Dynamic QoS Adaptation of Inter-Dependent Task Sets in Cooperative Embedded Systems

Luís Nogueira, Luís Miguel Pinho
IPP Hurray Research Group
School of Engineering, Polytechnic Institute of Porto, Portugal

E-mail: {luis,lpinho}@dei.isep.ipp.pt

Abstract

Due to the growing complexity and dynamism of many embedded application domains (including consumer electronics, robotics, automotive and telecommunications), it is increasingly difficult to react to load variations and adapt the system's performance in a controlled fashion within an useful and bounded time. This is particularly noticeable when intending to benefit from the full potential of an open distributed cooperating environment, where service characteristics are not known beforehand and tasks may exhibit unrestricted QoS inter-dependencies.

This paper proposes a novel anytime adaptive QoS control policy in which the online search for the best set of QoS levels is combined with each user's personal preferences on their services' adaptation behaviour. Extensive simulations demonstrate that the proposed anytime algorithms are able to quickly find a good initial solution and effectively optimise the rate at which the quality of the current solution improves as the algorithms are given more time to run, with a minimum overhead when compared against their traditional versions.

Categories and Subject Descriptors

J.7 [**Computers in Other Systems**]: Real time; D.1.3 [**Programming Techniques**]: Concurrent Programming—*Distributed programming*

General Terms

Algorithms, Management, Performance

Keywords

Open real-time systems, Anytime algorithms, QoS optimisation and adaptation, Service stability

1. Introduction

In open and dynamic real-time systems, resource requirements are inherently unstable and difficult to predict in advance, as independently developed services enter and leave the system at anytime [6]. As a consequence, the overall system's workload is subject to significant variations, which can result in an overload and degrade the entire system's performance in an unpredictable fashion. This situation is particularly critical for small embedded devices used in consumer electronics, telecommunication systems, industrial automation, and automotive systems. In fact, in order to satisfy a set of constraints related to weight, space, and energy consumption, these systems are typically built using small microprocessors with low processing power and limited resources.

For most of these systems, the classical real-time approach based on a rigid off-line design and worst-case assumptions would keep resources unused for most of the time. Online methods that react to load variations and adapt the system's performance in a controlled fashion overcome these shortcomings and provide the needed flexibility. To cope with dynamic environments, a system must be adaptive, that is, it must be able to adjust its internal strategies in response to changes in the environment in order to keep the system's performance at a desired level.

In this context, a cooperative execution of resource intensive services among neighbour nodes seems a promising solution. The CooperatES (Cooperative Embedded Systems) framework [11, 14, 15] facilitates the cooperation among neighbours when a service request cannot be satisfyingly answered by a single node. Nodes dynamically group themselves into a new coalition, allocating resources to each new service and establishing an initial Service Level Agreement (SLA) that maximises the satisfaction of the QoS constraints associated with the new service and minimises the impact on the global QoS caused by the new service's arrival [11, 14]. However, our previous work assumed that tasks sets were

independent. This paper extends our approach, assuming that services share resources [16] and their execution behaviour and input/output qualities are interdependent, i.e., a constraint on one quality or resource parameter can constrain other system's parameters.

Throughout the paper, two important requirements are considered to be essential for an efficient adaptive QoS control: the stability of service provisioning and the timeliness criteria that affects the usefulness of the system's adaptation behaviour. In fact, while some users or applications may prefer to always get the best possible instantaneous QoS, independently of the reconfiguration rate of their requested services, others may find that frequent QoS reconfigurations are undesirable [13]. This paper proposes a QoS reconfiguration policy that compares each possible service upgrade against the user's stability requirements, namely a minimum utility increment and a minimum granted stability period to upgrade the currently provided QoS level.

Furthermore, if the system adapts too late, it may not be useful and may even be disadvantageous. As the complexity of open real-time systems increases, it is also increasingly difficult to find an optimal resource allocation that deals with both users' and nodes' constraints within an useful and bounded time. This is true for many real-time applications, where it may be preferable to have approximate results of a poorer but acceptable quality delivered on time to late results with the desirable optimal quality. For example, it is better for a collision avoidance system to issue a timely warning together with an estimated location of the obstacle than a late description of the exact evasive action. Another example is video and sound processing. While poorer quality images and voices on a timely basis may be acceptable, late frames and long periods of silence often are not. Other examples can be found in route optimisation of automated vehicles [19, 18], computer games [7], and real-time control [2].

This paper proposes a novel anytime approach [22] to deal with a large number of dynamic tasks, multiple resources, and real-time operation constraints in open dynamic real-time systems. The increased complexity of negotiating service provisioning for task sets which exhibit unrestricted QoS dependency relations among their tasks makes it beneficial to be able to trade the quality of the achieved solution against its computation cost. The proposed anytime algorithms can be interrupted at any time and still provide a solution and a measure of its quality, which is expected to improve as the run time of the algorithms increases. This flexibility in the algorithms' execution times enables the system to timely adapt to the changing environmental conditions of dynamic open real-time systems. To the best of our knowledge, no other approaches exist to provide this adaptation, considering inter-dependent task sets.

2. The CooperatES framework

The goal of the CooperatES framework is to satisfy multiple QoS dimensions in a resource constrained environment forming temporary coalitions for a cooperative service execution among nodes. Each node has a significant degree of autonomy and it is capable of executing services and sharing resources with other nodes. A service can be executed by a single node or by a group of nodes, depending on the node's capabilities and on the user's imposed quality constraints. In either case, the service is processed in a transparent way for the user, as users are not aware of the exact distribution used to solve the computationally expensive service.

This paper considers the existence of dependencies among QoS dimensions and/or resource types, extending the work presented in [14]. Such dependency relations specify that a task offers a certain level of QoS under the condition that some specified QoS will be offered by the environment or by other tasks. For example, network bandwidth can be traded for processing power by using data compression techniques. Some compression techniques may not be lossless, thus, they may impact the quality of the output. Dependencies are modelled as a directed graph G_{ij}, where each graph node represents a task and the edges represent the data flow between the tasks.

A service $S_i = \{w_{i1}, w_{i2}, \ldots, w_{in}\}$ is then a collection of one or more work units w_{ij} that can be executed at varying levels of QoS to achieve an efficient resource usage that constantly adapts to the devices' specific constraints, nature of executing tasks and dynamically changing system conditions. Each work unit $w_{ij} = \tau_{i1}, \tau_{i2}, \ldots, \tau_{in}$ is a set of one or more tasks τ_{ij} that must be executed in the same node due to local dependencies. Correct decisions on service partitioning are made at run time when sufficient information about the workload and communication requirements become available [20].

