Implementing Atomic Objects with the RelaX Transaction Facility

Michael Mock and Reinhold Kroeger

German National Research Center for Computer Science (GMD)
Postfach 1240
D-5205 St. Augustin 1, Fed. Rep. Germany
e-mail: mock@gmdzi.gmd.dbp.de, kroeger@gmdzi.gmd.dbp.de

Abstract

RelaX offers an extensible transaction facility which isolates distributed (thus complex) transaction processing. A distributed system providing basic support for local recovery may easily be enhanced with flexible transaction functionality by linking it to the RelaX facility. The work desribed is being carried out in conjunction with the European ESPRIT Comandos project. In the Comandos platform, the transaction facility is used to implement atomic objects.

1 Introduction

Many experimental distributed systems have advocated a programming model in which atomic objects are manipulated on behalf of transactions. Usually, atomicity is an intrinsic property of all objects in the system and mostly either embedded in the language [7, 10] or related to a database style background store [2]. Together with several partners, we follow a different approach in the ESPRIT Comandos project (Construction and Management of Distributed Open Systems) [4]. Comandos identifies and constructs an integrated application support environment for developing and administrating distributed applications which can manipulate persistent objects. The Comandos platform provides the persistent and distributed object space as a generic, i.e. language independent system layer in which atomic and non-atomic objects co-exist. To achieve a uniform view and implementation of the system, the basic mechanisms to manipulate objects of both kinds are the same. Of course, management of non-atomic objects must not suffer from any overhead related to atomic objects. Thus, the problem is to enhance the system with transaction functionality for a subset of objects while keeping interference with the basic mechanisms low. Being generic, a purely language based approach does not work. Concentrating on the system aspects, we consciously neglect language level issues in this paper.

2 Basic Mechanisms

The basic mechanisms of the Comandos kernel architecture apply to non-atomic and atomic objects. They are described in this section as far as they relate to atomicity of objects. For a detailed description, see [1].

An activity is a sequence of object invocations executed in the global object space, independent of where the objects reside. A job is a set of related activities. A context is a collection of directly addressible objects on one site. An activity is represented by one or more processes in every context visited by the activity. Object invocations are initiated and executed at a language level. A language-independent virtual machine interface is provided by a generic run-time system (GRT) which enables the language specific run-time system to benefit from support for persistence and distribution. To communicate with the GRT, the language level has to provide for every considered object a predefined set of operations which are up-called from the GRT layer as required. The GRT maps objects into a context to make them accessible to the language. Accesses to non-mapped objects (object faults) are trapped and the GRT which will either map (activate) the object or initiate a cross-context invocation, e.g., if the object is already mapped in some other context (possibly on a remote site). The background representation of an object is maintained in a Storage Subsystem (SS) associated to one site in the system which is called the storage site of the object. On demand the object is activated and brought into the requesting context. The site of this context is called the activation site of the object. Objects are grouped into clusters to reduce the number of object faults and to increase the I/O rate. There is at most one active representation of an object at a time. Further accesses are

0-8186-2265-2/91 $1.00 © 1991 IEEE

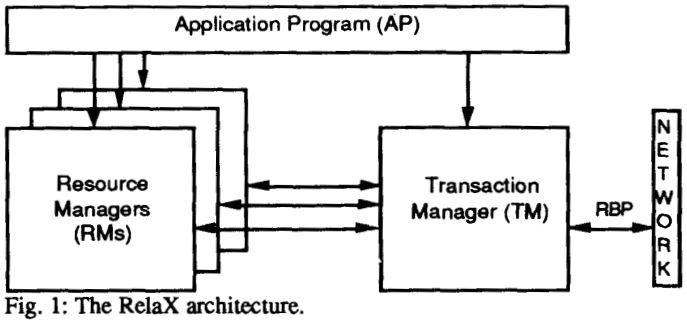

Fig. 1: The RelaX architecture.

directed to this representation until it is passivated. Passivating an object means deleting it from its context while writing its cluster back to the SS.

3 What is needed to implement atomic objects ?

Atomic objects provide consistent states even in the presence of concurrency and failures. Therefore, they must at least provide the following object local functionaltiy:

- synchronization of concurrent accesses to meet the serializability criterion,
- saving object states into recovery points, restoring recovery points to achieve the all-or-nothing property,
- saving object states stably to realize the permance of effect in spite of site failures.

But local functionality is not sufficient to define consistency on the basis of sets of operations grouped into transactions which either terminate successfully (commit) or leave no effect in the system (abort). For this, global coordination is necessary, which:

- detects failures/restarts of remote sites and forces affected transactions to abort,
- detects and cancels orphans created by timed-out remote invocations or site failures,
- runs a multi-phase commit protocol to achieve a common agreement on the outcome, i.e., a transaction must not commit at one site and abort at another site.

Conventionally, global transaction functionality is provided in a distributed database system. Unfortunately, it is irredeemably combined with the data-model of the database and cannot be extracted to be used in conjunction with atomic objects. In the RelaX facility, we identify and isolate transaction functionality which is independent of the underlying data-model and thus make it available outside of database systems.

