A CORBA-based Distributed Multimedia Database Management Layer: Design and Implementation Aspects

André Luís Vasconcelos Coelho and Ivan Luiz Marques Ricarte
School of Electrical and Computer Engineering (FEEC)
State University of Campinas (Unicamp)
P.O. Box 6101 — Campinas/SP 13083-970, Brazil
(coelho,ricarte)dca.fee.unicamp.br

Abstract

Multimedia applications executing on distributed environments lack a seamless approach to handle their components' persistent contents. In this paper, a persistence middleware layer that allows the integration of multimedia databases with media-dedicated storage servers is presented. The design of SGPOM follows the philosophy of RM-ODP and CORBA Persistent Object Service, thus enabling a clean integration with this distributed object platform. The main aspects related to the service implementation using the Java programming language are also described.

1. Introduction

Recently, many contributions have enabled the development of multimedia applications over distributed environments. Although key concepts, such as middleware [2] and platforms for distributed object computing [17], have been extended in many ways for supporting multimedia services [1,5,7,11], the endowment of persistence to multimedia objects (MOs) needs yet to be solved in this realm. For instance, CORBA provides the Persistent Object Service (POS) [10] as the generic infrastructure to target the platform components persistence, but leaves aside how to address many of the ordinary MO requirements.

There are two main approaches for MOs persistence: To extend object-oriented database management systems (OODBMS) to create multimedia database systems [8], or to design storage servers which are optimized to handle multimedia information [6]. In general, such servers take into account how to structure the multimedia contents into files in order to efficiently store and retrieve media streams. However, due to the multitude of media formats and standards [4], a general architectural solution for such specialized repositories seems somewhat difficult to be devised, and therefore, in a distributed multimedia platform, at least one OODBMS and many media servers should coexist.

In this context, there is a need for a mechanism to control the access to these distinct repositories. Such a gap is filled with the middleware service presented here. SGPOM is a management layer for persistent MOs initially conceived as a component of the Multiware Database [12]. Its design has been inspired by proposals for handling media streams in Open Distributed Processing [5] and by the CORBA POS, allowing for evolutionary trends of this platform [16].

The SGPOM proposal was motivated by the need to have a transparent way to integrate dissimilar repositories, possibly located at distinct servers and thus attending to disparate storage and retrieval requirements. Some SGPOM design goals included concurrent operation, robustness, and flexibility to easily add new media servers, either for load balancing purposes or for supporting new media types.

In this paper, Section 2 describes the general structure encompassing SGPOM, whereas Section 3 presents its main design aspects. Section 4 brings some implementation issues regarding an SGPOM prototype built with Java. Finally, Section 5 concludes the paper.

2. Structural Outline

The distribution of media-dedicated repositories brings about functionalities such as load balancing, data replication and sharing, trustability, availability, autonomy, and scalability. Following this tendency, the Multiware Database was conceived as a distributed object oriented database ensemble whose model is depicted in Fig. 1 [12].

In this architecture, SGPOM assumes the liability for the handling of the persistent states of multimedia documents components. It is a middleware layer mounted atop a group of heterogeneous media repositories dispersed across a distributed environment and is located via public interfaces, allowing applications that have access to an ORB implementation to register, store, recover, and optionally remove
their multimedia components’ active contents according to temporal restrictions. For such, its related architecture is ruled by a collection of managers, integrating a hierarchical and expandable framework (Fig. 2).

SGPOM commits to provide such a bearing instrument by extending and enriching the standardized CORBA POS in order to address and overcome its known weaknesses [15]. SGPOM, as POS, presents itself as a stratified service wherein each tier hides from its precedent the functionalities it subsumes. This effort towards the concurrent data access transparency support permits the design and implementation independence of a rank of layered modules mapped into a range of specialized components. These components are made accessible through the set of the public available IDL interfaces of their constituent subcomponents.

