Computing and Informatics, Vol. 34, 2015, 23–44

PASSIVE FAULT-TOLERANCE MANAGEMENT IN COMPONENT-BASED EMBEDDED SYSTEMS

Luís Nogueira

CISTER/INESC-TEC Research Centre
School of Engineering, Polytechnic Institute of Porto, Portugal
e-mail: `lmn@isep.ipp.pt`

Jorge Coelho

LIACC Research Centre – University of Porto
School of Engineering, Polytechnic Institute of Porto, Portugal
e-mail: `jmn@isep.ipp.pt`

Abstract. It is imperative to accept that failures can and will occur even in meticulously designed distributed systems and to design proper measures to counter those failures. Passive replication minimizes resource consumption by only activating redundant replicas in case of failures, as typically, providing and applying state updates is less resource demanding than requesting execution. However, most existing solutions for passive fault tolerance are usually designed and configured at design time, explicitly and statically identifying the most critical components and their number of replicas, lacking the needed flexibility to handle the runtime dynamics of distributed component-based embedded systems. This paper proposes a cost-effective adaptive fault tolerance solution with a significant lower overhead compared to a strict active redundancy-based approach, achieving a high error coverage with a minimum amount of redundancy. The activation of passive replicas is coordinated through a feedback-based coordination model that reduces the complexity of the needed interactions among components until a new collective global service solution is determined, hence improving the overall maintainability and robustness of the system.

Keywords: Component-based systems, embedded real-time systems, coordination model, fault-tolerance, passive replication

Mathematics Subject Classification 2010: 68M15

1 INTRODUCTION

The possibility of partial failures is a fundamental characteristic of distributed applications [22], even more so in open environments where the mix of independently developed applications and their aggregate resource and timing requirements are unknown until runtime. This becomes evident at the moment when services are no longer accessible due to faults or, even worse, when faulty results are provided to users [8, 11, 48].

A sub-domain of reliability, fault tolerance aims at allowing the system to survive in spite of faults, i.e. after a fault has occurred, by means of redundancy. Replication is an effective way to achieve fault tolerance for such type of failure [37, 27] and has some advantages over other fault tolerance solutions in distributed environments, providing the shortest recovery delays, it is less intrusive with respect to execution time, it scales much better, and is relatively generic and transparent to the application domain [19, 49].

In fault-tolerant real-time systems using active replication schemes, where several replicas run simultaneously, has been common [36]. Even if errors are detected in some of the replicas, the non-erroneous ones will still be able to produce results within deadlines. On the negative side, running several replicas simultaneously is costly and can be infeasible or undesirable in distributed embedded systems [7]. Embedded systems, unlike most general-purpose computing systems, often perform computations subject to various constraints, such as processor speed, amount of memory, power consumption, and reaction time [39].

Passive replication [6] minimizes resource consumption by only activating redundant replicas in case of failures, as typically, providing and applying state updates is less resource demanding than requesting the execution. As such, passive replication is appealing for soft real-time embedded systems that cannot afford the cost of maintaining active replicas and tolerate an increased recovery time [4]. Nevertheless, it may still be possible to tolerate faults within deadlines, thus improving the system reliability without using a more resource consuming fault-tolerance mechanism [47].

Furthermore, passive replication can be implemented without the use of complex replica consistency protocols [41, 9] and, in practice, a passive scheme has a simple structure and has no controller switching associated transients. Therefore, the additional real-time computational demand is low for a passive fault-tolerance control scheme [20].

However, most of the existing solutions for passive fault tolerance are usually designed and configured at design time, explicitly and statically identifying the most critical components and their number of replicas, lacking the needed flexibility to handle the runtime dynamics of open distributed real-time embedded systems [41]. Distributed real-time embedded systems often consist of several independently developed components, shared across applications and whose criticality may evolve dynamically during the course of computation. As such, offline decisions on the number and allocation of replicas may be inadequate after the system has been executing for some time already. Even if embedded systems are usually designed to run a well-defined set of applications, the increased complexity and requirements of the latter reduces the possibility to statically analyse and predict application behaviour and thus to apply static optimisations.

This paper is then motivated by the need to develop a cost-effective adaptive fault tolerance solution with an overhead significantly lower as compared to a strict active redundancy-based approach. The term *cost-effective* implies that we want to achieve a high error coverage with the minimum amount of redundancy. The paper proposes low runtime complexity heuristics to

1. dynamically determine which components to replicate based on their significance to the system as a whole;
2. determine a number of replicas proportional to the components significance degree; and
3. select the location of those replicas based on collected information about the nodes availability as the system progresses.

An extensive number of simulation runs was analysed to quantitatively study the effectiveness of the proposed approach. The results show that even simple heuristics with low runtime complexity can achieve a reasonably higher system availability than static offline decisions when lower replication ratios are imposed due to resource or cost limitations.

However, nothing can be said about the behaviour of the system when any failures beyond design basis occur. Consider the case where the quality of the produced output of a particular component depends not only on the amount and type of used resources but also on the quality of the inputs being sent by other components in the system [43]. If a primary replica is found to be faulty, a new primary must be elected from the set of passive backup ones and the execution restarted from the last saved state. However, it is not guaranteed that the new primary will be able to locally reserve the needed resources to output the same QoS level that was being produced by the old primary. In such cases, the need of coordination arises in order to preserve the correct functionality of the distributed service execution [3, 14, 29]. Complex interdependencies may exist among components such that the incorporation of one change can require the inclusion of several others for the change to work correctly. Complex problems may result from these chain reactions like infinite triggering of new adaptations or inconsistent configurations in different components [38], interference between the different self-management behaviours of components, conflicts over shared resources, sub-optimal system performance and hysteresis effects [16].

Coordination is then a key concept for developing self-adaptive distributed systems [18] and a wide spectrum of coordination strategies have been proposed. However, the limited applicability of existing coordination models to heterogeneous distributed real-time systems [15] provides the significant motivation for the development of a decentralised coordination model that also reasons about the duration and overhead of the coordination process. This paper builds upon the work presented in [29] and handles the coordinated activation of passive replicas through a feedback-based coordination model that reduces the complexity of the needed interactions among nodes until a new collective global service solution is determined. By exchanging feedback on the desired or imposed self-adaptive actions, components converge towards a global solution, even if that means not supplying their individually best solutions. As a result, each component, although autonomous, is influenced by, and can influence, the behaviour of other components in the system.

The rest of the paper is organized as follows. Section 2 describes the system model and used notation in the following sections. Section 3 proposes a set of heuristics to determine the number of replicas proportional to each component's significance degree to the overall system and to select the location of those replicas based on collected information about the availability of each node as the system progresses. Section 4 discusses the coordinated activation of passive replicas through a feedback-based coordination model, reducing the complexity of the needed interactions among nodes until a new collective global service solution is determined, while benefiting from a better performance on non-failure cases by only updating the backup replicas state on a failure of a primary one. The properties of the proposed coordination model are formally verified in Section 5. Section 6 shows and discusses, based on the results of extensive simulations, the efficiency of the proposed coordination model. Finally, Section 7 concludes the paper.