4 What is provided by the RelaX transaction facility ?

Implementing transactions at the system-level requires a flexible transaction concept not taylored to a specific application. In our approach, this flexibility is expressed by the concepts of the premature release of data objects as a basis for the optional use of uncommitted data, separating the successful completion of a transaction from its commitment, group commitment, extended nesting and non-strict two-phase locking. A non-extensible version of this transaction concept taylored to a hardware-supported object-oriented architecture has been implemented in our predecessor project Profemo. For a detailed description, see [8, 9].

Extensibility is achieved by a clear separation between distributed transaction management and management of atomic resources (Fig. 1). This approach is also supported by X/Open [13]. On every site in the system, distributed transaction management is isolated in a server (TM) which cooperates via a standard interface with an extensible set of so-called resource managers (RMs) and application programs (APs). The TM is responsible for transaction control at a site and executes together with the TMs at the other sites distributed transaction control functions, i.e. commit/abort protocols [11]. TMs communicate via a reliable broadcast protocol RBP [3,12], which also is in charge of detecting and announcing failures of remote sites to the TM.

From the TM's point of view, the active entity responsible for the computational progress is the "application program". The "resource manager" is a passive entity which maintains state in the system. Different RMs might provide different kinds of data (e.g. object management systems, file systems or specialized databases) with different interfaces to the AP. Of course, the distinction between AP and RM is only of logical nature and does not prevent them to be represented by the same entity (e.g. a process maintaining programs and state).

Every AP binds a TM-library which logically links it to the TM. The AP defines transaction boundaries via

calls to the TM-library. Inside these transactions, it uses resources of one or more RMs. If computations go remote, the TMs on both sites are informed via the library in order to keep track of distribution. According to the non-strict two-phase locking, the AP may announce the lockpoint of a transaction to the TM, which propagates the lockpoint to all relevant RMs (on every affected site). Calls to the TM-library finally go from the AP to the TM. In the opposite direction, the TM calls the library to set up the computational state of the AP if a transaction aborts (which might have been initiated remotely, e.g., by a site failure).

Every RM binds a RM-library logically linking it to the TM. This library provides local concurrency control and local recovery control. Accesses to atomic resources must be trapped into the library for concurrency and recovery control checking. This is based on the identifiers of the current transaction and the accessed resource. When the library gets notice of a transaction for the first time, it announces to the TM that its RM joins this transaction. Concurrency control is handled completely inside the library. The library may be configured to handle different numbers of lock- and compatibility modes for different resources. With respect to recovery control, the library keeps track of which transaction uses which resources and cooperates with the TM to coordinate state changes of transactions (commit/abort) with the corresponding state changes of resources. This is basically a local extension of the distributed protocols executed by the TM. The RM is expected to implement supporting operations which are called by the library. These are operations to save and restore resource states to handle transaction aborts. For commitment, background representations of resources have to be updated in two phases, according to the progress of the commit protocol execution.

Summarizing, the RelaX transaction facility covers concurrency control (local and distributed aspects) and the distributed aspects of recovery control. Support operations performing local recovery are left to be implemented by the RMs in the most convenient and efficient way. For a detailed description of the architecture and the distributed protocols the reader is referred to [6, 11].

5 How it fits together

In the first step, we are concentrating on an implementation based on Unix (Ultrix 4.1). A context is represented by a Unix process, a (Comandos) process is represented by a thread. On every site, the TM is a separate Unix process.

Basic structure

For the TM, a context acts as a resource manager for the objects it currently maps. Simultaneously, the context is the application program with respect to all object invocations executed in the context. Thus, the TM-library and the RM-library are both linked with every context and connect it to the local TM.

Managing program state

Transactions are always started/ended within the same activity (concurrent activites may be created additionally within the transaction). Creation of a transaction blocks the current process and sets up a new process for the transaction. Thus, adjusting program state on behalf of a transaction abort is achieved by killing the processes which are actually running in the transaction.

Creation/garbage collection of atomic objects

New objects can be created in a context without interacting with the SS. If an object is atomic, its creation is announced to the RM-library. The RM-library will initiate a stable save for the object only if the creating transaction commits and if the object is not detected to be garbage. Thus, detecting atomic garbage objects reduces the amount of data involved in transaction commit. Of course, the garbage collector must take references stored in recovery points into account because they might be restored.

Trapping accesses to atomic objects

The basic strategy to trap accesses to atomic objects is to re-use the mechanism provided to trap object faults from the language to the GRT level. Access checks are then performed in the RM-library based on the current transaction identifier and on the virtual memory address and size of the object. After that, the normal GRT invocation path is resumed. If an invocation within a transaction goes remote, the GRT informs the TM via the TM-library. Both mechanisms do not affect non-atomic objects. They induce the cost of a language to GRT switch on the invocation of a local atomic object and two additional local messages to the TMs on both sites for remote invocations. We are also considering language extensions for atomic objects enabling the language layer to invoke the RM-library and the TM-library directly.

