortium in the automotive domain [11]. Their purpose is to reduce production costs whilst satisfying the demands for increased functionality, customization, safety, efficiency, and performance.

The aspects of flexibility referred above apply only at design time, and give the designer more freedom to design functions and allocate them in the hardware architecture (*design flexibility*). However, further benefits can be achieved if the system is able to adapt at runtime to changes in the operational environment or system configuration, such as those arising from hazardous events, evolving requirements, environmental changes, and on-line Quality-of-Service (QoS) management [4, 9, 14, 5]. This level of flexibility (*operational flexibility*) can be exploited to increase the system survivability [14, 15], e.g., by supporting flexible modes and graceful degradation, as well as increasing efficiency in the use of system resources [5], particularly CPU and network bandwidth, carrying along an inherent potential to reduce system costs and improve its dependability.

Operational flexibility may be particularly interesting in cost-constrained safety-critical systems since it may provide support for fault-tolerance mechanisms that are less expensive than modular redundancy approaches to fault tolerance. For example, using functional replicas with different QoS

instead of modular redundancy may allow exploiting spare capacity in existing computing resources without the need for extra ones [15, 8].

Reconciling flexibility and safety as introduced above creates the problem that the system may evolve into an unsafe state during the reconfiguration. In order to prevent this from happening, operational flexibility must be adequately constrained to a space of safe configurations that have been pre-analyzed at design time and use a configuration change mechanism that assures continued safety. Two approaches can be followed, either starting from a static single configuration system or from an open systems perspective (Fig. 1). In the former case one needs to encode more configurations and an adequate switching logic at design time. We refer to it as *static scheduling-based approach*. In the latter case, it is necessary to restrict the space of reconfiguration requests that can be accepted at run-time. This will be referred to as *dynamic scheduling-based approach*.

This paper explores the latter approach only, and proposes a system model and design flow that is adequate to support the design of flexible safety-critical systems. Moreover, a specific contribution is proposed, namely the concept of local utilization bound, to support fast and efficient schedulability analysis for on-line resource management. Such bound is derived for each specific configurations space, but it is significantly higher than common bounds devised for general task sets, i.e., considering any relative offset, period and execution time. Higher utilization bounds mean that more configurations are accepted at run-time.

The remainder of the paper is organized as follows. Section 2 revisits previous related work, Section 3 presents the system model, Section 4 discusses the associated design flow, Section 5 presents the local utilization bound while Section 6 presents one quantitative example. Conclusions are drawn in Section 7.

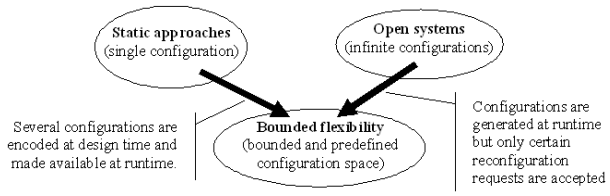

Figure 1: Two approaches to build systems with bounded operational flexibility

2. RELATED WORK

Despite the potential benefits of flexible and adaptive runtime approaches to complex embedded systems, as referred in Section 1, such approaches are not common because of safety concerns and absense of motivation to optimize resource efficiency. Looking at the work carried out within related lines we identify two main directions. On one hand we find the systems that encompass several operational modes to be selected at run-time. There are two issues associated with designing these systems: (1) selecting the modes to include in the design and (2) the mode-change protocol to manage the run-time mode switching. The former deals with both trading off memory with the number of modes and the practicality of verifying each mode individually for correctness and safety. The typical approach has been to strongly reduce the number of modes (i.e., the configuration space). For example, [16] analyses the reconfigurations induced by component failures but reduces the configuration space by first analysing reconfiguration in small subsystems and then composing subsystems into global system configurations. The latter, i.e., mode change protocols, is related with the need to ensure safety during switching from one configuration to another. A good survey of mode change protocols in uniprocessor systems and related issues can be found in [13]. Concerning distributed embedded systems, the Time-Triggered Protocol, for example, also allows defining up to 30 modes and includes a safe mode-change protocol [19].

Notice, however, that the previous approaches rely on static scheduling, either static cyclic table-based or static priorities-based, no matter the actual scheduling of tasks or messages being carried out off-line, as in [19], or on-line, as in [13]. In fact, the task and message sets of each mode are statically defined at design time and cannot change on-line. Other approaches use dynamic scheduling and are fundamentally different since the tasks executed in the system may vary on-line (e.g., the Spring Kernel [12]). In dynamic scheduling, an admission control assures continued timely operation by rejecting new tasks that would lead to unfeasible configurations. In these circumstances it is impossible to know beforehand which tasks the system will be executing at each instant. Despite assuring continued timeliness, this approach does not assure continued safety. In fact, at a given time, a new task that is essential for the system safety might be rejected because the bandwidth of a CPU or network is fully taken by other non-critical tasks/messages.

