Sharing Objects in a Distributed System

Alan Judge Vinny Cahill[*]

Distributed Systems Group, Dept. of Computer Science,

Trinity College, Dublin, Ireland

TCD-CS-93-35
December 1993

This paper also appears in *Proceedings of the Third International Workshop on Object Orientation in Operating Systems*, pages 136–140, Asheville, North Carolina. IEEE Computer Society Press, December 1993.

Abstract

This paper presents a design for the use of DSM *techniques and system-supported synchronisation to support shared access to persistent objects in a distributed environment. We adopt a hybrid approach where the system granularity is sometimes pages and sometimes objects. We are interested in providing shared access to small (i.e., less than a page) objects in a general purpose, language-independent environment, and supporting both* DSM *and* RPC *object access mechanisms.*

1 Introduction

Object-oriented systems have traditionally relied on remote procedure calls (RPC) as the fundamental method for accessing remote objects in distributed environments. However, the RPC model of shipping invocations to an object can be limiting, preventing, for example, simultaneous legal accesses to copies of an object on multiple nodes. Instead, all accesses to an object are sent to a single node.

Over the last few years, there has been an explosion of interest in distributed shared memory (DSM) [18] as a communications paradigm. The primary advantage that DSM gives over RPC is the ability to transparently migrate and replicate data so that it can be accessed on multiple nodes simultaneously. We are interested in applying DSM techniques in an object-oriented setting.

[*]E-mail: {**Alan.Judge,Vinny.Cahill**}**@cs.tcd.ie**

Most of the research done on DSM systems has concentrated on page-oriented shared memory, where the DSM system is closely linked with the virtual memory system. In page-based systems, there are good reasons for this link, because the virtual memory hardware on a machine frequently provides the only way to track accesses to shared memory and to allow or disallow access based on the DSM algorithm being used.

In an object-oriented system, however, things are somewhat different. Here, assuming strong encapsulation, the primary access path to a region of memory within an object is via object invocation; as a result, we have much more control over access to shared data, and, in fact, need not rely on hardware assistance to determine when (and how) a region of memory is being accessed. We also have access to extra information about how memory is laid out and how it is being used, which we can use as input to a shared memory system.

Much previous work on applying DSM techniques in an object-oriented environment has concentrated either on heavy-weight objects, frequently consisting of a large portion of an address space [4]; on the support of specific applications, such as databases, where object access involves an indirection or is transactional [13]; or on the support of single (specialised) programming languages [1, 3, 7]. We, however, are interested in providing shared access to small (i.e., less than a page) objects in a general purpose, language-independent environment. Similarly, previous work on supporting object mobility has produced quite complex single language systems [12] and systems that lack object replication [15]. We intend to provide integrated and transparent support for object mobility and replication in existing languages.

Closely related to the issue of sharing (be it objects or pages) is the issue of synchronisation. By pushing object synchronisation down to the system level, we can make use of the application-specific knowledge linking locks to specific areas of data. This extra semantic information can be used for two purposes. Firstly, the information reduces the need for page fault based synchronisation and consistency maintenance [17]. Secondly, we can use the finer granularity of the information to maintain shared memory consistency at the object level and therefore reduce *false sharing* effects when several objects share a page [5]. By allowing application control over the synchronisation and consistency mechanism, we can even work at a finer granularity and allow multiple inconsistent copies of an object to exist subject to application specific merging.

Taking a purely DSM based approach can be too limited in a flexible environment. For example, an RPC mechanism is still required to support objects that have a fixed physical location, such as those controlling physical devices. Similarly, providing support for heterogeneity introduces a whole range of complications — not least of which is that an object may have different sizes on different nodes — which change the tradeoffs so that RPC may be more attractive.

This paper presents an initial design for the use of DSM techniques and system supported synchronisation to allow shared access to persistent objects in a distributed environment. We adopt a hybrid approach where the system granularity is sometimes pages and sometimes objects. We will also show how this support can interface with a *Generic Runtime* library (GRT) [6] to allow object sharing in a language independent manner. We intend to support both RPC and DSM mechanisms for object access as we see both function shipping and data shipping to be valid and useful communication models.

