[50] [Schaffert86] Schaffert, C., T. Cooper, B. Bullis, M. Kilian, and C. Wilpolt, "An Introduction to Trellis/Owl", Proceedings of the Conference on Object-oriented Programming Systems, Languages and Applications (OOPSLA '86), 1986: Editor: N. Meyrowitz, Special Issue of SIGPLAN Notices, Vol: 21, Pages: 9-16.

[51] [Shapiro86] Shapiro, M., "Structure and Encapsulation in Distributed Systems: The Proxy Principle", Proceedings of the 6th International Conference on Distributed Computer Systems, May 1986.

[52] [Shapiro89] Shapiro, M., P. Gautron, and L. Mosseri, "Persistence and Migration for C++ Objects", Proceedings of the Third European Conference on Object-Oriented Programming (ECOOP '89), Nottingham, England, 1989.

[53] [Shrivastava87] Shrivastava, S. K., Dixon, G. N. and Parrington, G. D., "Objects and actions in reliable distributed systems", IEE Software Engineering Journal, vol. 2, no. 5, pp. 160-8, September 1987.

[54] [Skarra89] Skarra, A.H., and S.B. Zdonik, "Concurrency Control and Object-Oriented Databases", Object-Oriented Concepts, Databases and Applications. Editor: W. Kim and F.H. Lochovsky. New York: ACM Press, Pages: 395-422.

[55] [Stein89] Stein, L.A., H. Lieberman, and D. Ungar, "A Shared View of Sharing: The Treaty of Orlando", In: Object-Oriented Concepts, Databases, and Applications. Editors: W. Kim and F.H. Lochovsky. ACM Press, New York. 1989. Pages: 31-48.

[56] [Stroustrup86] Stroustrup, B., "The C++ Programming Language", Addison-Wesley, Reading, MA. 1986.

[57] [Sun85] Sun, "Remote Procedure Call Protocol Specification", Sun Microsystems Inc, 1985.

[58] [Tavanian87] Tavanian, A., "Architecture Independent Virtual MemoryManagement for Parallel and Distributed Environments: The Mach Approach", Technical Report CMU-CS-88-106, Department of Computer Science, Carnegie-Mellon, December 1987.

[59] [Turner85] Turner, D.A, "Miranda: A Non-strict Functional Language with Polymorphic Types", Proceedings of the IFIP International Conference on Functional Programming Languages and Computer Architecture, 1985.

[60] [Vaughan90] Vaughan, F., Schunke, T., Kock, B., Dearle, A., Marlin, C., Barter, C., "A Persistent Distributed Architecture Supported by the Mach Operating System", Proceedings of the 1990 Mach Applications Workshop, Usenix, Vermont 1990. USA.

[61] [Wegner87] Wegner, P, "Dimensions of Object-Based Language Design", Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '87), Editor: N. Meyrowitz, Special Issue of ACM SIGPLAN Notices, Vol: 22, Pages: 168-182. 1987.

[62] [Weightman91] Weightman, J., Lea, R., "Migration in Distributed Object Based Systems", Chorus Systems Technical Report, CS-TR-91- 1, Chorus Systems, France, January 1991.

[63] [Weihl88] Weihl, W.E, "Commutativity-based Concurrency Control for Abstract Data Types", IEEE Transactions on Computers Vol: 37 No.: 12, December 1988.

[64] [Xerox81] Xerox, "Courier: The Remote Procedure Call Protocol", Xerox System Integration Standard XSIS-038112, Stamford, Connecticut, 1981.

[65] [Zhou90] Zhou, S., Stumm, M., McInerney, T., "Extending Distributed Shared Memory to Heterogeneous Environments", Proceedings 10th International Conference on Distributed Computing Systems, CS press, California USA, June 1990.

[29] [Eliassen89] Eliassen, F, "FRIL Linguistic Support for Interoperable Information Systems", Technical Report , University of Tromso, Norway. March 1989.

[30] [Gallagher91] Gallagher, J.J., "Variations on a Theme", in Blair et al (eds), Object-Oriented Languages, Systems and Applications, Pitman Publishers, London, 1991.

[31] [Goldberg83] Goldberg, A., and D. Robson. "Smalltalk-80: The Language and its Implementation", Addison-Wesley, Reading, Mass, 1983.

[32] [Guedes91] Guedes, P., "Using C++ Proxies for Inter-Server Interaction", Proceedings of the OSF Micro-kernel Review, Boston, MA. Febuary 1990.

[33] [Guillemont91] Guillemont, M., Lipkis, J., Orr, D., Rozier, M., "Chorus: A Second Generation Micro-kernel based on Unix; Lessons in performance and Compatibility", Proceedings of the Usenix Winter'91 Conference, Dallas, Texas, January 1991.

[34] [Habert90] Habert, S., Mosseri, L., Abrosimov, V., "COOL: Kernel Support for Object-oriented Environments", Proceedings of OOPSLA'90, ACM Press, Canada,1990.

[35] [Halbert87] Halbert, D.C., and P.D. O'Brien, "Using Types and Inheritance in Object-Oriented Languages", Proceedings of the European Conference on Object-Oriented Programming, Pages: 23-34. 1987.

[36] [Herrmann88] Herrmann, F., et al, "Chorus, A New Technology for Building UNIX Systems", Proc. EUUG Autumn '88 Conference, October 1988.

[37] [Horn87] Horn, C., Krakowiak, S, "Object-Oriented Architecture for Distributed Office Systems", Proceedings of ESPRIT '87, North- Holland, 1987.

[38] [Hutto90] Hutto, P., "Weakening Consistency to Enhance Concurrency in Distributed Shared Memory", Technical Report GIT- ICS-89/39, Georgia Tech School of ICS, October 1990.

[39] [Johnston89] Johnston, G.M., Campbell, R.H., "An Object-Oriented Implementation of Distributed Virtual Memory", Internal Report, University of Illinois at Urbana Champaign, 1989.