This problem has been partly addressed with on-line QoS management approaches, which allow defining flexible parameters for tasks and messages as well as managing them as a whole instead of simply testing whether a given change can be accomplished with respect to a fixed running configuration. When an admission control returns a negative answer, these approaches manage the flexible parameters in order to find a configuration in which the requested change can be accepted. Using appropriate management policies, such as the elastic model [5] or the (m,k)-firm model [8] and bounding the task/message set as proposed in [1] it is possible to assure that critical tasks will always find enough resources to be accepted on-line, thus assuring continued safety.

The RoSES project (Robust Self-Configuring Embedded Systems) [16, 15] deserves a particular reference for its resemblance with the work presented here in terms of some of its aims. This project focused on achieving graceful degradation via reconfiguration caused by component failures using the concept of a product family. Conversely, we aim at improving resource efficiency in general, using flexible timing attributes of tasks and messages. Despite also supporting graceful degradation, this is not the main motivation for our work, which targets a higher level of reconfigurability, following the effective resource needs for each system state.

Finally, it is interesting to notice the convergence between the static scheduling-based and dynamic scheduling-based approaches which, despite starting from substantially different stand points, allow us to reach the same goal of introducing operational flexibility into the design of safety-critical systems. Previous work of the authors has explored both alternatives, basically with the use of stateful schedules [7] and

the Flexible Time-Triggered framework [10], respectively.

In this paper we follow the latter approach introducing a new contribution, namely the concept of local schedulability utilization bound to support fast and efficient schedulability analysis for on-line resource management. The overall approach works as follows. Subsystems that are temporarily not needed or at least not with high QoS are either shut down or turned into a low QoS state, thus releasing resources that are made available to a QoS manager. This manager distributes these resources among the currently active subsystems, using a schedulability test and an adequate policy. This corresponds to new configurations that are generated on-line. The schedulability test for the QoS policy is also executed on-line, and often within an iterative procedure, thus the importance of using a fast test.

Examples of situations in which this approach can be applied include all cases in which several subsystems coexist that are not effectively needed all the time, or at least, not with the highest QoS. In the automotive domain we can identify the ABS (anti-lock braking), needed when the brake is pressed, only, the cruise-control, needed when switched on by the user, only, and mutually exclusive with the ABS, the ESP (electronic stability program) that is needed when a lateral skid is detected, only, just to name a few. In robot control, similar situations can be found, e.g. concerning the obstacle detection, which can be more or less frequent depending on the robot speed, etc.

3. SYSTEM MODEL AND PROBLEM FORMULATION

In this paper we consider a complex system composed by a set of subsystems interconnected by a network. Each subsystem is characterised by a flexible set of parameters that depend on which mode each subsystem is in and represent different levels of QoS. The model considers thus a set of nodes $\mathcal{N}$ (Eq. 1) that run a task set $\mathcal{T}$ (Eq. 2) and exchange a set of messages $\mathcal{M}$ (Eq. 3) . We refer to the cardinality of the sets with $|\mathcal{N}|$, $|\mathcal{T}|$ and $|\mathcal{M}|$, respectively. The framework is time-triggered and tasks and messages follow a globally synchronised periodic model, with execution/transmission time (C), period (P), deadline (D) and offset (O). The tasks will also identify the node they reside in (n). There might be also a fixed priority (Pr) representing the importance of the task/message in the scope of the application.

A period of -1 means that the respective task/message is switched off. A period of 0 means infinity and the task/message will be released exactly once. Notice that, as usual in time-triggered models, the transmission of messages is carried out autonomously by the network and not by the tasks that produce the values to be exchanged. Information passing between tasks and messages is carried out through retentive buffers. Therefore, messages behave very similarly to tasks in terms of periodic release. On the other hand, we will consider single packet messages, only, and thus message transmission is non-preemptive.

$$\mathcal{N} \equiv \{n_i, i = 1..|\mathcal{N}|\} \tag{1}$$

$$\mathcal{T} \equiv \{t_i\,(C_i, O_i, P_i, D_i, n_i, Pr_i)\,, i = 1..|\mathcal{T}|\} \tag{2}$$

$$\mathcal{M} \equiv \{m_i\,(C_i, O_i, P_i, D_i, Pr_i)\,, i = 1..|\mathcal{M}|\} \tag{3}$$

One interesting aspect is that the time-triggered model allows decoupling all active resources in the system, i.e., nodes and network, in a way that each one can be analysed separately and appropriate off-sets can be derived to control the end-to-end latency of distributed transactions [6][17]. Therefore, from this point on and for the sake of clarity we will refer to one task set executing on one node. Finally, at this point it is irrelevant whether the deadlines can be larger than the periods or if they are constrained to be less than or equal to the periods. This issue is only relevant to determine the kind of off-line analysis that must be carried out to deduce the local utilization bound, as seen later on.

3.1 Representing different QoS levels

To represent the possible different values of QoS, we currently consider different task execution times and periods, only. Despite a possible, though complex, relationship between offset and QoS, this parameter is not considered as a QoS parameter in this work. Priorities could also be used to control the QoS but they are used herein to express the relative importance of each task in the scope of the application, as referred above.