It is assumed that applications provide different levels of service with distinct utilities and resource requirements. For example, consider the transmission of multiple audio/video streams over a network. This scenario involves a network with a given bandwidth and nodes serving and receiving the streams. Typical audio related parameters are the sampling rate (8, 16, 24, 44, 48 kHz), the sampling bits (8, 16), and the end-to-end latency (100, 75, 50, 25 ms), while in video are usually considered the picture dimension (SQCIF, QCIF, CIF, CIF4), colour depth (1, 3, 8, 16, ...) and frame rate (1, ..., 30).

Different configurations of a stream can have different utility values for different users and applications. For example, for a particular user a transmission of a music concert may place higher quality requirements on audio, although colour video may be also desirable, while another user of a remote surveillance system may require higher video quality with a minimum of gray scale images. The CooperatES framework allows a user's service request to be formulated through a relative decreasing order of importance of a set of QoS dimensions, their attributes, and possible values [11]. Users provide a single specification of their own range of QoS preferences Q_i for a complete service S_i, ranging from a desired QoS level $L_{desired}$ and the maximum tolerable service degradation, specified by a minimum acceptable QoS level $L_{minimum}$, without having to understand the individual work units that make up the service. As a result, the user is able to express acceptable compromises in the desired QoS and assign utility values to QoS levels. Note that this assignment is decoupled from the process of establishing the supplied service QoS levels themselves and determining the resource requirements for each level.

Given the spectrum of the user's acceptable QoS levels, if the service request cannot be satisfyingly answered by a single node, a cooperative execution request is broadcasted. Neighbours respond

to this cooperation request by recomputing their local set of QoS levels in order to accept a work unit of the new service. Although the general multiple QoS dimensions and multiple resources optimisation problem is NP-hard, the CooperatES framework offers a multi-attribute utility theory to achieve the best possible service configuration within a given and bounded time [14]. This paper proposes a QoS optimisation process that iteratively selects the resource allocation that maximises the satisfaction of the QoS constraints Q_i associated with the new service S_i and minimises the impact on the current QoS of previously accepted services. The proposed anytime algorithm can be interrupted at any time and provide a solution and a measure of its quality, which is expected to increase as the algorithm is given more time to run.

Having a set of service proposals, the dynamic selection of the new coalition's members is also influenced by the user's QoS preferences, tailoring the provided service to each user's specific needs [14]. The coalition allocates resources to the new service establishing an initial SLA which contains a service description whose parameters are within the range of the user's desired QoS level $L_{desired}$ and the maximum tolerable service degradation $L_{minimum}$.

Nevertheless, short term dynamic environmental changes impose that the promised SLA can never be more than an expectation of a best-effort service quality during long term periods [5]. Once a SLA is admitted, it may be downgraded to a lower QoS level (until $L_{minimum}$ is reached) in order to accommodate new service requests with a higher utility or upgraded (until $L_{desired}$ is reached) when the needed resources become available. This paper proposes a novel adaptive anytime QoS control method in which the online search for the best set of QoS levels is combined with the users' service stability preferences. While QoS downgrades can be mandatory in order to accommodate new services with a higher utility, upgrades to a higher QoS level are controlled by the user's stability requirements.

Concurrent QoS negotiations and adaptations are supported by the CooperatES framework. When a node recomputes its set of local SLAs, promised resources are pre-reserved until the node is notified about its service proposal's acceptance or rejection (or a timeout expires). This means that the amount of pre-reserved resources will not be available to other subsequent service negotiations when a new service arrives or when determining the possible service upgrades until the former negotiation's result is known. Note that the currently provided QoS levels only actually change at the end of a successful negotiation. If the service proposal is rejected, the reserved amounts are immediately released and the node continues to supply the offered SLAs prior to the failed cooperative negotiation. On the other hand, if the service proposal is accepted, the node imposes the new set of SLAs to all local services, assigning the pre-reserved resource amounts to services.

With several independently developed applications with different timing requirements coexisting in the same system, it is important to guarantee a predictable performance under specified load and failure conditions, and ensure a graceful degradation when those conditions are violated [3]. This is strictly related to the capacity of controlling the incoming workload, preventing abrupt and unpredictable degradations and achieving isolation among services, providing service guarantees to critical applications [1]. The Capacity Sharing and Stealing (CSS) scheduler [15] extends the resource reservation approach by proposing to handle overloads with additional capacity that is available from two sources: (i) by reclaiming unused allocated capacity when jobs complete in less than their budgeted execution time; and (ii) by stealing allocated capacities to non-isolated servers used to schedule sporadic best-effort jobs. The integration of the CSS scheduler into the CooperatES framework is discussed in detail in [12].

3. Supporting QoS adaptation and stability

We consider the users' influence on the stability of service provisioning to be essential in the dynamic QoS arbitration among competing services [13]. While some users or applications may prefer to always get the best possible instantaneous QoS, independently of the reconfiguration rate of their requested services, others may find that frequent QoS reconfigurations are undesirable. This suggests that while the system may not be able to avoid to downgrade the currently provided QoS level of some service in order to accommodate new services with a higher utility in a resource constrained environment, upgrades to a higher QoS level can and should be controlled by each user's stability requirements.

Possible attributes for such QoS stability dimension can be a minimum granted stability period Δ_{min} and a minimum increment in the service's reward U_{min} in order to upgrade the current service's QoS level. These can be interpreted as "do not change to a better service's quality state unless this gives me at least a reward's increment of U_{min} over a Δ_{min} period". The flexibility and expressiveness of the QoS scheme proposed in [11] allows the user's stability preferences and their relative order of importance to be expressed as any other QoS dimension [11].

These stability constraints will rule the possible service upgrades. The system starts by computing the possible SLA upgrades (see Section 5 for details) and forecasting the granted stability period (Section 3.1) for an entire service S_i. If the user's stability preferences are met, the current service's QoS is upgraded to the new determined level. Otherwise, the service is kept in its current QoS level and the pre-reserved resource amounts become immediately available for subsequent QoS negotiations.