[40] [Lea91b] Lea, R., "COOL-2: Initial Specification for the COOL-2 Object Support Platform", Chorus Systems Technical Report, CS-TR- 91-2, Chorus Systemes. France, January 1991.

[41] [Lea91a] Lea, R., Weightman, J., "COOL: an Object Support Environment Co-existing with Unix", Proceedings of the AFUU'91 Unix Convention, Paris, March 1991.

[42] [Lieberman85] Lieberman, H, "Delegation and Inheritance: Two Mechanisms for Sharing Knowledge in Object-Oriented Systems", Journes d'Etudes Langages Orients Objet, AFCET, Paris. 1985. Pages: 79-89.

[43] [Lieberman86] Lieberman, H, "Using Prototypical Objects to Implement Shared Behavior in Object-Oriented Systems", Proceedings of the Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA '86), Editor: N. Meyrowitz, Special Issue of ACM SIGPLAN Notices, Vol: 21, Pages: 214-223. 1986.

[44] [Liskov84] Liskov, B.H, "Overview of the Argus Language and System", Programming Methodology Group Memo 40, M. I. T. Laboratory for Computer Science. February 1984.

[45] [Magee89] Magee, J., Kramer, J., Sloman, M., "Constructing Distributed Systems in Conic", IEEE Transactions on Software Engineering, Vol. SE-15, No. 6, pp 663-675, June 1989.

[46] [Meyer86] Meyer, B., "Genericity versus Inheritance", Proceedings of the Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA '86), Editor: N. Meyrowitz, Special Issue of ACM SIGPLAN Notices, Vol: 21, Pages: 391-405. 1986.

[47] [Meyer88] Meyer, B. "Object-Oriented Software Construction", Prentice-Hall. 1988.

[48] [Mullender89] Mullender, S. (ed), "Distributed Systems", ACM Press, Frontier Series, Pages 65-86, 1989.

[49] [Ortega91] Ortega, M.I., "The Chorus Distributed Shared Memory Mapper", Technical Report, Chorus Systems, France, January 1991.

[9] [Black86] Black, A., N. Hutchinson, E. Jul, and H. Levy, "Object Structure in the Emerald System", Proceedings of the Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA '86), Editor: N. Meyrowitz, Special Issue of ACM SIGPLAN Notices, Vol: 21, Pages: 78-86. 1986.

[10] [Black87] Black, A., N. Hutchinson, E. Jul, H. Levy, and L. Carter, "Distribution and Abstract Types in Emerald", IEEE Transactions on Software Engineering Vol: SE-13 No.: 1, 1987, Pages: 65-76.

[11] [Blair90] Blair, G.S., "Distributed Invocation in Multimedia, Object-Oriented Systems" Position Paper, Workshop on Object- Orientation in Operating Systems, OOPSLA/ECOOP '90, Canada, October 1990.

[12] [Blair91a] Blair, G.S., Gallagher, J.J., Hutchison, D., Shepherd, W.D., "Object-Oriented Languages, Systems and Applications", Pitman Publishers, London, 1991.

[13] [Blair91b] Blair, G.S., "What are Object-Oriented Systems?", in Blair et al (eds), Object-Oriented Languages, Systems and Applications, Pitman Publishers, London, 1991.

[14] [Brachman83] Brachman, R.J., "What IS-A Is and Isn't: An Analysis of Taxonomic Links in Semantic Networks", IEEE Computer, pp 30- 36, 1983.

[15] [Bruce86] Bruce, K.M., and P. Wegner, "An Algebraic Model of Subtypes in Object-Oriented Languages", SIGPLAN Notices Vol: 21 No.: 10, 1986,

[16] [Burstall80] Burstall, R.M., D.B. McQueen, and D.T. Sannella, "HOPE: An Experimental Applicative Language", Internal Report CSR- 62-80, University of Edinburgh. May 1980.

[17] [Cahill91] Cahill, V., Horn, C., Kramer, A., Martin, M., Starovic, G., "C** and Eiffel**: Languages for Distribution and Persistence", Internal report, Comandos project, Trinity College, Dublin, January 1991.

[18] [Canning89] Canning, P.S., W.R. Cook, W.L. Hill, and W.G. Olthoff, "Interfaces for Strongly-Typed Object-Oriented Programming", Proceedings of OOPSLA '89, Pages: 457-467. 1989.

[19] [Cardelli85] Cardelli, L., and P. Wegner, "On Understanding Types, Data Abstraction, and Polymorphism", Computing Surveys Vol: 17 No.: 4, 1985, Pages: 471-522.

[20] [Comandos90] Comandos Kernel Team, "Comandos Kernel and Generic Run-time Interface", Chorus Systems Technical Report CS-TR-90-19, Chorus Systems, August 1990.

[21] [Cox86] Cox, B.J., "Object Oriented Programming: an Evolutionary Approach", Addison-Wesley 1986.

[22] [Cusack90] Cusack, E., and M. Lai, "Object Oriented Specification in LOTOS and Z or, My Cat Really is Object-Oriented!", presented at the REX/FOOL Workshop on the Foundations of Object Oriented Programming, Noordwijkerhout, The Netherlands, May 28th - June 1st, 1990.

[23] [Danforth88] Danforth, S., Tomlinson, C, "Type Theories in Object- Oriented Programming", ACM Computing Surveys Vol: 20 No.: 1, 1988, Pages: 29-72.

[24] [Decouchant88] Decouchant, D., A. Duda, A. Freyssinet, M. Riveill, X.R.d. Pina, G. Vandome, and R. Scioville, "Guide: An Implementation of the COMANDOS Object Oriented Distributed System Architecture on Unix", Proceedings of the '88 EUUG Conference, 1988.

[25] [Deshayes89] Deshayes, J.M., Abrossimov, V., Lea, R, "CIDRE, an Object Based Document Support Environment Running on Chorus", Proceedings of Tools'89 conference, Paris France, July, 1989.