2 SYSTEM MODEL

Service-oriented applications increasingly consist of a set of interacting software components $S = \{c_1, c_2, \ldots, c_n\}$ that jointly provide some service to end-users and operate in open dynamic environments [21]. Each component c_i is defined by its functionality, is able to send and receive messages, is available at a certain point of the network, and has a set of QoS parameters that can be changed in order to adapt service provisioning to a dynamically changing environment. Each subset of QoS parameters that relates to a single aspect of service quality is named as a *QoS dimension*. Each of these QoS dimensions has different resource requirements for each possible level of service quality. We make the reasonable assumption that services execution modes associated with higher QoS levels require higher resource amounts.

Components use results produced by other components, that may be running on other nodes, to produce their own output and hence are interdependent. Interdependency relationships among components of a service S can be represented as a directed acyclic graph (DAG) $\mathcal{G}_S = (\mathcal{V}_S, \mathcal{E}_S)$, where each vertex $v_i \in \mathcal{V}_S$ represents a component $c_i \in S$ and a directed edge $e_i \in \mathcal{E}_S$ from c_j to c_k indicates that c_k is functionally dependent on c_j. In this paper, we consider a failure to be when a software component stops producing output.

Within $\mathcal{G}_S = (\mathcal{V}_S, \mathcal{E}_S)$, we give particular names to three types of components. A *source component* models an input device and is not a consumer of the output produced by any other component in the DAG of the service. A *sink component* represents a component that is not a producer of any output consumed by other components in $\mathcal{G}_S$ and models a unit to display the global service output in the end-user's device. The source and sink components mark the limit of a set of components that must be managed in a coordinated manner. Finally, we call *cut-vertex* to a component $c_i \in \mathcal{V}_S$, if the removal of that component divides $\mathcal{G}_S$ in two separate connected graphs. Cut-vertexes may confine coordination operations to a subset of $\mathcal{G}_S$. Within a feasible QoS region, it may be possible to maintain the current output quality by compensating for a decrease in the input quality by an increase in the amount of used resources or vice versa [43].

Each component c_i is only aware of the set of inputs $I_{c_i} = \{(c_j, Q^j_{val}), \ldots, (c_k, Q^k_{val})\}$, describing the quality of all of its inputs coming from precedent components in $\mathcal{G}_S$ and the set of outputs $O_{c_i} = \{(c_l, Q^l_{val}), \ldots, (c_p, Q^p_{val})\}$, describing the quality of all of its outputs sent to successor components in $\mathcal{G}_\mathcal{W}$. As such, no global knowledge is required for coordinating the activation of a backup replica after a failure of a primary component c_i.

3 ADAPTIVE REPLICATION CONTROL

The problem consists in finding a replication scheme which minimizes the probability of failure of the most important components without replicating every software component. This involves the study of mechanisms to determine which components should be replicated, the quantity of replicas to be made, and where to deploy such replicas [24]. As such, the benefit of replication in open dynamic resource-constrained environments is a complex function of the number of replicas, the placement of those replicas, the selected replica consistency protocol, and the availability and performance characteristics of the nodes and networks composing the system. Since replica consistency protocols are relatively well understood [23, 9, 41], we will not consider them in the remainder of this paper.

Thus, assuming that a mechanism exists for keeping passive replicas consistent, how can we make use of passive replication to increase the reliability of distributed resource-constrained embedded systems where it may not be possible to replicate every available component? Our approach is based on a concept of significance, a value associated to each component which reflects the effects of its failure on the overall system. Intuitively, the more a component c_i has other components depending on it, the more it is significant to the system as a whole. Then, the significance degree w_i of a component c_i can be computed as the aggregation of the interdependencies of other components on it, determining the usefulness of its outputs to all components which depend on it to perform their tasks.

More formally, given $S_\mathcal{G} = \{\mathcal{G}_1, \ldots, \mathcal{G}_n\}$, the set of connected graphs of interdependencies between components for a given system, and $\mathcal{O}_{\mathcal{G}_j}(c_i)$, the out-degree of a node $c_i \in \mathcal{G}_j$, the significance wi of c_i is given by Equation (1).

$$w_i = \sum_{k=1}^{n} \mathcal{O}_{\mathcal{G}_k}(c_i) \tag{1}$$

Once the significance of each component to the system has been estimated, the decision on which components to replicate and the correspondent number of passive replicas must be taken. Equation (2) determines a number of replicas for a component c_i which is directly proportional to the component significance degree w_i and to the maximum number of possible replicas $\max_{c_i}$[1], and inversely, proportional to the sum of the significance degree of all components in the system W.

$$n^{c_i} = \left\lceil \frac{w_i * \max_{c_i}}{W} \right\rceil \tag{2}$$

The next step is to determine a strategy for placing those replicas in the network. Consider the effects of placing replicas on unreliable nodes. The resulting unreliability of those replicas will usually require replica consistency protocols to work harder [41], increasing network traffic and processing overheads. Thus, not only will the system performance suffer but its availability may actually decrease, despite the increased number of available components through replication [23]. However, an optimal replica placement in a distributed system can be classified as a NP-hard discrete location problem. Consequently, several heuristic strategies which do not have a guarantee in terms of solution quality or running time, but provide a robust approach to obtaining a high quality solution to problems of a realistic size in a reasonable time have been investigated, independently of the followed replication approach [45, 17]. Nevertheless, it is our belief that static offline approaches are inadequate for open real-time systems, where the environment dynamically changes as the system progresses. As such, a placement of a replica which was correct when a service started may be incorrect after it was executing for some time.

Two gross measures of the reliability of a node are its mean time to failure (MTTF) and its mean time to recovery (MTTR) [24]. We propose to use those measures to dynamically allocate the set of replicas of a component c_i based on the expected availability of nodes in the system. The utility $0 \leq u_k^{r_j^i} \leq 1$ of allocating a passive replica r_j^i of a component c_i to a node n_k is then defined by the probability of its availability during the system execution, given by Equation (3). Utilities range from zero, the value of a completely unavailable node, to one, the value of a totally available node.

$$u_k^{r_j^i} = \frac{MTTF_k}{MTTF_k + MTTR_k} \tag{3}$$

Having the utility of each possible allocation, the probability of failure of a given set of replicas $R_i = r_1^i, r_2^i, \ldots, r_{n^{c_i}}^i$ is determined by Equation (4).

$$F(R_i) = (1 - u_1^i) * (1 - u_2^i) * \ldots * (1 - u_{n^{c_i}}^i) \tag{4}$$

The system will then allocate the set of replicas $R_i = r_1^i, r_2^i, \ldots, r_{n^{c_i}}^i$ such that its probability of failure $F(R_i)$ is minimal among all the possible allocation sets. In

[1] $\max_{c_i}$ is given by the number of nodes in a heterogeneous environment which have the needed type of resources to execute the component c_i.

order to keep this allocation as up-to-date as possible, nodes have to be monitored as the system runs. If reliability of a replica set strays outside a predefined tolerance value a reconfiguration of the allocation set should be performed.

4 COORDINATED ACTIVATION OF PASSIVE REPLICAS

While passive replication is appealing for systems that cannot afford the cost of maintaining active replicas, the requirement to provide both high availability, strong state consistency, and satisfactory response times during the non-failure cases is conflicting in many ways. In fact, response times perceived by applications will depend on the time taken by the primary replica to synchronise its state with that of the slowest backup replica, even if low complexity replica consistency protocols are used [41, 9].