3.1 Promised stability periods

From the service provider's side, each proposed SLA should now be complemented with a stability period Δ_t, indicating that during that specific time interval the promised QoS level for a work unit w_{ij} of service S_i will be assured either on the arrival and departure of other services.

Note that service stability could be achieved by using a fixed, large enough value for Δ_t, but this would then result in lack of responsiveness in adaptability to environmental changes. Furthermore, fixed values only make sense when there is some knowledge about the tasks' traffic model, which is not the case in open real-time systems. Proposed stability periods should then be periodically updated in response to variations in the tasks' traffic flow and correspondent resource usage, efficiently adapting the system's behaviour to the observed environmental changes.

Time series analysis comprises methods that attempt to under-

stand a sequence of data points, typically measured at successive times, spaced at (often uniform) time intervals to make forecasts. The simple exponential smoothing (SES) model [4] has become very popular as a forecasting method for a wide variety of time series data as it is both robust and easy to apply. In fact, empirical research by Makridakis et al. [9] has shown SES to be the best choice for one-period-ahead forecasting, from among 24 other time series methods and using a variety of accuracy measures. Thus, regardless of the theoretical model for the process underlying the observed time series, simple exponential smoothing will often produce quite accurate forecasts.

Intuitively, past data should be discounted in a more gradual fashion, putting relatively more weight on the most recent observations. SES accomplishes exactly such weighting, with exponentially smaller weights being assigned to older observations. We use the SES model to forecast the length of the next stability period for a service S_i by combining the forecasts for each of the system's resources r_i it uses.

Equation 1 is used recursively to update (forecast) the smoothed series as new observations are recorded for each resource r_i. The observed minimum stability period for resource r_i during the period of observation t is denoted by x_t and $\Delta_t^{r_i}$ may be regarded as the best estimate of what the next value of x will be.

$$\Delta_t^{r_i} = \alpha x_t + (1-\alpha)\Delta_{t-1}^{r_i} \quad (1)$$

Each new forecast is then based on the previous forecast plus a percentage of the difference between that forecast and the actual value of x_t at that point. The percentage $0 \leq \alpha \leq 1$ is known as the smoothing factor. Values of α close to 1 have less of a smoothing effect and give a greater weight to recent changes in the data, while values of α closer to 0 have a greater smoothing effect and are less responsive to recent changes. α can then be adjusted by the system's designer to create a more reactive or conservative response to recent changes in the tasks' traffic flow. Alternatively, a statistical technique may be used to optimise the value of α, minimising the difference between the predicted and observed values. For example, the method of least squares may be used to determine α's value for which the sum of the quantities $(\Delta_{t-1}^{r_i} - x_t)^2$ is minimised [4].

Having the stability forecasts for each of the system's resources, the promised stability period for a work unit of a particular service S_i must be based on a coherent summary of the forecasts for each of the resources it uses. Any use of an arithmetic summarisation function that combines the values (such as a mean), will provide an incorrect stability period due to relative scaling. On the other hand, combination of several dynamical variables using logical operators has already been proposed to provide more expressive policies for SLAs [17].

Equation 2 determines the promised stability period for service S_i by aggregating the forecasted values for each of the resources r_i it uses through the fuzzy AND operator (the min function). It allows a quick and simple evaluation of stability periods for each locally accepted service and leads to a correct system behaviour.

$$\Delta_t = min(\Delta_{r_1}\ AND\ \Delta_{r_2}\ AND\ \ldots\ AND\ \Delta_{r_n}) \quad (2)$$

Having forecasted a stability period Δ_t, the system ensures that during the forecasted period no change occurs in the promised QoS level for service S_i. Nevertheless, services whose stability period has already expired can be downgraded to a lower quality level to accommodate new services with a higher utility or can be upgraded when the needed resources become available. These issues will be discussed in detail in the next sections.

4. Formulating an initial SLA

Requests for a cooperative service execution may arrive at any time. To guarantee the local execution of a work unit w_{ij} of the new requesting service S_i, each node executes a local QoS optimisation that aims to maximise the provided QoS level for the arriving work unit, while minimising the impact on the current QoS of the previously accepted services.

Each service request has associated a set of user-imposed QoS preferences Q_i, expressed in decreasing preference order. Each $Q_{kj}^i = \{Q_{kj}^i[0], \ldots, Q_{kj}^i[n-1]\}$ is a finite set of n quality choices for the j^{th} attribute of the k^{th} QoS dimension associated with service S_i.

Services share resources and their execution behaviour and input/output qualities are interdependent, i.e., a constraint on one quality or resource parameter can constrain other system's parameters. This means that the negotiation process must ensure that a source task provides a QoS which is acceptable to all consumer tasks and lies within the QoS range supported by the source task. As such, the system may have to adapt the quality of individual services according to some inter-service QoS dependencies when searching for the best overall service utility. The increased complexity of such negotiation makes it beneficial to propose an anytime approach that can trade the achieved solution's quality by its computational cost to ensure a timely answer to events.

Based on the new service's data flow graph G_i and on its set of inter-dependency relations $Deps_i$, the proposed anytime QoS optimisation algorithm (Algorithm 1) tracks QoS dependencies and propagates the performed changes in one attribute to all local affected attributes at each iteration. If, by following the chain of dependencies, the algorithm finds a task that is already in its list of resolved dependencies, a deadlock is detected and the service proposal formulation is aborted.

In order to be useful in practice, an anytime approach must try to quickly find a sufficiently good initial proposal and gradually improve it if time permits, conducting the search for a better feasible solution in a way that maximises the expected improvement in the solution's quality [22]. As such, the proposed QoS optimisation algorithm starts by keeping the QoS levels of previously accepted services and selects the lowest requested QoS level for the new tasks in w_{ij} that complies with any eventual QoS dependency with currently executing tasks. Note that this is the service configuration with the highest probability of being feasible without degrading the current level of service of previously accepted

tasks.

