1. Introduction

In a multimedia application, the processing and communication requirements are distinct from the temporal constraints, and reliability, among others, of an application. In order to maintain synchronization characteristics, the several SOE subsystems, known as orchestration, must coordinate their actions. The recursive sharing of the QoS responsibilities among the involved subsystems, known as service level agreement (SLA), is a key factor in the provision of Quality of Service (QoS). As a result of this approach, we expect that designers can define clear and unambiguous (QoS) orchestration. As a result of this approach, we expect that designers can define clear and unambiguous

Two main phases can be identified during the QoS provision: negotiation and tuning. The QoS negotiation phase involves mechanisms responsible for the admission of new user flows in an SOE. A new flow contract offers, both sides can break the previously negotiated QoS. In case of contract violation (from any side), it should fire actions to reestablish the QoS level offered to users. In case of contract violation (from any side), it should fire actions to reestablish the QoS

The LindaQoS domain-specific language was designed as a high-level notation for the specification of resource orchestration. As a result of this approach, we propose the use of Wright architecture description language (ADL) in the QoS provisioning domain, as the basis for the formal verification of QoS system properties. To smooth this task, the QoS orchestration can be translated into architectural descriptions (Wright, at present) and into programming languages (Java, at present). Two compilers complement our work, enabling the hierarchical translation of LindaQoS specifications into architectural hierarchies of QoS negotiation and tuning subsystems. In order to illustrate the use of LindaQoS, the paper presents an approach focused on the architectural description of the QoS negotiation and tuning environment. The process of controlling and maintaining the acceptable quality is a QoS orchestration problem, which presents an architectural description for a hypermedia system. The process of controlling and maintaining the acceptable quality is a QoS orchestration problem, which presents an architectural description for a hypermedia system. The process of controlling and maintaining the acceptable quality is a QoS orchestration problem, which presents an architectural description for a hypermedia system. The process of controlling and maintaining the acceptable quality is a QoS orchestration problem, which presents an architectural description for a hypermedia system.
Abstract

The increasing number of multimedia applications has motivated the construction of platforms with end-to-end Quality of Service (QoS) support. This work proposes the use of Wright architecture description language (ADL) in the QoS provisioning domain, as the basis for the formal verification of QoS system properties. To smooth this task, the LindaQoS domain-specific language was designed as a high-level notation for the specification of resource (QoS) orchestration. As a result of this approach, we expect that designers can define clear and unambiguous instantiated platforms, with support to multimedia application implementations, in a reduced development time.

1. Introduction

In the last years there has been an increase demand for platforms with multimedia application support. Since processing and communication requirements are distinct for each media type, different QoS guarantees are necessaries to maintain synchronization characteristics, temporal constraints, and reliability, among others, of an application. In a Service Offering Environment – SOE (processing and communication environment), such QoS guarantees must be end-to-end provided, which implies the recursive sharing of the QoS responsibilities among the several SOE subsystems, process known as QoS orchestration (or resource orchestration) [5].

Two main phases can be identified during the QoS provisioning: negotiation and tuning. The QoS negotiation phase involves mechanisms responsible for the admission control of new user flows in an SOE. A new flow admission characterizes the establishment of a service contract (or service agreement) between the user and the QoS provisioning environment. During the service offering, both sides can break the previously negotiated contract. The user may not respect anymore its initial load and the environment can be no more able to maintain the service level agreement (since resources are dynamically shared). The QoS tuning phase provides mechanisms responsible for monitoring the flow load and the QoS really offered to users. In case of contract violation (from any side), it should fire actions to reestablish the QoS negotiated level. These actions may even require a re-orchestration of the involved subsystem resources.

The SOE negotiation and tuning phases define a complex software-architecture. However, it can be specified by means of a basic architecture that can be adapted and recurrently applied in the several SOE abstraction levels. The basic architecture and its recurrence and adaptation mechanisms require a clear and unambiguous specification, as discussed in some proposals in the literature. The use of UML (Unified Modeling Language) in the description of the mentioned basic architecture was firstly proposed in [5]. However, this approach presents some limitations, mainly because UML does not allow the precise specification of the architecture semantics, since it is a semi-formal notation.

This paper presents an approach focused on the architectural description of the QoS negotiation and tuning mechanisms using Wright ADL [1]. Wright has a formal notation (CSP) associated to it. This enables Wright to offer tools for formal analysis and verification, which can be used to infer system properties (e.g., absence of deadlock among different connectors). In order to simplify the architecture comprehension and use, a domain-specific language (DSL), called LindaQoS, is proposed. Its main purpose is to provide a notation closer to the problem domain, being particularly oriented for defining hierarchies of QoS negotiation and tuning subsystems. Two compilers complement our work, enabling the translation of LindaQoS specifications into architectural descriptions (Wright, at present) and into programming languages (Java, at present).