Therefore, the QoS attributes herein considered are periods and execution times, which are expressed as vectors (Eq. 4 and 5) instead of scalars as usual. These vectors contain all possible values of the respective attribute.

$$\{P_i\}_{\mathcal{T}} = \left\{ \left[P_i^1 P_i^2 \dots P_i^{q_i^{\mathcal{T}}} \right] \right\}_{\mathcal{T}} \tag{4}$$

$$\{C_i\}_{\mathcal{T}} = \left\{ \left[C_i^1 C_i^2 \dots C_i^{d_i^{\mathcal{T}}} \right] \right\}_{\mathcal{T}} \tag{5}$$

The term $q_i^{\mathcal{T}}$ is the length of the period configurations vector, i.e., the number of different periods for task i, and $d_i^{\mathcal{T}}$ is the number of different execution times of task i.

We will designate the whole space of all possible combinations of parameters, the *QoS state space* (Ω). This space has a cardinality of $|\Omega| = \Pi_{i=1}^{|\mathcal{T}|} q_i^{\mathcal{T}} d_i^{\mathcal{T}}$, which represents the total number of instances of task sets, i.e., configurations, that can be produced by instantiating the specified QoS parameters.

However, in general, not the whole QoS space will be feasible. It is possible that many combinations of individual parameters of tasks lead to unschedulable sets. These are undesired and must be avoided during system operation. Thus, a first constraint that must be applied to the QoS space Ω is that of schedulability, resulting in the subset Ω^S of all schedulable configurations. Necessarily, $\Omega^S \subseteq \Omega$.

On the other hand, we define two other sets of constraints over the QoS space Ω, namely the *exclusion constraints* X and the *coherency constraints* H. These constraints capture control dependencies across application subsystems that result in forbidden or mandatory relationships among tasks periods or execution times. For example, the ABS in a car runs in mutual exclusion with the cruise control. This means that configurations in which specific tasks of both functions are active at the same time should be rejected. Also, when the rate of a task that produces a message is changed, the rate of that message should change consistently. These constraints must be enforced when generating on-line configurations.

DEFINITION 1 (EXCLUSION CONSTRAINT). *An* exclusion constraint $X = \langle t_i, t_j \rangle$ *specifies that if* t_i *is in state* (C_i, P_i) *and* t_j *is in state* (C_j, P_j)*, then* $P_i \geq 0 \Rightarrow P_j = -1$ *and* $P_j \geq 0 \Rightarrow P_i = -1$.

DEFINITION 2 (COHERENCY CONSTRAINT). *A period coherency constraint* $H_P = \langle t_i, t_j, k \rangle$ *specifies that if* t_i *is in state* $\left(C_i, P_i^k\right)$ *where* k *indicates the index in the* t_i *period vector* P_i*, then* t_j *must be in state* $\left(C_j, P_j^k\right)$ *with* $k \in \mathbb{Z}_{\geq 1}$ *and* $|P_i| = |P_j|$*. A* computation coherency constraint $H_C = \langle t_i, t_j, k \rangle$ *specifies that if* t_i *is in state* (C_i^k, P_i)*,* t_j *must be in state* (C_j^k, P_j) *where* k *has equivalent meaning to* k *in* H_P *and* $|C_i| = |C_j|$*.*

Exclusion constraints model configurations that are forbidden and they can be used to model, in a compact way, mutually exclusive subsystems, i.e., when one is operating the other is switched off. Consider, for example, two message streams m_1 and m_2 that are mutually exclusive. This is stated with $X = \langle m_1, m_2 \rangle$ meaning that $P_1 \geq 0 \Rightarrow P_2 = -1$ and vice versa. Notice that switching off both subsystems is still possible, i.e., $P_1 = -1 \wedge P_2 = -1$. The coherency constraints model tasks and messages of the same subsystem that must change coherently, e.g., keeping always a similar period or even a similar ratio as with oversampling. Only configurations that meet the constraints are allowed, so after applying these constraints to the QoS space Ω we obtain the subspace $\Omega^{H \cup X} \subseteq \Omega$.

Finally, there must be at least one configuration, called *nominal* and represented by $F = \{t_i, m_j, i = 1..|\mathcal{T}|, j = 1..|\mathcal{M}|\}$, which guarantees safe operation, even with degraded performance, no matter the operational system state. This configuration must be schedulable and fulfill the exclusion and coherency constraints. It represents the minimum resources needed by all subsystems for safe operation. It is likely that in this configuration all, or several, non-critical functions will be deactivated, which will only run when some critical functions can be temporarily switched off or executed at lower than nominal QoS, thus releasing the required resources.