SGPOM extends the capabilities of the POS router component, the Persistent Object Manager (POM), providing a growable hierarchy of specialized managers. POS POM only keeps in charge of the first connection phase between a CORBA persistent object (PO) and the lowest level datastore proxies (PDSs), which is restrictive and inadequate for scalability and concurrency purposes. A new management configuration is needed, formed by cooperative entities serving to multiple client concurrent I/O requests.

SGPOM-PO (named after the POS counterpart) is the unique visible component from the perspective of an SGPOM client, acting only as a higher level interface for the whole structure and preserving the semantics of a typical persistent object for the SGPOM managers. These managers (SGPOM_OMs) logically compose a top-down hierarchy rooted by the LEADER, the head component that behaves as the only CORBA server to which the I/O operations sent by the SGPOM-PO are forwarded. As every management component, the LEADER maintains information concerning its predecessors, which, in this case, are media-type managers (TYPES). These, in turn, compose a level of the hierarchy representing the myriad of existent media types (text, audio, video...). A manager under this group has associated with it a gamut of SUBTYPES, each representing a particular coding format (subtype) for a given media.

SGPOM_DSs are data-driven entities responsible for interfacing with the media-specialized repositories (Datastores). A SUBTYPE can keep a symbolic link with one or more SGPOM_DSs, each supporting a particular kind of Protocol. Because it converts the high-level I/O operations to the Datastore API, the functionalities of an SGPOM_DS resemble those of a database adapter, which should be partially implemented by the repository designer. A special component of this class is GENERIC, a hybrid element viewed both as an adapter and a Datastore to which VO operations relating to raw data should be forwarded. GENERIC can also act as a servant devoted to the maintenance of the persistent states of the other components, and jointly with an SGPOM-PO and a LEADER instances integrate the service default configuration.

3. Design Aspects

When of the formal conception of SGPOM, the following design rules were assumed as basal:

Distributed object platform. The CORBA brokerage service was selected as the communication foundation for the dynamic ensemble of the distributed elements that compose the management structure in the access to the repositories, keeping the focus on the RM-ODP set of transparencies.

Simple media active contents. The MOs to be handled by the service are limited to single media active contents (referred here simply as media objects), which encapsulate structured or streaming data codified in an available standardized format [4]. Therefore, it does not concern to SGPOM knowing either the origin of those objects or how they temporally and spatially relate to each other for a given pre-
3.1. Support to Five Basic I/O Operations

Supported operations are divided into management operations, Register() and Delete(), which do not incur the media object direct manipulation; and access operations, Comm.way.Store(), Store() and Restore(), implying the deployment of an adequate communication/middleware service to keep in charge of the media contents conveyance. Analogous to POS, these operations are announced with similar signatures in all the hierarchy layers.

The semantics of connect()/disconnect() in POS is not well specified, leading to dubious interpretations [15]. Hence, another operation regarding the subscription phase of the media object in a given repository was envisaged. A successful flow of Register() through all the SG-POM ranked layers results in the creation of a descriptive data location identifier, named after POS PID as an SG-POM_PID. Besides the logical branch of the structure (from the LEADER to the SG-POM_DS) determining a certain Datastore, this key guards a template pointing to the physical place of the media content inside the repository realm. In this phase, the CA must indicate the type/subtype pair of strings associated with the media content, besides denoting which Protocol the repository should support. Such protocol defines an agreement between the client and the underlying Datastore concerning which functionalities the media object should implement in order to be suitably handled.

The POS operation Store() needed to be mapped into a pair of two-phase-based ones, Comm.way.Store() and Store(). The first implies the creation of a Comm.way CORBA object, an abstraction resource incorporating one of the several available protocols or communication mechanisms to be supported by the SG-POM_DSS. For sockets, a Comm.way specialization was devised, known as Socket.port.host, containing the port and host (URL) of a media data server. On the other hand, Store() should only be invoked after the CA has already integrally forwarded the media object across the communication substratum. A boolean value indicating the effective placement of the media content inside the repository acknowledges the CA that the whole storage phase was successfully completed.