Algorithm 1 Formulate an initial SLA

Let τ^p be the set of previously accepted tasks
Let τ^e be the set of tasks whose stability period Δ_t has expired
Let $\tau^* = \tau^p \cup w_{ij}$ be the new set of tasks

Step 1: Improve the QoS level of each task $\tau_a \in w_{ij}$
Select $Q_{kj}[n]$, the lowest requested level of service for all k QoS dimensions, considering the dependencies with the previously accepted tasks τ^p, for all newly arrived tasks τ_a in w_{ij}
Keep the current QoS level of previously accepted tasks τ^p
while the new set of local tasks τ^* *is* feasible **do**
  for each task $\tau_a \in w_{ij}$ **do**
    for each attribute without dependencies with τ^p receiving service at $Q_{kj}[m] > Q_{kj}[0]$ **do**
      Upgrade attribute to the next possible value $Q_{kj}[m-1]$
      Follow attribute's dependencies in w_{ij} and change values accordingly
      Determine the utility increase of this upgrade
    end for
  end for
  Find task τ_{max} whose reward's increase is maximum and perform upgrade
end while

Step 2: Find the local minimal service degradation in τ^* **to accommodate each** $\tau_a \in w_{ij}$
while the new set of local tasks τ^* *is not* feasible **do**
  for each task $\tau_i \in \tau^e \cup w_{ij}$ receiving service at $Q_{kj}[m] > Q_{kj}[n]$ **do**
    for all QoS attributes **do**
      Degrade attribute j to the previous possible value $Q_{kj}[m+1]$
      Follow dependencies of attribute j in all local tasks τ^* and change values accordingly
      Determine the utility decrease of this downgrade
    end for
  end for
  Find task τ_{min} whose reward's decrease is minimum and perform downgrade
end while
return new local QoS optimisation

After quickly determining this initial service solution, the search of a better solution is guided, at each iteration, by the maximisation of the new service's QoS level and by the minimisation of the QoS degradation of the previously accepted services. When w_{ij} can be accommodated without degrading the QoS of the previously accepted tasks, the configuration that maximises the reward's increase for w_{ij} is selected (Step 1). On the other hand, when QoS degradation is needed to accommodate w_{ij}, the algorithm incrementally finds the minimal service degradation for the previously accepted services until a feasible solution is found.

At each iteration, the quality of the proposed solution can be measured by considering the reward achieved by the new arriving work unit $r_{w_{ij}}$, the impact on the provided QoS of the n previously accepted work units r_{τ^p} and the value of the previous generated feasible configuration Q'_{conf} (Equation 3). Initially, Q'_{conf} is set to zero and its value is only updated if the iteration's solution is feasible.

$$Q_{conf} = \left(r_{w_{ij}} * \frac{\sum_{i=0}^{n} r_{\tau^p}}{n} \right)^{(1-Q'_{conf})} \tag{3}$$

Rewards are computed by considering the proximity of each SLA with respect to the weighted user's QoS preferences expressed in decreasing relative order using Equation 4.

$$reward(S_i) = 1 - \sum_{j=0}^{\forall Q_{jk} < Q_{best_j}} \beta_j * penalty_j \tag{4}$$

The $0 \leq penalty \leq 1$ parameter can be fine tuned by the system's administrator and its value should increase with the distance to the user's preferred value for a particular quality attribute. The expressed relative order of preference determines the weight β of each dimension, encoding user's preferences in a qualitative way.

The algorithm can be interrupted at any time as a consequence of the dynamic nature of the environment [15, 16], or finishes when it finds a feasible set of QoS configurations whose quality cannot be further improved, or when it finds that even if all the tasks would be served at the lowest admissible QoS level it is not possible to accommodate the new requesting tasks in w_{ij}. In this later case, the service request is rejected and the previously accepted tasks continue to be served at their current QoS levels.

The proposed anytime QoS optimisation algorithm always improves or maintains the current solution's quality as it has more time to run. This is done by keeping the best feasible solution found so far, if the result of each iteration is not always proposing a feasible service configuration for the new task set. However, each intermediate configuration, even if not feasible, is used to calculate the next solution, minimising the search effort.

The next simple example denotes this behaviour. Admit that the algorithm runs to completion or it is interrupted after its fifth iteration (Table 1). With this set of iterations, the algorithm would return the solution found at the fifth iteration rather than the second one, since it is the one with the greatest quality for the new service under negotiation. The second solution would only be returned as the best feasible solution if the algorithm was interrupted before it was able to complete its fifth iteration.

Nevertheless, at the end of the QoS optimisation process, a measure of the node's global reward can be computed by combining the rewards of the n computed SLAs (Equation 5).

Iteration	Q_{conf}	Feasible?
1^{st}	$(0.1 * 0.8)^{(1-0)} = 0.08$	yes
2^{nd}	$(0.2 * 0.8)^{(1-0.08)} = 0.185$	yes
3^{rd}	$(0.3 * 0.8)^{(1-0.185)} = 0.313$	no
4^{th}	$(0.3 * 0.75)^{(1-0.185)} = 0.297$	no
5^{th}	$(0.3 * 0.7)^{(1-0.185)} = 0.280$	yes

Table 1. Iteratively QoS optimisation

$$R = \frac{\sum_{i=1}^{n} reward(S_i)}{n} \quad (5)$$

Note that unless all services are executed at their highest requested QoS level, there is a difference between the current node's global reward $R_{current}$ and the maximum theoretical global reward R_{max}. This difference can be caused by either resource limitations, which is unavoidable, or poor load balancing. The later can be improved by using these global rewards in the nodes' selection for the new cooperative coalition [11]. Selecting the node with a higher global reward for service proposals with a similar QoS level, not only maximises a particular user's satisfaction with the provided service, but also maximises the global system's utility, since a higher local reward clearly indicates that the node's previous set of tasks had to suffer less QoS degradation in order to accommodate the new tasks in w_{ij}.

4.1 Properties

Not every algorithm that can produce a sequence of approximate results is a well behaved anytime algorithm [22]. The conformity of the proposed anytime service proposal formulation algorithm with the desired properties of anytime algorithms is checked in the next paragraphs.

PROPERTY 4.1.1 (MEASURABLE QUALITY). *The solution's quality can be determined precisely*

Proof: Equation 3 determines the quality of the proposed set of SLAs by considering the proximity of the service proposals with respect to the user's request under negotiation and the impact of that proximity on the utility of previously accepted services.

□

PROPERTY 4.1.2 (RECOGNISABLE QUALITY). *The solution's quality can be easily determined at run time*

Proof: The utility of a feasible set of SLAs is determined using the rewards achieved by the selected service configurations. With Equation 4, computing the reward achieved by each SLA is strait forward and time-bounded.

