Technical Report

A Feedback-based Decentralised Coordination Model for Distributed Open Real-Time Systems

Luis Miguel Nogueira

Luis Miguel Pinho

Jorge Coelho

HURRAY-TR-120713

Version:

Date: 07-24-2012

A Feedback-based Decentralised Coordination Model for Distributed Open Real-Time Systems

Luis Miguel Nogueira, Luis Miguel Pinho, Jorge Coelho

IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail:

http://www.hurray.isep.ipp.pt

Abstract

Moving towards autonomous operation and management of increasingly complex open distributed real-time systems poses very significant challenges. This is particularly true when reaction to events must be done in a timely and predictable manner while guaranteeing Quality of Service (QoS) constraints imposed by users, the environment, or applications. In these scenarios, the system should be able to maintain a global feasible QoS level while allow- ing individual nodes to autonomously adapt under different constraints of resource availability and input quality.This paper shows how decentralised coordination of a group of autonomous interdependent nodes can emerge with little communication, based on the robust self-organising principles of feedback. Positive feedback is used to reinforce the selection of the new desired global service solution, while neg- ative feedback discourages nodes to act in a greedy fashion as this adversely impacts on the provided service levels at neighbouring nodes.The proposed protocol is general enough to be used in a wide range of scenarios characterised by a high degree of openness and dynamism where co-ordination tasks need to be time dependent. As the reported results demon- strate, it requires less messages to be exchanged and it is faster to achieve a globally acceptable near-optimal solution than other available approaches.

A Feedback-based Decentralised Coordination Model for Distributed Open Real-Time Systems

Luís Nogueira[a,*], Luís Miguel Pinho[a], Jorge Coelho[b]

[a] *CISTER Research Centre*
School of Engineering (ISEP), Polytechnic Institute of Porto (IPP)
[b] *Artificial Intelligence and Computer Science Laboratory (LIACC-UP)*
School of Engineering (ISEP), Polytechnic Institute of Porto (IPP)

Abstract

Moving towards autonomous operation and management of increasingly complex open distributed real-time systems poses very significant challenges. This is particularly true when reaction to events must be done in a timely and predictable manner while guaranteeing Quality of Service (QoS) constraints imposed by users, the environment, or applications. In these scenarios, the system should be able to maintain a global feasible QoS level while allowing individual nodes to autonomously adapt under different constraints of resource availability and input quality.

This paper shows how decentralised coordination of a group of autonomous interdependent nodes can emerge with little communication, based on the robust self-organising principles of feedback. Positive feedback is used to reinforce the selection of the new desired global service solution, while negative feedback discourages nodes to act in a greedy fashion as this adversely impacts on the provided service levels at neighbouring nodes.

The proposed protocol is general enough to be used in a wide range of scenarios characterised by a high degree of openness and dynamism where coordination tasks need to be time dependent. As the reported results demonstrate, it requires less messages to be exchanged and it is faster to achieve a globally acceptable near-optimal solution than other available approaches.

Keywords:

[*] Corresponding author.

Email addresses: `lmn@isep.ipp.pt` (Luís Nogueira), `lmp@isep.ipp.pt` (Luís Miguel Pinho), `jmn@isep.ipp.pt` (Jorge Coelho)

Open real-time systems, Self-organising decentralised systems, Decentralised coordination, Feedback control

1. Introduction

In recent years, many real-time systems have become open to unpredictable operating environments where both system workload and platform may vary significantly at run time. Open real-time systems allow a mix of independently developed real-time and non real-time applications to coexist. As such, the set of applications to be executed and their aggregate resource and timing requirements are unknown until runtime but, still, a timely answer to events must be provided in order to guarantee a desired level of performance [1].

This move from self-enclosed to open real-time systems is also one of the reasons for moving from static to dynamic environments. This calls for a different and more flexible approach than those typically used for building fixed-purpose distributed real-time embedded systems [2]. Static resource allocations might be appropriate for a situation at a single point in time, e.g. the initial deployment, but quickly become outdated as conditions change. Thus, the ability to adapt the system's behaviour at runtime is seen as a key attribute in a variety of new domains for real-time systems [3, 4, 5]. When reasoning in terms of Quality of Service (QoS) support in dynamic environments, which implies the establishment of contracts between clients and service providers, the idea is to design systems using QoS adaptation and renegotiation techniques and ensuring that, despite the uncertain factors that trigger the occurrence of changes in the environment, the QoS contracts remain valid [6].

While our previous work addresses multiple aspects of QoS support in open real-time systems [6, 7], this paper goes forward by focusing on a novel approach for coordinating self-adaptive peer nodes required to work cooperatively to solve computationally expensive services. Open real-time services increasingly consist of a set of interacting components that jointly provide some service to end-users [6, 8, 9]. Components use results produced by other components, that may be running on other nodes, to produce their own output and hence are interdependent. This means that a constraint on one quality or resource parameter at a component can constrain other parameters at other components [10]. Thus, a service's global QoS level must be defined as the set of compatible feasible QoS regions provided by all the interdependent components that compose the service. A feasible QoS region defines a valid region of output quality that a component can achieve when provided with sufficient input quality and resource allocation.

In these cases, a distributed system composed by self-adaptive nodes that optimise their behaviour towards some individual goals may not necessarily be optimised at a global level, as there is the possibility that conflicting greedy decisions may lead to interference between the different self-management behaviours of nodes, conflicts over shared resources, sub-optimal system performance and hysteresis effects [11]. Coordination is then a key concept for developing self-adaptive distributed systems [12] and a wide spectrum of coordination strategies have been proposed [13]. However, these strategies have typically relied on centralised or consensus-based approaches to globally adapt the system in a controlled manner. While these may be reasonable approaches for closed environments, they are not viable for open, dynamic, and heterogeneous systems [14, 15, 16].

One of the main challenges that an autonomous self-adaptation of individual nodes pose when designing a decentralised self-organising system is that we cannot anticipate the entire set of possible environmental conditions and their respective adaptation specifications. Components are thus faced with uncertainty as what, when, and how to adapt to a changing environment in order to maintain desired system-wide properties [17]. Managing this uncertainty, and its inevitable complexity, requires a tradeoff between flexibility and assurance that the critical goals of the system are always met [18]. This paper discusses these challenges and proposes a decentralised coordination model based on an effective feedback mechanism to reduce the time and complexity of the needed interactions among nodes until a collective adaptation behaviour is determined. By exchanging feedback on the desired self-adaptive actions, nodes converge towards a global solution, even if that means not supplying their individually best solutions. As a result, each node, although autonomous, is influenced by, and can influence, the behaviour of other nodes in the system.

The remainder of this paper is organised as follows. Section 2 discusses the challenges of binding individual autonomous activities into a collective acceptable behaviour. Section 3 introduces the CooperatES framework, a QoS-aware framework that addresses the increasing demands on resources and performance by allowing computationally expensive services to be cooperatively executed by temporary coalitions of nodes. Section 4 describes the system model and used notation, followed by a detailed description of the proposed decentralised coordination model, in Section 5. Section 6 analyses and validates the properties of the proposed model. Section 7 shows and discusses, based on the results of extensive simulations, the efficiency of the proposed coordination model to coordinate the self-adaptive behaviour of autonomous nodes and maintain desirable system-wide properties in decentralised cooperative systems with timing constraints. Finally, Section 8 concludes the paper.

2. Self-managed distributed systems

Managing the operation of increasingly complex and uncertain open distributed systems is a central challenge in real-time systems research [5]. Open distributed systems have to cope with remoteness of components, concurrency, the lack of a global state, asynchrony of state changes, heterogeneity, autonomous entities, and evolution of their configurations. It is now widely acknowledged that the needed degree of self-adaptiveness cannot be achieved without binding independent activities into a collective acceptable behaviour [19, 20, 21].

A self-adaptive system needs to observe its own behaviour, analyse those observations, and determine appropriate adaptation actions on one or more running services. However, as a service adapts and evolves, we face the problem of preserving an accurate and consistent model of the service's architecture and its constituent parts. 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. Dealing with such changes requires coordination so that the system's functionality is preserved. Coordination is then the logic that binds independent activities together into a collective activity [12].

Consider a distributed video conferencing application where the end-to-end latency increases due to network congestion. Examples of possible adaptation actions are reducing security, adding compression in the underlying network protocol, or reducing the picture's size, colour, or resolution at the application layer. Without coordination, both layers may independently make the decision on whether and by how much to adapt, potentially leading to unnecessary overcorrection. With coordination, however, the overall change can be more precisely tailored to a particular situation.

As another example, consider a mobile device with limited available power running several adaptive network-based applications. Whenever the battery level falls below some specified threshold, one of the applications is triggered to reduce its power consumption by reducing the network interface's utilisation. However, as the first application adapts and releases some of its share of the overall bandwidth, this action may be interpreted independently by the other applications as an increase in available network bandwidth and may consequently increase their use of the network. In this case, the request to utilise available bandwidth is in direct conflict with the initial goal of reducing power consumption.

As such, adaptation actions within a given host demand the coordination across local components and layers to avoid inconsistent changes or overreactions in response to the changing environmental conditions [22, 20, 23]. The problem is exacerbated, however, among interdependent autonomous hosts of a distributed system, where there is the need to ensure that local individual adaptation actions will produce a globally acceptable solution [24, 25]. Yet, coordinating autonomous interdependent runtime adaptations in distributed environments is not an easy task [26].

Guaranteeing globally optimal adaptation decisions requires a complete information about the state of all components, an optimal decision policy, and support for a strong consensus among components on the actions to be taken. Even when the goal is to reach a near-optimal global solution, coordination may require complex communication and synchronisation strategies. This is caused by interdependencies among components, as well as among components and their execution environment [8, 27, 28, 29].

In several architectures, the monitoring of the system's state and execution of adaptive actions is typically coordinated by a global manager [30, 31, 28, 32], a centralised component in the distributed system that monitors the state of all components and connectors and coordinates the determined adaptive actions on these components and connectors. While the use of centralised entities to support rule enforcement and self-managing behaviour is a reasonable approach for a closed environment, it has several disadvantages in open, dynamic, and heterogeneous systems [14, 33]. With the increasing size and complexity of open distributed systems, the ability to build self-managed distributed systems using centralised coordination models is reaching its limits [15], as solutions they produce require too much global knowledge. Thus, there is a trend towards using decentralised coordination techniques to establish and maintain system-wide properties over collections of components. How these components should interact in order to collectively solve a problem (and not what they were actually doing) became the focus of decentralised coordination research.

Georgiadis et al. [28] introduced the notion of a self-organising distributed system as "one in which components automatically configure their interactions in a way that is compatible with an overall architectural specification". However, although their work describes a software architecture that organises itself through local architectural constraints, it does not provide a model for building large decentralised systems, as the coordination model is consensus-based, requiring components to broadcast local changes to all other components in order to maintain common views on the system. This is known to produce communication overhead when establishing an agreement on changes to the shared model, which limits its scalability [34].