2 Background

The aim of the Amadeus project [10] is to a provide a general purpose, object-oriented, programming environment supporting distributed and persistent applications in a multi-user, heterogeneous, distributed system. Amadeus supports, through the use of a *Generic Runtime* library (the GRT) [6], a number of existing object-oriented programming languages, such as C**++** and Eiffel. One of the goals of the project is to support a number of languages on one distributed platform as well as facilitating language inter-working. Current versions of Amadeus permit objects to be active in only one address space at any one time and use RPC to achieve distributed access; synchronisation is achieved locally. This paper discusses extensions to the Amadeus system that will permit the replication and migration of objects while providing distributed synchronisation to maintain consistency.

In order to make the following explanation clear, we introduce a few key terms from the Amadeus vocabulary. An *object* consists of a *language object* together with its GRT header (which is itself a C**++** object). Objects are stored in groups of (related) objects called *clusters*. An *extent* is a protection domain and each cluster belongs to exactly one extent. All the objects within an extent are assumed to be trusted together and any given address space will only contain objects from a single extent.

3 Design outline

This section presents an outline of the new invocation system in Amadeus-2.

The most important point is that all object accesses are synchronised. This access synchronisation is enforced by the system by trapping invocations to unlocked objects. This trapping is achieved by binding different versions of code to the object when it is an unlocked state. This allows us to maintain consistency on a per-object, rather than a per-page basis.

An object in the system can be in one of three states. It can be *dormant* when it is not mapped into any address space and has no code bound to it. It can be *activated* when it has been mapped into an address space and has had code bound to it by the language specific runtime (LSRT). Finally, it can be *locked* when a lock has been obtained and the real code has been bound to the object. If the lock is released, the object will return to the *activated* state. Figure 1 shows a simple system with just one cluster in use.

3.1 Initialisation

When a process is created in Amadeus-2, its address space will initially contain no objects and the only information available is a reference to an object (called an *indicator*) and a description of an initial invocation to perform on that object.

When starting up, a new Amadeus-2 process will ask the GRT to convert the object indicator into a pointer to an object and perform the initial invocation. The GRT will determine in which cluster the desired object is stored and then contact the *extent manager* for the extent in which the cluster is currently resident. The extent manager is responsible for mapping objects in and

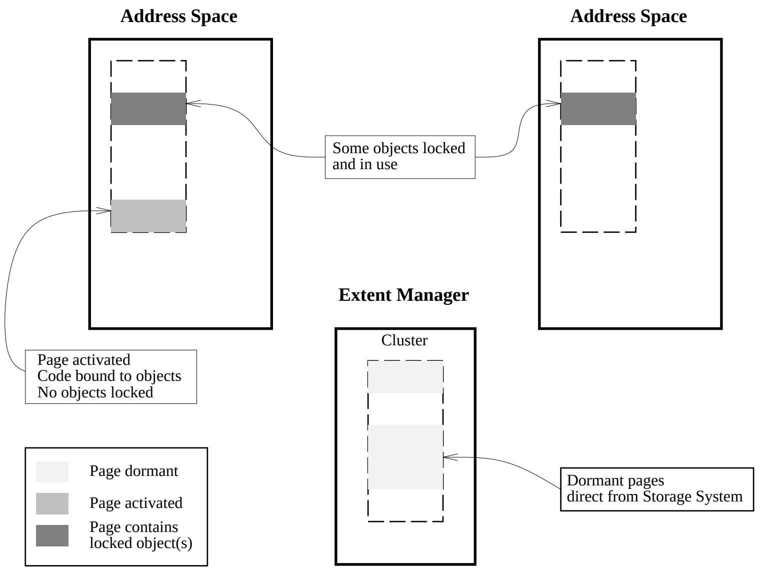
Address Space
Address Space
Some objects locked and in use
Extent Manager
Cluster
Page activated Code bound to objects No objects locked
Page dormant
Page activated
Page contains locked object(s)
Dormant pages direct from Storage System

Figure 1: Active system working with a single cluster

out of the address space, in a manner analogous to an external pager in a page-based system. The extent manager is a distributed entity consisting of a single primary process plus a number of local representatives. The primary process is responsible for interacting with secondary storage and maintaining consistency between different address spaces active in the extent. The local representative (one thread in each participating address space) is responsible for lock caching and language specific operations such as *swizzling*[1].

After being contacted by the GRT, the extent manager will allocate a mapping address in the shared extent space for the cluster and return this base address to the GRT. The GRT will then use the offset stored in the object reference to produce a pointer to the target object's header. It will then make an upcall to the LSRT to unmarshall the invocation parameters and perform the invocation. This access will cause a page fault.

