Chapter 8

Benchmarking Issues of a Distributed Multimedia Database Management Service

André Luís Vasconcelos Coelho and Ivan Luiz Marques Ricarte

DCA – FEEC – UNICAMP, PO Box 6101

Campinas, SP, Brazil, Zip-code 13083-970

Key words: Multimedia middleware service; distributed persistence management layer; CORBA; benchmarking.

Abstract: One of the most compelling yet still not properly accomplished tasks inside the distributed multimedia database arena is that of suitably fulfilling the disparate temporal requirements commonly imposed by the different types of persistent multimedia information. In such regard, an object-based multimedia management layer has already been conceived and implemented as a CORBA service towards the seamless integration of multimedia databases with “net-hidden” media-dedicated storage repositories. The goal of this paper is to present some quantitative results obtained through measurement sessions over a developed prototype of this distributed service in order to qualitatively assess the adequacy of its architectural proposal.

1. INTRODUCTION

One of the core issues in a distributed environment is to seamlessly integrate in a unified management framework a range of different specialised services. This is the typical case when multimedia functionalities should take place in. Such integration assumes an even more significant role if one strives to follow on the philosophy behind the openness precept; in this realm, integration has turned to be called interoperation and is the main target to be tackled by middleware tools [1].

Particularly within the distributed multimedia database systems (DMMDBMS) arena, there is one key factor regarding the manipulation of multimedia information that does still lack newer proposals for more suited architectural middleware solutions: How to endow persistence to multimedia objects (MOs). For such support, the approach followed by the Multiware Database platform [2] was to maintain within the same distributed multimedia environment at least one OODBMS and many media-dedicated repository servers. In this scenario, the OODBMS keeps in charge of the high-level media objects inter-relationship specifications while the servers should be specifically configured to take care of the media objects' active contents and streams retaining structural metadata information about their particular codification format.

In order to co-ordinate and control the access to these possibly distinct repositories, a distributed component-based mechanism had to be devised. The result was the design and implementation of a multimedia object persistence management layer called SGPOM [3], whose functional and operational features were motivated by proposals for handling media streams in Open Distributed Processing [4] and by the interoperability and portability trends advocated by the OMG CORBA standardised specifications (in particular, the CORBA Persistent Object Service [5]).

Aiming at the evaluation of this management service under real-work conditions, a prospective archetype was implemented in Java. Despite a few isolated initiatives towards the performance characterisation of some multimedia middleware services [6] or towards the adoption and launching of some benchmarking rules [7, 8], there is still much work to do until the stable maturation of assessment criteria to be employed over this new course of distributed technology. Hence, the contribution of this paper is to aid efforts in such endeavour as it tries to fill part of this gap when of the SGPOM performance evaluation. The expected effect is to set forth some initial results, which could serve as a basis for comparison with other CORBA-based distributed multimedia database systems [9].

The document is structured as follows. Section 2 is devoted to the general architectural overview of the proposed distributed persistence management layer, focusing at the inter-relationship between its (sub)components. Section 3 presents quantitative results obtained from an SGPOM prototypical implementation. These results are displayed by means of graphics and tables centred around some performance measures and parameters, confirming the expected service behaviour. Finally, Section 4 addresses some final considerations and points out some possible extension works.

2. SGPOM ARCHITECTURAL DESCRIPTION

The large-scale distribution of media-dedicated repositories seems to be a promising strategy for achieving distributed persistence for multimedia components, as it allows for a gamut of new functionalities: load balancing, data replication and sharing for fault-tolerance, increased trustability and availability, and autonomy and scalability to all repositories. SGPOM assumes the liability for the co-operative integration of these repository servers by orchestrating their activities and by controlling the access to them. It is a middleware layer with public interfaces, allowing applications to register, store, recover, and remove¹ their multimedia components' active contents in accordance with their temporal constraints. Its architecture is ruled by managers which integrate an expandable hierarchy based on CORBA POS, bringing transparency functionalities to the Multiware Database clients.

Fig. 1 depicts the SGPOM hierarchical framework, emphasising the inter-relationships between its management elements. Following POS philosophy, SGPOM presents itself as a stratified service composed by a rank of specialised components that inter-operate across an ORB substratum towards the accomplishment of the management and access activities. These entities are accessible through the set of public IDL interfaces of their constituent subcomponents and are each mapped to a level in the structure.