3.2 Design problem

Following from the previous section, the final configurations space, Ω^R, available for safe on-line use, is the intersection between $\Omega^{H \cup X}$ and Ω^S and, by definition, $F \in \Omega^R$. In other words:

$$\Omega^R \equiv \Omega^{H \cup X} \cap \Omega^S \text{ and } F \in \Omega^R$$

Figure 2 illustrates this relationship. It also allows to formulate our design problem. Our goal is two fold: on one hand to find all configurations, or as many as possible, within the space Ω^R, and on the other hand, to establish a mode switching logic that assures continued safety. Achieving these two goals allows maximizing efficiency in using system resources, because we will have more configurations to choose from in order to find one that better exploits the resources available at each instant.

In practice, there are two approaches to find configurations within the Ω^R space: (1) generate specific configurations at run-time according to requests managed by an on-line QoS management policy that integrates admission control or (2) generate safe configurations off-line and encoding them in memory. These two approaches have been designated, in Section 1, as the dynamic scheduling-based and the static scheduling-based, respectively. Nevertheless, both are normally sub-optimal in the sense that, generally, the actual space of configurations they generate, $\Omega^{R'}$, is just a subset of Ω^R (Fig. 2). This is due to the pessimism of the schedulability analysis used within the former approach and the space limitations in the generation and encoding of all possible configurations in the latter.

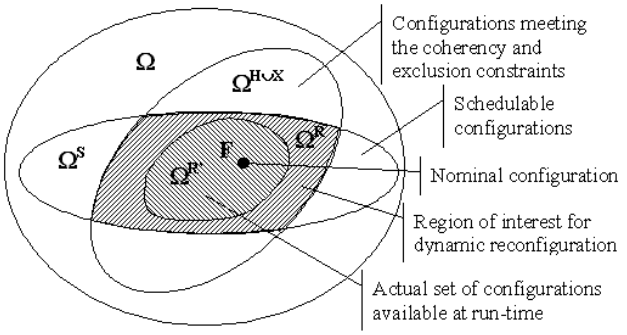

Figure 2: Constraining the configurations space to ensure safety

3.3 QoS management policy

It was referred above that one of the goals of this work is to establish a mode switching logic that assures continued safety. This is a fundamental goal since many of the configurations within $\Omega^{R'}$ will improve resource efficiency in certain system states [18], because resources usage will be tuned to actual needs, but may be unsafe for other system states in which resource needs might be different. The referred mode switching logic, which is part of the QoS management policy (μ), is responsible for tracking the current system state and force the adequate configuration changes (Fig. 3).

In each configuration, each task can be in one of two QoS states, *forced* or *managed*, being thus integrated into the respective set. These sets are defined as follows:

- **Forced QoS set**, set of tasks whose current QoS state has been explicitly set to their low values as a response to reconfiguration events generated aside the QoS management policy.
- **Managed QoS set**, set of tasks whose current QoS state is controlled by the QoS management policy, which tries to provide the best QoS possible to all such tasks with the level of resources currently available and considering any existing relative importance or value as well as the X and H constraints.
- **Reconfiguration event**, event that causes the transition of tasks from one of the sets defined above to the other.

To better illustrate these definitions consider again the ABS example in a car. During normal driving and after a certain period of stable conditions and no use of the braking function, the ABS, or part of it, is switched off (in the sense that certain functions will cease being invoked, despite remaining latent). This issues a reconfiguration event to move the associated tasks and messages from the *managed QoS* set to the *forced* one, releasing the associated resources that are then used by the QoS manager to improve the QoS of the tasks currently in the managed set. Later, there is a sudden need for braking, e.g. when the driver touches the brakes pedal), and thus another reconfiguration event is issued to move those tasks and messages back into the managed set to have the ABS system fully active again. In this case,

given the higher importance of this function, the QoS manager will allocate to it the required resources, switching off all tasks and messages with exclusion constraints, e.g., the Cruise Control system, and further reducing the QoS of less critical functions or even switching off non-critical ones if needed.

It is clear that the QoS management policy must be fast enough to allow prompt reconfigurations, mainly when responding to requests for awaking critical functions. The maximum time to carry out a QoS redistribution must, thus, be specified and used to select an adequate QoS management policy. These policies can be of different types, from fixed value-based, in which higher valued entities receive available resources before lower valued ones, to dynamic policies, such as the elastic model, in which the available resources are distributed among managed entities in a more even way, possibly weighted with a parameter [10]. This issue will be revisited in Section 4 but a thorough coverage of QoS management policies, including the strategy used to define the actual switching instant for replacing the period of a specific task, for adding a new task (previously switched off) or removing a running task (switching it off), is beyond the scope of this paper.

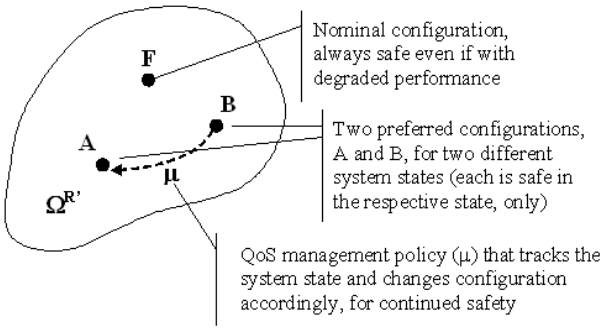

Figure 3: The role of the QoS management policy (μ) to ensure safety.