Goldin and Keil [33] describe a system built using a decentralised coordination model as a self-organising system whose system-wide behaviour is established and maintained solely by the local interactions of its components, which executes using only a partial view of the system. A commonly cited example of a complex distributed system with decentralised control comes from the world of biology - the socially intelligent colony of ants [35]. Without any central coordination, the colony displays emergent behaviours (e.g., finding optimal foraging paths) and structures (e.g., organised piles of dead ants appear around the nest site). While individual ants do not have a global view of the colony, intelligent global behaviour and functionality emerges solely from local interactions.

The benefits of such decentralised approach include the lack of centralised points of failure or attack [15], improved scalability, as well as a possible evolution of the system through evolving the local rules of their agents [36]. Furthermore, even if decentralised coordination models cannot achieve a strong consensus on the optimal adaptation actions to execute [37], it is still possible to achieve near-optimal decision policies through the localised coordination of components for certain classes of applications [38, 39, 21]. On the other hand, global solutions are typically accomplished through the exchange of multiple messages to ensure that all involved nodes make compatible decision about whether and how to adapt. Also, without proper control, it may become 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 its components can behave [40].

Control-theoretic approaches have been applied to a number of computing systems [41]. Of particular interest to distributed real-time embedded systems are, for example, the works reported in [5, 42]. The goal of the DEUCON algorithm [5] is to enforce, despite significant uncertainties in system workloads, the desired CPU utilisations on all the processors in a distributed system where a task may comprise a chain of subtasks on different processors. In contrast to the per-processor aggregate utilisation control approach of DEUCON, the DFCS framework [42] handles independent tasks via a combination of local QoS level adaptation and task (re)allocation to support data-driven applications with unpredictable and changing resource requirements. While we certainly share some concerns with these works, there are several important differences between our work and those projects. The CooperatES framework applies global multi-resource utility-based adaptation strategies to nodes with a significant degree of autonomy, capable of cooperatively performing tasks and sharing resources with other nodes.

In autonomic systems, Accord [43] is a component-based programming model and framework, designed to support the development of autonomic applications in closed distributed environments that can be managed at runtime using high-level rules defined by users. The authors identify challenges inherent in the construction of large-scale autonomic distributed applications including heterogeneity and dynamism in the availability and state of components, services and infrastructure. Behaviour rules control the runtime functional behaviour of autonomic components while interaction rules control the interactions between components, between components and their environments, and coordination within an autonomic application. In Accord, the centralised coordination of the adaptation of workflows is supported by a composition agent. However, there is no mechanism provided for the automated evaluation and updating of interaction rules at runtime, and Accord requires an administrator to manually evaluate the behaviour of the system and insert/remove/replace the appropriate interaction rules using the configuration agent. Accord also supports the definition and execution of decentralised coordination models specified as programmable reactive behaviours in a tuple-space. A reactive tuple consists of a condition for triggering some reactive behaviour, the reactive behaviour itself and a guard for defining the execution semantics of the reactive behaviour. There are no decentralised coordination models presented, however.

In [21], a collaborative reinforcement learning (CRL) approach is introduced as a decentralised coordination model that can coordinate the adaptive behaviour of groups of connected agents for the purpose of establishing system-wide autonomic properties in dynamic and uncertain environments. CRL models system optimisation problems as a set of discrete optimisation problems (DOP) that can be initiated at any agent and solved at any agent in the system. A DOP is defined as the combinatorial optimisation problem of finding the agent from a discrete set of agents that can solve a particular problem with the lowest cost. However, CRL assumes agents are homogeneous. In today's open distributed environments, the assumption that learning agents that join a system are homogeneous is becoming increasingly unrealistic.

Cholla [23] is a software architecture based on Cactus [44] that supports coordinated adaptations through separate centralised controllers that are constructed from composable fuzzy logic rule sets. These adaptation controllers encapsulate the adaptation policy decisions of how and when adaptive components react to changes in the environment. The architecture has its main focus on composing and coordinating control strategies for multiple adaptive components on a single host and only limited coordination between hosts is included in the runtime system of Cholla. Coordination among hosts follows the protocol detailed in [45], a consensus-based model that gathers information from remote nodes to allow controllers to determine a global adaptive action. The protocol, however, is unable to handle conflicting control desires of nodes and independent applications.

The limited applicability of these coordination models to heterogeneous distributed real-time systems provides the significant motivation for the development of a new decentralised coordination model that also reasons about the duration and overhead of the coordination process. As with all aspects of the design of distributed real-time systems, the development of a coordination model is constrained by stringent architectural properties. The proposed coordination model needs to fit these requirements but, nevertheless, be able to efficiently handle the challenges of an autonomous adaptation to environmental changing conditions.

Our goal is then to achieve a manageable, efficient, and predictable convergence to a global solution through a regulated decentralised coordination without overflowing nodes with messages. The next section details the proposed decentralised coordination model. The model is based on an effective feedback mechanism that reduces the complexity of the needed interactions among nodes until a collective adaptation behaviour is determined.

3. The CooperatES framework

The Cooperative Embedded Systems (CooperatES) framework [6] is primarily focused on open and dynamic environments where new services can appear while others are being executed, the processing of those services has associated real-time execution constraints, and component-based services can be executed by a coalition of neighbour nodes. In [46], we provide a complete description of the server-based schedulers used in the CooperatES framework, simultaneously dealing with capacity sharing, stealing and exchanging. Hard schedulability guarantees can be provided either for independent and interdependent task sets, even when hard and soft real-time tasks do share resources and exhibit precedence constraints.

The goal of the framework is to enable resource-constrained devices to solve computationally expensive services by redistributing parts of the service onto other devices, forming temporary coalitions for a cooperative service execution. Each node has a significant degree of autonomy and it is capable of performing tasks and sharing resources with other nodes. Nodes may either cooperate because they cannot deal alone with the resource allocation demands imposed by users and services or because they can reduce the associated cost of execution by working together.

A service $S = \{c_1, c_2, \ldots, c_n\}$ is considered to be formed by a set of interdependent software components c_i. Each component $c_i \in S$ is an entity that is defined by its functionality, is able to send and receive messages, is available at a certain node of the network, and has a set of QoS parameters that can be changed in order to achieve an efficient resource usage that constantly adapts to the specific constraints of the devices, nature of executing tasks, and dynamically changing system conditions.

The framework is composed by several modules, as depicted in Figure 1. A detailed description of the framework's model is outside the scope of this paper. However, a brief description helps to fully understand the proposed coordination model to maintain feasible QoS levels in distributed and dynamic open environments.

Whenever a user wants to execute some QoS-aware service S in a computational device, it requests execution to the framework through the *Application Interface*, thus providing explicit admission control, while abstracting the underlying middleware and operating system.

Given the heterogeneity of services to be executed, users' quality preferences, underlying operating systems, networks, devices, and the dynamics of

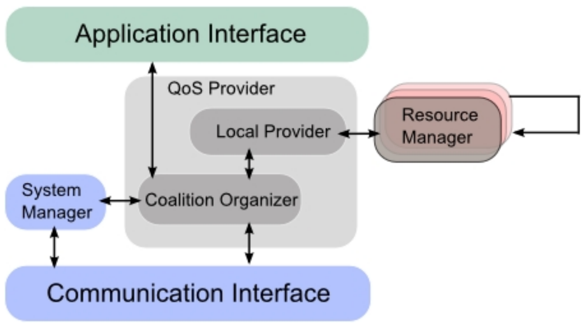

Figure 1: CooperatES Framework

their resource usages, QoS specification becomes an important issue in the context of a distributed QoS-aware cooperative service execution framework. The QoS scheme adopted in the CooperatES framework defines, for each application domain, its quality dimensions, attributes and values, as well as relations that map dimensions to attributes and attributes to values [6].

Having a QoS characterisation of a particular application domain, users and service providers are able to define service requirements and proposals in order to reach an agreement on service provisioning through a utility model. Several works associate with each pre-defined QoS level a utility function that specifies the user's benefit in obtaining service within those values [3, 10, 47]. However, it may be clearly infeasible to make the user specify an absolute utility value for every pre-defined quality choice. While we want a semantically rich request in order to achieve a service provisioning closely related to the user's quality preferences, we also want the user to actually be able to express personal QoS preferences in a service request. A more natural and realistic way is to simply impose a service request based on a qualitative, not quantitative, measure. With a relative decreasing order on quality dimensions, their attributes, and accepted values, a user is able to encode the relative importance of the new service's performance at the different QoS levels without the need to quantify every quality tradeoff with absolute values.

Thus, in the CooperatES framework users provide a single specification of their own range of QoS preferences $[L_{desired}, L_{minimum}]$ in decreasing preference order for a complete service S, ranging from a desired QoS level $L_{desired}$ to the maximum tolerable service degradation, specified by a minimum acceptable QoS level $L_{minimum}$, without having to understand the individual components that make up the service. It is assumed that a service can be executed at varying levels of QoS to achieve an efficient resource usage that constantly adapts to the specific constraints of devices, nature of executing tasks and dynamically changing system conditions. We make the reasonable assumption that the execution modes of services associated with higher QoS levels require higher resource amounts.

The service request will then be handled by the device's *QoS Provider*, which is composed by the *Local Provider* and *Coalition Organiser* components. The *Local Provider* is responsible for determining if a local execution of the new service is possible within the user's accepted QoS range, by executing a local gradient descent anytime QoS optimisation algorithm, quadratic in the number of tasks and resources and linear in the number of QoS levels. The goal is to maximise the satisfaction of the new service's QoS constraints while minimising the impact on the current QoS of previously accepted services [6]. Rather than reserving local resources directly, it contacts the *Resource Managers* to grant the specific resource amounts requested by the service. Each *Resource Manager* is a module that manages a particular resource, and interfaces with the actual implementation in a particular system of the resource controller, such as the network's device driver, the CPU scheduler, or even with the software that manages other types of resources (such as memory).

If the resource demand imposed by the user's QoS preferences cannot be locally satisfied, the coalition formation process is handled by the *Coalition Organiser*. There will be a set of interdependent components to be collectively executed, resulting from partitioning the resource intensive service. Correct decisions on service partitioning must be made at runtime when sufficient information about workload and communication requirements become available [48], since they may change with different execution instances and QoS preferences of users.

The *Coalition Organiser* is then responsible for broadcasting each of the service's components description along with the user's allowed QoS range, evaluating all the received service proposals, and deciding which nodes will be part of the coalition. The *Coalition Organiser* interacts directly with the *System Manager* to know which nodes are able to participate in the coalition formation process. Therefore, the *System Manager* is responsible for maintaining the overall system configuration, detecting QoS-aware nodes entering and leaving the network, and managing the coalition's operation and dissolution.