□

PROPERTY 4.1.3 (MONOTONICITY). *The solution's quality is a nondecreasing function of time*

Proof: The algorithm's first solution proposes a SLA for the new service with the minimum requested QoS level. Admit that this is a feasible initial solution and let σ_1 be its quality.

With spare resources (Step 1), the solution's quality is incrementally increased as the new service's reward increases, since the current QoS level of the previously accepted services is not changed. As such, at each iteration, the new feasible solution's quality σ_{new} is always greater than σ_1.

If, by increasing the new service's QoS level, an unfeasible set of SLAs is found, the incremental search of a better feasible solution σ_{next} is guided by the minimisation of the service degradation of the previously accepted services (Step 2). Let σ_u be the quality of an unfeasible set of SLAs.

Since a set of SLAs can only be considered useful within a feasible set of tasks, the algorithm always returns the best found feasible solution σ_{new}, rather than the last determined solution σ_u, if interrupted before computing σ_{next}.

According to Zilberstein [22], this characteristic in addition to a recognisable quality is sufficient to prove the monotonicity of an anytime algorithm.

□

PROPERTY 4.1.4 (CONSISTENCY). *For a given amount of computation time on a given input, the quality of the generated service configuration is always the same*

Proof: For a given amount of computation time Δt on a given input of a set of QoS constraints Q associated with a set of tasks τ, the quality of the determined set of SLAs is always the same, since the selection of services' attributes to improve or degrade is deterministic.

At each iteration, the QoS attribute to be upgraded is the one that maximises the reward's increase for the new arrived service, while the QoS attribute to be downgraded is the one that minimises the decrease in the rewards of the previously accepted tasks. As such, the algorithm guarantees a deterministic output quality for a given amount of time and input.

□

PROPERTY 4.1.5 (DIMINISHING RETURNS). *The improvement in the quality of the generated set of SLAs is larger at the early stages of the computation and it diminishes over time*

Proof: From the measurable quality property, we know that Equation 3 determines the quality of the proposed set of SLAs by considering the proximity of the service proposals with respect to the new user's request under negotiation and the impact of that proximity on the utility of previously accepted services.

When searching for a better feasible solution, the algorithm always improves the current solution's quality by upgrading the attribute that results in the greatest reward increase for the new service of by downgrading the attribute that minimises the global reward's decrease. As such, the improvement in the solution's quality is larger at the first iterations and this improvement's increase diminishes over time.

□

PROPERTY 4.1.6 (INTERRUPTIBILITY). *The algorithm can be stopped at any time and still provide a solution*

Proof: Let t' be the time needed to generate the first feasible solution. In interrupted at any time $t < t'$ the algorithm will return an empty service configuration for the new service, resulting in zero quality.

When stopped at time $t > t'$ the algorithm returns the best feasible set of SLAs generated until time t. Note that this may not be the last generated set of SLAs.

□

PROPERTY 4.1.7 (PREEMPTIBILITY). *The algorithm can be suspended and resumed with minimal overhead*

Proof: Since the algorithm keeps the best generated feasible solution found until time t and the current set of SLAs, it can be easily resumed after an interrupt.

□

5. Reconfiguring SLAs

As detailed in the previous section, the arrival of new services with higher utility may result in a QoS downgrade of previously accepted services. Since the goal is to maximise the provided QoS level for each of the existing services, it makes sense to reconfigure previously downgraded SLAs when the needed resources become once again available.

However, our basic viewpoint is that service stability can be more important for some users than some momentary maximum quality that does not take into consideration the services' reconfiguration rate. Although the framework ensures a fixed quality level during a dynamically determined period of time, an upgrade of the current quality level should be done according to each user's stability preferences.

Let L_t be the desired system's resource usage threshold to activate the dynamic QoS reconfiguration of previously degraded tasks whose stability period Δ_t has already expired. Let L be the current level of the system's load demanded by the n offered SLAs. Intuitively, $L < L_t$ indicates an underutilisation and the dynamic QoS reconfiguration will take place.

Possible QoS upgrades of previously downgraded SLAs are determined by Algorithm 2, which are then compared against the users' stability constraints. The anytime QoS reconfiguration algorithm tries to restore the initially provided SLAs by selecting, at each iteration, the new configuration that achieves the greatest reward increase.

Algorithm 2 Determine possible upgrades

```
Let τ^d be the set of previously downgraded tasks whose Δ_t has expired
Let τ^o be the set of all other tasks
Let Q_kj[init] be the initial quality level for attribute j of the k_th QoS dimension for task τ_i ∈ τ^d
Keep the current QoS level for all tasks in τ^o
while the new set of configurations is feasible do
  for each task τ_i ∈ τ^d do
    for each attribute without dependencies with τ^o receiving service at Q_kj[m] > Q_kj[init] do
      Upgrade attribute to the next possible value m − 1
      Follow attribute's dependencies and change values accordingly
      Determine the utility increase of this upgrade
    end for
  end for
  Find task τ_max whose reward's increase is maximum and perform upgrade
end while
```

The achieved solution's quality at each iteration is measured by Equation 6, considering the rewards achieved by the new SLAs for the previously downgraded tasks r_{τ^d} and the value of the previous generated feasible configuration Q'_{conf}. Initially, Q'_{conf} is set to zero and its value is only updated if the achieved solution is feasible.

$$Q_{conf} = \left(\frac{\sum_{i=0}^{n} r_{\tau^d}}{n} \right)^{(1-Q'_{conf})} \tag{6}$$

The algorithm can be interrupted at any time or finishes when it finds a feasible set of QoS configurations whose quality cannot be further improved due to resource limitations or all the initially granted SLAs are reached. A careful analysis of Algorithm 2 and Equation 6 allows us to conclude that the same desirable properties of anytime algorithms, checked for Algorithm 1 in Section 4.1, can be deduced for this dynamic QoS reconfiguration algorithm.

If Algorithm 2 produces a new set of upgraded SLAs, an actual upgrade of each of the currently provided SLAs only occurs if the

user's stability requirements, namely a minimum granted stability period Δ_{min} and a minimum increment in the SLA's reward U_{min}, are met. Clearly, as these constraints are stringent, it is harder to upgrade to better quality levels.

6. Analysis and Evaluation

The behaviour of the proposed anytime local QoS optimisation and adaptation algorithms in dynamic open real-time scenarios was evaluated through extensive simulations, with a special attention being devoted to introduce a high variability in the characteristics of the conducted simulations.