The operation Restore() should be propagated at the media retrieval stage and, as for Comm.way.Store(), it makes use of a Comm.way container for datastream conduction. The recovery task presents more significant temporal restrictions, once it directly relates to presentation tasks. Conversely, in case of media data loss or corruption during transmission, it is up to the CA deciding whether to abort the consuming activity, which may result in a new cycle of Restore() invocations through the hierarchy components.

The semantics of Delete() resembles that of its POS equivalent: To extinguish any piece of information (contents plus metadata) related to a particular media object. After removal, any attempts to perform access operations via an SG-POM_PID will be regarded as invalid, bringing about an exception. Therefore, the CA should manage to keep itself informed about how its peers intend to use the media content in the near future before judging out for its current displacement. For security purposes, a signed SG-POM_PID is returned during the subscription phase: The CA must keep with itself the authentication password (signature) and may pass along the other part of the identifier.

3.2. Subcomponents

Each of the three kinds of entities in the SG-POM hierarchy—SG-POM_PO, SG-POM_OMs and SG-POM_DSs—actually abstracts a particular layer within which lies a set of specialized elements defined by well delineated IDL interfaces (see Fig. 3 for the TYPE level).

A Factory is a CORBA object in constant execution (persistent activation [17]) responsible for the (re)creation of the instances of the other members of its layer—except for the Child objects—inside the same process. It follows the CORBA Lifecycle Service concept [15]. In turn, the Controller keeps in charge of local monitoring tasks, checking on some performance values regarding how its unvocally
The CORBA POS does not address or impose any issue regarding the orchestration of its standardized components; this aspect should be properly tackled by implementors. Within the SGPOM context, the deployment of a detached CORBA object, located outside the hierarchical layers and entitled as the Coordinator, was the devised approach. Basically, this new entity keeps in charge of three types of administrative tasks, (i) the dynamic launching of Central and Controller subcomponents, (ii) the periodic retrieval and assessment of performance parameters associated to each layer, and (iii) the regular checkpointing of all Central subcomponents, forcing them to keep their own executing state inside the GENERIC repository.

One or more SGPOM instances can be launched and dynamically incremented in order to service different CA domains. In such respect, the Coordinator assumes the central role of consistently mounting the whole hierarchies in accordance with the organizing rules directed by a Manager.client applet. During its spawning, a TYPE manager is provided with the name of the media it will represent, receiving an index value identifying the SGPOM instance it belongs to and two CORBA ObjRefs, one pointing to the Central element of the above layer (LEADER) and another to GENERIC. With the first remote address, the TYPE can subscribe via a special operation exported at the LEADER interface notifying its emergence and media type. By this means, the LEADER can be aware of the growing list of available TYPES for further requests forwarding. Moreover, the number of TYPE managers associated to a given media can vary according to load balancing policies, taking part in distinct logical branches leading to different media servers. Keeping the right proportions, the same aforementioned rules apply to the other hierarchy components.

The Coordinator offers to the service administrator a fashionable manner to analyze the dynamic behaviour of the layered components of each existent SGPOM instance. Monitoring information in the form of a special structure (Monitor_values) is frequently retrieved by a team of specific threads. Periodically, each thread will be liable for retrieving from a given Controller the performance measurements it has already collected. Besides this, a range of asynchronous alerting operations supplies a fault tolerance mechanism for the detection of anomalous conditions, such as Central failures, overloading, and low responsiveness. The Coordinator, following administrative policies, make arrangements for proper actions, such as displaying warn-

---

It could be applied to all other components.
mechanism that could alleviate the CAS from managing the treatment of such anomalies, bringing the new startup of the whole process. For this purpose, SGPOM-PO exports in its Childs' IDL interfaces three additional operations, dele-
gated.Comm.way.Store(), delegated.Store(), and delegated.Restore(), by means of which a client can delegate that responsibility.

For so, the client is obliged to extend and implement a callback IDL interface (INT.CLIENT) through the medium of which it could be notified by the SGPOM-PO about the result of its own attempts to perform the access operation on course. For a media object storage, it is necessary, firstly, to convey its contents to the SGPOM-PO for internal buffering. At a second step, the SGPOM-PO will try to accomplish the normal cycle of storage operations, Comm.way.Store() and Store(), forwarding the media data to the adequate repository. For the media recovery, a similar buffer will be used for temporarily caching the retrieved copy originated at the repository. After being notified, the client receives back such datastream via a chosen Comm.way instance.