Every neighbour node able to satisfy the request for executing a component $c_k \in S$ within the user's allowed QoS values, formulates a service proposal, according to the local anytime QoS optimisation algorithm discussed above and replies with both its service proposal P_k and its local reward R_k, resulting from its proposal acceptance. For the remaining of this paper, the service solution's reward indicates how close is the offered QoS level to the user's desired QoS level, according to the set of tasks being locally executed, their associated QoS constraints, and quality of received inputs. This distance is mapped into a value in the interval $[0, 1]$, with 1 being assigned to the service solution that completely satisfies the user's desired QoS level $L_{desired}$. How each node measures its local reward is detailed in [6].

The CooperatES framework differs from other QoS-aware frameworks by considering, due to the increasing size and complexity of distributed embedded real-time systems, the needed trade-off between the level of optimisation and the usefulness of an optimal runtime system's adaptation behaviour. Searching for an optimal resource allocation with respect to a particular goal has always been one of the fundamental problems in QoS management. However, as the complexity of open distributed systems increases, it is also increasingly difficult to achieve an optimal resource allocation that deals with both users' and nodes' constraints within an useful and bounded time. Note that if the system adapts too late to the new resource requirements, it may not be useful and may even be disadvantageous. This idea has been formalised using the concepts of anytime QoS optimisation algorithms, in which there are a range of acceptable solutions with varying qualities, adapting the distributed service allocation to the available deliberation time that is dynamically imposed as a result of emerging environmental conditions [46]. Nodes start by negotiating partial, acceptable service proposals that are latter refined if time permits, in contrast to a traditional QoS optimisation approach that either runs to completion or is not able to provide a useful solution. At each iteration, the proposed QoS optimisation tries to find a new feasible set of QoS levels with an increasing utility. This improvement is larger at the early stages of computation and diminishes over time.

The framework's anytime coalition formation process enables the selection of individual nodes that will constitute the best group to satisfy the user's QoS requirements. Such selection is based on their own resources and availability. As such, by best group, we mean the group formed by those nodes which offer service closer to the user's desired QoS level. Therefore, nodes dynamically group themselves into a new coalition, allocating resources to the new service and establishing an initial Service Level Agreement (SLA), a set of compatible QoS levels provided by all the components c_k that compose service S. As a result, each node n_i in a coalition commits to output a specific QoS level Q^k_{val} for a component c_k in a service S. Each of these commitments is represented by a triple (n_i, c_k, Q^k_{val}).

As an exemplifying scenario, let's consider a user that imposes a particular range of QoS levels to a QoS-aware application that captures, compresses and transmits frames of real-time video to end users, using its resource-constrained device. Assume that such request originated the dynamic formation of the coalition represented in Figure 2 and whose characteristics are detailed in the next table. The node receiving the final service is node **g** (the user's node). Recall that the achieved reward with the output QoS level at each node is a value in the interval $[0, 1]$, with 1 being assigned to the service solution that completely satisfies the user's desired QoS level $L_{desired}$.

Node	Reward of output QoS	Quality of received inputs
a	0.3	$\emptyset$
b	0.3	$\emptyset$
c	0.3	$\{(a, 0.3)\}$
d	0.3	$\{(b, 0.3)\}$
e	0.3	$\{(c, 0.3), (d, 0.3)\}$
f	0.4	$\{(e, 0.3)\}$

For the sake of clarity of presentation, the number of nodes in the coalition is small and each of those nodes is only executing one of the service's components. However, such limitations are not present in the framework. Users encode their own relative importance of the different QoS parameters for each service and the framework uses this information to determine the distributed resource allocation that maximises the satisfaction of those constraints, which may result in more than one component being executed in the same node.

Commitments on output QoS levels that originated the selection of nodes for the coalition formation process must be valid local solutions, in the sense that assumptions on resource availability for each executed component should not be mutually inconsistent, and each commitment should satisfy the user's acceptable QoS levels at the time it is made. However, assumptions on resource availability have an associated uncertainty in open dynamic systems

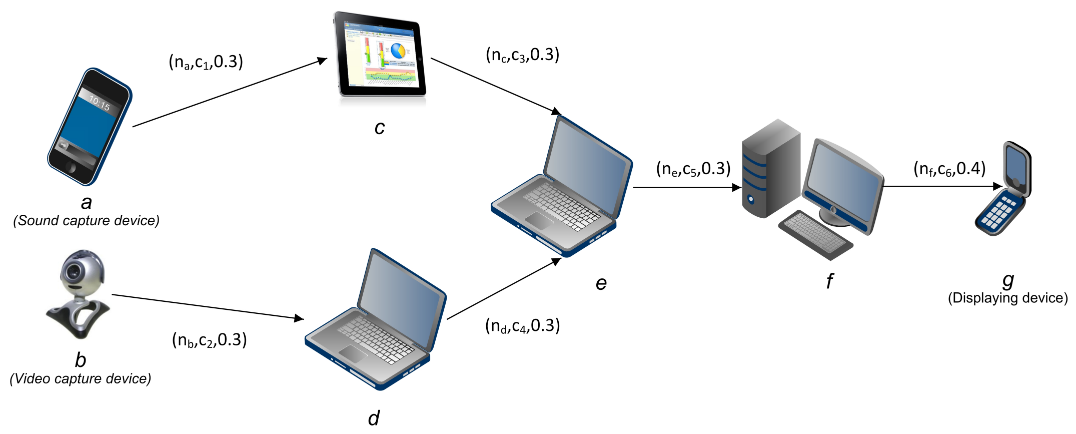

Figure 2: Dynamically formed coalition

due to the unpredictable arrival or departure of services. As such, the truth or falsity of a particular assumption may change, invalidating an existing commitment, forcing the node to adapt and make a new one.

Furthermore, QoS interdependencies among components of a service S imply that such autonomous adaptation actions of a node in a coalition may constraint current commitments at other coalition members. A QoS dimension Q_a is said to be dependent on another dimension Q_b if a change along the dimension Q_b will increase the needed resource demand to achieve the quality level previously achieved along Q_a [10]. Thus, the quality of the produced output of a component $c_i \in S$ may not only depend on the amount and type of locally reserved resources but also on the quality of the received inputs produced by other components.

QoS interdependencies explicitly express dependency relationships existing over the QoS characteristics of a service. Formally, there is a set of m QoS attributes $Attr = \{attr_1, attr_2, \ldots, attr_m\}$ of a service S, whose values are taken from the domains $D = \{D_1, D_2, \ldots, D_m\}$, respectively, and a set of interdependency constraints on their values. The constraint $p(attr_i, attr_j), \forall attr_i, attr_j \in Attr$ is a predicate that is defined on the Cartesian product $D_i * D_j, \forall D_i, D_j \in D$. This predicate is true if and only if the value assignment of these variables satisfies this constraint. Note that there is no restriction on the form of the predicate. It can be a mathematical or logical formula or any arbitrary relation defined by a tuple of acceptable values.

As a consequence of these interdependencies, the problem of maintaining a global consistent solution for a service S in the face of conflicting autonomous adaptation actions of nodes arises. This paper extends the CooperatES framework with a feedback-based decentralised coordination model that reduces the time and complexity of the needed interactions among interdependent nodes of a coalition until a collective adaptation behaviour is determined. The proposed model is general enough and completely independent from how the code to be executed on the original node's behalf arrives to the coalition members. It can be assumed that only nodes equipped a priori with a service's code blocks respond to a cooperation request, thus eliminating the need to migrate code at runtime and transfer the service's current state.

4. System model

We assume an open distributed system populated by several heterogeneous nodes, each with its specific set of resources. Requests for service, some of them with associated QoS constraints, can appear at any time while previously accepted services are being executed or terminated. Due to these characteristics, resource availability is highly dynamic and unpredictable in advance.

A service $S = \{c_1, c_2, \ldots, c_n\}$ is formed by a set of interdependent software components c_i. Each service has associated a range of acceptable QoS levels $[L_{desired}, L_{minimum}]$. To control the load imposed on system resources and, hence, guarantee a certain level of QoS, service acceptance and runtime adaptation must go through online admission control and resource reservation. Given a node n and a local set of QoS levels σ to be provided, it is considered that admission control is performed, and that therefore a system specific *feasibility* function (*e.g.* [49, 3, 50]) determines if a set of QoS requirements can be met with available resources.

Definition 1.

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

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

Thus, for some nodes, there may be a constraint on the type and number of components they can execute within acceptable QoS levels for their users. In this context, a cooperative QoS-aware execution of resource intensive services among neighbour nodes seems a promising solution [6]. The CooperatES framework enables a service to be executed either by a single or a group of nodes, depending on the capabilities of nodes and on the 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 services.

Interdependency relationships among components of a service S are 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.

Since the need for coordination arises from conflict-causing interdependencies, the feedback-based coordination protocol proposed in Section 5 uses these interdependencies to determine what, when, and to whom to communicate changes in current commitments. For the sake of simplicity, we present the following functions in a declarative notation with the same operational model as a pattern matching-based functional language.

Changes in interdependent commitments are propagated along $\mathcal{G_S}$ until a new globally acceptable service solution is found. For that, we define the *paths* and the $flatten$ functions. The *paths* function is a breadth first approach with cycle checking for determining nodes in paths. Visited nodes are added to the temporary set T in order to avoid infinite loops. The function outputs all the nodes in all the possible paths between two interdependent nodes n_i and n_j , or $\perp$ if there is no path between those two nodes. 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 nodes in these subsets will now belong to a simplified single set. Later in the paper, we use these functions in the proposed QoS upgrade and downgrade algorithms.

Given a DAG $\mathcal{G}_S = (\mathcal{V}_S, \mathcal{E}_S)$ and two nodes $n_i, n_j \in \mathcal{V}_S$, all the nodes in the possible paths between n_i and n_j are obtained as the result of the function:

$$\begin{array}{lcl}
paths(n_i, n_j) & = & flatten(paths(n_i, n_j, \emptyset)) \\
 & & \\
paths(n_i, n_j, T) & = & \emptyset, \text{ if } n_i = n_j \\
paths(n_i, n_j, T) & = & \{\{n_i, n_{k_1}\} \cup paths(n_{k_1}, n_j, T \cup \{n_{k_1}\}), \\
 & & \vdots \\
 & & \{n_i, n_{k_n}\} \cup paths(n_{k_n}, n_j, T \cup \{n_{k_n}\})\}, \\
 & & \forall n_{k_m} \in \mathcal{V}_S, \text{ such that } (n_i, n_{k_m}) \in \mathcal{E}_S \text{ and } n_{k_m} \notin T \\
paths(n_i, n_j, T) & = & \perp
\end{array}$$

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

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

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

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 service's DAG. An *user 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's output in the end-user's device. Source and user components mark the limit of a set of components that must be managed in a coordinated manner. Finally, we call *cut-vertex* to a node $n_i \in \mathcal{V}_S$, if the removal of that node 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 input quality by an increase in the amount of used resources or vice versa [9].

5. A feedback-based decentralised coordination model

Whenever the output QoS of some component depends not only on the amount and type of locally reserved resources but also on the quality of the received inputs sent by other components, it must be ensured that a source component provides a QoS which is acceptable to all consumer components and lies within the QoS range supported by the source component. This section proposes a decentralised, but still manageable, efficient, and predictable coordination model for open real-time systems.