[26] [Dixon89] Dixon, G.N., G.D. Parrington, S.K. Shrivastava, and S.M. Wheater, "The Treatment of Persistent Objects in Arjuna", The Computer Journal Vol: 32 No.: 4, 1989, Pages: 323-332.

[27] [Dtlefs88] Dtlefs, D.L., M.P. Herlihy, and J.M. Wing, "Inheritance of Synchronization and Recovery Properties in Avalon/C++", IEEE Computer December 1988, Pages: 57-69.

[28] [Ehrig85] Ehrig, H., Mahr, B. Fundamentals of Algebraic Specification. Springer-Verlag. 1985.

5 CONCLUDING REMARKS

The introduction of distribution is a significant and important step for object-oriented computing. As discussed in this paper, the move has a number of important consequences for the design and implementation of object-oriented systems. For example, distribution demands a broader view of the object model than captured by the concepts of object, class and inheritance. Similarly, distribution requires a wider range of implementation strategies to realise invocation.

These pressures are not unique to distributed object-oriented systems. As object-oriented techniques gain wider exposure, they are applied to an increasing number of application areas. This trend can be seen through the interest in object-oriented techniques in such diverse areas as artificial intelligence, databases, real-time systems, operating system design as well as distributed computing. With this widespread use, it is inevitable that there will be a level of divergence in the field. This divergence can already be seen. D-ifferent communities have adapted or modified the ideas behind object-oriented computing in order to meet their own specific demands. In the long term though this is a necessary stage for object-oriented computing. In the opinion of the authors, the divergence is to be welcomed; the end result will inevitably be a more mature and broader discipline. There is a clear demand though for a more fundamental understanding of object-oriented computing which is independent of any particular implementation strategy.

One of the main findings from research on distributed object- oriented computing is the need for underlying system support. This paper has looked at the approach taken in Chorus/ COOL to supporting object-oriented languages. This work has successfully been applied in a number of experimental systems. However, this area remains an important topic for research. In particular, it is important to integrate more flexible approaches to invocation into existing platforms. In addition, underlying support for type systems requires a great deal more attention. In conclusion, significant steps have been made in applying the object-oriented paradigm in the both distributed and shared environments. However, more research is required before the advantages of the object-oriented approach to software development can be realised in a distributed system.

References

[1] [Accetta86] Accetta M., Baron R., Bolosky W., Golub D., Rashid R., Tevanian A. and Young M., "Mach: a New Kernel Foundation for Unix Development", Proceedings of USENIX Summer Conference 1986, pp. 93-112, June 1986.

[2] [Albano85] Albano, A., L. Cardelli, and R. Orsini, "Galileo: A Strongly-Typed, Interactive Conceptual Language", Trans. Database Systems Vol: 10 No.: 2, June 1985, Pages: 230-260.

[3] [Amaral91] Amaral, P, R. Lea, and C. Jacquemot, "COOL-2: An object oriented support platform built above the Chorus Micro-kernel", Proceedings International Workshop on Object Orientation in Operating Systems, Palo Alto, Oct 17-18, 1991, pages: 68-73.

[4] [America87] America, P, "Inheritance and Subtyping in a Parallel Object-Oriented Language", Proceedings of The European Conference on Object-Oriented Programming, Pages: 281-289. 1987.

[5] [APM89] APM, "The ANSA Reference Manual", Version 01.01, Architecture Projects Management Ltd., Poseidon House, Castle Park, Cambridge, UK, 1989.

[6] [Bernabeu88] Bernabeu, J., Y.A. Khalidi, M. Ahamad, W.F. Appelbe, P. Dasgupta, R.J. LeBlanc, and U. Ramachandran, "Clouds - A Distributed Object Oriented Operating System: Architecture and Kernel Implementation", Proceedings of the 88' EUUG Conference, 1988.

[7] [Bershad89] Bershad, B., Anderson, T., Lazowska, E., Levy. H., "Lightweight remote procedure call", Proceedings of the 12th ACM SOSP, Arizona, December 1989.

[8] [Birrell84] Birrell, A. D. and Nelson, B. J., "Implementing remote procedure calls", ACM Transactions on Computer Systems, vol. 2, pp. 39-59, February 1984.

recipient actor. To provide further support for concurrency, the Chorus micro-kernel supports the notion of system level lightweight threads. Each thread is independently named, managed and (most importantly) scheduled by the micro-kernel. This feature allows system designers to exploit the rich send and receive opportunities of the multiple port model. The approach to memory management is one of the most interesting aspects of the micro-kernel. An actor in Chorus consists of a number of variable size regions, with each region being managed by a user level actor called a mapper. The micro- kernel provides support for the mapper in terms of basic mechanisms to bring in data in response to memory faults and to flush data when memory is no longer required. The mapper then provides the policy for memory management. Because this is a user level actor, this policy can be tailored for particular application domains. For example, the mapper could support the coherency required for distributed virtual memory.

COOL Base Layer The base layer is designed as a generic support platform for object based systems. The layer provides a set of abstractions mapped directly on to the Chorus micro-kernel which extend the micro-kernel interface to include the notions of objects and object management. More specifically, the base layer supports the creation of typed objects and the subsequent invocation of operations on objects. The layer also manages the mapping of objects into Chorus virtual address spaces (referred to as contexts in COOL) and the saving of such contexts as a form of large grained persistence. In addition, COOL allows the migration of objects between object spaces and across machine boundaries and augments this with an ability to migrate objects to and from secondary store.

COOL Run-Time The run-time layer provides a mapping from a language or system level object model to the COOL base facilities. Although the system is intended to be generic, experiments to date have focussed on a run-time layer for C++. This run-time layer transparently maps C++ objects on to COOL base facilities thus providing C++ programmers with the ability to develop dynamic distributed applications. The programmer writes C++ code as normal, but places some of the objects under the management of the COOL system. This additional management allows C++ objects to be enhanced with facilities of the COOL base layer. Thus COOL/C++ objects may be persistent, invoked both locally and remotely and may be migrated around a distributed system. Apart from this, distributed COOL objects appear exactly like other C++ objects.