SGPOM enhances the capabilities of the POS router element, the Persistent Object Manager (POM), by providing an extensible set of new specialised managers, each related to a particular media type/subtype. SGPOM_PO is the unique visible component from the perspective of an SGPOM client; it acts as a higher level interface preserving the semantics of a typical persistent object. The set of managers (SGPOM_OMs) is headed by the LEADER, which 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, the media-type managers (TYPEs). These, in turn, compose a level of the hierarchy representing existent media types, such as text, audio, and video. A manager of such class has associated with it a team of SUBTYPEs, each regarding a particular coding format (subtype) for a given media. The lowest layer is associated to the SGPOM_DSs, specialised "drives" responsible for interfacing with the media-dedicated repositories (Datastores). A SUBTYPE keeps with it a symbolic link to 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, an SGPOM_DS resembles a database adaptor. Finally, the GENERIC component is

¹ These will be referred as I/O operations.

an entity viewed both as an adaptor and a Datastore to which I/O operations relating to *raw data*² should be forwarded.

Figure 1. SGPOM framework of interoperable components

For SGPOM benchmarking purposes, the following concise list of design criteria [3] were considered. The basic I/O operations are divided into *management operations*, `Register()` and `Delete()`, which do not entail the direct handling of the MOs, and *access operations*, `Comm_way_Store()`, `Store()` and `Restore()`, which imply the deployment of an adequate communication middleware service to keep in charge of the media contents conveyance.

Each kind of entity in the SGPOM architecture—SGPOM_PO, SGPOM_OMs and SGPOM_DSs—actually abstracts one layer of the hierarchy encompassing a set of specialised elements. These elements are assigned to IDL interfaces defining four types of possible subcomponents, Factory, Controller, Central, and Child (Fig. 2). A Factory is a CORBA object in constant execution (persistent activation mode) responsible for the (re)creation of the instances of the other members of its layer—except for the Child objects—inside the same process. In turn, the Controller keeps in charge of local monitoring tasks. It constantly collects some performance values that quantitatively assess how its associated Central subcomponent has been servicing its addressed requests.

² Media contents that could not be either classified in pairs of types and subtypes or for which there does not exist a media repository available to handle their codification features.

The Central assumes the name of the layer it represents—SGPOM_XX, where XX is one of PO, OM or DS—, being conceived as its most important element. Each element in this class is only related to an SGPOM instance and may co-exist, in the same OS process, with peers representing other instances. Child objects were devised for scalability purposes: To support the forwarding of multiple concurrent I/O requests throughout the hierarchy. At each tier, there can exist an ever-growing pool of threads relating to Child subcomponents.

Figure 2. SGPOM subcomponents and their Coordinator

Within the SGPOM context, the deployment of a detached CORBA object, located outside the layered hierarchy and entitled as the Coordinator, was the chosen approach towards the (sub)components orchestration. Basically, this entity accomplishes the dynamic launching of Central and Controller instances, the periodic retrieval and assessment of performance parameters associated to each layer, and the regular checkpointing of all Central objects, forcing them to maintain their own executing state inside the GENERIC repository.

Aiming at a fair support to the concurrent access to the dispersed multimedia data, the POS POM functionalities was expanded, introducing new layers in the top-down structure. The result was the inclusion of groups of TYPEs and SUBTYPEs to represent, respectively, the MOs' media types and code formats, following the MIME standardised classification scheme. Besides, the idea of explicitly detaching the media streaming control and communication infrastructures [4] motivated the conception of an abstract container (named as Comm_way) to encapsulate a broad range of existing

transport protocols and services. The goal was to bring up an alternative way to surpass the performance constraints imposed by conventional ORBs [6].

3. PERFORMANCE EVALUATION

For the accomplishment of the benchmarking sequence, an application scenario (*multimedia playback*) was envisaged, regarding the real-time video-audio streams retrieval and exhibition. The applet of Fig.3 was employed for the performance measurement collections (I/O requests invocation times) and to emulate a playback application. With this interface, one can opt to test a given I/O operation or a continuous series of them all, comprehending the MO's typical lifecycle: registry, storage, recovery, and deletion. With a URL to a media archive, an MO can be created in consonance with a high-level database schema, encapsulating the media data-stream as its own class attribute (raw byte array).

Fig. 4 presents the physical environment where the performance measures were attained. To keep a more consistent and realistic analysis, a heterogeneous testbed composed by machines with differentiated resources was devised. The benchmarking data were obtained in many sessions and under uniform conditions.

Figure 3. The statistics interface for benchmarking and the SGPOM playback scenario.

Figure 4. SGPOM testbed.