Coordination involves managing the communication which is necessary due to the distributed nature of the system and QoS interdependencies among components. As a consequence of these interdependencies, the coordination problem becomes one of maintaining a global consistent solution for a service S in the face of conflicting autonomous adaptation actions of nodes. Thus, whenever a node autonomously changes its current commitment for a particular component $c_i \in S$, three questions must be answered: (i) is communication necessary?; (ii) what needs to be communicated?; and (iii) to whom should the communication be addressed?. In the next paragraphs, we outline a set of properties that a coordination protocol needs to enforce in order to answer these three questions in open and decentralised real-time systems.

A node's runtime adaptation to environmental changes can be viewed as a process of identifying alternative solutions and choosing the feasible resource allocation that maximises the satisfaction of the set of QoS constraints [6]. The new service solution becomes a *commitment* whenever it is communicated to its coalition partners. At this point, subsequent nodes in the coalition's DAG have the expectation that the commitment will be met and can use this knowledge in the elaboration of their own commitments. Thus, an *ordered sequence of changes* in the commitments of nodes answers the question of when communication is necessary. By ordering commitments according to interdependencies, commitments made by nodes that supply input for a component $c_i \in S$ must precede the commitment made by node n_i as only then has n_i incorporated the commitments of other nodes as constraints for its own local resource optimisation.

Coordinating a group of autonomous self-adaptive nodes, particularly in the presence of several different QoS dimensions of interest [6], has an overhead. In fact, in highly dynamic systems, the amount of degradation in QoS due to coordination could outweigh the benefits of adapting the system to changes. A coordination protocol should, therefore, ensure that the information that is exchanged among nodes is as concise as possible. *Conciseness* will then determine what should be communicated.

Completeness guarantees that, given a particular interdependency, all dependencies that follow from it can be determined. If the coordination protocol is not complete, then there is the possibility that inconsistencies in service provisioning will persist due to interdependencies that remain undetected. However, completeness does not guarantee that every detected interdependency will be a true interdependency. Thus, the set of nodes involved in the coordination process may contain nodes not affected by the new commitment. *Soundness* ensures that only true interdependencies will be detected and that the set of affected nodes is minimal. As such, *completeness and soundness* will then determine to whom should messages be sent.

In nature there is a well-known, pervasive notion that satisfies these desirable properties of coordination and can be successfully applied to decentralised software systems [51]: the notion of feedback. Nature provides plenty of examples of cooperative self-adaptive and self-organising systems, some of them are far more complex than distributed systems we design and build today [52]. These natural systems are often decentralised in such a way that participants do not have a sense of the global goal, but rather it is the interaction of their local behaviour that yields the global goal as an emergent property. Typically, when feedback is interpreted as negative feedback, it will act as a regulator on changes that deviate the system from some optimal state, whereas feedback perceived as positive will usually trigger adaptation actions that serve to increase changes in the same direction as the previous one.

These feedback loops can provide a generic mechanism for self-adaptation in distributed systems. To achieve this goal, the research community has increasingly explored the extent to which general principles of feedback are applicable when reasoning about self-adaptive software systems [51]. Nevertheless, to the best of our knowledge, feedback has not yet been adopted as a specific decentralised mechanism to coordinate the dynamic behaviour of an interdependent set of autonomous self-adaptive QoS-aware components working without any central control in open distributed real-time systems.

In the CooperatES framework, each of the components $c_i \in S$ may either have their currently output QoS levels downgraded (until $L_{minimum}$ is reached) in order to accommodate new service requests with a higher utility or upgraded (until $L_{desired}$ is reached) on an underutilisation of a particular coalition member. This implies that the beginning of a coordination process may either be a reflexive action, in case of a necessary downgrade, or a voluntary one, in case of an attempt to upgrade the currently provided QoS level of previously downgraded components.

Affected coalition partners then collect relevant data, both from the commitments of other nodes and from local resource reservations, that reflect the current state of the system. Subsequently, each involved node 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 coalition acts in a coordinated manner.

More formally, we model a self-managed coalition as a group of nodes that respond to interdependent changes on the output QoS of their components according to a decentralised coordination protocol defined by the following phases:

1. **Coordination request.** Whenever $Q^i_{val'}$, the needed downgrade or desired upgrade 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 other components $c_j \in S$, a coordination request is sent to the affected direct neighbours in $\mathcal{G}_S$.
2. **Feedback formulation.** Affected direct neighbours recompute their local set of SLAs using Algorithm 1 in order to formulate the corresponding feedback on the requested adaptation action. If a positive feedback is formulated, the coordination request is propagated to the next direct neighbour in $\mathcal{G}_S$, until the next cut-vertex n_c is reached. We assume that coalition partners are willing to collaborate in order to achieve a global coalition's consistency, even if this might reduce the utility of their local optimisations. However, a node only agrees with the requested adaptive action if and only if its new local set of SLAs is feasible.
3. **Coordinated adaptive action.** If the requested adaptive action is accepted by all the affected nodes, the new local set of SLAs is imposed at each of those coalition members. Otherwise, the currently global QoS level of service S remains unchanged.

A fundamental advantage of the proposed coordination model is that both the communication and adaptation overheads depend on the size of a node's neighbourhood until a cut-vertex is reached, instead of the entire coalition. This allows the proposed feedback-based coordination model to scale effectively to large distributed systems.

Although each individual node has no global knowledge about the coalition and its interdependencies as a whole (only knows to which direct neighbour(s) it sends its output), complex coordinated adaptations emerge from local interactions. Requests for coordination with a node $n_i \in \mathcal{G}_S$ may arrive dynamically at any time to any node $n_j \in \mathcal{G}_S$. The formulation of the corresponding positive or negative feedback at a node n_j depends on the feasibility of the new requested QoS level as a function of the quality of the new set of inputs I_{c_i} for the locally executed component c_i and the amount of locally available resources. Such feasibility is determined by the local QoS optimisation algorithm detailed in Algorithm 1.

To locally guarantee the coordination request through Algorithm 1, each node executes a gradient descent QoS optimisation, quadratic in the number of tasks and resources and linear in the number of QoS levels. The goal is to locally adapt the set of provided QoS levels in response to a coordination request and achieve a global consistency in a coalition's service execution, while minimising the impact on the current QoS levels of other locally accepted services. As a result, the solution's quality associated with the new local set of SLAs can be lower after the coordination request. However, recall that we make the assumption that, in cooperative environments, coalition partners are willing to collaborate in order to achieve a global coalition consistency, even if this coordination might reduce the global utility of their local QoS optimisations.

Whenever a coordination request arrives for a service S, the algorithm starts by maintaining the currently provided QoS levels of other services and by defining the new SLA for the component $c_i \in S$ according to the new quality of its inputs. The goal is to quickly find a feasible solution. If, on the other hand, QoS degradation is needed to accommodate the new SLA, the algorithm guides the search for a feasible solution by incrementally selecting the configuration that minimises the quality decrease of other previously accepted components, until a feasible solution is found (if it exists) or until it finds that, even at the lowest acceptable QoS level for each component, the resource demand imposed by the coordination request is not feasible. Note, however, that even if an unfeasible solution is determined at some iteration of the algorithm due to local resource availability, such solution is used to calculate the next possible solution, minimising the search effort.

When the algorithm terminates, if there are insufficient resources to accommodate the coordination request, a negative feedback is formulated and sent back in reply. On the other hand, if a positive feedback is formulated, the coordination request is propagated along the service's DAG. However, it may be possible to confine the coordination process to a subset of the

Algorithm 1 Feedback formulation

Let each locally running component c_i be composed by a set of tasks with an associated range of n user's acceptable QoS levels in decreasing preference order $[L_{desired}, L_{minimum}]$
Let each Q_{kj} be a finite set of n quality choices for the j^{th} attribute of the k^{th} QoS dimension, taken from the QoS characterisation of the particular service's domain
Let *desired*, *current*, and *minimum* be the indexes that give the respective value in Q_{kj}
Let $Q_{kj}[desired] \geq Q_{kj}[current] \geq Q_{kj}[minimum]$ be the currently provided level of service for the j^{th} attribute of the k_{th} QoS dimension
Let the granularity of quality decrease for the j^{th} attribute of the k_{th} QoS dimension from $Q_{kj}[current]$ to $Q_{kj}[current+1]$ be determined by the possible values in Q_{kj}
Let σ be the determined set of SLAs, updated at each step of the algorithm

```
1:  Define $SLA'_{c_i}$ as a function on the new input values $I_{c_i}$ and the required output level $Q^i_{val'}$ for component $c_i$
2:  Update the current set of SLAs $\sigma$ $\{\sigma \leftarrow \sigma \setminus SLA_{c_i} \cup SLA'_{c_i}\}$
3:  while $feasibility(\sigma) \neq$ TRUE do
4:    if there is no task $\tau_i$ being served at $Q_{kj}[current] > Q_{kj}[minimum]$, for any $j$ attribute of any $k$ QoS dimension then
5:      return FALSE
6:    end if
7:    for each component $c_d \subseteq \sigma \setminus c_i$ do
8:      for each task $\tau_i \in c_d$ do
9:        for each $j^{th}$ attribute of any $k$ QoS dimension in $\tau_i$ with value $Q_{kj}[current] > Q_{kj}[minimum]$ do
10:         Downgrade attribute $j$ to the previous possible value $Q_{kj}[current+1]$
11:         Determine the utility decrease of this downgrade
12:       end for
13:     end for
14:   end for
15:   Find task $\tau_{min}$ whose reward decrease is minimum
16:   Define $SLA'_{c_{min}}$ for the component where task $\tau_{min}$ belongs, setting the QoS values of all affected tasks according with the new value $Q_{kj}[current+1]$ for attribute $j$ of the QoS dimension $k$ for task $\tau_{min}$
17:   Update the current set of promised SLAs $\sigma$ $\{\sigma \leftarrow \sigma \setminus SLA_{c_{min}} \cup SLA'_{c_{min}}\}$
18: end while
19: return TRUE
```

service's DAG, if a cut-vertex is able to maintain its current output quality, despite the changes on the quality of its inputs. Recall that within a feasible QoS region, it may be possible to maintain the current output quality by compensating for a decrease in input quality by an increase in the amount of used resources or vice versa.

The *coordinated adaptive action* phase is initiated whenever a global coordination is successful. In this phase, each node commits to produce the new output quality and allocates the needed resources to achieve it. The resource allocation is always possible because sufficient resources were reserved during the feedback formulation phase. Once resources are allocated, the node commits to produce the announced output level until either the service terminates or adaptation occurs.

Decentralised control is then a self-organising emergent property of the system. By exchanging feedback on the performed self-adaptations, nodes converge towards a global solution, overcoming the lack of a central coordination and global knowledge. Negative feedback loops occur when a change in one coalition member triggers an opposing response that counteracts that change at other interdependent node along $\mathcal{G}_S$. On the other hand, positive feedback loops promote global adaptations. The snowballing effect of positive feedback takes an initial change in one node and reinforces that change in the same direction at all the affected partners, requiring only one negotiation round between any pair of interdependent nodes. As such, the uncertain outcome of iterative decentralised control models whose effect may not be observable until some unknowable time in the future is not present in the proposed regulated coordination model.

