SOME APPROACHES TO OBJECT CACHING IN CORBA

Uttam Kumar Roy, Samiran Chattopadhyay
Department of Information Technology,
Jadavpur University Salt Lake Campus
Block-LB, Plot-8, Sector – III,
Kolkata-700 098, India

ABSTRACT

Traditional distributed CORBA applications incur high latency due to excessive marshalling-demarshalling, and overuse of network bandwidth. It is necessary to reduce response time to support applications that require high degree of performance. Caching is an important technique to enhance performance, letting clients to access distributed objects locally instead of accessing them remotely.

In this paper, we have provided a number of approaches to object caching architecture using event service, object wrapper and interceptor showing relative advantages and disadvantages. Implementation to these approaches shows a quantitative comparison among them. A salient feature is that different types of remote objects can be cached with a little modification of client and server code. These approaches are based on cache-invalidate cache consistency policy, per process caching, variable grained data shipping and replication management. We have demonstrated that the proposed approaches of caching result in a significantly improved performance for applications that exhibit a reasonable amount of read operations.

Keywords: *CORBA, Distributed System, Caching, Consistency, Object Graph, Event Service, Wrapper, Interceptor.*

1. INTRODUCTION

Common Object Request Broker Architecture (CORBA) provides several advantages (e.g. language, OS, network, location transparencies), over existing protocols (e.g. RMI, socket) in the domain of distributed object computing ([7]). CORBA provides these transparencies by performing remote invocation for every request causing excessive marshalling-demarshalling, and overuse of network bandwidth resulting high latency and low scalability. It has become necessary to improve quality of service, CORBA provides, for applications that require high degree of performance. Caching objects lets clients to invoke methods on distributed objects locally instead of invoking them from remote servers. By storing (i.e. caching) objects locally at the point of usage, communication bandwidth, excessive messaging and unnecessary marshalling-demarshalling can be reduced significantly and hence response time may be shortened.

In this paper we have attempted to improve the performance of operations on objects implemented remotely by providing a variety of architectures for caching variable sized objects with little modifications of the existing code, based on event service, object wrapper and interceptor and made a relative comparison among them. Experimental results have also been given that shows significant improve in response time for a reasonable amount of read operation.

These architectures for caching have several features. First, architectures (except event service based) are designed to be used to implement caching with consistency for many types of objects. This is desirable because new types of objects can be introduced in the system at any time and we want to make the benefits of caching easily available to implementations of theses objects. This is different from systems in [9,17,11,18]. Second caching and consistency are completely transparent to client and server applications respectively. Third, server side caching has been proposed in addition to the client side caching, expecting further reduced response time whenever the server uses a remote database. Fourth, to accommodate large number of clients in a single node, cache replacement policy (LRU) has been used expecting high percentage of hit ratio (HR) for a moderate cache capacity (CC). Our architectures (using wrapper and interceptor) can cache directed object graph instead of caching standalone object using per-process caching mechanism. A hit ratio of 94.16% has been obtained for 6% cache size using traditional LRU replacement algorithm.

A Roadmap: This paper is organized as follows. The problems involved in building a caching system, possible solutions and the one we adopted are discussed in Section 2. We present an overview of our Architecture in Section 3. Section 4 contains the implementation details. We give experimental performance results in Section 5. The future research direction has been given in Section 6. We conclude the paper in Section 7.

2. ISSUES TO BUILD A DISTRIBUTED CACHING SYSTEM

Caching: Caching (Figure 2.1) lets clients to invoke methods on distributed objects locally instead of invoking them from remote servers and hence shortens response time ([5, 1, 2, 17, 16]). A cache is essentially a container of the object, which resides with users, or physically closer to them than the original objects, e.g. within a separate server in the case of the Web ([1,10]). Clients use the cache rather than the original object(s) as illustrated in Figure 2.1.

Consistency: Systems such as banking need strong consistency where slightest inconsistency is not tolerable. There are some applications where the system is allowed to inconsistent temporarily but the system should not be inconsistent for long time. Example includes railway reservation system. Other systems such as University information systems, where students read the information from the university database but they are not allowed to fire any write operations, are read-only systems that do not need to implement consistency-maintaining mechanism.