4.3 Experiences of Chorus/COOL

The experiences with COOL to date have been positive. In particular, it was relatively easy to map the computational model of C++ on to the COOL facilities. COOL has also been used successfully in the development of a major distributed application in the CIDRE project [Deshayes89], i.e.an advanced distributed office automation environment.

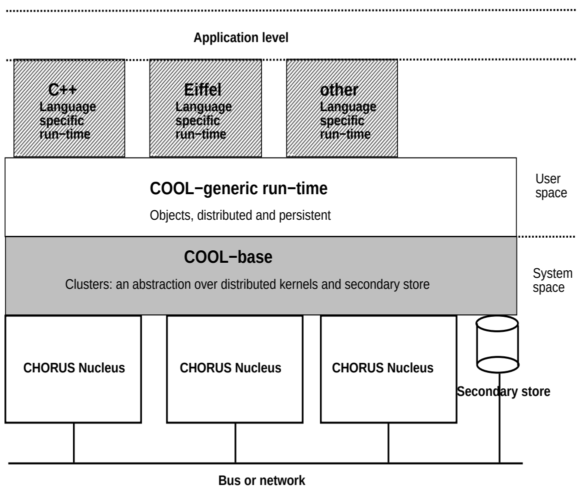
However, a number of a shortcomings with COOL have been identified and several modifications are under consideration. For example, preliminary investigations have been carried out into a more flexible invocation mechanism (as discussed in section 3.2.4.). The intention is to retain the abstraction of invocation whilst allowing the COOL system to select one of several mechanisms to achieve the invocation. This research is currently concentrating on migration techniques and shared distributed virtual memory to optimise invocation [Lea91b, Amaral91].

This research can be seen as contributing to the continuing developments in the provision of support for distributed object- oriented systems. The COOL system, whilst currently a research project, will form a major part of a future distributed object- oriented operating system.

existing distributed operating systems. This tension is examined in more detail below in a case study of supporting an object-oriented environment (COOL) on top of the Chorus distributed operating system.

4.2 Case Study: COOL

The Chorus Object-Oriented Layer (COOL) [Habert90, Lea91a] is a system designed to provide generic support for distributed object- oriented programming. The system was developed by Chorus systems and INRIA[4] , as part of a development for a research project with the SEPT[5] and is also being used by Lancaster University in research into object migration [Weightman91]. The architecture of the COOL implementation is shown in figure 4.



Application level
C++ Language specific run-time
Eiffel Language specific run-time
other Language specific run-time
COOL-generic run-time
Objects, distributed and persistent
User space
COOL-base
Clusters: an abstraction over distributed kernels and secondary store
System space
CHORUS Nucleus
CHORUS Nucleus
CHORUS Nucleus
Secondary store
Bus or network

Figure 2: The COOL architecture

Figure 4: Chorus/ COOL Architecture

The various layers in this diagram are discussed in turn below.

Chorus Distributed Operating System The COOL system is developed on top of the Chorus distributed operating system kernel [Guillemont91]. Chorus is representative of the current state of the art in distributed operating systems. The approach in Chorus is to provide a minimal kernel (or micro-kernel) which implements the basic functionalities required to support distribution. Other facilities, such as filing systems, are layered on top as distributed services[6]. The Chorus kernel supports a model of actors interacting through message passing. Two forms of message passing are provided: synchronous (where the sender blocks awaiting a reply to the message) and asynchronous (where the message is transmitted then the sender continues). Note that the messages are actually directed to ports rather than actors. Each actor is actually capable of supporting any number of individually named ports. Similarly, ports can be shared between more than one actor. Messages arriving at a port are queued and received by threads executing within the

[4]Institut National de Recherche en Informatique et en Automatique, Rocquencourt, France

[5]Service d'Etudes Communes des Postes et Telecommunications, Caen, France.

[6]Chorus have implemented a version of UNIX (MiX) on top of the micro-kernel. the system is made up of the following actors: a process manager, a pipe manager, a socket manager, a device manager and a file manager. Each of these are independently coded and can be distributed across several machine to provide a distributed Unix system.

remote procedure calls. It is now becoming clear, however, that systems will require a greater degree of flexibility in the future. As distributed object-oriented systems are applied to a wider range of application areas, it will become necessary to select the most appropriate mechanism for a given context [Lea90, Blair90]. Thus it can be foreseen that systems in the future will provide a wider range of mechanisms and will have the ability to dynamically select from this portfolio of techniques (this trend can already be seen in some distributed system platforms [Comandos91]).

There is also an interesting trend to deal with persistence and additional levels of transparency within the invocation framework. This implies that all the major problems of distribution are dealt within behind the implementation of the invocation mechanism. This can lead to clean and elegant designs. Thus the task of invocation now involves the following tasks:-

- discover the location of an object,
- decide on the most appropriate access method,
- decide whether the object should be re-located,
- maintain the integrity of persistent data, and
- bring in additional distribution management as required by the application.

This development requires a more general interpretation of invocation. Rather than implying a particular mechanism, invocation should be viewed as an abstraction which can be realised in a variety of ways. Thus, yet again, distribution demands a more general view of the basic concepts of object- oriented computing (c.f. section 3.1.3). In distributed object- oriented computing, it is not possible to equate abstractions with one particular implementation strategy.

4 SYSTEM SUPPORT FOR DISTRIBUTED OBJECT-ORIENTED SYSTEMS

4.1 The Importance of Underlying Support

From the previous discussion, it should be clear that the demands of distributed object-oriented computing are considerable. It is now recognised by the community that this requires systems support. It is simply not possible to develop distributed object-oriented environments on existing operating systems. There are two reasons for this:-

Need for efficiency Features such as remote procedure call protocols and distributed virtual memory demand a high performance underlying message passing mechanism for efficient implementation. This is particularly true in object-oriented computing where the patterns of interaction are rich. This observation has motivated a number of researcher-s to experiment with object-oriented systems layered on top of specialised distributed platforms [Vaughan90]. This work builds on the experience from the distributed systems community in developing distributed operating systems.