In order to illustrate the use of LindaQoS, the paper presents an architectural description for a hypermedia system. The process of controlling and maintaining the presentation of a hyperdocument with an output of acceptable quality is a QoS orchestration problem, which needs to be treated by the hypermedia formatter in two levels. First, there is the inter media-object synchronization issue, related with the spatial and temporal relationships specified by the author, the duration assumed by each media object presentation, and the instants when each presentation action (play, stop, pause, etc.) takes place. The second level is the intra-object synchronization for each media object, mainly
associated with the moment that contents are obtained from storage locations, the transfer rate, the presentation rate and the scheduling policies in the involved operating systems. In fact, the intra media-object orchestration is a fundamental support in order to correctly maintain the document inter-object synchronization. Both synchronization issues are modeled as QoS negotiation and tuning phases using LindaQoS.

The paper is organized as follows. Section 2 introduces the resource orchestration concept present in QoS negotiation and tuning phases. The concept, already well discussed in the literature, is just briefly explained, since it is necessary for understanding the architectural description in the Wright ADL, presented in Section 3. Section 4 describes LindaQoS and illustrates its usage when defining hypermedia system architectures. Section 5 discusses related work and presents our final remarks.

2. Resource orchestration

The division of QoS provisioning responsibility among the different SOE subsystems, what is called resource orchestration in Section 1, is the core of the QoS negotiation and tuning phases.

The orchestration mechanisms manage resources based on information queried from a data structure called virtual resource tree. A virtual resource tree abstractly denotes the hierarchical usage division for one or more resources. The leaf nodes are called final virtual resources and the other nodes are the schedulers, which divide their own usage portion of the resource among their children nodes.

When the root resource is a single resource or a set of resources viewed as a single one (e.g. communication links, CPU, etc.), the admission of new user flows is named primitive, because there is no need for negotiation among other mechanisms. Otherwise, the flow admission is recursively delegated to virtual resource trees of the lower abstraction levels. The process stops when a virtual resource with primitive admission is achieved. Some examples of non-primitive admission are the operating system (involving CPU, network subsystem, memory management, etc) [7], the network (admission done on each router, switch, etc), a hypermedia system (involving the formatter, network and servers) [8], etc.

Three component types compose the basic QoS negotiation architecture: Admission Controllers, QoS Negotiators and QoS Mappers. A request for a service contract is done through a call to an admission controller with parameters that describe the user flow, through the load characterization and the QoS specification. If this admission controller is related to a composite resource, it starts the negotiation mechanisms, which can be centralized or distributed, through a call to its associated QoS negotiator. This QoS negotiator identifies the internal resources that can be involved in the service provisioning and divides the portions of QoS responsibility among them. For this purpose, an associated QoS mapper translates the service request parameters into low-level parameters. After this translation, a new service contract request is performed on each internal resource, through their corresponding admission controllers. The admission process recurrently repeats in each resource, until primitive admission controllers are reached, when the test of admittance is directly executed over the resource (primitive admission controller has no negotiation agent).

Four elements compose the basic QoS tuning architecture: Adjustment Controllers, QoS Tuners, QoS Mappers and Monitors. During the service operation, the QoS tuning architecture models the mechanisms responsible for maintaining the negotiated QoS level (through resource reallocation). It also models the monitoring mechanisms that detect contract violations.

After the service admittance, monitors measure the QoS being actually offered. QoS tuners or primitive adjustment controllers (when both are associated to monitors) periodically evaluate these measurements to detect QoS violations. If a primitive adjustment controller detects a QoS violation, it tries to reallocate resources directly over its associated virtual resource tree. If it is not possible, the adjustment controller alerts a QoS tuner from a higher abstraction level, who is responsible to maintain the desired service level. If a QoS tuner detects a violation or receives an alert, it can reallocate its internal resources or it can recursively alert another QoS tuner from a higher abstraction level or from the same level (distributed QoS tuning). The process stops when primitive adjustment controllers are able to reallocate their virtual resources or when a QoS tuner succeeds on reallocating its internal resources. If the system cannot maintain the negotiated service level, the user is notified through a highest abstraction level alert.

3. Architectural description of resource orchestration

ADLs are languages that can be used to represent software-system architectures. An architectural description is composed by basic elements (or main blocks), which are: components, connectors and architectural configurations [2]. Components are computation or data storage units, normally with an associated state. Depending on the abstraction level, a component can be either a simple procedure or a complete application. Ports define the component interface, useful for specifying interactions with other components. Connectors model the rules that coordinate these interactions. They export with their interfaces (called roles) the services they expect from the components that bind to them. Finally, architectural
configurations, or topologies, are connected graphs of components and connectors, which describe the architectural structure.