An application that captures, compresses and transmits frames of real-time data to end users, which may use a diversity of end devices and have different sets of QoS preferences, was used as a scenario for the simulations. The application was composed by a set of source units to collect the data, a compression unit to gather and compress the data that came from the multiple sources, a transmission unit to transmit the data over the network, a decompression unit to convert the data into each user's specified format, and an user unit to display the data in the user's end device.

The number of simultaneous nodes in the system varied from 10 to 100 while the number of simultaneous users varied from 1 to 20, generating different amounts of load and resource availability in the system. Each node was running a prototype implementation of the CooperatES framework, with a fixed set of mappings between requested QoS levels and resource requirements. The code bases needed to execute each of the streaming application's units was loaded a priori in all the nodes.

The characteristics of end devices and their more powerful neighbour nodes was randomly generated, creating a distributed heterogeneous environment. This non-equal partition of resources affected the ability of some nodes to singly execute some of the application's units and has driven nodes to a cooperative service execution.

Requested QoS levels were randomly generated, at randomly selected end devices and at randomly generated times, expressing the spectrum of acceptable QoS levels in a qualitative way, ranging from a randomly generated desired QoS level to a randomly generated maximum tolerable service degradation. The relative decreasing order of importance imposed in dimensions, attributes and values was also randomly generated. Similarly, interdependency QoS relations among tasks were randomly generated for each service.

The QoS domain used to generate the users' service requests was composed by the following list of QoS dimensions, attributes, and possible values:

```
QoS dimensions: {Container,Video,Audio}
Container: {format}
Video: {colour depth,frame size,frame rate}
Audio: {sampling rate,sample bits}
format: {3GP,ASF,AVI,QuickTime,RealVideo,WMV}
colour depth: {1,3,8,16,24}
frame size: {240x180,320x240,640x480,720x480,
            1024x768, 1280x1024}
frame rate: {[1,30]}
sampling rate: {8,11,32,44,88}
sample bits: {4,8,16,24}
```

The reported results were observed from multiple and independent simulation runs, with initial conditions and parameters, but different seeds for the random values used to drive the simulations, obtaining independent and identically distributed variables, with a reasonably good statistical performance [8]. The random values were generated by the Mersenne Twister algorithm [10].

6.1 Performance profiles

Anytime algorithms correlate the output's quality with time in a performance profile [22], a function that maps the time given to an anytime algorithm (and in some cases also input quality) to the quality of the algorithm's produced solution. Since there are many possible factors affecting the execution time of an algorithm, rather than measuring the algorithms' absolute execution time on every simulation run, we have normalised it with respect to its completion time [21]. As such, in the next figures the algorithms' computation times are represented as a percentage of their respective completion times. Nevertheless, both algorithms needed an average time lower than 1 second to compute their optimal solutions on a Intel Core Duo T5500 at 1.66 G Hz.

A first study evaluated the behaviour of the anytime QoS optimisation algorithm, executed on the arrival of a new service request, by measuring its performance profile as well as the impact generated by the arrival of a new service on the QoS level of previously accepted tasks. Recall from Section 4 that the reward of a specific proposal measures how useful it will be for a particular user and that the local reward expresses a degree of global satisfaction for all the users that have tasks being executed at a particular node.

The results were plotted by averaging the results over several independent runs of the simulation, divided in two categories. Figure 1 presents the scenario where the average amount of available resources per node is greater than the average amount of resources demanded by the services being executed. The opposite scenario is represented in Figure 2, where the average amount of resources per node is smaller than the average amount of demanded resources.

In Figure 1, the increase in the solution's quality Q_{conf} results from the increase in the new task's reward (Step 1 of the algorithm). Recall that with spare resources the QoS levels of previously accepted tasks remains the same. As such, this increase in the new service's reward also increases the node's local reward, that was affected by the initially proposed solution of serving the new arrived service at the minimum requested QoS level. However, due to resource limitations (Figure 2), when trying to upgrade the reward achieved by the new service, the generated configuration may result in an unfeasible set of SLAs. The Step 2 of service formulation algorithm is then executed in order to try to find a new feasible solution that presents a higher satisfaction for the service request under negotiation.

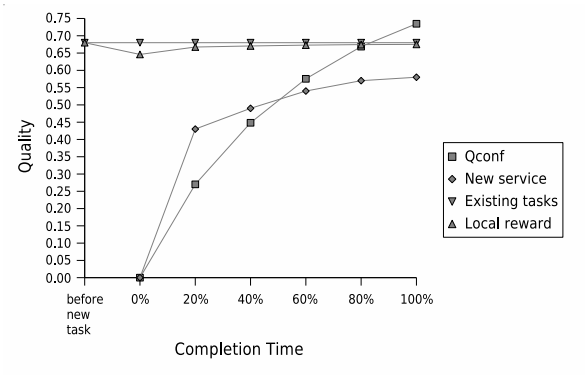

Figure 1. Initial SLA with spare resources

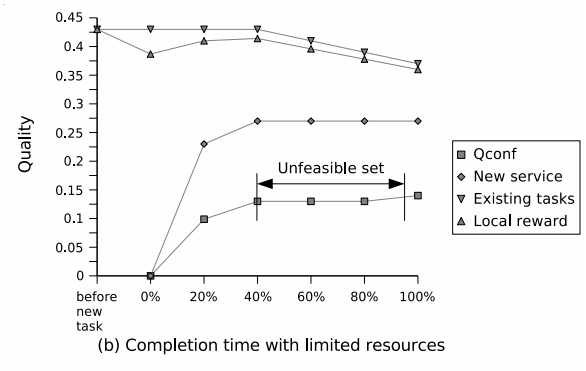

Figure 2. Initial SLA with limited resources

Note that, in both scenarios, the proposed algorithm optimise the rate at which the quality of the current solution improves over time. With spare resources (Figure 1), at only 20% of the computation time, the solution's quality for the new arrived task is near 74% of the achieved quality at completion time. When QoS degradation is needed to accommodate the new task (Figure 2), its service proposal achieves 85% of its final quality at 20% of computation time.

Also note that, the solution's quality, identified by $Qconf$ in both figures, quickly approaches its maximum value at an early stage of the computation. The diminishing returns property was then empirically verified, after is formal analysis in Section 4.1. The same is true for the monotonicity property, since the best feasible configuration is only replaced if, and only if, another feasible solution is found and has a higher quality for the user's request under negotiation (Figure 2).