Need for tailored support Most operating systems are based around the traditional abstraction of a process, supplemented by mechanisms for interprocess communication. This provides developers of object-oriented systems with a dilemma as there is no clear mapping between an object and a process. This has often led to inelegant and expensive implementations of object- oriented systems. It is important therefore to consider the most appropriate systems support for object-oriented computing. This is one of the prime motivations for research into distributed virtual memory.

The most interesting feature of providing systems support is the need to balance these two requirements. It is important to consider the abstractions required to support object-oriented computing; in parallel, it is also important to consider the facilities offered by

Introducing Persistence Many current implementations of object- oriented languages are naive in their treatment of persistence and hence are not suitable for shared, multi-user environments. For example, persistence in Smalltalk is modelled by taking complete snapshots of the system state to be stored in a filing system. The move to distributed systems, however, demands a more sophisticated treatment of persistence. Essentially, persistent objects provide the unit of sharing in multi-user environments. There is a substantial research community working on this problem and significant progress has been made. In particular, there are a number of persistent object-oriented programming languages in existence, e.g. Arjuna [Dixon89], SOS [Shapiro89] and Galileo [Albano85]. The most interesting work in this field is the work that treats persistence and data type as orthogonal dimensions of a value (or object) [Morrison87]. This concept is often referred to as orthogonal persistence. It is likely that orthogonal persistence will increase in importance in the object- oriented community and will be adopted by more and more language developers. One of the main repercussions of orthogonal persistence is that it becomes possible to integrate two previously separate concepts, namely the programming language and the system environment.

Additional Transparencies Some distributed applications will require higher levels of transparency than provided by the invocation mechanisms described above. More specifically, certain applications will prefer problems of replication, migration, failure and concurrency to be dealt with by the system. This is normally achieved by the introduction of an appropriate manager. For example, a transaction manager is often provided to deal with issues of concurrency and failure. Distribution management is now a relatively mature area with a range of well known algorithms existing for transaction management, replication management and migration management. There is evidence though that object-oriented computing will have a major impact on these areas. The main reason for this is that there is much more semantic information available in object- oriented systems. Management strategies can be designed to exploit this semantic information in order to reach more optimal decisions. The clearest evidence of this trend is in the area of transaction management. A number of researchers are working on semantic transactions which exploit information on the behaviour of objects to increase the level of concurrency and the flexibility of transactions [Skarra89]. Early steps in this work was carried out in the Argus project [Liskov84]. Argus allows types to be created out of in-built atomic types. Atomic types provide fairly traditional mechanisms for maintaining consistency of the data. This is then enhanced by allowing users to define their own data types with their own semantics of consistency. Other projects are extending this work by taking into account the characteristics of different objects. For example, the ANSA transaction model employs path expressions to specify constraints on the concurrent execution of operations on an object. Other systems looking at flexible transaction mechanisms include Arjuna [Dixon89] and Avalon [Dtlefs88]. Researchers at Tromso University in North Norway are taking an interesting approach of determining consistency constraints from the formal semantics of data types [Eliassen89]. The semantics are written in the algebraic notation of ACT-ONE [Ehrig85]. Similar research is also being carried out by Weihl at MIT [Weihl88].

3.2.4 Towards a General Framework for Invocation

A number of techniques therefore exist to realise invocation in a distributed system. Each technique has its own strengths and weaknesses. For example, distributed virtual memory can be very efficient if there are many accesses to an object by a single client. However, remote procedure calls might be more appropriate if an object is heavily shared by a number of client objects.

Most systems developed to date have opted for one particular approach to invocation. For example, Choices is based on distributed virtual memory whereas Emerald relies on portion of the address space. If it is not present, an exception is raised (c.f. a page fault). The object is then located and brought into the local address space. Invocation can then proceed using standard procedure calls. In effect, distributed invocation introduces a further optimisation over proxies by migrating objects into a local address space as opposed to simply the local node. Distributed virtual memory systems also support sharing of objects by either copying objects or migrating an object from node to node. The power of this approach is its implicit ability to deal with the addressing problem outlined above. Because virtual addresses are shared across a number of address spaces, it is not necessary to convert object references; thus, the migration process is greatly simplified. However, there are a number of problems with distributed virtual memory. Firstly, the cost of maintaining consistency of shared virtual memory in a distributed system can be high. Current research is therefore considering weaker forms of consistency to lessen this overhead [Hutto90]. Secondly, it is recognised that distributed virtual memory has limits in terms of scaleability [Ortega91]. At present, it is possible to implement distributed virtual memory on small local area networks with homogeneous hardware. Moving to larger systems, with slower communications networks and with heterogeneous hardware, is still a research issue [Zhou90].

Comparison of Techniques At first sight, the list of techniques for implementing invocation can appear bewildering. However, it is possible to analyse the various approaches by considering two different aspects of their design:-

- access method All the approaches described above provide a particular mechanism to access a (potentially remote) object. For example, some approaches use remote procedure call protocols whereas others use standard procedure calls. Some systems may also provide lightweight remote procedure calls optimised for the local environment.
- location strategy Some of the approaches attempt to enhance the performance of the system by re-locating the object at another site. For example, proxies may decide to migrate an object to a local site or temporarily cache the object. Distributed virtual memory in contrast always re-locate the object into the local virtual address space. Thus, some approaches re-locate as a form of optimisation whereas others re-locate as a necessity.

The various design options are summarised below.

	Remote procedure calls	Proxies	Distributed Virtual memory
Access method	RPC's or lightweight RPC's	RPC's or lightweight RPC's	Procedure calls
Location strategy	Leave at original site	May optimize by caching or migration	Map into local address space

This diagram highlights the differences between the various techniques. Both distributed virtual memory and remote procedure calls have a fixed strategy for dealing with objects. Proxies though have more flexibility as they can make management decisions to improve the performance of the invocation.