In order to increase reuse and expressiveness, some ADLs offer support for architectural styles. A style defines a family of systems in terms of an organizational structure pattern or, more specifically, it determines the vocabulary of components and connectors that can be used in instances of that style, together with a set of constraints on how they can be joined. Such constraints can be topological (e.g. without cycles) or relative to the execution semantics.

In this paper, styles are presented using an informal graphical notation. In our proposed notation, circles represent component types whose names appear into the circle. Boxes around the circles represent their ports. The port cardinality can be found near the port name, inside parentheses, in such a way that “(1)” means the existence of just one port and “(1..*)” expresses a variable number of ports (parameterized in the component type). The drawing of just one circle for a component type indicates the existence of a style restriction that permits the instantiation of just one component of that type. Ellipses suggest the instantiation of a variable number of components of that type. Finally, pipes symbolize connectors with their names and cardinalities written near each pipe. Port names also have a nomenclature. The \textit{intra} prefix indicates the binding between components of the same subsystem. The \textit{inter} prefix indicates the binding between components of different subsystems. The \textit{Input} and \textit{Output} suffixes in the interface type name are used to distinguish interfaces that start and that receive a request, respectively.

The specification of QoS negotiation mechanisms requires the definition of four styles that we developed: \textit{LowestNQoS}, \textit{CentralizedNQoS}, \textit{DistributedNQoS} and \textit{HierarchyNQoS}.

The \textit{LowestNQoS} style is presented in Figure 1(a). It describes subsystems that make the admission control and resource allocation directly over primitive virtual resource trees. In such subsystems, the \textit{AdmCtrl} (admission controller) component type is the unique present. This component type is responsible for the bindings between the basic QoS negotiation architecture and the SOE (main service), through its \textit{metainterface} port. Admission controllers can be attached to different subsystems through \textit{interlevel} ports.

The \textit{CentralizedNQoS} style is illustrated in Figure 1(b). It is provided for subsystems that centralize the negotiation mechanisms in one QoS negotiator. This component can be attached to one or more admission controllers, through \textit{Intralevel} connectors (linking the intralevel ports of each component type). For each underlying admission controller attached to the QoS negotiator (via interlevel ports), a QoS mapper is instantiated and attached to the central element, through an intermediary \textit{NegToMapper} connector (linking the translate port of both components).

The \textit{DistributedNQoS} style is presented in Figure 1(c). This style is provided for more complex subsystems, where negotiation mechanisms are delegated to more than one QoS negotiator. The \textit{IntraNeg} connector regulates the association rules among QoS negotiators (attached by their \textit{intraneg} ports). The other interface, component and connector types are similar to the ones described in the \textit{CentralizedNQoS} style.

Once subsystems are separately described using one of the three previous styles (\textit{LowestNQoS}, \textit{CentralizedNQoS} and \textit{DistributedNQoS}), a fourth style is necessary to describe the bindings among them. The \textit{HierarchyNQoS} style defines the \textit{Interlevel} connector, which regulates the interaction among QoS negotiators (\textit{InterlevelOutput}) and admission controllers (\textit{InterlevelInput}) on different subsystems. The \textit{HierarchyNQoS} style is described in Wright ADL in Figure 2. In [1], the Wright ADL is used to demonstrate how an ADL based on a formal and abstract system behavior model can provide practical means for describing and analyzing software architectures and styles. By offering tools oriented to configuration analysis, Wright became a natural choice for our work.

As can be noticed in Figure 2, the style only declares the connector type that binds different subsystems. This particular definition of the \textit{HierarchyNQoS} style is result of the lack of composite style support in ADLs, which would be essential for this nested subsystem organization.
This kind of recursive representation could be done using our graphical notation, but was omitted due to the lack of space. Indeed, this also obliged us to not show in this paper the Wright specification for the other three styles (LowestNQoS, CentralizedNQoS and DistributedNQoS).

4. LindaQoS

The specification of QoS tuning mechanisms has a structure similar to that of the QoS negotiation mechanisms. The QoS tuning is also described using four styles: LowestTQoS, used in subsystems that act directly over primitive virtual resource trees (Figure 3(a)); CentralizedTQoS, used to describe subsystems where one component is responsible for all the tuning process (Figure 3(b)); DistributedTQoS, describing more complex subsystems, where some components interact with each other to provide the tuning mechanism (Figure 3(c)); and HierarchyTQoS, which specifies the bindings among subsystems (similar to Figure 2).

A complete LindaQoS description of the QoS negotiation basic architecture is composed of some sections System and one section Hierarchy. Each section System describes the instantiation of components pertaining to a particular level of abstraction (subsection Instances) and the bindings between components of that level (subsection Attachments). The section Hierarchy, by its turn, describes all relationships between the different systems. There are three basic types of components in LindaQoS, used in subsection Instances: AdmCtrl (admission controller); QoSNeg (QoS negotiator) and QoSMap (QoS mapper).