Save/Restore management in virtual memory

Transaction aborts are handled in virtual memory. Triggered by the RM-library, a recovery point of an object is saved by copying the current object representation into a seperate memory area (possibly newly allocated). The address of the copy is returned to the library and is used as source to restore the object in case of an abort.

Two-phase update using logs at activation sites

We log after images onto a stable log in the first phase of the commit protocol execution, thus avoiding log writes during the normal execution of a transaction. Under the control of the RM-library, a context stores its affected objects stably onto a log located at the activation sites (or at an associated log-server site if the activation site is diskless). Thus commit processing for objects activated at the same site is performed locally on that site. The possible alternative to use logs at the corresponding storage sites would make commit processing costly and would increase the probability of being blocked/aborted because the ability to commit then would depend on the availability of these storage sites.

To increase availability of clusters after a failure of a storage site, the storage site stably marks activated clusters. If activated clusters were not marked in the SS, a restarting SS would have no means to determine the current activation site and could not decide whether there is a new committed (or prepared) state of the cluster in the log. Especially, if any other site was not available, the SS must pessimistically assume that this site has been activation site and already maintains a new committed (or prepared) state of the cluster in its log. Thus, with no marking of activated clusters, a restarting SS could only respond to any activation request if all sites in the system were available.

6 Status

A first version of the Comandos virtual machine together with an extended C++ run-time system, called the Amadeus system, has been implemented by Trinity College, Dublin, one of our partners in the Comandos project. This version, which so far has no support for atomic objects, is the starting point for integrating transaction functionality. Implementation of the RelaX transaction facility is currently underway. The fully integrated Unix version is expected to be operational in early 1992.

7 Conclusion

The simplicity of the mechanisms described in section 5 shows that implementing atomic objects is fairly easy with the RelaX transaction facility, especially because the implementor is not burdened with the distributed aspects of transaction management. The next step will be to move to a micro-kernel based implementation taking advantage of virtual memory management facilities for access trapping and save/restore management. The implementation will also serve as a testbed for extensions of the implemented transaction concept with respect to type-specific concurrency and recovery control, and integrated replication schemes to achieve highly available objects.

Acknowledgements

We would like to thank our partners in the Comandos consortium who have been involved in the design of the kernel. These include Bull-Imag at Grenoble, Chorus at Paris, INESC at Lisbon and Trinity College at Dublin. Special thanks go to Vinny Cahill and Paul Taylor from TCD for many fruitful discussions on the integration of transactions into the Comandos system.

References

[1] V. Cahill, C. Horn, G Starowic, R. Lea and P. Sousa: Supporting Object Oriented Languages on the Comandos Platform. Accepted, Esprit Conference 1991.

[2] M. Carey, D. DeWitt, D. Frank, G. Graefe, J. Richardson, E. Shekita, M. Muralikrishna: The Architecture of the Exodus Extensible DBMS, First Int. Workshop on Object-Oriented Database Systems, Pacific Grove, 1986.

[3] J. Chang, N. Maxemchuk: Reliable Broadcast Protocols, ACM Transactions on Computer Systems, Vol. 2, No. 3, Aug. 1984.

[4] Comandos Consortium: A Guide to the Comandos Platform, Esprit Project 2071, Deliverable D1-T2.2, 1991.

[5] K.P. Eswaran, J.N. Gray, R.A. Lorie, I.L. Traiger: On the Notions of Consistency and Predicate Locks, CACM, Vol. 19, No. 11, 1976.

[6] R. Kröger, M. Mock, R. Schumann, F. Lange: RelaX - An Extensible Architecture Supporting Reliable Distributed Applications, 9th Symposium on Reliable Distributed Systems, Huntsville, 1990.

[7] B. Liskov, R. Scheifler: Guardians and Actions: Linguistic Support for Robust Distributed Programs, Proc. 9th ACM Symp. on Operating System Principles, Bretton Woods, 1983.

[8] E. Nett, J. Kaiser, R. Kröger: Providing Recoverability in a Transaction Oriented Distributed System, Proc. of 6th Int. Conf. on Distributed Computing Systems, Cambridge, Mass. 1986.

[9] E. Nett, K.-E. Grosspietsch, A. Jungblut, J. Kaiser, R. Kröger, W. Lux, M. Speicher, H.-W. Winnebeck: PROFEMO - Design and Implementation of a Fault Tolerant Distributed System Architecture, GMD-Studie 100, 1985.

[10] G. D. Parrington: Reliable Distributed Programming in C++: The Arjuna Approach, 2nd Usenix C++ Conference, San Francisco, 1990.

[11] R. Schumann, R. Kröger, M. Mock: The Decentralized Non-Blocking RelaX Commit Protocol, 11th ITG/GI-Fachtagung Architektur von Rechensystemen, Baden-Baden, 1990.

[12] R. Vonthin: Spezifikation des PROFEMO-Reliable Broadcast Protokolls in Unix 4.2 BSD, GMD-Studie 127, 1987.

[13] X/Open Preliminary Specification: Distritbuted Transaction Processing: The XA Interface, 1990.