To overcome this limitation there is a possibility for the state of backup replicas to be made consistent only during a failure recovery, which significantly improves response times and saves resources during the non-failure cases. We recognise, however, that extra time must be spent to activate a new primary due to the significantly weaker consistency model. The problem is even more challenging when activating replicas in interdependent component-based systems where the output produced by a component may depend not only on the amount and type of used resources but also on the quality of received inputs. Nevertheless, the complexity of the needed interactions among nodes until a new collective global service solution is determined can be reduced through a feedback-based protocol, while benefiting from a better performance on non-failure cases by only updating the backup replicas state on a failure of a primary one [29].

Ideally, whenever a primary component fails, such backup replica is elected as a new primary that is able to obtain the needed resources to output the same QoS level that was being produced by the old primary replica. However, due to the heterogeneity and dynamically varying workloads of nodes in open embedded systems, it is not guaranteed that at least one of the backups will be able to locally reserve the needed resources to output such a quality level. On the other hand, the failing of a component can be seen as an opportunity to upgrade the output QoS level of the failing component c_i in an attempt to upgrade the global QoS that is being supplied to the user [29]. Note that the failing component could have been running on a device that was forced to downgrade the output QoS of c_i in order to accommodate new services with a higher reward to the system or to the user [31] and now the selected passive replica can be starting on a more powerful or less congested node.

Therefore, whenever a component fails, the activation of a passive replica may prompt the need for coordination activities that ensure a new globally acceptable solution for the entire distributed service [18]. While there has been a great deal of research in several aspects of runtime coordination in embedded real-time systems [12, 10, 5, 13], to the best of our knowledge we are the first to address the specific problem of coordinating the activation of passive replicas in interdependent distributed environments with real-time constraints. Here, the term *coordinated activation* refers to the ability of a distributed system to invoke adaptive actions on multiple nodes in a coordinated manner so as to achieve a new service configuration.

Without any central coordination entity, the collective adaptation behaviour must emerge from local interactions among components. This is typically accomplished through the exchange of multiple messages to ensure that all involved components make the same decision about whether and how to adapt [13]. One main challenge is controlling this exchange of information in order to achieve a convergence to a globally consistent solution. It may be difficult to predict the exact behaviour of the system taken as a whole due to the large number of possible non-deterministic ways in which the system can behave [42]. Whenever real-time decision making is in order, a timely answer to events suggests that after some finite and bounded time the global adaptation process converges to a consistent solution.

Therefore, whenever $Q^i_{val'}$, the proposed change of the currently output QoS Q^i_{val} for a component $c_i \in S$, has an impact on the currently output QoS level of direct neighbours in $\mathcal{G}_S$, a request for coordination in the adaptation process is sent to those affected components. Naturally, the formulation of the corresponding positive or negative feedback depends on the feasibility of the new requested QoS level as a function of the quality of the new set of inputs I_{c_j} for component c_j and the amount of locally available resources.

Definition 1. Given a node n and a set of QoS levels σ_n to be provided for all the components being locally executed at n, it is considered that admission control is performed, and that therefore a system specific *feasibility* function (e.g. [30, 1, 34]) determines if a set of QoS requirements can be met with available resources.

$$\mathit{feasibility}(\sigma_n) = \begin{cases} \text{true} & \text{if node } n \text{ has sufficient resources} \\ & \text{to supply the set of QoS levels } \sigma \\ \text{false} & \text{otherwise} \end{cases}$$

If there are insufficient resources to accommodate the change to the new requested QoS level, a negative feedback is formulated and sent back in reply and the global QoS adaptation fails. In this case, the former QoS level, that was being output by the older primary, must be used.

On the other hand, positive changes in interdependent commitments are propagated along $\mathcal{G}_S$, until the next cut-vertex c_c is reached. For that, we define the *paths* and the *flatten* functions. For the sake of simplicity of presentation, we present all the functions in a declarative notation with the same operational model as a pattern matching-based functional language. The reader should note that although a node is only aware of its nearest neighbours in a coalition of cooperating nodes for the collective execution of service S, we deal with the complete interdependency graph only to prove that all the conducted actions are correctly propagated until the final result is determined.

The *paths* function is a breadth first approach with cycle checking for determining components in paths. Visited components are added to the temporary set T in order to avoid infinite loops. The function outputs all the components in all the possible paths between two interdependent components c_i and c_j , or $\perp$ if there is no path between those two components. If there are more than one path between them, the result is a set of sets, each of them corresponding to a distinct path. The $flatten$ function is then used to make the resulting set of sets flat, meaning that all the components in these subsets will now belong to a simplified single set.

Definition 2. Given a DAG $\mathcal{G}_S = (\mathcal{V}_S, \mathcal{E}_S)$ and two components $c_i, c_j \in \mathcal{V}_S$, all the components in the possible paths between c_i and c_j are obtained as the result of the function:

$$\begin{aligned}
\mathit{paths}(c_i, c_j, \mathcal{G}_S) &= \mathit{flatten}(\mathit{paths}(c_i, c_j, \mathcal{G}_S, \emptyset)) \\
\mathit{paths}(c_i, c_j, \mathcal{G}_S, T) &= \emptyset, \text{ if } c_i = c_j \\
\mathit{paths}(c_i, c_j, \mathcal{G}_S, T) &= \{\{c_i, c_{k_1}\} \cup \mathit{paths}(c_{k_1}, c_j, \mathcal{G}_S, T \cup \{c_{k_1}\}) \\
&\quad \vdots \\
&\quad \{c_i, c_{k_n}\} \cup \mathit{paths}(c_{k_n}, c_j, \mathcal{G}_S, T \cup \{c_{k_n}\})\}, \\
&\quad \forall c_{k_m} \in \mathcal{V}_S, \text{ such that } (c_i, c_{k_m}) \in \mathcal{E}_S \text{ and } c_{k_m} \notin T \\
\text{paths}(c_i, c_j, \mathcal{G}_S, T) &= \perp, \text{ otherwise}
\end{aligned}$$

Definition 3. Given a set A containing other sets, the function *flatten(A)* is defined as:

$$\begin{aligned}
\mathit{flatten}(\emptyset) &= \emptyset \\
\mathit{flatten}(A) &= \{a\} \cup flatten(A \setminus \{a\}), \text{ if } a \in A
\end{aligned}$$

Affected components collect relevant data, both from the commitments of other components and from local resource reservations, that reflect the current state of the system. Subsequently, each involved component analyses the collected data and takes a decision on whether and how to adapt in order to reach a global desired state. Finally, to implement the decision, the set of components acts in a coordinated manner. A coordinated adaptive phase is initiated whenever a coordination phase is successful, i.e. whenever the component that initiated the coordination request receives a positive feedback in reply. During the coordination phase, each node pre-reserves the needed resources to achieve the new output QoS. Therefore, resource allocation is always possible at this stage [31]. Once resources are allocated, the node commits to produce the announced output QoS either until the service terminates or adaptation occurs.