Note that the normal operation of nodes continues in parallel with the *coordination request* and *feedback formulation* phases. Every time a node recomputes its set of local SLAs, promised resources are pre-reserved until the global negotiation's outcome is known (or a timeout expires) but the currently provided QoS levels only actually change at the *coordinated adaptive action* phase, as a result of a successful global coordination, avoiding race conditions that could overcommit resources. Due to the environment's dynamism, more than one coalition member can give origin to a coordinated adaptation process that spans multiple nodes at a given time. As discussed above, such coordination requests can either be caused by a downgrade or an upgrade of a node's current commitment for a component $c_i \in S$. Even with multiple simultaneous coordinated global adaptations for the same service S, only one of those will succeed since, due to local resource limitations, only the minimum globally requested SLA will be accepted by all coordination participants. In order to manage these simultaneous negotiations, every negotiation has a unique identifier, generated by the requesting node. For the sake of clarity of presentation, this negotiation identifier is absent from the algorithms, functions, and proofs in this paper.

There is one downside of pre-reserving resources to avoid overcommitments when two or more upgrades are being handled over the same subset of nodes concurrently. It is possible that some of them be unnecessarily rejected, a problem known as distributed livelock. While handling distributed livelocks is an important issue, it is outside the scope of the work presented in this paper. Solving this livelock on the CooperatES framework can easily be done by expressing preferences or priorities among services and employ, for instance, a Criticality-Based Back-Off retry scheme [53]. Under this scheme, an aborted upgrade request will make another upgrade request again, after a waiting time based on its importance/criticality level.

5.1. The feedback-based coordination model in action

Let us consider the coalition presented earlier in Section 3 detailed in Figure 2. Assume that node **a** has now the sufficient resources to upgrade the output QoS level for component $c_1 \in S$ to a new QoS level with a reward for the user of 0.4. It propagates that intention to node **c**, its direct neighbour in $\mathcal{G}_S$. Assume that node **c**, through the execution of Algorithm 1 determines that it is able to follow the proposed upgrade, even with node **d** keeping its output QoS. Based on the positive feedback, the coordination request is propagated to the next direct neighbour in $\mathcal{G}_S$, node **e**.

Assume that the cut-vertices **e** and **f** are also able to follow the proposed upgrade. Since all the involved nodes in the path to the end-user's device were queried and the conjunction of their efforts results in a upgraded QoS level being delivered to the user, the entire coalition reaches the new global state represented in Figure 3 and whose characteristics are detailed in the next table.

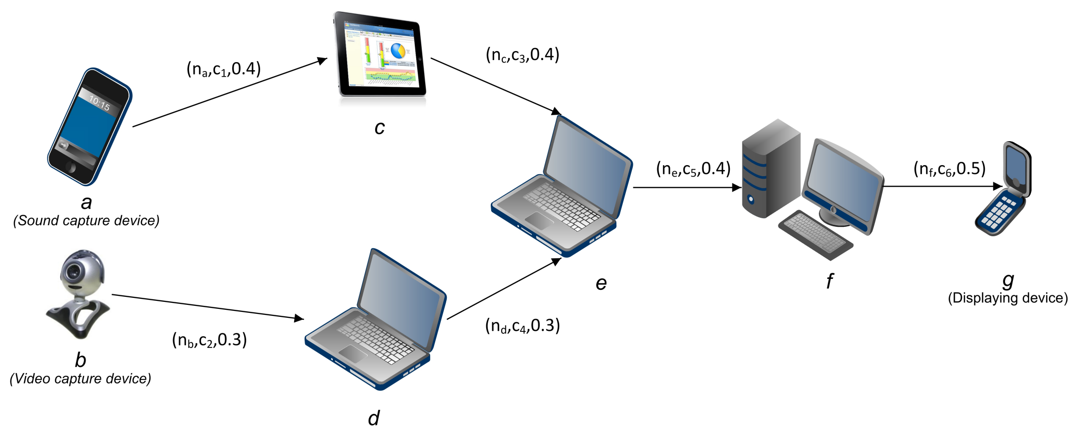

Figure 3: Coordinated QoS upgrade

Node	Reward of output QoS	Quality of received inputs
a	0.4	$\emptyset$
b	0.3	$\emptyset$
c	0.4	$\{(a, 0.4)\}$
d	0.3	$\{(b, 0.3)\}$
e	0.4	$\{(c, 0.4), (d, 0.3)\}$
f	0.5	$\{(e, 0.4)\}$

In another scenario, assume that the cut-vertex **e** has no sufficient available resources to follow the proposed upgrade (for instance, due to limitations imposed by the amount of remaining battery life). Then, all the previous nodes (**a** and **c**) that were pre-reserving resources for the upgrade, are informed to stay in their previous respective QoS levels, as their effort is useless. In this case, the global coalition state remains the same, as the one presented in Figure 2.

Lets now consider a needed downgrade of the output QoS level at one coalition member. Assume that the coalition is at the global state represented in Figure 3 and that node **b** is forced to change its output QoS for a new level with reward 0.1 for the user. A coordination request is sent to node **d**, its direct neighbour in $\mathcal{G}_S$. Now, suppose that the cut-vertex **d** cannot compensate this degraded output and is also forced to downgrades its output QoS for a new level with reward 0.1. Then, the coordination request is propagated to **d**'s direct neighbour in $\mathcal{G}_S$, node **e**.

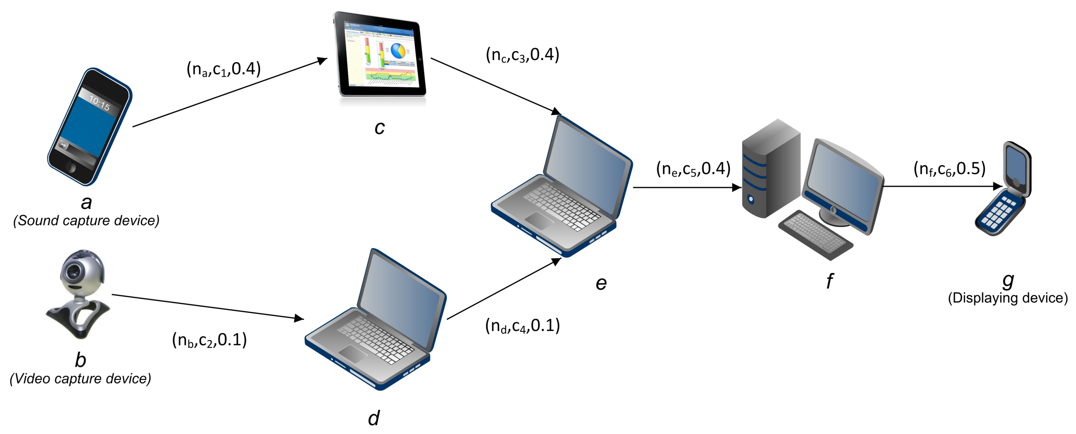

Figure 4: Coordinated compensated downgrade

Now, assume that the cut-vertex **e** in $\mathcal{G}_S$, even with a reduced input quality from node **d**, is able to keep its current QoS level, rewarded with 0.4, by compensating the reduced input quality with an increased local resource usage. In this case, the coordination request is restricted to the subgraph between node **b** and node **e** and the coalition reaches the new global state represented in Figure 4. The list of the properties follows

Node	Reward of output QoS	Quality of received inputs
a	0.4	$\emptyset$
b	0.1	$\emptyset$
c	0.4	$\{(a, 0.4)\}$
d	0.1	$\{(b, 0.1)\}$
e	0.4	$\{(c, 0.4), (d, 0.1)\}$
f	0.5	$\{(e, 0.4)\}$

6. Properties of the proposed decentralised coordination model

In this section we analyse the most important properties of the proposed feedback-based decentralised coordination model. Although a node is only aware of its nearest neighbours in a coalition, in this section, we deal with the complete interdependency graph to prove that all the conducted actions are correctly propagated until the final result is determined. We start with some auxiliary definitions and proofs.

Lemma 1 (Correctness of feedback formulation). *Algorithm 1 always terminates and returns true if the new required set of SLAs for outputting the QoS level $Q^i_{val'}$ at component c_i is feasible or false otherwise.*

Proof 1. *Termination comes from the finite number of tasks τ_i being executed in node n_i and from the finite number of the k QoS dimensions and j attributes being tested. The number of QoS attributes being manipulated decreases until a task τ_i is configured to be served at its lowest admissible QoS level $Q_{kj}[n]$, thus leading to termination.*

Correctness comes from the heuristic selection of the QoS attribute to downgrade at each iteration of the algorithm. Thus, after a finite number of steps the algorithm either finds a new set of feasible SLAs that complies with the coordination request or returns false if, even when all tasks are configured to be served at their lowest requested QoS level, the requested SLA for a component c_i cannot be supplied.

Remark 1. *Given a node n_i, a component c_i, the set of local SLAs $\sigma = \{SLA_{c_0}, \dots, SLA_{c_p}\}$ for the p locally executed components, $Q^i_{val'}$ as the new requested QoS level for c_i, and $I_{c_i} = \{(n_j, c_j, Q^j_{val}), \dots, (n_k, c_k, Q^k_{val})\}$ as the set of QoS levels given as input to c_i, for the remaining of this section and for the sake of clarity, we refer to Algorithm 1 applied to node n_i as $test\_feasibility(n_i, c_i, Q^i_{val'}, I_{c_i})$*

Given a connected graph $\mathcal{G}_S = (\mathcal{V}_S, \mathcal{E}_W)$, such that the component $c_i \in S$ is being processed by node $n_i \in \mathcal{V}_S$, and $I_{c_i} = \{(n_j, c_j, Q^j_{val}), \dots, (n_k, c_k, Q^k_{val})\}$ 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:

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

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

Given a node n_i and a component c_i, we define the function $get\_input\_qos(n_i, c_i)$ as returning the set of elements (n_j, c_j, Q^j_{val}), where each of these elements represents a component c_j being executed at node n_j with an output QoS level of Q^j_{val} used as an input of the component c_i at node n_i.

6.0.1. Coordinating upgrades

Given the connected graph $\mathcal{G}_S = (\mathcal{V}_S, \mathcal{E}_S)$ with a set of cut-vertices $\mathcal{C}_S$ and an end-user node n_u receiving the final outcome of the coalition's processing of service S, whenever a node $n_i \in \mathcal{V}_S$ is able to upgrade the output QoS of a component $c_i \in S$ to a new QoS level $Q^i_{val'}$, subsequent nodes in the coalition respond to this upgrade request according to Algorithm 2.