Many cache protocols employ a *"write-through"* policy, whereby updates, which are applied to the cache, are sent to the original object first. The server, in turn, *updates/invalidates* the other replicas that reside in different clients cache. Sometimes (e.g. web applications) updates can only be made to the original object (document), and caches are not informed of updates; each document has a *"time-to-live"* associated with it and a cache will only fetch a new copy if it believes its current copy is out-of-date.

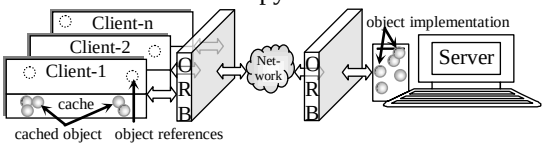
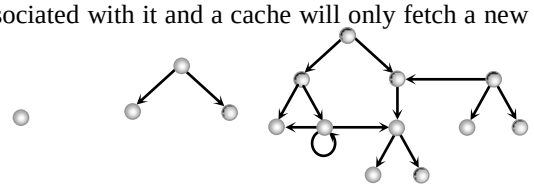


Figure-2.1: Object Caching Model



i) Stand alone ii) Simple iii) Complex

Figure-2.2: Different types of object graph

Consistency Initiation: Consistency policy determines how to keep all the replicas up to date. Responsibility can be imposed on the server or the client side. Both have their own set of advantages and disadvantages. If implementer takes the responsibility, more load on the server and hence longer the response time. On the other hand, designing a strict consistent system is more difficult if we impose consistency mechanism on the client.

Cache Replacement: Whenever the cache is full, some object needs to be replaced. Several cache replacement algorithms (e.g. FIFO, LRU, LFU etc.) have been implemented in the systems from processors to Web. These policies exploit data localities (temporal or spatial). We can take into account many other parameters—such as frequency of use, download time and document size—for placement and replacement ([2]) policies resulting more complex policies such as LRU-MIN, LRU-HOLD, LRU+ ([19]). In some distributed applications (e.g. banking, railway reservation, airline reservation, hospital system, and University information system etc.), user requests come following a different locality of reference pattern that needs more sophisticated cache replacement policies.

Sharing Mode: According to the number of replicas general classifications are: (i) Single Reader/Single Writer (SRSW) (ii) Multiple Reader/Single Writer (MRSW) (iii) Multiple Reader/Multiple Writer (MRMW). Coherence maintenance mechanisms for multiple writers are more complex and expensive.

Granularity: The basic data-item size used to cache is generally referred to as "*granularity".* Granularity level can be one byte, a word, a page or a data structure. In our implementation, we use the coarse grain size for the following reason. Transporting only the state of the object is not enough. Object may have references to other objects, which, in turn, may have references to still more objects resulting a directed graph as shown in Figure 2.2.

The object "*serialization"* and "*deserialization"* facilities should be designed to work correctly in these scenarios. If we want to serialize the root object, all other referenced objects are recursively located and serialized. Similarly, during the process of "*deserialization"*, all of these objects and their references are correctly stored.

3. PROPOSED ARCHITECTURE

The following section describes various components of the proposed architectures and how they address all issues discussed in the previous section.

Object Wrapper: Object wrapper feature allows us to define methods that are called when a client application invokes a method on a bound object or when a server application receives a request.