Definition 4. Given a connected graph $\mathcal{G}_S = (\mathcal{V}_S, \mathcal{E}_W)$, such that component $c_i \in \mathcal{V}_S$, and $I_{c_i} = \{(c_j, Q^j_{val}), \ldots, (c_k, Q^k_{val})\}$ as the current set of QoS inputs of c_i, and given T as the set of changed QoS inputs in response to the coordination request, the function *update*(I,T) updates I with the elements from T:

$$\begin{array}{rcl}
\mathit{update}(\emptyset,T) & = & \emptyset \\
\mathit{update}(I,T) & = & \{(c_i,Q^i_{val'})\} \cup \mathit{update}(I \setminus (c_i,Q^i_{val}),T), \text{ if } (c_i,Q^i_{val}) \in I \\
 & & \text{and } (c_i,Q^i_{val'}) \in T \\
\mathit{update}(I,T) & = & \{(c_i,Q^i_{val})\} \cup \mathit{update}(I \setminus (c_i,Q^i_{val}),T), \text{ if } (c_i,Q^i_{val}) \in I \\
 & & \text{and } (c_i,Q^i_{val'}) \notin T
\end{array}$$

Definition 5. Given the connected graph $\mathcal{G}_S = (\mathcal{V}_S, \mathcal{E}_S)$ with a set of ordered cut-vertices $\mathcal{C}_S$ and the subgraph that connects component c_i to next cut-vertex $c_c \in \mathcal{C}_S$, the function *test_change*$(c_i, c_c, \mathcal{G}_S, Q^i_{val'})$ is defined by:

$$\begin{array}{rcl}
\mathit{test\_change}(c_i,c_c,\mathcal{G}_S,Q^i_{val'}) & = & T, \text{ if} \\
 & & T' = \{(c_i,Q^i_{val'})\}, \\
 & & P = \mathit{paths}(c_i,c_c,\mathcal{G}_S), \\
 & & I_{QoS} = \mathit{get\_input\_qos}(c_i), \\
 & & \mathit{test\_change\_component}(I_{QoS},P,T') = T, \\
 & & T \neq \bot \\
\mathit{test\_change}(c_i,c_c,\mathcal{G}_S,Q^i_{val'}) & = & \bot, \text{ otherwise.}
\end{array}$$

$$\begin{array}{rcl}
\mathit{test\_change\_component}(I_{QoS},\emptyset,T') & = & \emptyset \\
\mathit{test\_change\_component}(I_{QoS},P,T') & = & \mathit{test\_change\_component}(I_{QoS},P\backslash\{c_j\},T') \\
 & & \cup \{(c_j,Q^j_{val'})\}, \text{ if} \\
 & & c_j \in P, \\
 & & S = \mathit{update}(I_{QoS},P,T'), \\
 & & \mathit{test\_feasibility}(c_j,Q^j_{val'},S) = \text{true}, \\
\mathit{test\_change\_component}(I_{QoS},P,T') & = & \bot, \text{ otherwise}
\end{array}$$

Furthermore, we use the following auxiliary functions which, for the sake of clarity of presentation, we do not define formally:

***test_feasibility*$(c_i, Q^i_{val'}, I_{c_i})$:** Given a node n_x where a component $c_i \in S$ is currently being executed, $Q^i_{val'}$ as the new requested QoS level for c_i, and $I_{c_i} = \{(c_j, Q^j_{val}), \ldots, (c_k, Q^k_{val})\}$ as the new set of QoS levels given as input to c_i, *test_feasibility* refers to the specific feasibility function of the system given by Definition 1 applied to node n_x to determine if the new set of QoS requirements can be met with available resources.

***set_qos_level*$(Q^i_{val'}, c_i, S)$:** Sets the new QoS level $Q^i_{val'}$ for component $c_i \in S$.

***get_cut_vertices*($\mathcal{G}_S$):** Returns an ordered set of all the cut-vertices in $\mathcal{G}_S$. This function is based on the depth-first algorithm for finding cut-vertices which was first presented in [46]. The cut-vertices are found and stored ordered through the DAG from a source component until the sink component.

***get_input_qos*(c_i):** Given a component $c_i \in S$, returns the set of elements (c_j, Q^j_{val}), where each of these elements represents a component c_j with an output QoS level of Q^j_{val} used as an input in component c_i.

***get_qos_level*(c_i):** Returns the current QoS level output by component c_i.

***head*(S):** Returns the first element of set S.

Having this background, given the connected graph $\mathcal{G}_S = (\mathcal{V}_S, \mathcal{E}_S)$, an end-user sink component c_u, the set of components S and the set of cut-vertices $\mathcal{C}_S$ computed by function *get_cut_vertices*($\mathcal{G}_S$), whenever a component $c_i \in \mathcal{V}_S$ is able to change its output QoS to a new QoS level $Q^i_{val'}$, subsequent components in $\mathcal{G}_S$ respond to the coordination request according to a decentralised feedback-based coordination protocol, formally presented in Algorithm 1.

Algorithm 1 Service coordination

$$\begin{aligned}
\mathit{service\_coordination}(c_i, c_u, \mathcal{G}_S, \mathcal{C}_S), &= \mathit{change}(F, S),\ \text{if} \\
& \mathit{try\_coordination}(c_i, \mathcal{C}_S \cup \{c_u\}) = F \\
& \text{and } F \neq \perp \\[1em]
\mathit{try\_coordination}(c_i, \emptyset) &= \emptyset \\
\mathit{try\_coordination}(c_i, \mathcal{C}_S) &= \mathit{try\_coordination}(c_c, \mathcal{C}'_S) \cup T, \mathit{if} \\
& c_c = \mathit{head}(\mathcal{C}_S \backslash \{c_i\}), \\
& Q^i_{val'} = \mathit{get\_qos\_level}(c_i), \\
& \mathit{test\_change}(c_i, c_c, \mathcal{G}_S, Q^i_{val'}) = T, \\
& T \neq \perp \\
\mathit{try\_coordination}(c_i, \mathcal{C}_S) &= \perp,\ \text{otherwise.} \\[1em]
\mathit{change}(\emptyset, S) &= \text{true} \\
\mathit{change}(F, S) &= \mathit{change}(F', S),\ \text{where} \\
& (c_j, Q^j_{val'}) \in F, \\
& \mathit{set\_qos\_level}(Q^j_{val'}, c_j, S), \\
& F' = F \backslash \{(c_j, Q^j_{val'})\}.
\end{aligned}$$

By exchanging feedback on the performed self-adaptations, components converge towards a global solution, overcoming the lack of central coordination and global knowledge. Negative feedback loops occur when a change in one service component triggers an opposing response that counteracts that change at other interdependent component along $\mathcal{G}_S$. On the other hand, the positive feedback loops promote global adaptations. The snowballing effect of the positive feedback takes an initial change in one component and reinforces that change in the same direction at all affected partners.

A fundamental advantage of the proposed coordination model is that both the communication and adaptation overheads may depend on the size of a component neighbourhood until a cut-vertex is reached, instead of the entire set of components, if a cut-vertex is able to maintain its current output quality, despite the changes on the quality of its inputs. This allows the proposed feedback-based coordination model to scale effectively to large distributed systems.

5 PROPERTIES OF THE COORDINATION MODEL

Proposition 1. Given a node n and a set of QoS levels σ to be satisfied, the function *feasibility*(σ_n) always terminates and returns true if σ is feasible in n or false otherwise.