3.6. Exceptions and Communication Infrastructure

One of the POS 1.0 remarkable drawbacks is the lack of a minimal set of exceptions in the I/O operations signatures for the handling of extraordinary occurrences within the service lifetime. Due to the SGPOM enriched structure, such absence would have introduced some operational side-effects, such as functional inconsistencies regarding the layered subcomponents interaction, bringing about performance setbacks. For this reason, all SGPOM IDL interfaces—from the INT.CLIENT to the Coordinator to the subcomponents—present a particular exception type [17] encompassing envisaged exceptional circumstances. The aim of such a measure is to make it feasible to determine which (sub)component has fired a given exception and to delimit the source of error.

Coulson et al. [5] standed for the explicit separation of the media streaming control and communication infrastructures. This advise has motivated the conception of an extensible abstraction, the Comm.way, to encapsulate a broad range of existing transport protocols and services [3, 7, 9], thus overcoming the limitations currently imposed by conventional ORB implementations [14].

4. Implementation Aspects

To validate the proposed set of design criteria, an SGPOM prototype has been made available. Qualitative results obtained through benchmarking sessions over the developed prototype corroborated the adequacy of the whole approach. The purpose here is to briefly enumerate the most prominent resolutions addressed at this implementation stage: (i) Java was employed as the underlying programming environment; (ii) Sockets were regarded as the default media communication mechanism and Java Serialization as the default storage protocol applied to codify the contents of the

---

Figure 5. Checkpointing phase.
media objects; (iii) Employment of data structures whose lengths could be dynamically increased when needed; and (iv) Each element in the service, from the subcomponents to the Coordinator, was assigned to a publicly available IDL interface announced in a proper module.

Owing to its clean package-adapted design of classes and its intrinsic bearing to portability, modularity, concurrency, and interoperability [13,17], Java was the programming language of choice. Likewise, Java has been constantly extended to allow for different levels of persistence, bringing since file serialization to relational-database adaptors. Besides that, it has provided a fruitful and robust infrastructure to CORBA-based implementations, offering a gamut of facilities such as an extensive hierarchy of socket and thread-related classes and the support to expandable collections.

In SGPOM, all object servers are launched under the CORBA shared activation mode policy [17]. Therefore, as all subcomponents of a given SGPOM tier reside inside the same OS process and can be, at the same time, attending to different types of requests, there must be available a multiplexing mechanism based on threads. IONA Orbix family of products [14] address such requirements by means of a filter facility. In this scope, a filter is a software entity that carries out some operational transformations over incoming CORBA requests or outcome results, very much resembling the current CORBA interceptors facility. Such strategy was pervasively applied at all levels of the SGPOM prototype, enabling for scalability and fault tolerance issues.

Fig. 6 brings the IDL modules hierarchy describing the proposed middleware framework. Following the guidelines posed by POS, each SGPOM subcomponent was assigned to a given IDL interface, and closely interrelated interfaces were kept behind the same module umbrella.

5. Conclusion

This paper reports on the design and implementation criteria applied during the conception of a data storage middleware service. SGPOM was conceived as an underlying management layer to support the persistence requirements commonly imposed by distributed multimedia applications and tools. It follows the philosophy behind the Multiware Database and, therefore, is itself arranged as a layered based framework of specialized managers whose location is delimited by the boundaries of the distributed environment.

SGPOM does not address any new solution based only on the extensions of a particular language capability. Conversely, it is a detached system-environment provision devoted to cater for concurrent I/O requests via its public IDL API. Moreover, it constitutes an enhancement of the OMG CORBA Persistent Object Service stating a new hierarchical arrangement of its components. Such approach is novel and comprehends an extensible and richer alternative to other proposed distributed multimedia database middleware services [1,11].

References