4. DESIGN FLOW

The design flow associated with the scheme proposed in this paper is shown in Fig. 4. Steps 1 and 2 deal with the system requirements analysis and constratints specification. They provide the necessary information to expand all possible configurations in terms of tasks and messages parameters, as carried out in Step 3. Step 4 consists on executing a schedulability analysis, as accurate as possible, to filter out any non-schedulable configurations. If this schedulability test is necessary and sufficient then all schedulable configurations, and only them, will pass it, thus generating the full safe configurations space (Ω^R). On the other hand, if the test is just sufficient then a few schedulable configurations might be left out by the test, thus leading to a subset of all safe configurations ($\Omega^{R'}$).

Finally, Step 5 deals with the definition of the QoS management policy, including the definition of an integrated schedulability test to be used on-line, which assures that the configurations generated at run-time are all schedulable. It is this test that defines the final configurations space that will actually be used ($\Omega^{R'}$). Therefore, in the general

1. Analyze system requirements
   (a) Deduce tasks and messages
   (b) Deduce their periods, execution and transmission times, modes (for each subsystem)
2. Specify constraints H and X, and the nominal vector F.
3. Expand all possible configurations considering all exclusion and coherency constraints ($\Omega^{H \cup X}$).
4. Analyze off-line the schedulability of all the configurations within $\Omega^{H \cup X}$ and define the configurations space (Ω^R or $\Omega^{R'}$).
5. Define the QoS management policy (μ) that will establish the switching logic to control the changes of configuration.
   (a) Define a suitable on-line schedulability test that is fast and accepts as many configurations as possible from those selected in Step 4, redefining the configuration space $\Omega^{R'}$.

Figure 4: Proposed design flow

case, it is possible to skip Step 4 since the exclusion of potentially non-schedulable configurations is carried out on-line. However, we show, in this paper, that using both tests, off-line and on-line, might lead to an improved efficiency of the latter, namely using the concept of local utilization bound presented in discussed in Section 5.

5. THE LOCAL UTILIZATION BOUND

When deciding about which schedulability test to use on-line, the utilization-based bounds stand up as an attractive solution because of the simplicity of the calculations involved and thus the low latency imposed by their use. A good example of a utilization bound is the one for EDF scheduling with preemptive independent task systems and D=P, which is necessary and suficient. However, this is not the general case and all other similar tests, for different polices or task models, are typically poor in terms of efficiency, generating substantial levels of pessimism thus rejecting several schedulable configurations.

One problem associated with those bounds is that they are general in the sense that they were devised to be applied to task sets generated with any relative offset, period and execution time, i.e., an infinite space of possibilities. In our case, we want to apply a schedulability test to a well defined space of configurations, only, namely those in the space $\Omega^{H \cup X}$ (Fig. 2). Therefore, we conjecture that the utilization threshold for schedulability within this space will be more favourable than in the general case. The reasoning behind is that the configurations with the most unfavourable parameters, in general, will not be present in the set $\Omega^{H \cup X}$. If they unfortunately are, then the schedulability utilization threshold will be equal to the general one, but this seems rather infrequent in most practical designs.

We call such bound a *local utilization bound* for schedulability (LUB^{Ω}) and notice that the validity of such bound is naturally confined to the configurations space (Ω) from

which it was derived. Different spaces will exhibit different bounds. The schedulability test for a set of tasks $\mathcal{T}$ will then be of the form:

$$\sum_{i=1}^{|\mathcal{T}|} U_i = \sum_{i=1}^{|\mathcal{T}|} \frac{C_i}{P_i} \leq \text{LUB}^{\Omega} \quad (6)$$

The determination of LUB^{Ω} is carried out by off-line analysis, within Step 4 of the design flow shown in Fig. 4. All configurations within $\Omega^{H \cup X}$ are sorted in ascending order of utilization and their schedulability is determined using an analysis with good accuracy. Three regions can normally be identified, a region of lower utilization in which all configurations are schedulable, a mid utilization region in which there are schedulable and non-schedulable configurations mixed together, and finally a higher utilization region with non-schedulable configurations, only (Fig. 5). These are called the guaranteed schedulability, mixed schedulability and guaranteed non-schedulability regions, respectively. With necessary and sufficient conditions, the mixed schedulability region will not exist. LUB^{Ω} will then be the highest utilization of the configurations in the guaranteed schedulability region, i.e. the highest utilization below which all configurations are schedulable, thus providing a utilization-based schedulability test that is correct by construction.