To illustrate the use of LindaQoS, Figure 4 shows the graphical representation of inter media-object and intra media-object QoS negotiation subsystems for multimedia presentation systems. The flow (in this case a multimedia document) admission is done by the SOE subsystem, which is in charge of dividing the inter-object QoS (inter media-object synchronization) among the Document Server (Inter_Server), the communication provider (Inter_
Provider) and the exhibition platform (Inter_Client). The orchestration is centralized. The server and the provider subsystems are considered primitives, only to make the example simpler.

The Inter_Client subsystem, by its turn, in order to orchestrate the synchronized presentation of the document objects, acts as an intra-object QoS (intra media-object synchronization) user of the exhibition platform (CPU and Buffer) and the SOE environment (for loading each individual document media object in a calculated pre-fetching time).

The Intra_SOE, upon receiving a request from the Inter_Client subsystem must divide the intra-object QoS responsibility among the Intra_Server, Intra-Provider and Intra_Client subsystems. This orchestration is constituted by a distributed mechanism. Again, to simplify, the server and the provider subsystems are considered primitives.

The Intra_Client subsystem divides (centralized orchestration) its QoS responsibility between the CPU and Packet_Queue (OS network subsystem) primitive subsystems, finishing the whole orchestration process.

![Diagram](image)

**Figure 4.** Hypermedia system inter and intra media-object QoS negotiation: graphical representation.

The basic architecture of QoS tuning is described through a single MtHierarchy section and some MtSystem sections. Analogous to the QoS negotiation, each subsystem (MtSystem) has component instances of a particular abstraction level (Instances subsection) and the corresponding bindings for that level (Attachments subsection). The tuning component basic types are: AdjCtrl (adjustment controller) and QoS_Tun (QoS tuner and an implicit monitor associated).

Due to the similarities with the negotiation architecture, the illustration of an example of a QoS tuning architecture was omitted. More details on inter media-object and intra media-object QoS provisioning in hypermedia systems can be found in [8].

![Diagram](image)

**Figure 5.** Hypermedia system inter and intra media-object QoS negotiation: LindaQoS specification.

5. Related work and final remarks

Some architectural description languages, such as Xelha [4] and CBabel [6], offer QoS support, but they are oriented to specific environments and service categories. Their main function is to make easier the development of a service with QoS, through language built-in resource management mechanisms. These languages may be the natural choice for developing QoS services (main services), since they facilitate the resource management. However, they are not suitable for developing QoS basic architectures (meta services). Since this work is focused on describing the architecture of the negotiation and
tuning mechanisms, and not the services they provide, we chose to use the Wright ADL.

SCM (Service-Composition Model) [5] offers a base structure for analysis of service adaptation and service programming mechanisms in communication systems. The model provides suitable abstractions for representing and programming QoS aspects in communication services. Gomes et al. [5] applies the SCM concepts on modeling QoS provisioning mechanisms, through the use of UML frameworks. However, as already mentioned, UML is not precise for architecture family descriptions.

A list of DSLs applied to several domains can be found in [3]. DSSA (Domain-Specific Software Architecture) [9] is a methodology for domain engineering very close to our proposal, combining object-oriented frameworks, software architecture, generators and domain analysis for increasing reuse in the whole system development cycle.

We propose an architectural description of QoS provisioning mechanisms as a supporting tool for multimedia application developments. Through a formal notation, free of ambiguities, QoS negotiation and tuning subsystems can be defined in the Wright ADL. To simplify the development, the LindaQoS DSL allows designers to define resource orchestration hierarchies using a higher-level notation, specifically oriented to the QoS problem domain.

The use of a formal description for resource orchestration brings several benefits. First, it is possible to infer properties with precision. Second, it is possible to run formal analysis. Finally, the design can be based on principles, instead of intuitions. However, it is well known that it is difficult to translate formally proven properties into implementations. Moreover, there is an increase in complexity when trying to understand formal notations. LindaQoS aims at avoiding the difficulty on dealing with a formal notation, while its compiler offers the automatic code generation feature, providing a specialization of the frameworks described in [5] as a starting point for QoS system implementation.

One minor contribution of this paper is the LindaQoS example of usage. We offer an integrated view of the inter-media-object and intra-media-object QoS orchestration in hypermedia presentations. The structure suggested for the inclusion of QoS orchestration mechanisms inside hypermedia systems uses a recurrent approach, repeating in a higher abstraction level the same concepts presented in traditional infrastructures (network and operating systems). We expect with this middleware proposal to facilitate the process of negotiating and maintaining QoS within infra-structure providers and to offer a good support for hypermedia document and application developers.

6. Acknowledgments

This work was developed with support of FUNTTEL (Brazilian Fund for the Telecommunications Technological Development), through the contract 0594/02.

7. References