References

[1] Sabine Habert, Laurence Mosseri, and Vadim Abrossimov. COOL: Kernel support for object-oriented environments. In *ECOOP/ OOPSLA'90 Conference*, volume 25 of *SIGPLAN Notices*, pages 269–277, Ottawa (Canada), October 1990. ACM.

[2] Deshayes, J.M., Abrossimov, V. and Lea, R. The CIDRE distributed object system based on Chorus. Proceedings of the TOOLS'89 Conference, Paris, France. July 1989.

[3] The Integrated Systems Architecture project. ISA - Esprit project 2267. The ISA consortium, APM ltd, Castle Park, Cambridge, UK.

[4] Vinny Cahill, Rodger Lea and Pedro Sousa. Comandos: generic support for persistent object oriented languages. To appear, Proceedings of the Esprit Conference 1991. Brussels, November 1991. also Chorus systèmes technical report CS-TR-91-56.

[5] Lea, R. and Weightman, J., COOL: An object support environment co-existing with Unix. Proceedings of Convention Unix '91, AFUU, Paris France. March 1991.

[6] Lea, R. and Weightman, J. Supporting Object Oriented Languages in a Distributed Environment: The COOL approach. Proceedings of TOOLS USA'91, July 29-August 1, 1991. Santa Barbara, CA. USA.

[7] Shapiro, M., Collet., P., Lea, R., Amaral, P. and Jacquemot C. Soul: an object-oriented OS framework for persistent object support Submitted to 1991 conference on system sciences, Hawaii, 1991, also available as CS-TR-91

[8] Campbell, R. H. and Madany, P. W. Considerations of Persistence and Security in Choices, an Object-Oriented Operating System. Procs. of International Workshop on Computer Architectures to Support Security and Persistence of Information. May 1990, Bremen (Germany).

- For clusters mapped into an existing cluster space, instead of using an RPC call, we are able to de-map the cluster and re-map into the calling cluster space, or into a cluster space on the same machine allowing us to use the light weight form of the standard RPC call.

COOL-base is capable of using this range of mechanism to carry out the invocation. The choice of mechanism will be dependent on higher level policy, but a simple approximation, using invocation efficiency as a criteria allows us to build a lightweight, default policy into the base level.

5.3 Clustering Policy

Based on our experience in COOL-1 we have moved the notion of objects out of the base level, replacing them with a larger grained entity. (Which could be viewed as a system level object). These larger grained entities represent clusters of application level objects. The policy choosen to cluster such objects is of paramount importance for efficiency, but is also extremely delicate.

For efficiency reasons, all objects that invoke each other should be clustered together, however, this causes problems in a distributed persistent model because the groupins we create at any particular time may not remain valid during longer periods of execution.

Our current approach is to adopt a simplified approach whereby we scan source code noting object interaction and cluster objects which have multiple inter-references. This is augmented by a user interface that allows explicit creation of new clusters and creation of objects in named clusters.

We hope in the future to explore the problems of dynamic clustering based on the execution pattern of objects.

6 Conclusion and current status

The COOL project is building an object oriented kernel above the CHORUS micro-kernel. Its aims are to provide a generic set of abstractions that will better support the current and future object oriented languages and applications.

While COOL defines an entire object oriented architecture which is common to work in Comandos and SOUL, COOL differs in its advanced use of the CHORUS micro-kernel and with its exploitation of the CHORUS virtual memory model.

Our premise is that the abstractions we provide at the lowest level will support both the model of construction for operating systems, and that of application level via the intermediery run-time levels.

We currently have a limited COOL platform running above the CHORUS micro-kernel, running native on 386 based machine. This platform implements the basic cluster level, but lacks light weight RPC and a sophisticated distributed virtual memory mapper. The COOL GRT has minimal functionality, lacking the protection system, and full support for persistence. We are currently testing this implementation and have just begun work on a simple language specific run-time for C++.

In particular, dealing with the conversion of *in memory pointers* to *persistent pointers*, when bringing objects to and from persistent store, and managing the dispatch model of a particular language, all use the upcall table to allow the GRT to ask the language specific run-time with help for semantic operations.

Again the reader is referred to [4] for a better description of these mechanisms.

5 Main research areas

While the project covers a number of areas of interest in distributed, persistent systems, the architecture poses a number of problems at the lowest level.

5.1 Distributed memory model

Each cluster space represents a logical distributed address space, with each cluster mapped into a number of physical address spaces. The model makes a coupling between virtual memory addresses and object addresses only during the time that clusters are mapped. It makes no statement about the coupling between these addresses when a cluster is moved to persistent store. Thus we can support a model where a cluster always occupies a set of addresses and that range does not change when it moves between persistent store, or we may adopt a model whereby, the binding is only maintained whilst a cluster is mapped into a cluster space. Of course wh need higher level (GRT) support for relocation of objects within clusters if we adopt this approach.