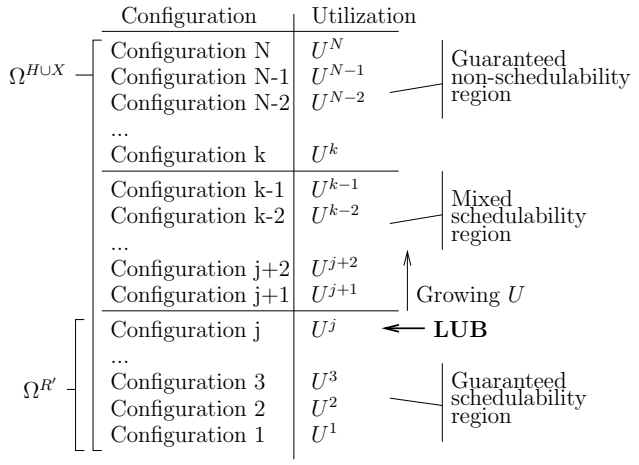

Figure 5: The local utilization bound.

In any case, the final efficiency of the local utilization bound depends on the particular configurations space (Ω) and on the accuracy of the off-line schedulability analysis carried out in Step 4 of the design flow. In fact, we can say that the local utilization bound approximates, from below, the efficiency of the schedulability analysis used off-line.

One relevant aspect is the time taken by the off-line analysis, given the potentially large dimension of the configurations space. However, for many practical cases it is likely that many tasks and messages will be related to each other through exclusion and coherency constraints reducing drastically the whole (raw) configuration space, i.e., $|\Omega^{H \cup X}| << |\Omega|$. Moreover, there are new analysis proposed recently that are particularly efficient and fast, such as those proposed in [3] for fixed priorities systems. Finally, if the configurations space still grows beyond what can be handled off-line, it is always possible to disable the reconfiguration of some subsystems and make them operate at an adequate fixed QoS level.

6. EXAMPLE

To allow a better understanding of the model exposed before, namely the respective design flow and the local utilization bound concept, we present the following illustrative example. Let us consider a distributed system in which the execution of tasks and transmission of messages is time-triggered and is under the control of a QoS management scheme that keeps the load on the nodes and network under adequate limits. The global QoS management is a complex and important topic that, nevertheless, is beyond the scope of this paper. For the sake of simplicity we focus on the following set of tasks ($\mathcal{T}$) executing on one particular node using fixed priority preemptive scheduling. Time is expressed in an arbitrary time unit. The different QoS levels are enforced with different periods.

$$\begin{aligned}
&\mathcal{T} \equiv \{t_i\,(C_i, P_i, Pr_i)\,, i = 1..12\} \equiv \{\\
&t_1\,(0.25, [2, 8]\,, 1)\,,\\
&t_2\,(0.2, [2, 8]\,, 2)\,,\\
&t_3\,(0.3, [-1, 2]\,, 3)\\
&t_4\,(0.15, [-1, 2]\,, 4)\\
&t_5\,(0.25, [3, 4]\,, 5)\\
&t_6\,(0.2, [3, 4]\,, 6)\\
&t_7\,(0.25, 2, 7)\\
&t_8\,(0.25, 2, 8)\\
&t_9\,(0.1, [-1, 2]\,, 9)\\
&t_{10}\,(0.2, [-1, 4]\,, 10)\\
&t_{11}\,(0.15, [-1, 4]\,, 11)\\
&t_{12}\,(0.5, [-1, 4]\,, 12)\\
&\}
\end{aligned}$$

For all messages, offsets are not considered, i.e., taken as arbitrary, and $D = P$. The exclusion and coherency constraints are the following $X \equiv \{\langle t_3, t_4 \rangle\}$, $H \equiv \{\langle t_1, t_2 \rangle, \langle t_5, t_6 \rangle\}$. The nominal configuration F corresponds to the following periods:

$$F \equiv \left[P^F_{1..12}\right] = [8, 8, 2, -1, 4, 4, 2, 2, -1, -1, -1, -1]$$

The total QoS space $\Omega^{H \cup X}$ has 192 configurations with different period combinations. Figure 6 then shows the sets that result from the LUB. The set $\Omega^{R'}$ contains the configurations from 0 to 180. The remaining 12 configurations are beyond the bound; this set can be characterized as $\Omega^{H \cup X} \setminus \Omega^S$. This figure shows the total utilization of each configuration as well as its schedulability assessment. The solid impulses mark schedulable configurations, the dashed ones are non-schedulable. The total utilization varies between $U^{min} = 0.4188$ and $U^{max} = 1.0375$. The nominal configuration is not shown but has index 22 with utilization $U^F = 0.5687$.