Algorithm 2 Coordinating upgrades

```
1:  $temp := n_i$
2:  $U := \emptyset$
3:  for each $n_c \in \mathcal{C}_S \cup \{n_u\}$ do
4:      if $upgrade(temp, n_c, \mathcal{G}_S, Q^{tmp}_{val'}) = (\textbf{TRUE}, T)$ then
5:          $temp := n_c$
6:          $U = U \cup T$
7:      else
8:          $U = \emptyset$
9:          return
10:     end if
11: end for
12: for each $(n_i, Q^i_{val'}) \in U$ do
13:     Set the new QoS level $Q^i_{val'}$ for component $c_i \in S$
14: end for
```

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 node n_i to next cut-vertex $n_c \in \mathcal{C}_{\mathcal{S}}$, the function $upgrade(n_i, n_c, \mathcal{G}_S, Q^i_{val'})$ is defined by:

```
function upgrade($n_i, n_c, \mathcal{G}_S, Q^i_{val'}$)
$T := \{(n_i, c_i, Q^i_{val'})\}$
for each $n_j \in paths(n_i, n_c) \setminus \{n_i\}$ do
  $S := update(get\_input\_qos(n_i, c_i), T)$
  if $test\_feasibility(n_j, c_j, Q^j_{val'}, S) =$ TRUE then
    $T := T \cup \{(n_j, Q^j_{val'})\}$
  end if
end for
$S := update(get\_input\_qos(n_c, c_c), T)$
if $test\_feasibility(n_c, c_c, Q^c_{val'}, S) =$ TRUE then
  return (TRUE, $T$)
end if
return (FALSE, $\emptyset$)
```

Lemma 2. *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 node n_i to next cut-vertex $n_c \in \mathcal{C}_\mathcal{S}$ and a new upgraded QoS level value $Q^i_{val'}$, the call to upgrade$(n_i, n_c, \mathcal{G}, Q^i_{val'})$ terminates and returns true if n_c is able to output a new QoS level $Q^c_{val'}$ or false otherwise.*

Proof 2. *Since $\mathcal{V}$ is a finite set, and since by Proposition 2 paths terminates, and by Proposition 3 update terminates, the number of iterations is finite due to the finite number of elements in the path. Thus, upgrade terminates.*

For any element n_j in the path between n_i and n_c, the new required QoS level $Q^j_{val'}$ is tested and, by Lemma 1, the upgrade is possible if and only if the new local set of SLAs is feasible. After considering all nodes in the path, the upgrade function returns true and the set of nodes able to upgrade, if node n_c is able to upgrade to $Q^c_{val'}$, or false otherwise. Thus, the result follows by induction on the length of the set of elements in the paths between n_i and n_c.

Theorem 1 (Correctness of Upgrade). *Given the connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ representing the QoS interdependencies of a service S being executed by a coalition of nodes, such that $n_u \in \mathcal{V}$ is the end-user node receiving the service at a QoS level Q_{val}, whenever a node n_i announces an upgrade to $Q^i_{val'}$, Algorithm 2 changes the set of SLAs at nodes in $\mathcal{G}$ such that n_u receives S upgraded to the QoS level $Q^u_{val'}$ or does not change the set of local SLAs at any node and n_u continues to receive S at its current QoS level Q^u_{val}.*

Proof 3. *Termination comes from the finite number of elements in $\mathcal{C} \cup n_u$ and from Lemma 2.*

Algorithm 2 applies the function upgrade iteratively to all nodes in the subgraph starting with n_i and finishing in n_u. The base case is when there are no cut-vertices and there is only one call to upgrade. It is trivial to see that the result of upgrade will consist in true and a set of nodes that will upgrade for the new QoS level $Q^j_{val'}$ or false and an empty set and, by Lemma 2, it is correct. The remaining cases happen when there are one or more cut-vertices between n_i and n_u. Here, upgrade will be applied to all subgraphs starting in n_i and finishing in n_u. Each of these subgraphs are sequentially tested. Only if all of them can be upgraded will service S be delivered to node n_u at the new upgraded QoS level $Q^u_{val'}$. The result follows by induction in the number of cut-vertices.

6.0.2. Coordinating downgrades

Given the connected graph $\mathcal{G}_S = (\mathcal{V}_S, \mathcal{E}_S)$ with a set of cut-vertices $\mathcal{C}_S$ and an end-user node n_u receiving the final outcome of the coalition's processing of service S, whenever a node $n_i \in \mathcal{V}_S$ is forced to downgrade the output quality of a component $c_i \in S$ from its current QoS level of Q^i_{val} to a downgraded QoS level $Q^i_{val'}$, subsequent nodes in the coalition respond to this downgrade request according to Algorithm 3.

Algorithm 3 Coordinating downgrades

```
1: $temp := n_i$
2: for each $n_c \in \mathcal{C}_S \cup \{n_u\}$ do
3:     if $downgrade(temp, n_c, \mathcal{G}_S, Q^{tmp}_{val'}) = $ FALSE then
4:         $temp := n_c$
5:     else
6:         Downgrade was compensated and $n_c$ continues to output $Q^c_{val}$
7:         break
8:     end if
9: end for
```

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 node n_i to next cut-vertex $n_c \in \mathcal{C}_S$, the function $downgrade(n_i, n_c, \mathcal{G}_S, Q^i_{val'})$ is defined by:

```
**function** downgrade($n_i, n_c, \mathcal{G}_S, Q^i_{val'}$)
$T := \{(n_i, Q^i_{val'})\}$
**for each** $n_j \in paths(n_i, n_c) \setminus \{n_i\}$ **do**
    $D := update(get\_input\_qos(n_j, c_j), T)$
    **if** $test\_feasibility(n_j, c_j, Q^j_{val}, D)$ = **TRUE then**
        $T := T \cup \{(n_j, Q_{val})\}$
    **else**
        $set\_qos\_level(n_j, c_j, Q^j_{val'})$
    **end if**
**end for**
$D := update(get\_input\_qos(n_c, c_c), T)$
**if** $test\_feasibility(n_c, c_c, Q^c_{val}, D)$ = **TRUE then**
    **return TRUE**
**else**
    **for each** $n_j \in paths(n_i, n_c) \setminus \{n_i\}$ **do**
        $set\_qos\_level(n_j, c_j, Q^j_{val'})$
    **end for**
    **return FALSE**
**end if**
```

Lemma 3. *Given the connected graph $\mathcal{G}_S = (\mathcal{V}_S, \mathcal{E}_S)$ with a set of cut-vertices $\mathcal{C}_S$, the subgraph that connects node n_i to next cut-vertex $n_c \in \mathcal{C}_S$, and a new downgraded QoS level value $Q^i_{val'}$, the call to downgrade$(n_i, n_c, \mathcal{G}_S, Q^i_{val'})$ terminates and returns true if n_c is able to keep its current output level Q^c_{val} or false otherwise.*

Proof 4. *Since $\mathcal{V}_S$ is a finite set, and since by Proposition 2, paths terminates, and by Proposition 3 update terminates, the number of iterations is finite due to the finite number of elements in the path. Thus, downgrade terminates.*

For any element n_j in the possible paths between n_i and n_c, it is tested if that node, given its new set of inputs I_{c_j}, can continue to output its current QoS level Q^j_{val}. After considering all j nodes in the possible paths, the downgrade function returns true, if node n_c is able to continue to output Q^c_{val}, or sets all the j previous nodes in the possible paths between n_i and n_c to the downgraded QoS level $Q^j_{val'}$ and returns false. Again the result follows by induction on the length of the set of elements in the paths between n_i and n_c.

Theorem 2 (Correctness of Downgrade). *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 coalition of nodes such that $n_u \in \mathcal{V}_S$ is the end-user node receiving S at the QoS level Q^u_{val}, whenever a node n_i is forced to downgrade the output quality of a component $c_i \in S$ from its current QoS level of Q^i_{val} to a degraded QoS level $Q^i_{val'}$, Algorithm 3 changes the set of SLAs at nodes in $\mathcal{G}_S$ such that n_u continues to receive S at its current QoS level Q^u_{val} or sets all nodes to a degraded QoS level of $Q^j_{val'}$.*

Proof 5. *Termination comes from the finite number of elements in $\mathcal{C}_S \cup n_u$ and from Lemma 3.*

The correctness trivially follows by the correctness of Lemma 3 and by induction on the number of elements in $\mathcal{C}_S \cup \{n_u\}$.

6.1. Number of exchanged messages

Given the formalisation of upgrades and downgrades of the currently supplied QoS level for a service S, here we analyse the number of exchanged messages in such coordination operations. We start with some definitions.

Definition 2. *Given a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the in-degree of a node $n_i \in \mathcal{V}$ is the number of edges that have n_i as their destination.*

Definition 3. *Given a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the out-degree of a node $n_i \in \mathcal{V}$ is the number of edges that have n_i as their starting node.*

Whenever an upgrade to a new QoS level $Q^i_{val'}$ is requested by a node n_i, if the next cut-vertex n_c in $\mathcal{G}$ cannot supply the requested upgrade, then all the precedent nodes between n_i and n_c are kept in their currently supplied feasible QoS level Q^j_{val}. Thus, the number of needed messages is given by the number of edges in the paths between the n_i and n_c where it was determined that the requested upgrade was not possible. On the other hand, if the upgrade is possible, the number of needed messages is twice the number of edges between n_i and the end-user node n_u. This is because an upgrade is only possible after all the involved nodes are queried and the conjunction of their efforts results in a upgraded QoS level being delivered to n_u.

Whenever, due to resource limitations, a node n_i announces a downgrade to $Q^i_{val'}$, the next nodes in the sub-graph from n_i to the next cut-vertex n_c try to compensate the downgraded input quality in order to keep outputting the previous QoS level Q^j_{val}. When the cut-vertex n_c is reached two scenarios may occur. In the first one, the cut-vertex cannot compensate the service degradation, although some of its precedent nodes may. In this case, all the precedent nodes are informed that they can downgrade their current QoS level to $Q^j_{val'}$ since their compensation effort is useless. Note that, in the worst case, this can be propagated until the final node n_u is reached and all the coalition members will downgrade their current QoS level. As such, in the worst case, a message is sent from each node to its adjacent ones and a reply is received, which demands a total number of messages of two times the number of edges between n_i and n_u. On the other hand, in the second possible scenario, some cut-vertex n_c may be able to compensate the downgraded input quality and continue to output is current QoS level Q^c_{val}. In this case, the coordination process is restricted to the subgraph between n_i and n_c. As such, coordination messages are exchanged in this subgraph only.

Thus, in the worst case, the maximum number of exchanged messages in a coordination operation is given by Equation 1.

$$\sum_{n \in \mathcal{V}} (out\_degree(n) + in\_degree(n)) \tag{1}$$

7. 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 self-adaptive behaviour of autonomous nodes and maintain desirable system-wide properties in decentralised cooperative systems with timing constraints.