3.2.3 Further Concerns

A number of other issues must be considered in a distributed implementation of invocation. The two most important problems are dealing with persistence and the provision of additional transparencies. These issues are discussed in turn below.

To hide the complexities of distributed programming, it is therefore necessary to resolve each of these issues. This clearly adds great complexity to the issue of distributed invocation. There is however a level of experience from existing distributed object-oriented implementations. This work is described below.

3.2.2 Implementing Transparent Invocation

Invocation Techniques It is not desirable to implement full distribution transparency for all classes of application. This would be too costly and would not be required for most distributed applications. The general approach is therefore to concentrate on a basic set of transparencies, namely location and access transparency, in implementing invocation. This level of transparency effectively allows methods to be invoked on objects in the same way, whether they are local or remote, without the caller being aware of their location. A number of techniques have been developed to achieve location and access transparency. The principal techniques are discussed below:-

Remote Procedure Calls Remote procedure calls (RPC's) [Birrell84] extend the semantics of local procedure calls into a distributed environment, allowing programmers to construct distributed application based on a conceptually local view of the world. The remote procedure call model exploits the underlying message passing facilities of a distributed system and uses this to create the abstraction of ordinary procedure calls. When a local procedure call to a remote object is made, special code, often called stub code, is executed. This packages up the parameters into a message and sends this message to the receiving object. At the receiving object, a similar piece of stub code unpackages the arguments and calls the correct local procedure. This process of packing and unpacking of parameters is referred to as marshalling. Marshalling can be quite complicated; for example, it is important to ensure that addresses sent to remote sites have a valid interpretation at that site [Mullender89]. There is now considerable experience in implementing remote procedure call protocols [Sun85, Xerox81] and a number of optimisation techniques have been proposed. For example, Washington University [Bershad89] have observed that, in practical systems, the vast majority of invocations are local and typically involve small data packets. They have therefore developed a lightweight RPC package which is optimised for this special case. Intuitively, this approach seems attractive for distributed object-oriented systems because of the nature of object interaction, however, it should be noted that the basis for the Washington work was traditional client/server based systems and not object oriented systems. A more straight-forward optimisation can also be achieved by trapping invocations within the same address space and mapping these calls directly on to a local procedure call [APM89].

Proxies Invocation based on RPC, although well established, often fails to recognise possible management optimisations. The concept of proxies [Shapiro86] was therefore introduced to overcome this problem. A proxy is a representative of a remote object in the local address space. Like stub code, they therefore provide a local interface to a remote object; however, unlike stubs, they can contain a degree of intelligence. For example, a proxy for a remote file object may cache recently accessed data to speed up access [Guedes91]. Proxies also provide a platform for additional forms of transparency (see below). For example, proxies can make use of local information and decide to migrate the remote object it represents from its remote context to the local one. This action is generally performed with the help of the underlying operating system. Whilst conceptually simple, a number of complicating factors arise because of problems of addressing. It is important to retain the integrity of addresses when an object is moved [Cahill91].

Distributed Virtual Memory This final technique extends the concept of virtual memory into a distributed environment [Tavanian87, Johnston89]. The heart of the technique is the creation of a virtual address space spanning a number of nodes in a distributed system. When accessing an object, a check is made to see if the object resides in the local the objectives of object-oriented systems are equally valid.

The authors believe that, in order to capture the essence of object-oriented computing, it is necessary to consider the properties that an object-oriented system should possess rather than the precise mechanisms used to realise object-oriented systems. This issue is discussed in more depth in [Blair91b]. In particular, this paper proposes a set of properties that object- oriented systems should possess. The semantics of object-oriented computing is an ongoing debate [Stein89]. However, the benefits of reaching agreement on a wider view of object-oriented computing are clear. Such a step would enable the object-oriented community to respond to questions such as:-

i) what are the fundamental benefits of object-oriented computing in various fields of application?

ii) what are the most appropriate mechanisms to realise these benefits in the different application areas?

3.2 Implementation Considerations

There are also some major implementation difficulties in developing distributed object-oriented computing. As we shall see, many of these problems are concerned with the implementation of a distributed invocation mechanism. This issue is discussed in depth below.

3.2.1 The Problems of Distributed Invocation

As described in section 3.1.1, the term invocation is used to describe the calling of a method on an object. In non-distributed systems, this is actually a relatively straight-forward task and is often described conceptually as a simple message passing operation to the object.

At first sight, it might appear that a similar approach would be suitable for distributed invocation as the message passing model is at the heart of distributed systems. However, it is now recognised that invocation in a distributed system is a much more difficult problem. The added complexity stems from the wish to realise a level of distribution transparency, hiding the problems of a distributed implementation.

In more detail, it is now recognised [APM89] that distribution transparency can be broken down into a number of individual transparency issues.

Transparency	Central Issue	Result of Transparency
Location	The location of an object in a distributed environment	User unaware of the location of services
Access	The method of access to objects in a distributed system	All objects are accessed in the same way
Migration	The re-location of an object in a distributed environment	Objects may move without the user being aware
Concurrency	Shared access to objects in a distributed environment	Users do not have to deal with problems of concurrent access
Replication	Maintaining copies of an object in a dustributed environment	System deals with the consistency of copies
Failure	Partial failure in a distributed environment	Problems of failure are are masked from the user

have emerged [Turner85, Burstall80]. This work has now been applied in object-oriented languages with interesting results [America87, Schaffert86, Halbert87]. These techniques have found particular application in distributed object-oriented languages. The concept of subtyping[3] [Bruce86] has emerged along with abstract data types as the fundamental building blocks of distributed object- oriented systems [Black87, Decouchant88].