Proposition 2. Given a DAG $\mathcal{G}_S = (\mathcal{V}_S, \mathcal{E}_S)$ and two components $c_i, c_j \in \mathcal{V}_S$, *paths*$(c_i, c_j)$ terminates and returns all the components in the possible paths between c_i and c_j, $\emptyset$ in case $c_i = c_j$, or $\perp$ in case there is no path between $c_i, c_j \in \mathcal{V}_S$.

Proposition 3. Given two sets I and T, both with elements of the form (c_i, Q^i_{val}), *update(I,T)* terminates and returns a new set with the elements of I such that whenever $(c_i, Q^i_{current}) \in I$ and $(c_i, Q^i_{new}) \in T$ the pair stored in the returned set is (c_i, Q^i_{new}).

Lemma 1. Given the connected graph $\mathcal{G}_S = (\mathcal{V}_S, \mathcal{E}_S)$ with a set of cut-vertices $\mathcal{C}_S$ and the subgraph that connects component c_i to next cut-vertex $c_c \in \mathcal{C}_S$ and a new updated QoS level value $Q^i_{val'}$, the call to *test_change*$(c_i, c_c, \mathcal{G}_S, Q^i_{val'})$ terminates and succeeds if c_c is able to output a new QoS level $Q^c_{val'}$ or fails otherwise.

Theorem 1 (Correctness of service coordination)**.** Given the connected graph $\mathcal{G}_S = (\mathcal{V}_S, \mathcal{E}_S)$ representing the QoS interdependencies of a service S being executed by a group of components, such that $c_u \in \mathcal{V}$ is the end-user sink component receiving the service at a QoS level Q_{val}, whenever a component c_i announces an update to $Q^i_{val'}$, Algorithm 1 changes the set of SLAs at components in $\mathcal{G}$ such that c_u receives S changed to the QoS level $Q^u_{val'}$ or does not change the set of local SLAs at any node and c_u continues to receive S at its current QoS level Q^u_{val}.

Proof. Termination comes from the finite number of elements in $\mathcal{C}_S \cup c_u$ and from Lemma 1. Algorithm 1 applies the function *test_change* iteratively to all nodes in the subgraph starting with c_i and finishing in c_u. The base case is when there are no cut-vertices and there is only one call to *test_change*. It is trivial to see that the result of *test_change* will consist in a set of components that will update for the new QoS level $Q^j_{val'}$ or it will fail and, by Lemma 1, it is correct. The remaining cases happen when there is one or more cut-vertices between c_i and c_u. Here, *update* will be applied to all subgraphs starting in c_i and finishing in c_u. Each of these subgraphs are sequentially tested. Only if all of them can be updated service S will be delivered to component c_u at the new updated QoS level $Q^u_{val'}$. The result follows by induction in the number of cut-vertices. □

Definition 6. Given a directed graph $\mathcal{G}_S = (\mathcal{V}_S, \mathcal{E}_S)$, the in-degree of a component $c_i \in \mathcal{V}$ is the number of edges that have c_i as their destination.

Definition 7. Given a directed graph $\mathcal{G}_S = (\mathcal{V}_S, \mathcal{E}_S)$, the out-degree of a component $c_i \in \mathcal{V}$ is the number of edges that have c_i as their starting node.

Whenever a change to a new QoS level $Q^i_{val'}$ is requested by a component c_i, if the next cut-vertex c_c in $\mathcal{G}$ cannot supply the requested level, then all the precedent components between c_i and c_c are kept in their currently supplied feasible QoS level Q^j_{val}. Thus, the number of needed messages is given by the in-degree and out-degree of the nodes in the paths between c_i and c_c where it was determined that the requested new QoS level was not possible. On the other hand, if the requested change is possible, the number of needed messages is given by the number of edges between c_i and the end-user sink component c_u. This is because a change is only possible after all the involved components are queried and the conjunction of their efforts results in a newer QoS level being delivered to c_u. Thus, the maximum number of exchanged messages in a coordination operation is given by Formula (5).

$$\sum_{c \in \mathcal{V}_S} (\deg^+(c) + \deg^-(c)) \tag{5}$$

Applications with real-time requirements can benefit from decentralised coordination mechanisms, as long as those mechanisms support the timing and QoS requirements of applications. Therefore, coordination tasks need to be time dependent since handling time is necessary to specify (and enforce) given levels of quality of service [44, 31].

Access to the system resources can be modelled as a set of isolated constant-bandwidth servers, either related to CPU [2, 28] or network [32, 35] scheduling. Furthermore, feedback can be formulated as a result of a local anytime QoS adaptation process that can trade off the needed computation time by the achieved quality of solution, within a useful and bounded time [30].

Based on these guarantees, it is possible to determine a time-bounded convergence to a globally accepted solution. As such, the uncertain outcome of iterative decentralised control models whose effect may not be observable until some unknowable time in the future [42] is not present in the proposed regulated coordination model.

Proposition 4. Given the connected graph $\mathcal{G}_S = (\mathcal{V}_S, \mathcal{E}_S)$ representing the QoS interdependencies of a service S, such that $c_u \in \mathcal{V}$ is the end-user sink component receiving the service at a QoS level Q_{val}, if a source component starts a service coordination process involving all the nodes in the graph, this means that for a graph of n components we have to exchange the number of messages given by Formula (5). If the longest exchange of messages between two components takes t_m time units and the longest time for a node to compute its feedback is δ then the convergence of the system for the worst case scenario is given by the following formula:

$$\left(\sum_{c\in\mathcal{V}_S} \left(\deg^+(c) + \deg^-(c)\right)\right) * t_m * \delta \tag{6}$$

6 EVALUATION

We have conducted extensive simulations in order to evaluate the performance of the proposed decentralised feedback-based coordination model in highly dynamic open scenarios. The objective was to show that it can be efficiently used to coordinate the activation of passive replicas and maintain desirable system-wide properties in decentralised component-based systems with timing constraints.

An application that captures, compresses and transmits frames of video to end users, which may use a diversity of end devices and have different sets of QoS preferences, was used to evaluate the efficiency of the proposed passive replication mechanism with coordinated activations, with a special attention being devoted to introduce a high variability in the characteristics of the considered scenarios. The application is composed by a set of components to collect the data, a set of compression components to gather and compress the data sent from multiple sources, a set of transmission components to transmit the data over the network, a set of decompression components to convert the data into the user's specified format, and a set of components to display the data in the end device.

The number of simultaneous nodes in the system randomly varied, in each simulation run, from 10 to 100. For each node, the type and amount of available resources were randomly generated, creating a distributed heterogeneous environment. Nodes failed and recovered according to their MTTF and MTTR reliability values, which were randomly assigned when the nodes were created (it was ensured that each node had an availability between 60 % and 99 %). Each node was running a prototype implementation of the CooperatES framework [25], with a fixed set of mappings between requested QoS levels and resource requirements. At randomly selected nodes, new service requests from 5 to 20 simultaneous users were randomly generated, dynamically generating different amounts of load and resource availability. Based on each user's service request, services of 4 to 20 components were formed and randomly a percentage of the connections among those components was selected as a QoS interdependency.