The key concepts underlying the proposed feedback-based coordination model have been prototyped and evaluated using an application that captures, compresses and transmits frames of real-time video to end users, which may use a diversity of end devices and have different sets of QoS requirements. The application is composed by a set of source components to collect the data, compression components to gather and compress the data sent from multiple sources, transmission components to transmit the data over the network, decompression components to convert the data into the user's specified format, and a user component to display the data in the end user's device.

An important requirement for the experiments was to introduce a high variability in the characteristics of the considered topologies of the randomly generated coalitions. The characteristics of end devices and their more powerful neighbour nodes was randomly generated, creating a distributed heterogeneous environment. At randomly selected end devices, multiple new service requests with random ranges of acceptable QoS levels and lifetimes were generated, dynamically imposing different amounts of load and resource availability. The non-equal partition of resources imposed to nodes affected the ability of some nodes to singly execute all of the application's components and has driven nodes to a coalition formation for a cooperative service execution, using the anytime coalition formation and service proposal formulation algorithms of [6].

At each simulation run, the number of simultaneous nodes in the system varied from 10 to 100, from which a coalition is dynamically formed at each new service request, while the number of simultaneous users varied from 1 to 20. Each node was running a prototype implementation of the CooperatES framework [54], 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. After a coalition was dynamically determined, a varying number of QoS interdependencies among the application's components was randomly generated, creating topologies with different degrees of complexity.

The performance of the proposed decentralised feedback-based coordination protocol was compared to representative examples of classical methodologies that rely on centralised [31] or consensus-based [23] approaches to establish and maintain system-wide properties, and are implemented using techniques such as group communication protocols that ensure that all coalition members receive all messages sent to the group in the same total order or centralised configuration managers that monitor the state of all components and connectors and coordinate the determined adaptive actions on these components and connectors in dynamic software architectures.

The reported results were observed from multiple and independent simulation runs, with initial conditions and parameters, but different seeds for the random values[1] used to drive the simulations, obtaining independent and identically distributed variables. The mean values of all generated samples and their standard deviation were used to produce the charts, with a confidence level of 99,9% associated to each confidence interval.

The first study compared the total number of messages that had to be exchanged among nodes when using the three coordination approaches until a global adaptation solution was determined. Since the number of exchanged messages among nodes increases proportionally with the number of interdependent nodes in a coalition, the coordination model should help a coalition to globally adapt in a coordinated manner with a minimal number of messages. The average results of all simulation runs for different coalition sizes are plotted in Figure 5.

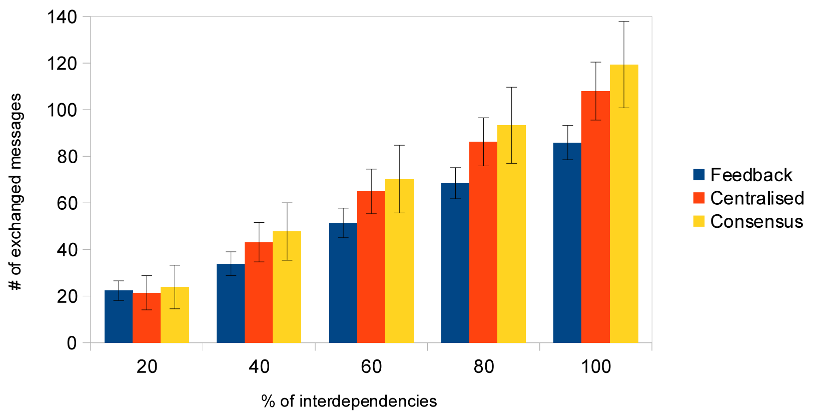

Figure 5: Number of exchanged messages

As expected, all three coordination approaches require more messages to be exchanged as the complexity of the service's topology increases. Nevertheless, it is clearly noticeable that the number of exchanged messages until all a global adaptation solution is determined is strongly affected by the chosen coordination model. As the reported results show, the proposed decentralised feedback-based coordination model outperforms the other evaluated coordination approaches, minimising the number of exchanged messages among

[1]The random values were generated by the Mersenne Twister algorithm [55].

nodes, particularly with higher levels of complexity of the evaluated topologies.

In several application scenarios characterised by a high degree of openness and dynamism, coordination tasks need to be time dependent since handling time is necessary to specify (and enforce) given levels of quality of service [6]. A second study measured the needed average time from the moment a node announced a change in its output QoS level for a particular component until the outcome of the global adaptation process was determined. The obtained results are plotted in Figure 6.

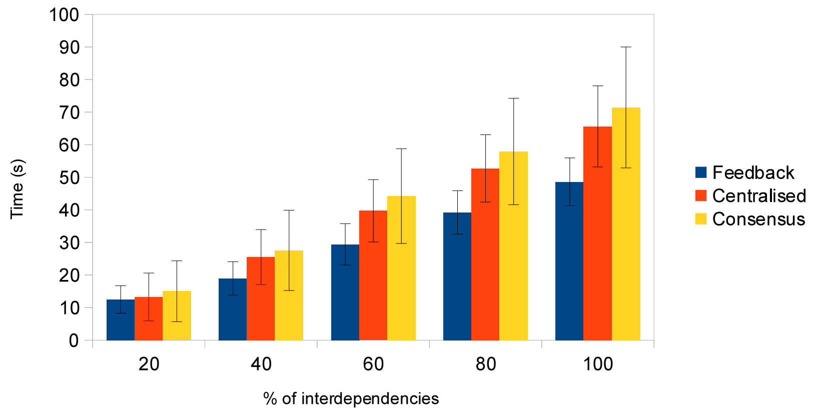

Figure 6: Needed time for a global coordinated adaptation

Clearly, the choice of a coordination model also has an impact on the needed time to determine a globally acceptable solution in all the evaluated topologies, with varying degrees of complexity. Not only the proposed decentralised feedback-based coordination model is faster than the other two approaches but also has tighter bound, a consequence of the model's guarantees on the exact behaviour of the coalition taken as a whole due to the deterministic way in which nodes can behave on a coordination request.

A decentralised coordination model should not only reduce message passing costs and guarantee a timely convergence to ensure that real-time constraints are met, but also guarantee an eventual convergence to a near-optimal service solution. It is known that guaranteeing globally optimal adaptation decisions requires a complete knowledge about the state of all components, typically relying on centralised approaches to globally adapt the system in a controlled manner. On the other hand, optimal decentralised control is known to be computationally intractable [37]. Thus, it is important to measure the impact of the proposed feedback-based coordination model on the quality of the achieved solution.

In this sense, a third study measured how far from the optimal solution, determined with an optimal centralised approach with complete knowledge, was the global adaptation solution resulting from the proposed feedback-based decentralised coordination model. The utility of both solutions was measured as an weighted sum of the differences between the user's preferred values and the determined values after the local QoS optimisation on each node [6].

The results were plotted in Figure 7 and normalised to the utility of the solution's determined by the optimal centralised approach. The study was divided in two categories: (i) when the average amount of available resources per node is greater than the average amount of resources demanded by the services being executed; and (ii) when the average amount of resources per node is smaller than the average amount of demanded resources. Thus, in the first scenario, QoS upgrade coordinations are more likely to occur when a service leaves the system, while in the second, nodes have a higher probability of needing to downgrade the quality of previously accepted services in order to accept new ones.

It is clearly noticeable, in both scenarios and for varying degrees of topology complexity, that a near-optimal service solution's quality can be achieved when using the proposed feedback-based coordination model, despite its simpler approach and absence of global system knowledge. We are confident that the proposed feedback-based coordination model can be successfully tested and used in a variety of domains, from large-scale grid computing systems to small-scale wireless and sensor networks, where power usage and radio transmission usage should be minimised.

8. Conclusions

The engineering of self-organising decentralised systems is a major challenge, particularly if predictability and timeliness are desired. Autonomous distributed self-adaptive systems are highly context-dependent. Whenever a component's context changes it has to decide whether or not it needs to

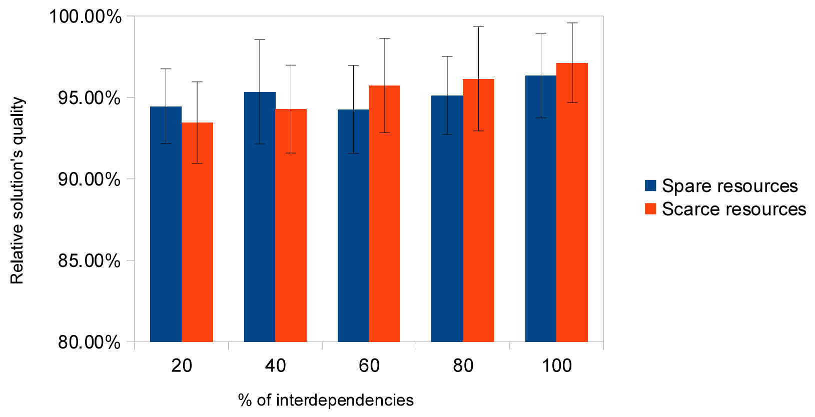

Figure 7: Relative solution's utility

adapt. Whenever it decides to adapt, this may prompt the need for coordination activities that ensure a globally acceptable solution for the entire distributed service.

This paper addressed this challenge and proposed a decentralised coordination model based on an effective feedback mechanism to reduce the time and complexity of the needed interactions among nodes until a collective adaptation behaviour is determined. Feedback is formulated at each affected coalition member as a result of a local anytime QoS adaptation process that evaluates the feasibility of the new requested service solution. This way, positive feedback is used to reinforce the selection of the new desired global service solution, while negative feedback will act as a regulator on changes that deviate the system from some near-optimal state.

The proposed approach aims to be general enough to be used in a wide range of scenarios characterised by a high degree of openness and dynamism where coordination tasks need to be time dependent. As the reported results demonstrate, the proposed feedback-based coordination model requires less messages to be exchanged and it is faster to achieve a globally acceptable near-optimal solution.

We plan to extend this work in several ways. One direction is to extend the coalition formation algorithm of the CooperatES framework so that a coalition's formation can be post-adjusted while the service is being executed in response to several internal and external events. The decentralised coordination of such adjustment process poses an interesting research question that we plan to investigate. We believe that further research can yield valuable additional insights about when and what kinds of communication are most critical and which communication policies are most effective in various domains.

Another direction is to support the notions of computational trust and reputation models. In this paper, there is an assumption that every node in the system is trusted. However, in open distributed systems this will not always be the case. Future developments of the proposed decentralised coordination model involve the development of local trust models of nodes that have been encountered in the past. The proposed model would then take into account the trust models when determining the collective adaptation behaviour and mechanisms could be developed to adjust the advertised feedback to take a neighbouring node's trust profile into account.

Acknowledgments

This work was supported by the CooperatES project, ref. PTDC/EIA/71624/2006, funded by national funds (3599 - PPCDT) through the FCT-MCTES (Portuguese Foundation for Science and Technology)

References

[1] J. A. Stankovic, Real-time and embedded systems, in: The Computer Science and Engineering Handbook, CRC Press, 1997, pp. 1709–1724.