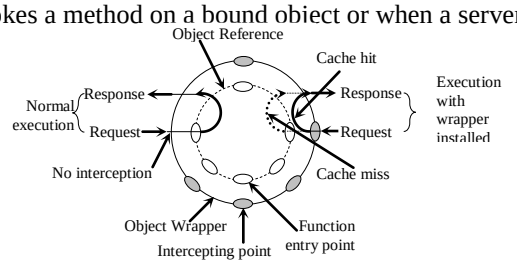


Figure3.1: Operation of an Object wrapper

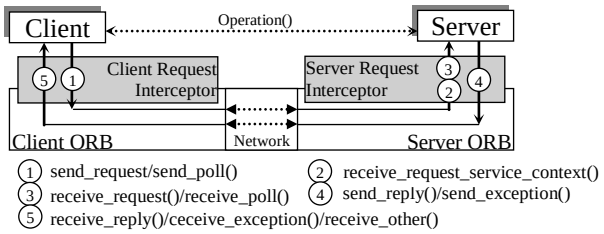


Figure3.2: Operation of an Interceptor

Object wrappers ([7]) are invoked before an operation request has been marshaled. Using wrappers, we can have finer control on the method invocations in the fact that only necessary requests (figure 3.1) can be intercepted that eliminates unnecessary function call. Any number of object wrappers can be installed in any side (client/server) depending upon the application requirements.

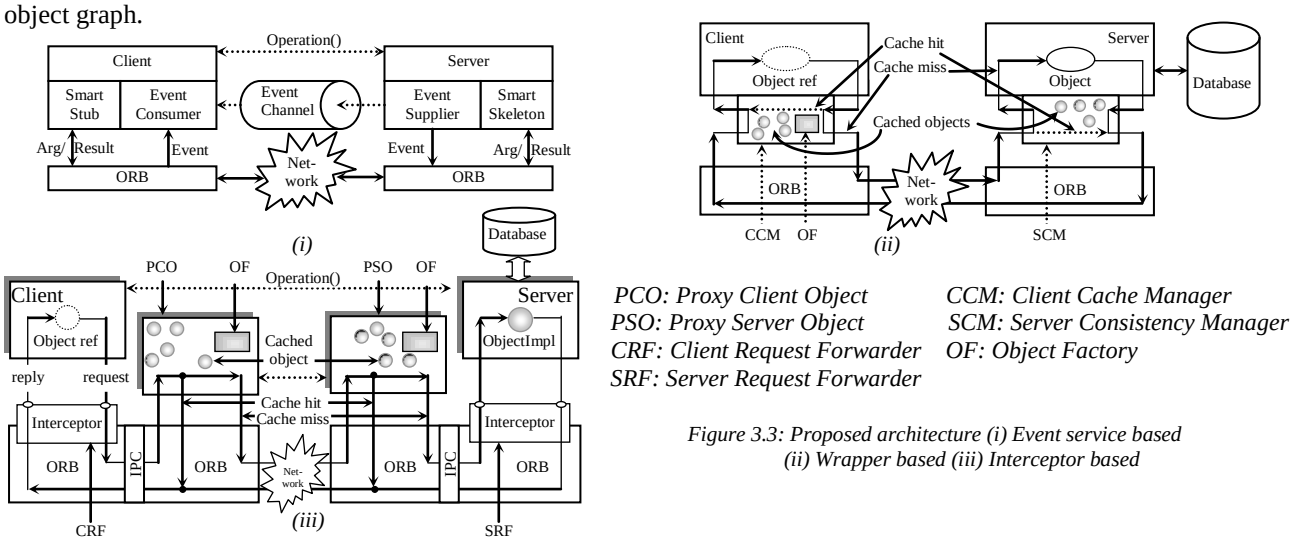
Interceptors: Interceptors are a set of interfaces, which provide a framework for plugging in additional ORB behavior such as caching, security, transactions, or logging. Using interceptors, users can notify the communications between clients and servers, and modify these communications if they wish, effectively altering the behavior of the ORB. PIs can perform operations at different points as shown in Figure 3.2 during request processing.

Event Service Based Architecture: This architecture comprises of following components [Figure 3.3 (i)]:

- *Client Cache Manager* - Responsible to store objects.
- *Event Suppliers and Smart Skeleton* - Responsible for notifying clients the change of state of object(s).
- *Event Consumers and Smart Stub* - Responsible for receiving the updated object state and keeping the object state synchronized using CORBA's standard event service.
- *Event Channel* - This is the CORBA component that is responsible for dispatching events from *Event Suppliers* to all the subscribing *Event Consumers* (smart stub in our case).