The Role of Inheritance Inheritance has often been considered to be the principal feature of object-oriented systems. However, distributed object-oriented languages have often avoided the concept of inheritance in their design. The lack of interest in inheritance can often be traced back to the difficulties of implementing inheritance in a distributed environment. A number of projects have experimented with distributed inheritance; however, in general the overheads of searching class hierarchies at run time have proved to be too costly. Indeed, this problem has led Wegner to state 'inheritance is incompatible with distribution' [Wegner87]. It should be pointed out however that, strictly speaking, the problem is not inheritance but is dynamic binding. Inheritance would not be a major problem if method bindings were determined statically. However, there has been little enthusiasm for static forms of inheritance in the distributed community.

Several distributed object-oriented systems have replaced the concept of inheritance with subtyping. This led to a debate on the relative merits of the two approaches. Initially, there was great confusion between the of subtype and inheritance (or subclassing). However, this has largely been resolved. It is now appreciated that both are manifestations of behaviour sharing: subtyping is concerned with the sharing of abstract behaviour in the form of interfaces whereas subclassing extends this to the actual sharing of implementations. In the words of Black [Black87], 'In Smalltalk, a subclass does not necessarily conform to its superclass; for example, it may override some of the operations of the superclass so that they expect different classes of argument. Moreover, one class may conform to another without a subclass relationship existing between them. What a subclass and its superclass do have in common is part of their representation and some of their methods. In short, inheritance is a relationship between implementations, while conformance [subtyping] is a relationship between interfaces'. Canning et al extend this analysis further to incorporate the style of behaviour sharing found in delegation based systems [Canning89]. In general, distributed systems designers have been attracted much more to the style of behaviour sharing supported by subtyping than found in inheritance.

The problems of distributed inheritance have motivated a number of researchers to consider alternative ways of achieving code re-use. It is now recognised that inheritance is only one way of composing objects from existing object implementations. A number of alternative schemes more suited to distribution have thus been developed including delegation [Lieberman85] and hierarchical composition [Magee89].

3.1.3 Towards a General Framework for Objects

The developments in the distributed object-oriented community reflect a wider trend. As object-oriented techniques are introduced in more and more areas, there is a tendency to introduce new mechanisms more suitable for the particular fields of applications. This can be seen in distributed computing with the emphasis on mechanisms based on abstract data typing and subtyping. The same trend can also be witnessed in, for example, the AI community with the interest in more general forms of semantic relationships [Brachman83]. This development however challenges the very basis of object-oriented computing. In particular, it becomes very difficult to define precisely what is meant by object-oriented computing. It is clearly no longer possible to equate object-orientation directly with the concepts of object, class and inheritance. Other approaches to meeting

[3]Intuitively, an abstract data type A1 is a subtype of abstract data type A2, iff A1 provides at least the behaviour of A2, i.e. it supports at least the operations of A2.

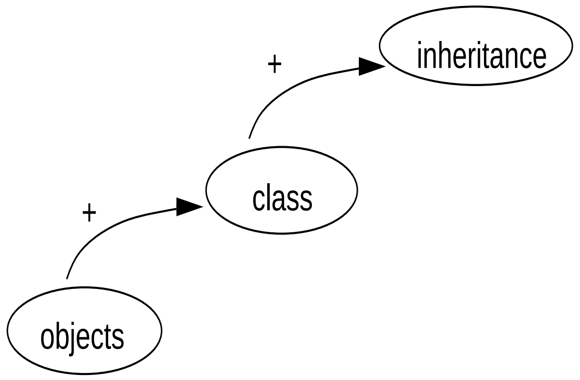


Figure 1: Traditional view of object-orientation

This view of object-oriented computing, whilst not universally accepted [Gallagher91, Lieberman85], has provided the basis for a large number of object-oriented systems and languages. Hence, the model will be used as the basis for the subsequent analysis.

3.1.2 Impact of Distribution on the Object Model

As mentioned above, there is now a strong body of research in the field of distributed object-oriented computing. Experience has shown that existing mechanisms are often inappropriate for distributed computing and hence new techniques have emerged. Thus the work on distributed object-oriented systems is challenging the very principles of object-oriented computing. The main areas of challenge are discussed below.

Typing and Type Checking Traditionally, object-oriented languages do not provide static type checking. Rather, they produce exceptions at run-time if inappropriate methods are invoked on objects. This feature of object-oriented systems has proved to be undesirable for distributed systems developers. There are a number of reasons for this concern. For example, distributed systems introduce a number of additional problems for application programmers to deal with (e.g. partial system failure, the location of objects). The trend has been to mask out such problems (distribution transparency). Run-time type errors go against this trend. In addition, many distributed applications have stringent requirements for correctness (e.g. process control). For such applications, run-time errors are clearly unacceptable. This concern about type correctness has stimulated a body of work on type systems for object-oriented computing [Danforth88][2]. This work is generally based around the notion of an abstract data type. In particular, the external behaviour of an object is defined by an abstract data type signature denoting the available methods and their associated parameters and results. This research has also highlighted the important distinction between type and class. The type of an object is a definition of its external behaviour whereas the class of an object describes the external behaviour plus its implementation.

To retain the inherent flexibility of object-oriented systems, type systems for object-oriented computing are generally polymorphic. Polymorphism [Cardelli85] is defined as the ability of an object to have more than one type, for example a polymorphic stack could describe a stack of integers or a stack of reals. The topic has been the subject of intense research for a number of years and a number of polymorphic type systems

[2] Note that type checking does not provide absolute guarantees of run-time behaviour of a system. Run-time errors may still occur if, for example, an object is not available at a particular time. Type checking however guarantees that there will be no run-time errors due to typing errors in the distributed program.

a fundamental aspect of the computational model. This evolutionary approach to programming is unique to object-oriented computing [Cox86].

Data abstraction is of obvious benefit to the distributed systems community. The loose coupling and tight cohesion implied by data abstraction is precisely what is required in a distributed system. Similarly, the benefits of behaviour sharing provide exactly the flexibility required in a distributed system. Finally, evolution has long been a major concern in distributed computing. It is recognised that a distributed system must be capable of changing its functionality in terms of the introduction of new components, partial system failure or new software requirements.