The schedulability assessment is carried out using the worst-case response time analysis for fixed priorities preemptive scheduling [2]. Thus, we use the following equation to derive upper bounds to the worst-case response time of each task, assuming the task set sorted by decreasing priorities.

$$\begin{aligned}
R_i^{wc} &= I_i^{wc} + C_i\\
I_i^{wc} &= \textstyle\sum_{j<i} \lceil \frac{R_i^{wc}}{P_j} \rceil \times C_j
\end{aligned}$$

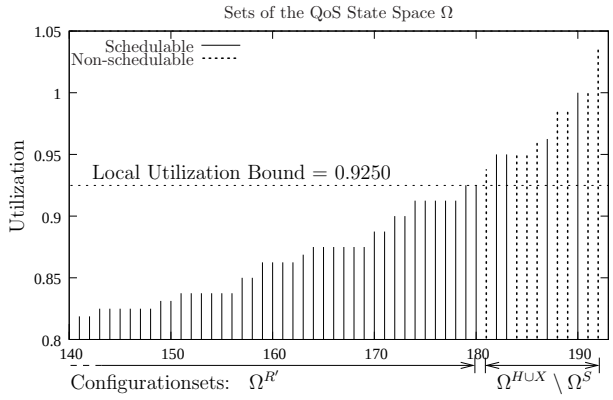

Figure 6: Sets $\Omega^{R'}$ and $\Omega^{H\cup X} \setminus \Omega^S$ in relation to the local utilization bound of the example.

A trivial schedulability test is then carried out by checking whether $\forall t_i \in \mathcal{T} : R_i^{wc} < D_i$. This allows defining the local utilization bound (LUB^Ω) as shown there in, too, which equals 0.925. To give a notion of the relative efficiency achievable with this bound, we can compare it with the Liu and Layland's utilization bound for Rate-Monotonic, which computes to 0.7241 for a set of 8 tasks[1]. This corresponds to an extra 20% of CPU utilization that are made available by the local utilization bound with respect to Liu and Layland's bound, thanks to the higher efficiency inherited from the response-time schedulability test carried out off-line. The whole test, including expansion of the configurations space, application of exclusion and coherency constraints and response-time analysis, was carried out using Matlab and took approximately 80ms in a 1.7GHz Centrino laptop with energy management disabled.

Once the local bound is established, the associated schedulability test (as in Eq. 6) can be integrated within the QoS management policy (μ) to be used on-line. As an example, consider that a prioritized QoS management policy is used, according to which the available bandwidth is preferably assigned to the higher priority messages, unless they are in the *forced QoS* set.

Therefore, suppose that at a given instant the system is operating in the following configuration, with total utilization 0.925, i.e., just at the local bound (LUB^Ω).

$$[P_{1..12}] = [2, 2, -1, 2, 3, 3, 2, 2, 2, 4, -1, 4]$$

Notice that tasks $t_{3,4}$ are mutually exclusive and task t_{11} is in the forced QoS set, i.e., there was a previous reconfiguration event to switch it off. Then, at a given instant, there is a new reconfiguration event to switch on task t_3, which has higher priority than t_4. This event will cause t_4 to be switched off due to their mutual exclusion. However, swapping tasks $t_{3,4}$ causes the total utilization to grow to 1, i.e., 0.075 above the local bound. Therefore, the QoS management policy starts reducing the QoS of the tasks in managed QoS set, starting from the lower priority ones and until enough bandwidth is accumulated. This requires switching off task t_{12}, releasing a bandwidth of 0.125, which is enough to accomodate the request leading to the following configuration, with a total utilization of 0.875.

$$[P_{1..12}] = [2, 2, 2, -1, 3, 3, 2, 2, 2, 4, -1, -1]$$

[1]Notice that the number of active tasks varies dynamically but we consider an average of 8 being active simultaneously among the 12 that compose the task set

7. CONCLUSIONS AND FUTURE WORK

Dynamic QoS management is appearing as a possible framework to develop more flexible and more resource efficient embedded systems, particularly networked ones. However, the achieved level of flexibility may conflict with timeliness and safety requirements. Therefore, this paper proposes a framework to deploy dynamic QoS management with constrained flexibility in order to harmonize the referred attributes. In this respect, the paper proposes defining a confined configuration space and a switching logic that assures that a safe configuration is always used and, at the same time, the resources usage is maximized, particularly CPU and network bandwidth.

In order to support a prompt switching between configurations, a fast on-line QoS management policy must be used, which must include a fast schedulability analysis. Utilization-based schedulability tests are fast but normally inneficient. Therefore, the paper proposed the concept of Local Utilization Bound (LUB), i.e., a bound that applies to a confined configurations space, only, avoiding pernicious general configurations and resulting in substantially higher efficiency. Such bound must be determined off-line for each particular configurations space. An example is shown that highlights the use of dynamic QoS management based on the concept of the local utilization schedulability bound.

Nevertheless, there are still issues to be further explored in future work, namely the use and comparison of different local utilization bounds, e.g., considering blocking, non-preemption and inserted idle-time, their integration with different dynamic QoS management policies, the use of the proposed approach in several practical case studies and, also, within a holistic system design perspective, considering simultaneously task scheduling in the nodes and message scheduling in the network. Finally, the proposed framework is also being integrated with static scheduling-based approaches, as referred in Sections 1 and 2, facilitating the formal verification of the nominal configuration, the switching to and from it, and the possibility to include in the configurations space those configurations that are schedulable but are left out by the residual pessimism in the schedulability analysis used within the dynamic scheduling based approach followed in this paper.