3.2 Page Activation

Since Amadeus-2 supports direct language-level references to objects, through which operations may be invoked, certain parts of a page must be valid when it is first faulted into an address space. We call this *page activation*. Since any language or GRT object within the page may be invoked without warning, code binding must have been performed. This binding is achieved (if necessary) at page fault time by the local representative of the extent manager, using upcalls to the LSRT. If necessary, the dormant page is obtained from the primary process. For previously unbound objects, code is bound to the object so that invocations will be trapped.

[1]Swizzling is the process by which disc format object references are translated into valid language pointers within and address space.

Note that objects within the page may be both locked and unlocked and may not have been swizzled as yet. It depends on whether the objects in question have been used before.

3.3 Object Locking and Swizzling

When the object is actually invoked, the invocation will be trapped by the code bound at page activation and the GRT will be asked to obtain a lock for the object. Since no lock is currently cached, the GRT will ask the extent manager to obtain a lock for the object. At this time, the extent manager will also check to see if the object has been swizzled and, if not, it will swizzle the object before returning the lock. Thus, swizzling can be handled on a per-object basis and the GRT does not need to initiate swizzling; indeed, swizzling policy can even vary from extent to extent or cluster to cluster. Once the lock for the object has been obtained, it can be cached by the GRT and the object's code binding changed so that invocations can proceed unhindered[2].

3.4 Object based consistency

As we are implementing object-based locking, we will allow multiple writable copies of *pages* to co-exist, subject to the constraints of the object level locking. Objects, and the pages that contain them, will be replicated and migrated on demand as objects are accessed from other nodes. Since we know where objects are within pages, we can merge changes from multiple pages when they return to the primary pager.

4 Issues

We intend to evaluate this approach and compare it both with previous versions of Amadeus and with similar work in the object-oriented, DSM, and OODBMS fields. A number of additional interesting research avenues open up when considering the issues raised by this paper.

There is considerable scope for work in applying complex locking systems, such as intent locking [2] and call back or lease-based locking [19]. There is also scope for integrating application supplied locking and consistency management with the system and we intend to investigate extending the GRT interface to provide for passing the necessary locking information down into the system.

The availability of a large amount of application specific semantic information at the system level allows us much more scope in the algorithms used to maintain object consistency. Release consistent[3] [8] and causal memory [11] algorithms seem particularly appropriate here.

Given that swizzling is now under the sole control of the extent manager, the possibility of having both swizzled and un-swizzled extents cohabiting in a system arises. The advantages of single level stores are well known [9, 14], but people have shied away from using them in 32-bit systems, because of the limitations that they imply. Allowing single level addressed extents to

[2]Except for tracking which locks are in use.

[3]In fact, the design outlined already implements a form of object-oriented release consistent memory.

cohabit with traditional swizzled extents allows us to gain many of the advantages of a single level system for sets of interrelated objects that can fit in a single address space, while still allowing us to access larger object sets.

Pages which have been activated or swizzed may be sent back to the extent's primary pager to allow it to provide pre-processed pages to other clients working in the same extent. Depending on the cost of activation and swizzling, this may be worthwhile.

We are also interested in the effects of clustering and lock granularity on the caching of locks. For example, when an application maps a page or cluster should it be immediately given locks for all the objects in the cluster or page; should this behaviour depend on what else is happening in the system; and what heuristics should be used to make these decisions?

Finally, we are interested in whether the concept of a cluster as it existed before is redundant given that whole clusters are no longer (necessarily) read and mapped in one operation. Also the original intention of the cluster keeping related objects together no longer seems relevant when parts of a cluster may be simultaneously in use on several different nodes. Or should we take the opposite approach and use dynamic information from the system as input into a reclustering algorithm?

5 Conclusions

We are currently implementing the design outlined in this paper, applying DSM techniques and system synchronisation to object sharing, and intend to evaluate the results against the previous version of Amadeus and other similar systems. The new system will interface with the Amadeus *Generic Runtime* library (GRT) allowing object sharing in a language independent manner and supporting both RPC and DSM mechanisms for object access. We believe that application supplied locking offers scope for improving the performance of the system and plan to extend the GRT interface to support this.

Amadeus-1 [6, 10] has been implemented on top of a number of different Unix platforms and is described in more detail in the references. Work has begun on a re-engineering of Amadeus in a micro-kernel environment based on Mach 3.0 and the OSF-1 server and this implementation will incorporate the design ideas outlined in this paper.