Wrapper Based Architecture: This architecture comprises of following components [Figure 3.3 (ii)]:

- *Client Cache/Consistency Manager (CCM):* Being an object wrapper, this module is responsible to managing the cache in the client side. For weaker consistent system CCM also takes responsibility to maintain consistency.
- *Server Cache/Consistency Manager (SCM):* This component is also an object wrapper. SCM module is responsible to make the system strongly consistent. SCM can also be used for server side caching.
- *Object Factory (OF):* It constructs an object from the serialized data. CCM/SCM uses this module whenever an object faulting occurs. We have used Java's serialization (readObject/ writeObject) procedure to serialize the entire object graph.



PCO: Proxy Client Object
PSO: Proxy Server Object
CRF: Client Request Forwarder
SRF: Server Request Forwarder
CCM: Client Cache Manager
SCM: Server Consistency Manager
OF: Object Factory

Figure 3.3: Proposed architecture (i) Event service based (ii) Wrapper based (iii) Interceptor based

Interceptor Based Architecture: This architecture comprises of following components [Figure 3.3 (iii)]:

- *Proxy Client Object (PCO):* Non-collocated proxy object, responsible to cache the objects in the client side. PCO has some additional methods (that are used to work the caching mechanism properly) to the object for which it is used to be a proxy. PCO has a reference to PSO, which is used to maintain the system consistent.
- *Proxy Server Object (PSO):* Non-collocated proxy object, responsible to cache the objects in the server side. For each object, PSO has list of PCOs containing a copy of the object. Whenever the object's state changes, PSO sends an invalidate message to all registered PCOs. PCO and PSO are isolated from client and server applications so that they can be used by other applications that are also using caching facility.
- *Client Request Forwarder (CRF):* Client request interceptor. It intercepts client requests, and redirects the request to PCO by throwing a *"ForwardRequest"* exception.
- *Server Request Forwarder (SRF):* Server request interceptor, responsible to intercept client requests.
- *Object Factory (OF):* It constructs an object from the serialized data. PCO/PSO uses this module whenever an object faulting occurs.

4. IMPLEMENTATION

This section describes implementations of the architectures proposed in the previous section using Visibroker CORBA. We considered a bank application scenario with multiple read and update operations. A typical scenario involves an instance of a bank manager that creates account objects by opening accounts and exports that for the clients to use. An object wrapper/interceptor/smart skeleton around the account manager is installed in the server side that notifies when the users update their accounts and sends requests to the registered clients to invalidate their cached copies. To obtain object graph level granularity we have used Java's serialization procedure where the entire object graph can be shipped from the server side and can be reconstructed in the client side.

Caching: For all architecture, process level cache (requires less cache memory but more complexity in the operating system call and greater overhead with respect to time) management has been used that uses Hash table (which will be main memory in Java during execution) as the cache memory. For event service based architecture, we have cached data items instead of object graph. CCM/SCM, PCO/PSO maintain their own Hash tables to cache the objects.

Cache Replacement: In all implementations, LRU cache replacement policy has been used. Whenever an object is referred that resides in the cache, the current system time is attached with the page containing the object. The oldest page is replaced whenever the cache is full. We see that LRU policy gives ~94% hit ratio for 6% cache size. As mentioned earlier that a more sophisticated replacement policy (which we left for the future work) has to be devised for practical applications that does not follow traditional locality of reference.

Sharing Mode: Number of readers/writers can also vary greatly with respect to an application type. As for example, for railway reservation system, an enquiry about a train may occur from a number places while enquiry about the balance of a bank account will perhaps be from one place. In general, performance of an algorithm greatly depends upon the number of readers/writers. In this paper, we have implemented MRMW sharing mode. As the number of readers/writers increases, the performance decreases for a fixed percentage of read operation. We have shown that the rate of the increase of accessing an object decreases with the increase of percentage of read.

Consistency Maintenance: For event service based architecture, consistency is maintained using event consumers and event suppliers. To intercept requests, we have explicitly altered the stub and skeleton code generated by the idl compiler. Whenever an object's state changes, smart skeleton intercepts this and event supplier dispatches an *"invalidate"* message to event channel, which eventually delivers this message to all event consumers.