In order to assess the efficiency of the proposed dynamic replication control as opposed to an offline static replication in dynamic resource-constrained environments, we considered the number of component-based services which were able to recover from failures and conclude their cooperative executions as a function of the used replication ratio. 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[2]. The mean values of all generated samples were used to produce the charts.

In the first study, we evaluated the achieved system availability with the proposed dynamic replication control based on components significance and with a static offline approach in which the components to replicate and number of their replicas is fixed by the system designer at the service composition phase [24]. At each simulation run, if the primary replica of a component c_i failed during operation, a new primary was selected among the set of passive backups. If this was not possible, all the coalitions depending on c_i were aborted. In this study, replicas were also randomly allocated among eligible nodes with the dynamic replication control policy.

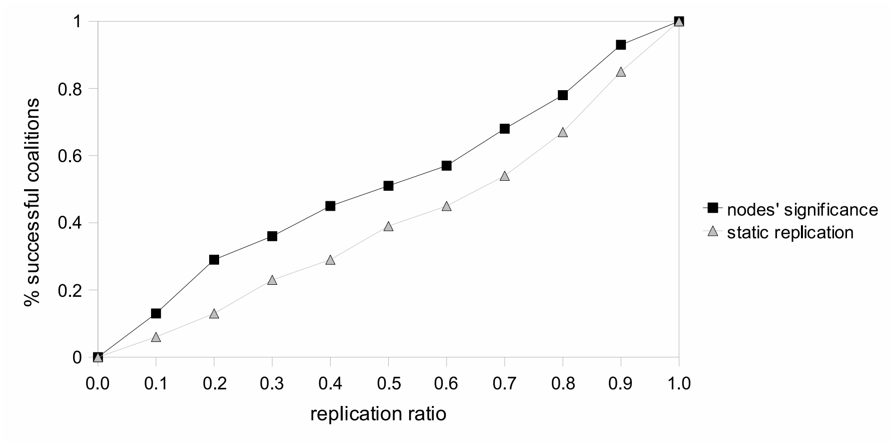


Figure 1. Impact of the chosen replication control strategy on the system availability

Figure 1 clearly shows that our strategy is more accurate to determine and replicate the most significant components than a static offline one, particularly with lower replication ratios. Thus, when lower replication ratios are imposed due to resource or cost limitations, a higher availability can be achieved if the selection of which components to replicate and number of their replicas depends on their significance to the system as a whole. In open and dynamic environments, such

[2] The random values were generated by the Mersenne Twister algorithm [26].

significance can be determined online as the aggregation of all the other components that depend on a particular component to perform their tasks.

The second study evaluated the impact of the selected replicas placement strategy on the achieved system availability for a given replication ratio. The study compared the performance of the proposed allocation heuristic based on collected information about the nodes availability as the system evolves with a random policy in which the placement of the generated replicas is fixed offline [33]. The decision on which components to replicate and their number of replicas followed the same dynamic and static approaches of the first study. For the dynamic allocation strategy, a tolerance value for the availability of each replica set was randomly generated at each simulation run. If this tolerance was surpassed, a reassignment of replicas was performed. The results were plotted in Figure 2.

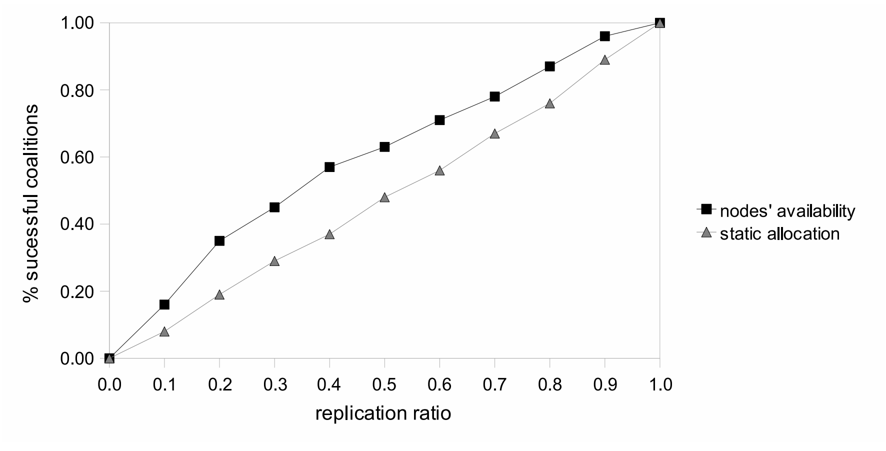


Figure 2. Impact of the chosen replica allocation strategy on the system availability

It is then possible to conclude that the location of replicas is a relevant factor for the system availability as a whole. The proposed dynamic replicate allocation that takes into account the nodes reliability over time always achieves a better performance than an offline static allocation policy in open and dynamic environments. Furthermore, a comparison of Figures 1 and 2 shows that even though an improvement in availability can be achieved by increasing the replication ratio, the impact of replicas placement is quite significant.

The third study evaluated the efficiency of the proposed coordinated activation of interdependent passive replicas in comparison to a typical centralised coordination approach [40] in which a system-wide controller coordinates resource allocations among multiple nodes. The average results of all simulation runs for the different coalition sizes and percentages of interdependencies among components are plotted in Figure 3. As expected, both coordination approaches need more time as the complexity of the services topology increases. Nevertheless, the proposed decentralised coordination model is faster to determine the overall coordination result in all the evaluated services topologies, demanding approximately 75 % of the time spent by the centralised near-optimal model.

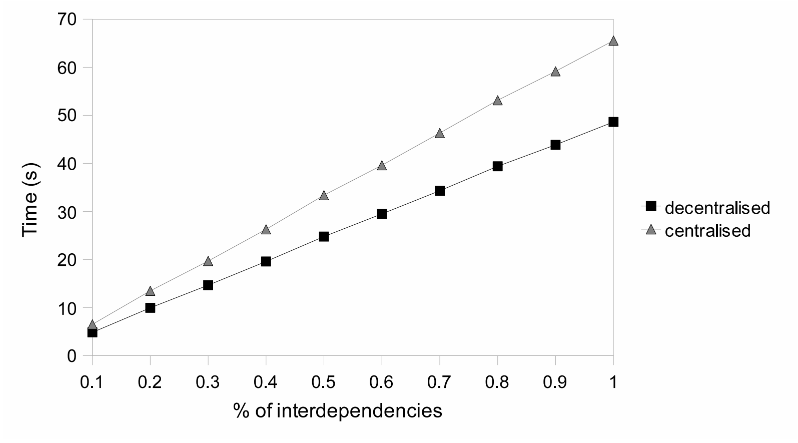


Figure 3. Time for a coordinated replica activation

7 CONCLUSIONS

The increased complexity of embedded real-time systems leads to increasing demands with respect to requirements engineering, high-level design, early error detection, productivity, integration, verification and maintenance, which increases the importance of the efficient management of life-cycle properties such as maintainability, portability, and adaptability. Therefore it is necessary to have a systematic approach to embedded software development and maintenance. A natural choice is to use the well-established component-based design in which the system can be seen as a set of interacting components, each providing a well-defined subset of functionalities and whose integration produces the final system behaviour.

At the same time, when developing embedded systems, certain constraints regarding extra-functional properties have to be guaranteed, such as: timing, resource usage and higher reliability and availability. These extra-functional properties have significant importance, and have to be addressed explicitly when developing embedded systems.