Both of these models impose a criteria for distributed memory allocation, since allocating a new cluster requires that all machines in the cluster space allocate the same space. Currently we adopt a simple model where portions of an address space are initially allocated to different machines. Creation of clusters initially uses this space and uses a standard distributed virtual memory to ensure that the allocation is propagated to all machines represented in the cluster space. When a machine exhausts this initial space, it must arbitrate with others to allocate space from a common pool.

5.2 Single invocation model

The base level abstractions include a invocation mechanism that works between clusters. Invocation falls into one of three cases. Local invocation, ie that which stays within an address space. Invocation local to a machine but between address spaces, and standard remote invocations (RPC). In a persistent, distributed system, there are a number of possibilities when invocation takes place concerning the location of the object.

In particular, the interaction between the invocation model and the cluster model provides us with the ability to optimise invocation:

- For a cluster that is held in persistent store, the cluster is mapped into the calling cluster space.

The *COOL-base* maps in clusters on behalf of the upper layers. It can be used to enforce an invoking thread to carry on execution in a remote address space. In addition, because clusters are persistent, the *COOL-base* provides a mechanism to locate non-active clusters, i.e., clusters currently swapped-out on secondary storage and load them transparently into a cluster spaces. A mapper is used to store and retrieve passivated clusters to and from secondary storage.

Therefore, the *COOL-base* level supports a single-level, persistent cluster store with synchronous and asynchronous invocation between clusters, and distributed cluster sharing.

4.2 The COOL generic run-time

Above the *COOL-base* level we provide a generic run-time level. The majority of the generic run-time code executes in users space.

The generic run-time implements a generic object oriented computational model, and has the following basic components.

- the **Execution Subsystem** (ES) provides support for object execution, including activities (lightweight threads) and jobs (distributed execution of activities);
- the **Virtual Object Memory** (VOM) handling all operations related with the manipulation of objects within clusters;
- the **Storage Subsystem** (SS) provides support for persistent objects;
- the **Communication Subsystem** (CS) is responsible for providing a generic RPC interface which is mapped onto the *COOL-base* invocation primitives;
- the **Protection Subsystem** (PS) ensures the specified level of protection during application execution.

A significant aspect of the GRT, and one that allows us to reduce the semantic gap between a generic object model and language specific ones, is the use of an upcall table associated with each object which is called by the GRT when it needs to access language specific information.

Since we wish to concentrate in this paper on the base level mechanisms, the reader is referred to [4] for a better description of the generic run-time layer.

4.3 The language specific run-time

The language specific run-time maps a particular language object model to the generic run-time model. This may be achieved through the use of pre-processors to generate the correct stub code and the use of the upcall table.

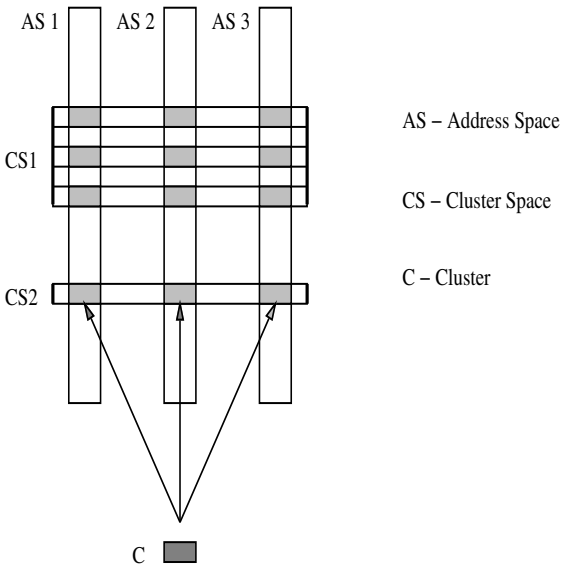
As discussed above, the GRT will, in the process of operations such as map/unmap, invoke each call into the language specific run time responsible for that object by using the upcall table associated with the object and generated by the language specific run-time.

this layer have a close relationship with CHORUS itself and they are intended to benefit from the performance of a highly mature micro-kernel.

The *COOL-base* provides address spaces were objects can exist, a way to share these objects in a distributed manner, a way to communicate between them, an execution model and a single level persistent store.

Using our previous experience with COOL-1 we have moved the notion of object out of our base layer and replaced it with two more generic abstractions, *clusters* and *cluster spaces.*

A cluster is viewed from higher levels as a place where related objects exist. When mapped into an address space, it is simply a collection of virtual memory regions. The mapping can be done on an arbitrary address. The collection of regions that belong to a mapped cluster is a set of CHORUS regions backed by segments, and forms a semantic unit managed by the base layer. By using a distributed virtual memory mapper, regions and hence clusters, can be mapped into multiple address spaces, which leads us to the notion of cluster space.



A cluster space is a collection of distinct address spaces on a non-empty set of nodes. The relationship between clusters spaces and address spaces is orthogonal, i.e., a cluster space can have arbitrary numbers of address spaces as well as clusters. Any cluster belonging to a cluster space is mapped into all address spaces of that cluster space. In this case, we must enforce that the cluster is mapped always at the same address. Therefore, a cluster space represents a distributed virtual address space, and this is a means to share clusters among threads of execution of a particular cluster space.