3.1 Metrics

For the service evaluation, the following group of *measures* was defined, focusing on the temporal aspects of the I/O requests attendance. The *Invocation Time* for a simple request means the elapsed time interval between two actions: The first comprises the event generated when clicking at the statistics interface “GO!” button, and the latter refers to the same interface receiving the request answer—for both management operations and for the storage one, this answer translates to a simple “success or failure” acknowledgement, while for the recovery sequence it consists of the MO effective restore. In case of a complete cycle of requests (registry–storage–recovery–removal of the MO), the start-up time of one operation (exception made to the subscription one) will be delimited by the final time of the previous one.

For comparison purposes, the focus here will centre upon the arithmetic average of those intervals (*Average Invocation Time*³). The *Management Time* refers to the Invocation Time for either one of the management

³ In this section, T_x will represent such measure.

operations, i.e., registry or removal. With this measure, it can be inferred the time uniquely spent by the SGPOM layer components (that is, owing exclusively to the service arrangement), as there is no direct manipulation of the MO contents. The *Access Time* alludes to the Invocation Time for either one of the two types of MO access operations (storage or recovery). It provides a direct means for detecting the time spent both in the MO contents conveyance and repository handling phases. For storage purposes, this comprehends the total time involved in attending to the `Comm_way_Store()` and `Store()` operations. The *Efficiency* is devised as the percentage difference between the unit and the ratio between the sum of the mean management times (registry plus removal) and the sum of the mean access times (storage plus recovery) to a same MO.

Some parameters influenced on the data set obtained in the test sequence. *MO size* regards the number of bytes of a well-formatted media file sample. For a prospective analysis, five different sized archives (in MPEG, AVI and Quicktime formats) were chosen for comparison purposes, ranging from 1 to 25 MB. A higher value for the *number of Child objects* implies a higher number of possible I/O requests being simultaneously serviced by the same SGPOM component. At the same time, a larger internal data structure of the Central object is required to keep the relative Object References (ObjRef) to its Child related threads. Moreover, the time spent in the search for a free Child for the attendance of a new request would be increased. For the the measures presented here, a constant number for this parameter (20) was established with respect to all service component layers. The *number of clients* refers to the number of applications performing concurrent I/O requests over the same SGPOM instance. In this analysis, a unique statistic applet is used.

Unless for the SGPOM_DS, which directly handles the OM contents, the time spent in each tier can be deduced from the sequence of actions of Fig. 5: (i) A Child from the previous layer sends a `Create()` request to SGPOM_XX; (ii) SGPOM_XX looks up the table for the first available Child; (iii) SGPOM_XX returns the chosen Child's ObjRef; (iv) The Child of the previous layer sends an I/O request to the chosen peer; (v) This one restarts the cycle described above and then enters in a "blocked" state awaiting the forthcoming result; and (vi) the request result is propagated and the Child is relieved.

Figure 5. Action sequence chart concerning the time spent in each SGPOM layer.

3.2 Results

For the benchmarking accomplishment, IONA's OrbixWeb [10] was chosen as a supportive toolkit as it provides a robust IIOP-based ORB implementation with some additional facilities (such as *filters* and *loaders*), being compliant with the standardised Java-IDL mapping. Java Sockets were employed as the default media streaming communication infrastructure and Java Serialization as the default MO storage and codification protocol.

Fig. 6 (Layout #1) brings values for the “Invocation Time” measure regarding the four types of I/O operations—*Register()*, *Comm_way_Store() + Store()*, *Restore()* and *Delete()*—and the service *Basic Configuration*, set by an instance of *SGPOM_PO*, a LEADER and a GENERIC, each in a different host. This is done for the five sample archives (“MO Size”) through complete cycles of I/O requests. The average values were calculated over a series of ten execution cycles, and the MO transmission, for storage/recovery purposes, is done exclusively over the Fast-Ethernet subnet via socket channels between similar machines (UltraSparc 256MB). Table 1 shows “Efficiency” values calculated by applying the ratio:

$$Efficiency = 1 - \frac{Tm_{management}}{Tm_{access}}, (1)$$

where $Tm_{management}$ is the arithmetic average between the mean registry invocation time and the mean deletion invocation time, whereas Tm_{access} is the average between the mean storage invocation time and the mean restore invocation time.

Figure 6. “Average Invocation Time” for the four I/O request types (Basic Configuration).

Table 1. “Efficiency” measure for the data set of Layout #1.