The availability and performance of the open distributed embedded system is significantly affected by the choice of the replication control strategy and placement of the generated replicas. Due to its low resource consumption, passive replication is appealing for embedded real-time systems that cannot afford the cost of maintaining active replicas and need not assure a hard real-time performance. The proposed heuristics based on the components significance to the overall system and on nodes reliability history have a low runtime complexity and achieve a reasonably higher system availability than static offline decisions, particularly when lower replication ratios are imposed due to resource or cost limitations.

Another challenge is to transfer the state of a distributed service to a new globally acceptable configuration whenever a new elected primary cannot provide the same QoS level that was being output by the old primary that was found as faulty. The proposed distributed coordination model reduces the complexity of the needed interactions among nodes and is faster to converge to a globally acceptable solution than the traditional centralised coordination approach.

Acknowledgments

This work was partially supported by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology) and when applicable, co-financed by ERDF (European Regional Development Fund) under the PT2020 Partnership, within project UID/CEC/04234/2013 (CISTER Research Centre); also by the North Portugal Regional Operational Programme (ON. 2 – O Novo Norte) under the National Strategic Reference Framework (NSRF), through the European Regional Development Fund (ERDF), and by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology), within project NORTE-07-0124-FEDER-000063 (BEST-CASE, New Frontiers). Jorge Coelho was partially funded by LIACC through Programa de Financiamento Plurianual of FCT.

REFERENCES

[1] Abdelzaher, T. F.—Atkins, E. M.—Shin, K. G.: QoS Negotiation in Realtime Systems and Its Application to Automated Flight Control. IEEE Transactions on Computers, Best of RTAS '97 Special Issue, Vol. 49, 2000, No. 11, pp. 1170–1183.

[2] Abeni, L.—Buttazzo, G.: Integrating Multimedia Applications in Hard Realtime Systems. Proceedings of the 19th IEEE Real-Time Systems Symposium, Madrid, Spain, December 1998, p. 4.

[3] Allen, G.—Dramlitsch, T.—Foster, I.—Karonis, N. T.—Ripeanu, M.—Seidel, E.—Toonen, B.: Supporting Efficient Execution in Heterogeneous Distributed Computing Environments with Cactus and Globus. Proceedings of the 2001 ACM/IEEE Conference on Supercomputing, November 2001, pp. 52–52.

[4] Balasubramanian, J.—Tambe, S.—Lu, C.—Gokhale, A.—Gill, C.—Schmidt, D. C.: Adaptive Failover for Real-Time Middleware with Passive Replication. Proceedings of the 15th IEEE Real-Time and Embedded Technology and Applications Symposium, IEEE Computer Society, April 2009, pp. 118–127.

[5] Boutilier, C.—Das, R.—Kephart, J. O.—Tesauro, G.—Walsh, W. E.: Cooperative Negotiation in Autonomic Systems Using Incremental Utility Elicitation. Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence, Acapulco, Mexico, August 2003, pp. 89–97.

[6] Budhiraja, N.—Marzullo, K.—Schneider, F. B.—Toueg, S.: The Primary-Backup Approach. Distributed Systems (2nd Ed.), 1993, pp. 199–216.

[7] Cai, Zh.—Kumar, V.—Cooper, B. F.—Eisenhauer, G.—Schwan, K.—Strom, R. E.: Utility-Driven Proactive Management of Availability in Enterprise-Scale Information Flows. Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware, Springer-Verlag, 2006, pp. 382–403.

[8] Crnkovic, I.: Component-Based Software Engineering for Embedded Systems. Proceedings of the 27th International Conference on Software Engineering, St. Louis, MO, USA, May 2005, pp. 712–713.

[9] de Juan-Marin, R.—Decker, H.—Munoz-Esco, F. D.: Revisiting Hot Passive Replication. Proceedings of the 2nd International Conference on Availability, Reliability and Security, April 2007, pp. 93–102.

[10] De Wolf, T.—Holvoet, T.: Towards Autonomic Computing: Agent-Based Modelling, Dynamical Systems Analysis, and Decentralised Control. Proceedings of the IEEE International Conference on Industrial Informatics, August 2003, pp. 470–479.

[11] Distler, T.—Popov, I.—Schröder-Preikschat, W.—Reiser, H. P.—Kapitza, R.: Spare: Replicas on Hold. Proceedings of the 18th Network and Distributed System Security Symposium, February 2011.

[12] Dorigo, M.—Di Caro, G.: The Ant Colony Optimization Meta-Heuristic. New Ideas in Optimization, 1999, pp. 11–32.

[13] Dowling, J.—Haridi, S.: Decentralized Reinforcement Learning for the Online Optimization of Distributed Systems. Chapter in Reinforcement Learning: Theory and Applications, I-Tech Education and Publishing, Vienna, Austria, 2008, pp. 142–167.

[14] Ensink, B.—Adve, V.: Coordinating Adaptations in Distributed Systems. Proceedings of the 24th International Conference on Distributed Computing Systems, Tokyo, Japan, March 2004, pp. 446-455.

[15] Farinelli, A.—Rogers, A.—Petcu, A.—Jennings, N. R.: Decentralised Coordination of Low-Power Embedded Devices Using the Max-Sum Algorithm. Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems, 2008, Vol. 2, pp. 639–646.

[16] Friday, A.—Davies, N.—Cheverst, K.: Utilising the Event Calculus for Policy Driven Adaptation on Mobile Systems. Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks, Washington, DC, USA, IEEE Computer Society, 2002, p. 13.

[17] Fu, W.—Xiao, N.—Lu, X.: A Quantitative Survey on QoS-Aware Replica Placement. Proceedings of the 7th International Conference on Grid and Cooperative Computing, Shenzhen, China, October 2008, pp. 281–286.

[18] Gelernter, D.—Carriero, N.: Coordination Languages and Their Significance. Communications of the ACM, Vol. 35, 1992, No. 2, pp. 96–107.

[19] Guerraoui, R.—Schiper, A.: Software-Based Replication for Fault Tolerance. IEE Computer, Vol. 30, 1997, pp. 68–74.

[20] Jiang, J.—Yu, X.: Fault-Tolerant Control Systems: A Comparative Study Between Active and Passive Approaches. Annual Reviews in Control, Vol. 36, 2012, No. 1, pp. 60–72.

[21] Jin, J.—Nahrstedt, K.: QoS-Aware Service Management for Component-Based Distributed Applications. ACM Transactions on Internet Technology, Vol. 8, 2008, No. 3, pp. 14:1–14:31.

[22] Kopetz, H.: Real-Time Systems: Design Principles for Distributed Embedded Applications. Kluwer, 1997.

[23] Little, M. C.: Object Replication in a Distributed System. Ph.D. thesis, Department of Computing Science, Newcastle University, September 1991.

[24] Little, M. C.—McCue, D. L.: The Replica Management System: A Scheme for Flexible and Dynamic Replication. Proceedings of the 2nd International Workshop on Configurable Distributed Systems, April 1994, pp. 46–57.