For wrapper based architecture, CCM maintains the system weakly consistent while strong consistency is maintained by the SCM. This architecture employs "*write-through"* policy, where updates, which are applied to the cache, are sent to the original object first. The SCM, in turn, *invalidates* the other replicas that reside in different registered client's cache

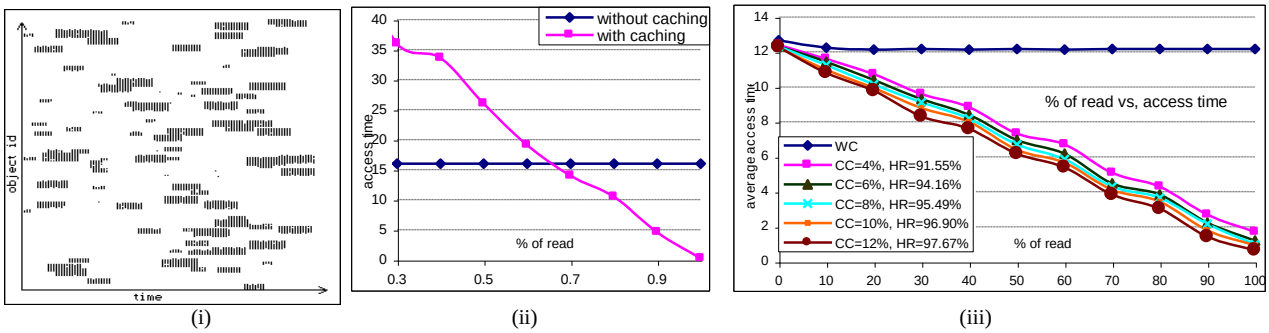
In case of interceptor-based architecture, two interceptors CRF (Client Request Forwarder) and SRF (Server Request Forwarder) are used to intercept requests in the client and server side respectively. CRF redirects request to the PCO by throwing a *"ForwardRequest"* exception for the first time. Successive requests will be forwarded to the PCO by ORB itself. Consistency is maintained in a similar way as wrapper based architecture by PCO and PSO using *write-through-invalidate* policy. For each object PSO has a list references to registered PCOs containing a replica. Whenever object's state changes, PSO sends *"invalidate"* message to all registered PCOs.

Consistency Initiation: Event service based architecture uses event supplier to initiate strict consistency. For weakly consistent system the consistency is initiated by the CCM in wrapper based architecture. CCM updates its cache content in the regular interval. This interval time (determines the tightness of consistency) should be very long than the average interval between two read/write accesses. If the CCM updates its cache content frequently, more tight consistent system is obtained while the performance degrades, because the system remains busy to update its cache most of the time. For interceptor based architecture consistency is initiated by PSO.

Granularity: We have implemented object graph level granularity for object shipping for wrapper and interceptor based architecture while data level granularity for event service based architecture. Object graph level granularity provides robust solutions whereas data or single object level granularity provides better response time. Java's serialization procedure, which is responsible to ship the entire object graph recursively, has been used here taking the factors mentioned earlier into account.

5. PERFORMANCE ANALYSIS

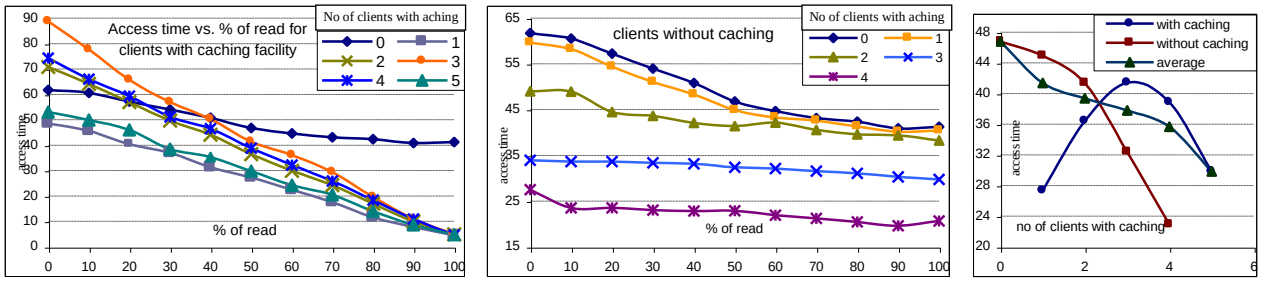
In this section we show experimentally measured performance with different cache capacity (cc) using LRU cache replacement policy. All measurements were carried out using average over large number (~4000) of iterations.