Archive	<i>Tm_{management}</i> (ms)	<i>Tm_{access}</i> (ms)	Efficiency (%)
Sample1 (930 KB)	204.68	731.55	72.02
Sample2 (2140 KB)	201.43	1150.15	82.49
Sample3 (6.5 MB)	230.9	2687.7	91.41
Sample4 (15.5 MB)	247.67	6198.4	96.00
Sample5 (23.17 MB)	256.99	9587.25	97.32
Average	228.33	4071.01	94.39

The management operations, despite small differences, present a uniform (deterministic) temporal behaviour, centring around the 230 ms bounds. On the other hand, the access requests demonstrate, as expected, a continuous growth for the invocation times, in proportion to “MO Size”. Large values for this parameter implies large “Efficiency” results, as the proportional cost of time due exclusively to the SGPOM layers’ forwarding activities will be less significant to the total amount. Consequently, the “Efficiency” measure turns to be a good cost/benefits indicator for SGPOM deployment.

A detailed report of the “Average Invocation Time” was undertaken by separating the lots of time in agreement with the following independent activities: MO transmission (average time spent with sockets, *T_{socket}*), MO management and handling at the repository (*T_{Datastore}*) and MO memory encapsulation (*T_{memory}*). By this way, the temporal cost associated exclusively

to the SGPOM Basic Configuration activities ($\underline{T}_{SGPOM}$) could be deduced, both for the access and management operations. In this respect, Layout #2 (Fig. 7) depicts, in detail, the percentage quota relative to each of the activities for the “Average Invocation Time” measure, regarding the MOs management requests. The $\underline{T}_{SGPOM}$ can be inferred, in this case, by the following simple algebraic equation:

$$\underline{T}_{SGPOM} = \underline{T}_{Total} - \underline{T}_{Datastore}, \quad (2)$$

where $\underline{T}_{Total}$ are the values of “Average Invocation Time” in Layout #1.

Figure 7. Detail of the “Average Invocation Time” measure for the management operations, in SGPOM Basic Configuration, where (A) indicates the $\underline{T}_{Total}$ and (B) indicates the $\underline{T}_{Datastore}$.

Layout #3 (Fig. 8) follows the guideline stated in the previous paragraph. However, here, $\underline{T}_{SGPOM}$ can be inferred, for the MOs storage and recovery requests, from their respective associated expressions:

$$\underline{T}_{SGPOM} = \underline{T}_{Total} - \underline{T}_{Datastore} - \underline{T}_{socket} - \underline{T}_{memory}, \quad (3)$$

$$\underline{T}_{SGPOM} = \underline{T}_{Total} - \underline{T}_{Datastore} - \underline{T}_{socket}. \quad (4)$$

$\underline{T}_{memory}$ concerns with the execution time spent in reading (for the memory) the flow of media composing the MO, giving the proper URL to the media archive. For that reason, it is only examined within the storage activities.

Figure 8. Detail of the “Average Invocation Time” measure for the access operations, in SGPOM Basic Configuration, where (A) indicates the T_{Total} , (B) indicates the $T_{Datastore}$, (C) refers to T_{socket} , and (D) refers to T_{memory} .

At the time of the benchmarking sessions, the availability of a complete implementation of a media-dedicated repository (with design optimisations and assured temporal disk access restrictions) could not be fulfilled. Hence, the network file system was deployed as the default Datastore and the Java Object Serialization as the default data serialisation protocol. For this reason, $T_{Datastore}$ is largely related to the time spent in the search and access to the file that contains the MO raw contents within the appropriate directory (usually remote); such access is made possible through NFS support.

For registry activities, the T_{memory} temporal contribution stays soundly constant, as it is limited to the creation of a null content file and of an identification string (SGPOM_PID). For the removal task, the quote due to $T_{Datastore}$ grows as the handled MO volume increases, too, which agrees with what is expected: The time spent in the physical removal of the object bytes in the disk will be higher. $T_{Datastore}$ as much as T_{socket} are the most contributors to the storage and recovery temporal costs (MOs direct handling) whereas the time strictly spent with the SGPOM hierarchy stayed within the limits of a uniform range (columns 2 and 3 of Table 2). T_{memory} has an insignificant contribution for the storage task and therefore could be despised without damage for the analysis consistence.

In the storage and recovery processes, a great part of T_{SGPOM} is owed to the MO allocation and retention in intermediate buffers inside the SGPOM_DSs (in this case, inside GENERIC). For the developed prototype, no special technique for buffers implementation was used. This could grant an optimised access to the MO contents, entailing better service performance. Extrapolating to the handling of huge MOs, this limitation could become an

accentuated bottleneck and, therefore, should deserve a special attention. Another measure source (given, for instance, by T_{buffer}) could indicate the relative temporal quota associated to the buffering activities, which might be interesting for the media repositories designers.