[25] Maia, C.—Nogueira, L.—Pinho, L. M.: Experiences on the Implementation of a Cooperative Embedded System Framework: Short Paper. Proceedings of the 8th International Workshop on Java Technologies for Real-Time and Embedded Systems, Prague, Czech Republic, August 2010, pp. 70–72.

[26] Matsumoto, M.—Nishimura, T.: Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator. ACM Transactions on Modeling and Computer Simulation (TOMACS), Vol. 8, 1998, No. 1, pp. 3–30.

[27] Nikolow, D.: Semantic-Based Storage QoS Management Methodology – Case Study for Distributed Environments. Computing and Informatics, Vol. 31, 2012, No. 6, pp. 1345–1366.

[28] Nogueira, L.—Pinho, L. M.: A Capacity Sharing and Stealing Strategy for Open Real-Time Systems. Journal of Systems Architecture, Vol. 56, 2010, No. 4-6, pp. 163–179.

[29] Nogueira, L.—Coelho, J.: Service-Wide Adaptations in Distributed Embedded Component-Based Systems. Proceedings of the 7th International Symposium on Intelligent Distributed Computing, Prague, Czech Republic, September 2013, pp. 141–150.

[30] Nogueira, L.—Pinho, L. M.: Dynamic QoS Adaptation of Inter-Dependent Task Sets in Cooperative Embedded Systems. Proceedings of the 2nd ACM International Conference on Autonomic Computing and Communication Systems, Turin, Italy, September 2008, p. 97.

[31] Nogueira, L.—Pinho, L. M.: Time-Bounded Distributed QoS-Aware Service Configuration in Heterogeneous Cooperative Environments. Journal of Parallel and Distributed Computing, Vol. 69, 2009, No. 6, pp. 491–507.

[32] Nolte, T.—Lin, K.-J.: Distributed Real-Time System Design Using CBS-Based End-to-End Scheduling. Proceedings of the 9th International Conference on Parallel and Distributed Systems, December 2002, pp. 355–360.

[33] On, G.—Schmitt, J.—Steinmetz, R.: Quality of Availability: Replica Placement for Widely Distributed Systems. Proceedings of the 11th International Workshop on Quality of Service, Monterey, CA, June 2003, pp. 325–342.

[34] Park, J.—Ryu, M.—Hong, S.: Deterministic and Statistical Admission Control for QoS-Aware Embedded Systems. Journal of Embedded Computing, Vol. 1, 2005, pp. 57–71.

[35] Pedreiras, P.—Gai, P.—Almeida, L.—Buttazzo, G. C.: FTT-Ethernet: A Flexible Real-Time Communication Protocol That Supports Dynamic QoS Management on Ethernet-Based Systems. IEEE Transactions on Industrial Informatics, Vol. 1, 2005, No. 3, pp. 162–172.

[36] Pinho, L. M.—Vasques, F.—Wellings, A.: Replication Management in Reliable Real-Time Systems. Real-Time Systems, Vol. 26, 2004, No. 3, pp. 261–296.

[37] Powell, D. (Editor): A Generic Fault-Tolerant Architecture for Real-Time Dependable Systems. Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[38] Rajkumar, R.—Lee, C.—Lehoczky, J.—Siewiorek, D.: A Resource Allocation Model for QoS Management. Proceedings of the 18th IEEE Real-Time Systems Symposium, IEEE Computer Society, 1997, pp. 298.

[39] Rasche, A.—Poize, A.: Dynamic Reconfiguration of Component-Based Real-Time Software. Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems, February 2005, pp. 347–354.

[40] Rohloff, K.—Schantz, R.—Gabay, Y.: High-Level Dynamic Resource Management for Distributed, Real-Time Embedded Systems. Proceedings of the 2007 Summer Computer Simulation Conference, July 2007, pp. 749–756.

[41] Rubel, P.—Gillen, M.—Loyall, J.—Schantz, R.—Gokhale, A.—Balasubramanian, J.—Paulos, A.—Narasimhan, P.: Fault Tolerant Approaches for Distributed Real-Time and Embedded Systems. Proceedings of the 2007 Military Communications Conference, Orlando, Florida, USA, October 2007, pp. 1–8.

[42] Di Marzo Serugendo, G.: Autonomous Systems with Emergent Behaviour. Chapter in Handbook of Research on Nature Inspired Computing for Economy and Management, Idea Group, Inc., Hershey-PA, USA, September 2006, pp. 429–443.

[43] Shankar, M.—de Miguel, M.—Liu, J. W. S.: An End-to-End QoS Management Architecture. Proceedings of the 5th IEEE Real-Time Technology and Applications Symposium, Washington, DC, USA, 1999, IEEE Computer Society, pp. 176–191.

[44] Stankovic, J. A.—Abdelzaher, T. E.—Lu, Ch.—Sha, L.—Hou, J. C.: Real-Time Communication and Coordination in Embedded Sensor Networks. Proceedings of the IEEE, Vol. 91, July 2003, No. 7, pp. 1002–1022.

[45] Streichert, T.—Glass, M.—Wanka, R.—Haubelt, C.—Teich, J.: Topology-Aware Replica Placement in Fault-Tolerant Embedded Networks. Proceedings of the 21st International Conference on Architecture of Computing Systems, Dresden, Germany, February 2008, pp. 23–37.

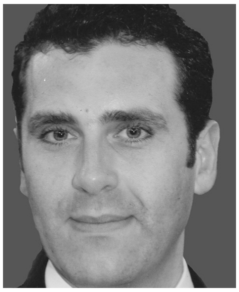
[46] Tarjan, R. E.: Depth-First Search and Linear Graph Algorithms. SIAM J. Comput., Vol. 1, 1972, No. 2, pp. 146–160.

[47] Tjora, A.—Skavhaug, A.: A General Mathematical Model for Run-Time Distributions in a Passively Replicated Fault Tolerant System. Proceedings of the 15th Euromicro Conference on Real-Time Systems, Porto, Portugal, July 2003, pp. 295–301.

[48] Wiklander, J.—Eliasson, J.—Kruglyak, A.—Lindgren, P.—Nordlander, J.: Enabling Component-Based Design for Embedded Real-Time Software. Journal of Computers, Vol. 4, 2009, No. 12, pp. 1309–1321.

[49] Zhang, J.—Liu, L.—Pu, C.—Ammar, M.: Reliable Peer-to-Peer End System Multicasting through Replication. Proceedings of the 4th International Conference on Peer-to-Peer Computing, Zurich, Switzerland, August 2004, pp. 235–242.

Luís Nogueira received his B.Sc. degree in computing engineering in 2000, his M.Sc. degree in informatics – systems and networks in 2002 and his Ph.D. degree in computer science in 2009 from University of Porto. He is Assistant Professor at the Department of Computer Engineering, School of Engineering of the Polytechnic Institute of Porto and a researcher at CISTER/INESC-TEC. His main research interests include open real-time systems, self-organising decentralised systems, QoS support, and real-time scheduling algorithms.



Jorge Coelho received his B.Sc. degree in computer science in 2000, his M.Sc. degree in informatics – systems and networks in 2002 and his Ph.D. degree in computer science in 2007 from University of Porto. He is Assistant Professor at the Department of Computer Engineering, School of Engineering of the Polytechnic Institute of Porto. His main research interests include open real-time systems, self-organising decentralised systems, XML processing languages, and verification of web content.