(i) (ii) (iii)

Figure 5.1: (i) Generated locality of reference data. Performance of (i) event service based (ii) wrapper based architecture.

The experiments were done on HP & IBM running Windows2000 Server and IBM PC running Windows98, connected by 10Mbps Ethernet. The server is started on hp tc2100 server (1.13 GHz) that accesses the Oracle 9i database running on IBM server (xSeries 220, 1.2 GHz) and clients are started on IBM pcs. The clients then make the CORBA calls satisfying locality of reference as shown in Figure 5.1 (i).



(i) *(ii)* *(iii)*

Figure 5.2: Performance of interceptor based architecture.

Figure 5.1(ii) shows the performance result obtained for architecture based on event service. The access time is less than the access time of the architecture that does not use caching for 65% or more read operations.

Figure 5.1(iii) shows the performance result obtained for the architecture based on object wrapper. Here the access time is always less than the access time for the architecture without caching (WC) for single client. We have used traditional LRU cache replacement algorithms and obtained a Hit Ratio (HR) of 94.16% for a cache size of 6%. Figure 5.2(i) and 5.2(ii) shows the performance result obtained for the architecture based on interceptor. Here we have shown

the effect of % of read operation on the access time in two ways. Figure 5.2(i) shows the access time for the clients that use caching facility. Figure 5.2(ii) shows the same for the clients that do not use caching facility.

Figure 5.2(iii) summarizes Figures 5.2(i) and 5.2(ii) showing average access time, considering all clients. We see that the average access time for all clients decreases with the increase in number of clients with caching facility. We claim that the interceptor-based architecture gives best result if all the clients use caching facility.

6. FUTURE RESEARCH DIRECTION

In this paper we use coarse grained (object graph) sized data shipping. As the size of the object graph increases, the overhead due to shipping procedure becomes significant that suggests partial object graph shipping. If a higher throughput or wider distribution is required then the platform is fully replicated and installed in the client site. In most of the cases, this approach is not feasible due to limited cache size. With the above scheme, a larger fraction of all objects can be cached resulting in improved performance. A sophisticated cache replacement policy has to be devised for systems that do not follow traditional locality of reference pattern. Our implementation cannot handle the situation whenever a fault (object or server) occurs. A fault tolerance mechanism has been left for future implementation.

Related Work: In [19] Zahir Tari, Herry Hamidjaja and Qi Tang Li have outlined a caching architecture that uses extension Least Recently Used algorithm, or Enhanced LRU (they call it LRU+) to minimize the overall overhead with a sorting process distribution and a variation of the Optimistic Two-Phase Locking (Optimistic 2PL) algorithm for consistency. In [22, 18], authors presented a caching approach (COCC) that is a transactional-based model. There is an overhead to install the interface repository that can negate the performance over a critical processing. In [8], authors developed a caching system (ScaFDocs) considering both strict and casual consistency. Cache replacement policy, effect of cache size and hit ratio on access time has not been indicated there. In [17], authors mainly concentrated on effect of page size on access time. Though our system can cache object graphs with variable size, the effect of graph size on access time is left as a future exercise. In [2], a qualitative discussion between distributed shared memory caching and proxy caching has been made without any implementation and hence quantitative discussion.

7. CONCLUSION

In this paper, we provided a number of object caching architecture and implementation based on event service, object wrapper and interceptor that can be used to cache different types of remote objects with little modifications of client code. These approaches were based on cache-invalidate cache consistency policy, per process caching, object graph based data shipping and replication management. We demonstrated that caching results in improved performance for applications that exhibit a reasonable amount of locality.