Another study evaluated the behaviour of the anytime QoS adaptation algorithm. During the conducted simulations, when a system's utilisation below 60% was detected, an upgrade of the currently provided SLAs whose stability period had already expired was done. Promised stability periods were determined by taking into consideration the observed variations in the tasks' traffic flow and correspondent resource usage, adapting the system to the observed environmental changes. The value of the smoothing factor α was optimised using the method of least squares. The average solution's quality increase was measured and the results are plotted in Figure 3. Similar conclusions regarding the desirable properties of anytime algorithms can be taken with respect to the proposed QoS adaptation algorithm.

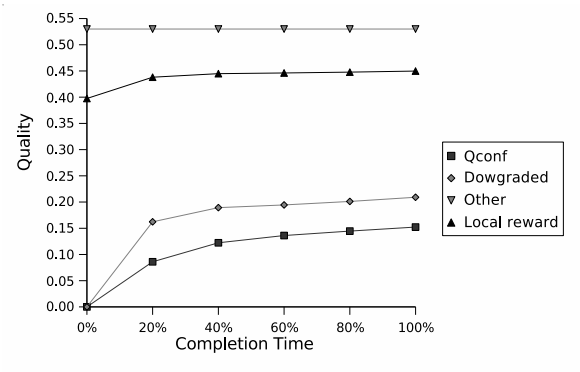

Figure 3. Dynamic QoS reconfiguration

Both studies clearly demonstrate that the proposed anytime algorithms can be useful even when there is no time to compute an optimal local resource allocation. The solution's quality can be improved if the algorithms have more time to run, but it rapidly approaches its final value at an early stage of the needed computation time.

6.2 Adaptation behaviour

A third study evaluated the users' influence on the services' adaptation behaviour. Three permanent service requests were added to the dynamic traffic that was randomly generated at each simulation run. The same randomly generated spectrum of acceptable QoS values, in the same decreasing preference order, was used in the three requests. The requests only differed on the users' QoS stability constraints for the minimum utility increase and stability period, $User_1 = \{0, 0s\}$, $User_2 = \{0.2, 10s\}$, $User_3 = \{0.3, 30s\}$, respectively.

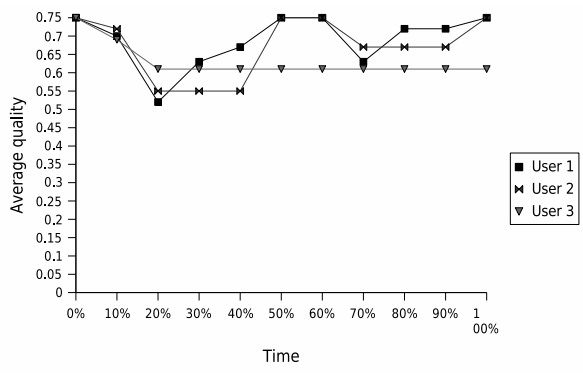

Figure 4. QoS reconfiguration rate

The influence of personal constraints on the system's adaptation behaviour is clearly observable in Figure 4. As the user's constraints for a service upgrade are harder to achieve there is less probability to change and stay in a better quality level. These results clearly demonstrate that the users' influence can be extended to the services' adaptation behaviour.

6.3 Overhead

A fourth study compared the computational cost of the proposed anytime approach when compared against the traditional version of the algorithms to reach their optimal solutions. The results obtained by the local QoS optimisation are reported in the next paragraphs. Similar results were obtained for the proposed QoS adaptation mechanism.

The traditional version of a local QoS optimisation approach proposed in [11] was extended to resolve any QoS dependencies present in its optimal solution and used in this comparison. The algorithm starts by selecting the user's preferred QoS level for the new service and stops when it finds a feasible solution that minimises the impact on the provided global level of service caused by the new service's arrival. The comparison's results were normalised with respect to the completion time of the longest solution and plotted in Figures 5 and 6.

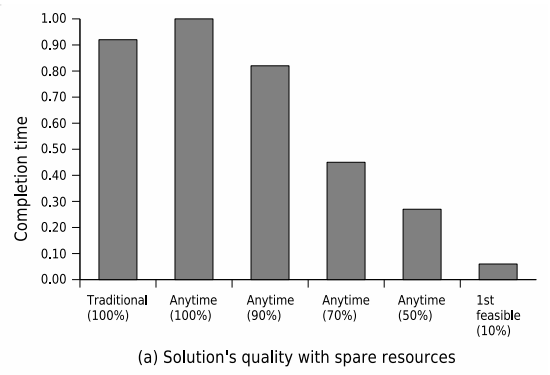

Figure 5. Needed time with spare resources

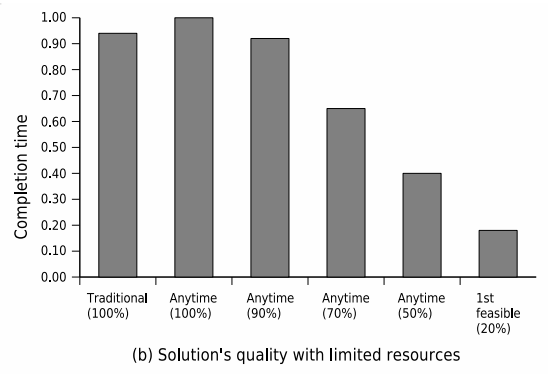

Figure 6. Needed time with limited resources

Both figures show that the anytime version can take more time to achieve the same optimal solution in both scenarios. Two main reasons explain this difference. First, the anytime version resolves QoS dependencies at each iteration. Recall that the goal is to be able to interrupt the algorithm at any time and still be able to return a valid solution. Without any restriction on the needed time to compute its optimal solution, the traditional version only has to resolve any eventual dependencies after finding the best configuration for the individual tasks. Dependencies were resolved by relaxing the optimal values of some of the individual tasks to the maximum allowed value, according to the constraints of the inter-dependency relations. Second, the different approaches used to achieve an optimal solution can have an impact on the number of needed iterations, particularly with spare resources. Since the anytime version tries to quickly find a feasible solution, it starts by considering the worst requested QoS values for the new service and iteratively improves that solution until the optimal one is found. On the other hand, the traditional version starts by trying to provide the best requested level of service for the new tasks and iteratively degrades all tasks, stopping when it finds a feasible, optimal solution.