Table 2. T_{SGPOM} (in ms) for the four I/O request types and five MO samples, in SGPOM Basic Configuration, according to (2), (3) and (4).

	Registry	Storage	Recovery	Deletion
Sample1.mov	177.73 (71.9%)	323.5 (33.4%)	148 (29.9%)	116.6 (71.9%)
Sample2.mpg	168.67 (69%)	302.29 (20.8%)	126.12 (14.9%)	111.42 (70.2%)
Sample3.avi	213.36 (75.4%)	317.43 (9.7%)	162.87 (7.7%)	116.54 (65.1%)
Sample4.avi	236 (79.3%)	271.6 (3.69%)	133.2 (2.64%)	110.9 (56.04%)
Sample5.avi	236.69 (79.6%)	387.1 (3.7%)	248.03 (2.8%)	119.32 (54.7%)

4. FINAL REMARKS

This paper reports on some performance validation issues regarding a datastorage multimedia middleware service prototype. Such validation comprises the qualitative analysis of a set of benchmarking results of some established performance metrics obtained during different session cycles.

The performance assessment accomplished was based on temporal measurement values obtained for a service prototype built over a single CORBA product. As the OMG has not yet specified neutral performance evaluation criteria for both existent and future middleware tools, it comes to be very constrained to infer how would be its behaviour over other ORB toolkits. Besides this, there is current a strong appeal towards the deployment of real-time brokering infrastructures [6]. In such context, another set of metrics should be additionally devised in order to embrace other non-flexible temporal requirements.

Plášil et al. [8] emphasise the hardness of investigating benchmarking measures regarding complex actions over a conventional ORB (such as sequences of requisitions), as it implies the need for the separate identification of the implicit influence of certain non-atomic factors. The solution proposed by the authors was to explicitly define elementary actions, such as single marshalling and dispatching operations, which would be of primal relevance to the particular study scenario, such as for throughput, scalability, latency and robustness affairs. For the SGPOM archetype, such a library of basic benchmarking procedures would be very helpful, as for the estimation of the temporal parcels due only to the I/O requests transmission through the framework layers.

Owing to the peculiarities of the proposed service and the lack in the literature of similar published results, the analysis made here does still need

to be further improved. In other multimedia branches of research, as the one referring to the human sensorial perception to synchronisation phase-outs between separate media flows, it seems to be more easy to settle down comparison initiatives as they could count on standardised values for certain basic metrics. By other means, it can be stipulated that the performance contemplation introduced here may serve as a first referential for other database benchmarking works, mainly for those regarding the management and access requirements associated to the distributed persistent media data allocated to distributed repository servers.

ACKNOWLEDGEMENT

CNPq has sponsored this research through MSc. and PhD. scholarships to the first author.

REFERENCES

- [1] P. Bernstein. Middleware: A Model for Distributed System Services. *Communications of the ACM*, 39(2): 86-98, February 1996.
- [2] I. L. M. Ricarte and C. M. Tobar. Towards an Architecture for Distributed Multimedia Databases. *Procs. of ICIIMS*, pp. 65-68, USA, June 1996.
- [3] A. L. V. Coelho and I. L. M. Ricarte. A CORBA-based Distributed Multimedia Database Management Layer: Design and Implementation Aspects. *To be published at Procs. of ISCC2000*, France, July 2000.
- [4] G. Coulson et al. Supporting the Real-time Requirements of Continuous Media in Open Distributed Processing. *Computer Networks and ISDN Systems*, 27:1231-1246, 1995.
- [5] OMG. Persistent Object Service Specification 1.0. TR 97-12-12, December 1997.
- [6] D. Schmidt, D. Levine and S. Mungie. The Design and Performance of Real-Time Object Request Brokers. *Computer Communications*, 21(4):294-324, April 1998.
- [7] P. Carando. Toward the Development of an Object Request Broker Evaluation Instrument. In *OOPSLA-96 Workshop: Objects in Large, Distributed and Persistent Systems*, 1996.
- [8] F. Plášil, P. Tüma and A. Buble. CORBA Benchmarking, TR OMG 98-10-04, Charles University, 1998.
- [9] T. Plagemann et al. Adaptive QoS Aware Binding of Persistent Multimedia Objects. In *Procs. of DOA*, Scotland, September 1999.
- [10] IONA Tech. *OrbixWeb Programmer's Guide*, 3.0 edition, November 1997.