Acknowledgement: This work is partially funded by U.G.C sponsored M.R.P entitled "Performance Analysis of Distributed Component Platforms & Introduction of a Unified Distributed Architecture" (Ref. No. F.14-16/2000(SR-I) dated Oct 12, 2000).

8. REFERENCES

[1] Charu Aggarwal, Joel L. Wolf and Philip S. Yu. Caching on the World Wide Web. *IEEE Transactions on knowledge and data processing, vol 11, no 1, January-February, 1999*

[2] Juan-Carlos Cano, Ana Pont and Julio Sahuquillo. The Difference between Distributed Shared Memory Caching and Proxy Caching. *IEEE Transaction on Computers, July-September 2000.*

[3] Gregory Chockler, Danny Dolev, Roy Friedman and Roman Vitenberg HUJI and Technion: Implementing a Caching Service for Distributed CORBA Objects (CASCADE).

[4] Pascal Felber, Benoit Garbinato and Rachid Guerraoui. The Design of a CORBA Group Communication Service. *Proc. Of the 15th Symposium on Reliable Distributed Systems (SRDS-15), Niagra-on-the-lake, Canada, 1996.*

[5] Amitranjan Gantait, Samiran Chattopadhyay and Uttam Roy. Event Service Based Architecture for Object Caching in CORBA.

[6] Aniruddha S. Gokhale and Douglas C. Schmodt. Measuring and Optimizing Latency and Scalability Over High-speed Networks. *IEEE Transaction on Computers, Volume 47, No. 4, April 1998.*

[7] Object Management Group (http://www.omg.org/). The Common Object Request Broker: Architecture and Specification.

[8] R. Kordale and M. Ahamad. Object Caching in a CORBA Compliant System. *Technical Report: GIT-CC-95-23.*

[9] M. C. Little and S. K. Shrivastava. Implementing High Availability CORBA Applications with Java. *IEEE Workshop on Internet Applications, WIAPP'99, San Jose, July 1999.*

[10] C Marchetti, L. Verde and R. Baldoni. CORBA Request Portable Interceptors, A Performance Analysis. *Proc. of the Third International Symposium on Distributed-Objects and Applications (DOA'01), 2001.*

[11] Veljko Milutinovic. Caching in Distributed Systems. *Guest Editor's Introduction, IEEE Concurrency, July-September, 2000*

[12] T. Mowbray and R Malveau. *CORBA Design pattern.* John Wiley Computer Publishing, 1997

[13] Priya Narasimhan, Louise E. Moser and P. M. Melliar-Smith. Using Interceptors to Enhance CORBA. IEEE Transactions on Computers, 1999

[14] Michael N. Nelson, Graham Hamilton and Yousef A. Khalidi. A framework for Caching in an Object Oriented System. *The SMLI Technical Report series, 1993, Sun Microsystems Inc.*

[15] Robert Orfali and Don Harkey. Client Server Programming with Java and CORBA. *2nd Edition, John Wiley, 1998*

[16] Thomas Sandholm, Stefan Tai, Dirk Slama and Eamon Walshe. Design of Object Caching in a CORBA OTM System. IONA Technologies plc

[17] S. Selvakumar. Implementation and Comparison of Distributed Caching schemes. T*he IEE International Conference on Networks, 2000.*

[18] Zahir Tari, Slimane Hammoudi and Stephen Wagner. A CORBA Object-based Caching with Consistency. *Proc of the International Conference on Database and Expert Systems, Florence, September 1999.*

[19] Zahir Tari, Herry Hamidjaja.,Cache Management in CORBA Distributed Object Systems. *IEEE Concurrency, July-September, 2000, p48-55.*

[20] Borland Technology. Visibroker for Java Programmers Guide, v5.2.1. www.borland.com.

[21] Oliver Theel and Markus Pizka. Distributed caching and Replication. *The 32nd Hawaii International Conference on System Sciences-1999.*

[22] Stephen Wagner and Zahir Tari. A Caching Protocol to Improver CORBA Performance. *Proc. Of the Australian Database conference, 31st January-3rd February, Canberra, Australia, 2000.*