[2] V. Subramonian, G. Deng, C. Gill, J. Balasubramanian, L.-J. Shen, W. Otte, D. C. Schmidt, A. Gokhale, N. Wang, The design and performance of component middleware for qos-enabled deployment and configuration of dre systems, Journal of Systems and Software 80 (2007) 668–677.

[3] T. F. Abdelzaher, E. M. Atkins, K. G. Shin, Qos negotiation in real-time systems and its application to automated flight control, IEEE Transactions on Computers, Best of RTAS '97 Special Issue 49 (11) (2000) 1170–1183.

[4] P. Li, B. Ravindran, Proactive qos negotiation in asynchronous real-time distributed systems, Journal of Systems and Software 73 (2004) 75–88.

[5] X. Wang, D. Jia, C. Lu, X. Koutsoukos, DEUCON: Decentralized end-to-end utilization control for distributed real-time systems, IEEE Transactions on Parallel and Distributed Systems 18 (7) (2007) 996 –1009.

[6] L. Nogueira, L. M. Pinho, Time-bounded distributed qos-aware service configuration in heterogeneous cooperative environments, Journal of Parallel and Distributed Computing 69 (6) (2009) 491–507.

[7] L. Nogueira, L. Pinho, J. Coelho, Flexible and dynamic replication control for interdependent distributed real-time embedded systems, in: Distributed, Parallel and Biologically Inspired Systems, Vol. 329 of IFIP Advances in Information and Communication Technology, Springer Boston, 2010, pp. 66–77.

[8] F. Kon, R. H. Campbell, Dependence management in component-based distributed systems, IEEE Concurrency 8 (1) (2000) 26–36.

[9] M. Shankar, M. de Miguel, J. W. S. Liu, An end-to-end qos management architecture, in: Proceedings of the 5th IEEE Real-Time Technology and Applications Symposium, IEEE Computer Society, Washington, DC, USA, 1999, pp. 176–191.

[10] R. Rajkumar, C. Lee, J. Lehoczky, D. Siewiorek, A resource allocation model for qos management, in: Proceedings of the 18th IEEE Real-Time Systems Symposium, IEEE Computer Society, 1997, p. 298.

[11] A. Friday, N. Davies, K. Cheverst, Utilising the event calculus for policy driven adaptation on mobile systems, in: Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks, IEEE Computer Society, Washington, DC, USA, 2002, p. 13.

[12] D. Gelernter, N. Carriero, Coordination languages and their significance, Communications of the ACM 35 (2) (1992) 96–107.

[13] G. A. Papadopoulos, F. Arbab, Coordination models and languages, Tech. rep., Centre for Mathematics and Computer Science, Amsterdam, The Netherlands (1998).

[14] T. Kielmann, Designing a coordination model for open systems, in: Proceedings of the 1st International Conference on Coordination Languages and Models, 1996, pp. 267–284.

[15] A. Montresor, H. Meling, Ö. Babaoglu, Toward self-organizing, self-repairing and resilient distributed systems, in: Future Directions in Distributed Computing, 2003, pp. 119–126.

[16] R. Khare, R. N. Taylor, Extending the representational state transfer (REST) architectural style for decentralized systems, in: Proceedings of the 26th International Conference on Software Engineering, 2004, pp. 428–437.

[17] T. De Wolf, G. Samaey, T. Holvoet, D. Roose, Decentralised autonomic computing: Analysing self-organising emergent behaviour using advanced numerical methods, in: Proceedings of the 2nd International Conference on Autonomic Computing, 2005, pp. 52 –63.

[18] J. Kramer, J. Magee, Self-managed systems: an architectural challenge, in: Proceedings of the International Conference on Software Engineering, Workshop on the Future of Software Engineering, 2007, pp. 259–268.

[19] P. Bridges, W.-K. Chen, M. Hiltunen, R. Schlichting, Supporting coordinated adaptation in networked systems, in: Proceedings of the 8th Workshop on Hot Topics in Operating Systems, Oberbayern, Germany, 2001, p. 162.

[20] C. D. Gill, J. M. Gossett, D. Corman, J. P. Loyall, R. E. Schantz, M. Atighetchi, D. C. Schmidt, Integrated adaptive qos management in middleware: A case study, Real-Time Systems 29 (2-3) (2005) 101–130.

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

[22] B. Li, K. Nahrstedt, A control-based middleware framework for quality-of-service adaptations, IEEE Journal on Selected Areas in Communications 17 (9) (1999) 1632–1650.

[23] P. G. Bridges, M. A. Hiltunen, R. D. Schlichting, Cholla: A framework for composing and coordinating adaptations in networked systems, IEEE Transactions on Computers 58 (11) (2009) 1456–1469.

[24] G. Allen, T. Dramlitsch, I. Foster, N. T. Karonis, M. Ripeanu, E. Seidel, B. Toonen, Supporting efficient execution in heterogeneous distributed computing environments with cactus and globus, in: Proceedings of the 2001 ACM/IEEE conference on Supercomputing, 2001, pp. 52–52.

[25] B. Ensink, V. Adve, Coordinating adaptations in distributed systems, in: Proceedings of the 24th International Conference on Distributed Computing Systems, Tokyo, Japan, 2004, pp. 446–455.

[26] T. W. Malone, K. Crowston, The interdisciplinary study of coordination, ACM Computing Surveys 26 (1) (1994) 87–119.

[27] M. Clarke, G. S. Blair, G. Coulson, N. Parlavantzas, An efficient component model for the construction of adaptive middleware, Lecture Notes in Computer Science 2218 (2001) 160–178.

[28] I. Georgiadis, J. Magee, J. Kramer, Self-organising software architectures for distributed systems, in: Proceedings of the 1st Workshop on Self-healing systems, 2002, pp. 33–38.

[29] K. Whisnant, Z. T. Kalbarczyk, R. K. Iyer, A system model for dynamically reconfigurable software, IBM Systems Journal 42 (1) (2003) 45–59.

[30] R. Allen, R. Douence, D. Garlan, Specifying and analyzing dynamic software architectures, in: Proceedings of the 1st International Conference on Fundamental Approaches to Software Engineering, Lisbon, Portugal, 1998, pp. 21–37.

[31] D. Garlan, B. Schmerl, Model-based adaptation for self-healing systems, in: Proceedings of the 1st Workshop on Self-healing Systems, 2002, pp. 27–32.

[32] J. S. Bradbury, J. R. Cordy, J. Dingel, M. Wermelinger, A survey of self-management in dynamic software architecture specifications, in: Proceedings of the 1st ACM SIGSOFT Workshop on Self-managed systems, ACM, New York, NY, USA, 2004, pp. 28–33.

[33] D. Goldin, D. Keil, Toward domain-independent formalization of indirect interaction, in: Proceedings of the 13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative En-

terprises, IEEE Computer Society, Washington, DC, USA, 2004, pp. 393–394.

[34] R. Van Renesse, K. P. Birman, W. Vogels, Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining, ACM Transactions on Computer Systems 21 (2) (2003) 164–206.

[35] M. Dorigo, G. D. Caro, The ant colony optimization meta-heuristic, New ideas in optimization (1999) 11–32.

[36] J. Holland, Hidden Order: How Adaptation Builds Complexity (Helix Books), Addison Wesley Publishing Company, 1998.

[37] T. De Wolf, T. Holvoet, Towards autonomic computing: agent-based modelling, dynamical systems analysis, and decentralised control, Proceedings of the IEEE International Conference on Industrial Informatics (2003) 470–479.

[38] M. Jelasity, A. Montresor, O. Babaoglu, A modular paradigm for building self-organizing peer-to-peer applications, in: In Engineering Self-Organising Systems, G. Di Marzo Serugendo, Springer, 2004, pp. 265–282.

[39] I. Dusparic, V. Cahill, Research issues in multiple policy optimization using collaborative reinforcement learning, in: Proceedings of the 2007 International Workshop on Software Engineering for Adaptive and Self-Managing Systems, IEEE Computer Society, Washington, DC, USA, 2007, p. 18.

[40] G. D. M. Serugendo, Autonomous Systems with Emergent Behaviour, Idea Group, Inc., Hershey-PA, USA, 2006, Ch. Handbook of Research on Nature Inspired Computing for Economy and Management, pp. 429–443.

[41] T. Abdelzaher, J. Stankovic, C. Lu, R. Zhang, Y. Lu, Feedback performance control in software services, Control Systems, IEEE 23 (3) (2003) 74 – 90.

[42] T. He, J. A. Stankovic, M. Marley, C. Lu, Y. Lu, T. Abdelzaher, S. Son, G. Tao, Feedback control-based dynamic resource management in distributed real-time systems, Journal of Systems and Software 80 (2007) 997–1004.

[43] H. Liu, M. Parashar, S. Hariri, A component-based programming model for autonomic applications, in: Proceedings of the First International Conference on Autonomic Computing, 2004, pp. 10–17.

[44] C. S. D. The University of Arizona, The cactus project, Available at `http://www.cs.arizona.edu/projects/cactus/`.

[45] W.-K. Chen, M. A. Hiltunen, R. D. Schlichting, Constructing adaptive software in distributed systems, in: Proceedings of the 21st International Conference on Distributed Computing Systems, 2001, pp. 635–644.

[46] L. Nogueira, L. M. Pinho, A capacity sharing and stealing strategy for open real-time systems, Journal of Systems Architure 56 (4-6) (2010) 163–179.

[47] C. Li, L. Li, Utility-based qos optimisation strategy for multi-criteria scheduling on the grid, Journal of Parallel and Distributed Computing 67 (2) (2007) 142–153.

[48] C. Wang, Z. Li, Parametric analysis for adaptive computation offloading, in: Proceedings of the ACM SIGPLAN 2004 Conference on Programming Language Design and Implementation, ACM Press, 2004, pp. 119–130.

[49] L. Nogueira, Time-bounded adaptive quality of service management for cooperative embedded real-time systems, Ph.D. thesis, University of Porto (2009).

[50] J. Park, M. Ryu, S. Hong, Deterministic and statistical admission control for qos-aware embedded systems, Journal of Embedded Computing 1 (2005) 57–71.

[51] Y. Brun, G. Marzo Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, M. Shaw, Engineering self-adaptive systems through feedback loops, Software Engineering for Self-Adaptive Systems (2009) 48–70.

[52] S. Camazine, J.-L. Deneubourg, N. R. Franks, J. Sneyd, G. Theraulaz, E. Bonabeau, Self-Organization in Biological Systems, Princeton Studies in Complexity, Princeton University Press, 2002.

[53] J. Huang, Y. Wang, F. Cao, On developing distributed middleware services for qos-and criticality-based resource negotiation and adaptation, Real-Time Systems Journal 16 (1999) 187–221.

[54] C. Maia, L. Nogueira, L. M. Pinho, Experiences on the implementation of a cooperative embedded system framework: short paper, in: Proceedings of the 8th International Workshop on Java Technologies for Real-Time and Embedded Systems, Prague, Czech Republic, 2010, pp. 70–72.

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