Each cluster is uniquely identified in the system as the unit of persistence. Clusters can have references to other clusters and they are subject of garbage collection.

The *COOL-base* also provides a low level mechanism for communication between clusters. This can be used to implement invocation of objects that exist inside the cluster. Transparent remote invocation is achieved with a simple communication model which uses the CHORUS communication primitives and protocols.

This provided the C++ programmer with the means to create globally known objects, dynamically link these into existing applications, invoke such objects across a network, attach activity to these objects, migrate objects between address spaces and machines and store these objects in a persistent store.

COOL-1 was used as a basis for a number of projects, in particular, the CIDRE project that has built a large distributed office document application running above the COOL-1 platform [2].

However, our initial implementation suffered from a number of drawbacks, in particular,

- The programmer was forced to explicitly deal with COOL-1 mechanisms for storage, distributed invocation and migration, ie we lacked transparency.
- COOL was designed to support multiple object models, however, experience with the fine grained model of C++ showed that in fact COOL really only supported a medium grained object model and that the cost to the kernel, even at such a granularity, were too great.
- Supporting (multiple) sophisticated object models with a generic set of mechanisms leads to a large semantic gap and hence inefficiency.

These problems and some interim experimentation with COOL-1 are reported in [5] [6].

3 COOL-2

In an attempt to address these problems and move the COOL platform from a toy towards a full object oriented operating system we began a redesign of the COOL abstractions in 1990. This work was carried out in conjunction with two European research projects, both building distributed object based systems, the Esprit ISA project and the Esprit Comandos project [3] [4] and with ongoing work at INRIA [7].

The result of this work has been the specification of the COOL-2 system and its initial implementation in the summer of '91.

4 The COOL architecture

COOL-2 is composed of three functionally separate layers, the *COOL-base* layer, the COOL generic run-time and the COOL language specific run-time layer.

4.1 COOL base

The *COOL-base* is the system level layer. It has the interface of a set of system calls and encapsulates the CHORUS micro-kernel. It acts itself as a micro-kernel for object-oriented systems, on the top of which the generic run-time layer can be built. The abstractions implemented in

Abstract

The CHORUS Object Oriented Layer (COOL) is a layer built above the CHORUS micro-kernel designed to extend the micro-kernel abstractions with support for object oriented systems. COOL-2, the second iteration of this layer provides generic support for clusters of objects, in a distributed virtual memory model. We discuss experiences with COOL-1 that have led to our current model and in particular, with our decision to build a two layer system where the lowest layer supports only clusters and the upper layers supports objects. We describe a number of problems that we are addressing with this new design and present the current status.

1 Introduction

COOL is an ongoing research project designed to explore the issues in building efficient object support mechanisms for distributed systems.

Our main goals are to:

- Explore the use of the CHORUS distributed micro-kernel and in particular its virtual memory model.
- Provide low level abstractions suitable for supporting distribution and persistence.
- Provide a common base onto which multiple object oriented models can be layered.

In this paper we discuss how the COOL system has been designed to exploit the unique features of the CHORUS operating system model to provide an efficient set of abstractions that are well suited to supporting the object oriented metaphor. A significant feature of the COOL project is that we have been able to exploit the design methodology of CHORUS to allow us to co-exist with the CHORUS Unix implementation (CHORUS/MiX) yet still build our abstractions along side Unix and thus exploit the CHORUS Nucleus functionality directly. This provides a very fast bootstrap environment (we can use the Unix file store and compilation chain) but because it is layered directly onto the CHORUS micro-kernel, is efficient.

2 History

The COOL project is now in its second iteration, our first platform, COOL-1[1], was designed as a testbed for initial ideas and implemented in late '88 [1].

In particular, COOL-1 supported a simple object model, an encapsulation of code and data as the base entity in the system. The COOL kernel provided mechanisms to create, name, invoke and migrate these entities within a locally area distributed system. To test out this base set of mechanisms, we built a minimal C++ support layer that mapped C++ objects onto our underlying COOL kernel objects.

[1] COOL-1 was built as a joint project between Chorus Systèmes, SEPT and INRIA

Contents

1 Introduction **1**

2 History **1**

3 COOL-2 **2**

4 The COOL architecture **2**

4.1 COOL base 2

4.2 The COOL generic run-time 4

4.3 The language specific run-time 4

5 Main research areas **5**

5.1 Distributed memory model 5

5.2 Single invocation model 5

5.3 Clustering Policy 6

6 Conclusion and current status **6**

CS/TR-91-103

COOL-2: an object oriented support platform built above the CHORUS Micro-kernel

Rodger Lea, Paulo Amaral, Christian Jacquemot

abstract: In: Proc. of 1991 IWOOOS, Palo Alto, CA, October 1991

© Chorus systèmes, 1991

© Chorus systèmes, 1991

October 1991