8. ACKNOWLEDGMENTS

This work was carried out during a visit of the first author to the University of Pennsylvania which was partially supported by Fundação Calouste Gulbenkian, Portugal, and by the European Commission (NoE ARTIST2, IST-2-004527). The other authors were supported in part by NSF CNS-0509327, NSF CNS-0509143, ARO W911NF-05-1-0182, and OEAW APART-11059.

9. REFERENCES

[1] L. Almeida. A Word for Operational Flexibility in Distributed Safety-Critical Systems. *Proceedings of IEEE Workshop on Object-Oriented, Real-Time and Dependable Systems (WORDS 2003)*, 2003.

[2] A. N. Audsley, A. Burns, M. Richardson, and K. Tindell. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8:284–292, 1993.

[3] E. Bini and S. Baruah. Efficient computation of response time bounds under fixed-priority scheduling. In *Proc. of 15th International Conference on Real-Time and Network Systems - RTNS'07)*, March 2007.

[4] B. Bouyssounouse and J. Sifakis. *Embedded Systems Design: The ARTIST Roadmap for Research and Development*, volume 3436 of *Lecture Notes in Computer Science*. Springer, 2005.

[5] G. C. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni. Elastic scheduling for flexible workload management. *IEEE Transactions on Computers*, 51(3):289–302, 2002.

[6] M. Calha and J. Fonseca. Data Streams: an Analysis of the Interactions Between Real-Time Tasks. In *Proc. IEEE Conference on Emerging Technologies and Factory Automation - ETFA'05)*. IEEE Computer Society, September 2005.

[7] S. Fischmeister, O. Sokolsky, and I. Lee. Network-Code Machine: Programmable Real-Time Communication Schedules. In *Proc. of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*, 2006.

[8] J. Li, Y. Song, and F. Simonot-Lion. Providing real-time applications with graceful degradation of qos and fault tolerance according to (m, k)-firm model. *IEEE Transactions on Industrial Informatics*, 2(2):112–119, May 2006.

[9] C. Lu, J. Stankovic, G. Tao, and S. Son. Feedback control real-time scheduling: Framework, modeling and algorithms. *Special issue of RT Systems Journal on Control-Theoretic Approaches to Real-Time Computing*, 23(1/2):85–126, Jul/Sept 2002.

[10] P. Pedreiras and L. Almeida. The Flexible Time-Triggered (FTT) Paradigm: An Approach to QoS Management in Distributed Real-Time Systems. In *Proceedings of the Workshop on Parallel and Distributed Real-Time Systems (WPDRTS'03)*, pages 123–130, 2003.

[11] R. Racu, R. Ernst, and K. Richter. The need of a timing model for the autosar software standard. In *Proc. of 1st Int. Workshop on Models and Analysis for Automotive Systems (WMAAS'06)*, December 2006. Satellite of IEEE RTSS 2006.

[12] K. Ramamritham and J. Stankovic. Scheduling strategies adopted in spring: An overview. In A. von Tilborg and G. Koob, editors, *Foundations of Real-Time Computing: Scheduling and Resource Management*, pages 277–306. Kluwer Academic Publishers, 1991.

[13] J. Real and A. Crespo. Mode change protocols for real-time systems: A survey and a new proposal. *Real-Time Systems*, 26:161–197, 2004.

[14] D. Schmidt, R. Schantz, M. Masters, J. Cross, D. Sharp, and L. DiPalma. Towards Adaptive and Reflective Middleware for Network-Centric Combat Systems, CrossTalk, November. 2001. `http://www.cs.wustl.edu/~schmidt/PDF/crosstalk.pdf`; accessed February 21, 2005., 2001.

[15] C. Shelton and P. Koopman. Improving system dependability with functional alternatives. In *Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN'04)*, page 295. IEEE Computer Society, 2004.

[16] C. Shelton, P. Koopman, and W. Nace. A framework for scalable analysis and design of system-wide graceful degradation in distributed embedded systems. In *Proc. of Workshop on Object-oriented, Real-time and Dependable Systems (WORDS'03)*. IEEE Press, January 2003.

[17] V. Silva, R. Marau, L. Almeida, J. Ferreira, M. Calha, P. Pedreiras, and J. Fonseca. Implementing a distributed sensing and actuation system: The CAMBADA robots case study. *Proceedings of ETFA'2005* 10^{th} *EEE International Conference on Emerging Technologies and Factory Automation, T8 Intelligent Robots and Systems transaction*, September 2005.

[18] E. A. Strunk and J. C. Knight. Dependability through assured reconfiguration in embedded system software. In *Proc. IEEE Conf on Dependable Systems and Networks - DSN'05*, June 2005.

[19] TTTech. Time-Triggered Protocol TTP/C High-Level Specification Document (edition 1.0). *http://www.ttagroup.org*, 2002.