References

[1] P. Amaral, R. Lea, and C. Jacquemot. A model for persistent shared memory addressing in distributed systems. In *Proceedings of the 2nd International Workshop on Object Orientation in Operating Systems*, pages 2–12, Dourdan, France, Sept. 1992. IEEE.

[2] S. Baker. *System Issues in Persistent Programming and OODBMS Integration*. PhD thesis, Department of Computer Science, Trinity College, Dublin, July 1992.

[3] H. Bal. *Programming Distributed Systems*. Silicon Press, Prentice Hall, 1990.

[4] J. M. Bernabéu-Aubán, P. W. Hutto, M. Y. A. Khalidi, M. Ahamad, W. F. Appelbe, P. Dasgupta, R. J. LeBlanc, and U. Ramachandran. *Clouds* — A Distributed, Object-Based Operating System

Architecture and Kernel Implementation. In *Proceedings of the EUUG Autumn Conference*, pages 25–37, Oct. 1988.

[5] W. J. Bolosky, R. P. Fitzgerald, and M. L. Scott. Simple But Effective Techniques for NUMA Memory Management. In *Proceeding of the* 12^{th} *ACM Symposium on Operating Systems Principles*, pages 19–31. Dec. 1989.

[6] V. Cahill, S. Baker, C. Horn, and G. Starovic. The Amadeus GRT — Generic Runtime Support for Distributed Persistent Programming. In *OOPSLA*, 1993. To Appear.

[7] J. S. Chase, F. G. Amador, E. D. Lazowska, H. M. Levy, and R. J. Littlefield. The Amber System: Parallel Programming on a Network of Multiprocessors. In *Proceeding of the* 12^{th} *ACM Symposium on Operating Systems Principles*, pages 147–158. Dec. 1989.

[8] S. Dwarkadas, P. Keleher, A. L. Cox, and W. Zwaenepoel. Evaluation of Release Consistent Software Distributed Shared Memory on Emerging Network Technology. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pages 144–155, San Diego, California, May 1993.

[9] G. Heiser, K. Elphinstone, S. Russell, and G. R. Hellestrand. A Distributed Single Address-Space Operating System Supporting Persistence. SCS&E Report 9302, School of Computer Science and Engineering, University of New South Wales, Mar. 1993.

[10] C. Horn and V. Cahill. Supporting Distributed Applications in the Amadeus Environment. *Computer Communications*, 14(6):358–365, July/August 1991.

[11] P. W. Hutto and M. Ahamad. Slow Memory: Weakening Consistency to Enhance Concurrency in Distributed Shared Memories. In *Proceedings of the* 10^{th} *International Conference on Distributed Computing Systems*, pages 302–309. May 1990.

[12] E. Jul, H. Levy, N. Hutchinson, and A. Black. Fine-Grained Mobility in the Emerald System. *ACM Trans. Comput. Syst.*, 6(1):109–133, Feb. 1988.

[13] W. Kim. *Introduction to Object-Oriented Databases*. Computer Systems Series. The MIT Press, Cambridge, Massachusetts, 1990.

[14] E. J. Koldinger, J. S. Chase, and S. J. Eggers. Architectural Support for Single Address Space Operating Systems. In *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 175–186, Boston, Massachusetts, Oct. 1992.

[15] S. E. Lucco and D. P. Anderson. Tarmac: A Language System Substrate based on Mobile Memory. In *Proceedings of the* 10^{th} *International Conference on Distributed Computing Systems*, pages 46–51. May 1990.

[16] M. Mock, R. Kroeger, and V. Cahill. Implementing Atomic Objects with the RelaX Transaction Facility. *Computing Systems*, 5(3):259–304, 1992.

[17] U. Ramachandran, M. Ahamad, and M. Y. A. Khalidi. Unifying Synchronization and Data Transfer in Maintaining Coherence of Distributed Shared Memory. In *Proceedings of the 1989 Conference on Parallel Processing*.

[18] M.-C. Tam, J. M. Smith, and D. J. Farber. A Taxonomy-Based Comparison of Several Distributed Shared Memory Systems. *ACM Operating Systems Review*, 24(3):40–67, July 1990.

[19] Y. Wang and L. A. Rowe. Cache Consistency and Concurrency Control in a Client/Server DBMS Architecture. In J. Clifford and R. King, editors, *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data*, pages 367–376, May 1991.