Nevertheless, in both scenarios the anytime version is by far quicker to find a feasible solution. With spare resources, the first feasible solution with a quality near 10% of its optimal value is almost immediately found, and at near 20% of the running session the solution's quality is already around 50% of its optimal value. With limited resources, the anytime version takes about 20% of its computation time to reach a feasible solution with 20% of its optimal solution's quality, and at near 40% of the running session it achieves 50% of its optimal value. These results, in addition to the performance profiles plotted in Figures 1 and 2, further validate the ability of the proposed anytime algorithm to quickly find a feasible solution and maximise the improvement in the expected solution's quality at each iteration.

7. Conclusions

Adaptive real-time is a relatively new approach to embedded systems design that allows an online reaction to load variations, adapting the system to the specific constraints of devices and users, nature of executing tasks and dynamically changing environmental conditions. This ability is essential to efficiently manage the provided Quality of Service (QoS) in domains such as telecommunication, consumer electronics, industrial automation, and automotive systems. However, due to the growing complexity and dynamism of these systems, it is increasingly difficult to determine an optimal resource allocation within an useful time.

This paper proposes anytime QoS optimisation and adaptation algorithms that can trade off the needed computation time by the achieved solution's quality. Both algorithms are able to quickly find a sub-optimal solution at an early stage of the computation, which are then iteratively refined as the algorithms are given more time to run. Such flexibility allows a higher adaptation to the dynamically changing conditions of open real-time embedded systems and, as the achieved results demonstrate, can be obtained with an overhead that can be considered negligible when compared against the introduced benefits.

Particular attention was devoted to maximise each user's influence on the services' adaptation behaviour during execution. Upgrades of provided QoS levels are done by comparing each user's stability requirements against the system's dynamically forecasted stability periods and possible utility increments. Promised stability periods are periodically updated in response to fluctuations in the tasks' traffic flow, relating observations of past and present environmental conditions. The achieved results clearly demonstrate that such influence can be achieved.

Acknowledgements

The authors would like to thank the anonymous referees for their helpful comments. This work was supported by FCT (CISTER Research Unit - FCT UI 608 and CooperatES project - PTDC/ EIA/ 71624/ 2006), and by the European Commission through the ArtistDesign NoE (IST-FP7-214373).

8. References

[1] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the 19th IEEE RTSS*, page 4, Madrid, Spain, December 1998.

[2] R. Bhattacharya and G. J. Balas. Anytime control algorithm: Model reduction approach. *Journal of Guidance, Control, and Dynamics*, 27(5):767–776, October 2004.

[3] S. A. Brandt and G. J. Nutt. Flexible soft real-time processing in middleware. *Real-Time Systems*, 22(1):77–118, 2002.

[4] R. G. Brown. *Smoothing, forecasting and prediction of discrete time series*. Prentice-Hall, Englewood Cliffs, NJ, 1963.

[5] M. Burgess. On the theory of system administration. *Science of Computer Programming*, 49:1, 2003.

[6] S. Ghosh, R. R. Rajkumar, J. Hansen, and J. Lehoczky. Integrated resource management and scheduling with multi-resource constraints. In *Proceedings of the 25th IEEE Real-Time Systems Symposium*, pages 12–22, Lisbon, Portugal, December 2004.

[7] N. Hawes. *Anytime Deliberation for Computer Game Agents*. PhD thesis, School of Computer Science, The University of Birmingham, November 2003.

[8] A. M. Law and W. D. Kelton. *Simulation modeling and analysis*. McGraw-Hill, 3rd edition, 2000.

[9] S. Makridakis, A. Andersen, R. Carbone, R. Fildes, M. Hibon, R. Lewandowski, J. Newton, E. Parzen, and R. Winkler. The accuracy of extrapolation (time series) methods: Results of a forecasting competition. *Journal of Forecasting*, 1:111–153, 1982.

[10] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998.

[11] L. Nogueira and L. M. Pinho. Dynamic qos-aware coalition formation. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, page 135, Denver, Colorado, April 2005.

[12] L. Nogueira and L. M. Pinho. Building adaptable, qos-aware dependable embedded systems. In *Proceedings of the 3rd International Workshop on Dependable Embedded Systems*, pages 72–77, Leeds, United Kingdom, October 2006.

[13] L. Nogueira and L. M. Pinho. Dynamic adaptation of stability periods for service level agreements. In *Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 77–81, Sydney, Australia, August 2006.

[14] L. Nogueira and L. M. Pinho. Iterative refinement approach for qos-aware service configuration. *IFIP From Model-Driven Design to Resource Management for Distributed Embedded Systems*, 225:155–164, 2006.

[15] L. Nogueira and L. M. Pinho. Capacity sharing and stealing in dynamic server-based real-time systems. In *Proceedings of the 21th IEEE International Parallel and Distributed Processing Symposium*, page 153, Long Beach,CA,USA, March 2007.

[16] L. Nogueira and L. M. Pinho. Shared resources and precedence constraints with capacity sharing and stealing. In *Proceedings of the 22th IEEE International Parallel and Distributed Processing Symposium*, page 97, Miami,Florida,USA, April 2008.

[17] G. Rodosek. Quality aspects in it service management. In *Proceedings of the 13th IFIP/IEEE Internation Workshop on Distributed Systems: Operations and Management*, pages 82–93, Montereal, Canada, October 2002.

[18] J. Shackleton, D. Cofer, and S. Cooper. Anytime scheduling for real-time embedded control applications. In *Proceedings of the 23rd Digital Avionics Systems Conference*, volume 2, pages 101–110, Salt Lake City, UT, USA, October 2004.

[19] J. van den Berg, D. Ferguson, and J. Kuffner. Anytime path planning and replanning in dynamic environments. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2366– 2371, Orlando, Florida, USA, May 2006.

[20] C. Wang and Z. Li. Parametric analysis for adaptive computation offloading. In *Proceedings of the ACM SIGPLAN 2004 Conference on Programming Language Design and Implementation*, pages 119–130. ACM Press, 2004.

[21] S. Zilberstein. *Operational Rationality Through Compilation of Anytime Algorithms*. PhD thesis, Department of Computer Science, University of California at Berkeley, 1993.

[22] S. Zilberstein. Using anytime algorithms in intelligent systems. *Artificial Inteligence Magazine*, 17(3):73–83, 1996.