In general, however, the object model is naive in terms of distributed systems. The problems of moving from a single workstation environment to the general case should not be under- estimated. The following section examines the impact of distribution in some depth.

3 IMPACT OF DISTRIBUTION

A number of projects have addressed the problems of distribution in object-oriented computing, e.g. Emerald [Black86], Comandos [Horn87], Arjuna [Shrivastava87] and Clouds [Bernabeu88]. There is therefore a level of awareness of the problems raised by this class of object-oriented system. A large number of problems have been identified. We categorise the issues under the following headings:- i)conceptual issues, and ii) implementation issues. These two areas are discussed in depth below.

3.1 Conceptual Issues

This section discusses the impact of distribution on the conceptual understanding of the object model. It is argued that distribution demands a wider perspective on what is meant by object-oriented computing. Firstly, however, it is necessary to consider what is meant by the term object-oriented computing.

3.1.1 The Object Model

It is notoriously difficult to give precise definitions of terminology in this field. However, a good starting point is provided by a paper by Wegner [Wegner87]. This describes object-oriented systems as containing three important components, namely objects, class and inheritance (see figure 1).

An object is defined as an encapsulation of a set of methods with state that remembers the effect of the methods (hence implementing data abstraction). This is then extended by the concept of a class, i.e. a template from which objects can be created. Classes therefore define the full behaviour of objects in terms of a list of methods and their implementations. Finally, a form of behaviour sharing is supported through an inheritance mechanism. With inheritance, a class can inherit the behaviour of another class but then alter or extend the behaviour to meet new requirements. In this way, a class hierarchy can be formed of classes and associated subclasses. Note that this has the repercussion that particular methods defined on objects can belong to its class or one of the superclasses in the hierarchy. Hence, it is necessary to search the class hierarchy when a method is invoked. This search is normally carried out at run- time and is referred to as dynamic binding[1].

[1]Further details of the object model is beyond the scope of this paper; the interested reader is referred to the literature more detailed descriptions [Blair91a].

This trend can be seen through the interest in object-oriented techniques in such diverse areas as artificial intelligence, databases, real-time systems, and operating system design. This paper focuses on the interest in object-oriented techniques in one particular community, namely distributed systems. The topic of distributed systems is now established as a fundamental concern in the computing community. An increasing number of organisations are recognising the benefits that can be gained from distributed computing and hence are opting for distributed environments.

The area of distributed computing has attracted considerable attention over the past decade. The subject is therefore reasonably mature. In particular, there is now great experience in the techniques required to build distributed systems. This maturity is evident by the increasing number of products now available [Accetta86, Herrmann88]. However, there is still some concern in the distributed community on the abstractions provided by existing systems. In particular, attention has focussed on provided more support for the construction of distributed applications. Distributed applications tend to be large and complex and hence it is vitally important to provide good tools to aid the software development process. This concern has resulted in an increasing interest in the perceived benefits of the object-oriented approach.

Distribution poses some difficult problems for object-oriented computing. For example, issues such as the location of objects and the possibility of partial failure become important concerns. This paper considers the impact of distribution on the object- oriented paradigm. The paper is structured as follows. Section 2 discusses the fundamental reasons for the success of object- oriented computing and suggests the same benefits can translate to the distributed systems community. This is followed, in section 3, with an in depth examination of the issues raised by distribution. This section considers the impact on the object model as well as implementation techniques required to realise the object-oriented approach in a distributed environment. Section 4 highlights the importance of underlying system support for the future of distributed object-oriented systems and provides a case study of an operating system designed to support distributed object-oriented computing. Finally, some concluding remarks are presented in section 5.

2 THE OBJECT-ORIENTED APPROACH TO SOFTWARE ENGINEERING

Many claims have been made about the benefits of the object- oriented approach, particularly for large and complex applications. Most of these claims can be traced to the particular approach to abstraction found in object-oriented computing. Abstraction has historically been an important tool in dealing with complexity. Similarly, in computing, a great deal of attention has been directed towards finding the right abstractions to aid problem solving. In our opinion, the object- oriented approach to abstraction is captured by the following fundamental principles [Blair91a]:-

- Data Abstraction The essence of data abstraction is that the programmer is presented with a higher level of abstraction over both the data and the algorithms required to manipulate the data. The user view is that of a number of operations which collectively define the behaviour of the abstraction; the user is not necessarily aware of the internal details of implementation.
- Behaviour Sharing Behaviour sharing is concerned with allowing items of behaviour to span more than one object or grouping of objects. For example, it is advantageous for generic behaviour such as print operations to have wide interpretation. It is now recognised that behaviour sharing can increase the flexibility of systems.
- Evolution Over a period of time, systems change and additional functionality may be required. The object-oriented approach is to support this process of evolution as

THE IMPACT OF DISTRIBUTION ON THE OBJECT-ORIENTED APPROACH TO SOFTWARE DEVELOPMENT

Gordon S. Blair* University of Lancaster
Rodger Lea† Chorus systèmes

May 2, 1992

Abstract

Object-oriented computing is now an established technology for software development. However, a number of challenges must be met before the topic can claim to be fully mature. One of the most demanding challenges is posed by the move from single workstation environments to the more general case of a distributed system. There is now considerable interest in the distributed systems community in object-oriented techniques; it is recognised that the features of object-oriented computing are highly appropriate for developing complex distributed applications. This paper considers the importance of object-oriented techniques to distributed system designers and discusses the repercussions of such a move to distributed environments. The impact of distribution on object-oriented computing is examined in depth. This analysis raises a number of important issues such as the need for a broader view of object-oriented computing and the requirement for flexible approaches to invocation. Problems relating to persistence and distribution transparency are also discussed. The importance of underlying system support for distributed object- oriented computing is stressed and a case study of the Chorus/ COOL environment presented. The paper concludes by assessing the state of the art in distributed object-oriented computing. It is argued that further research is required before the advantages of the object-oriented approach to software development can be realised in a distributed environment.

Published in the IEE/BCS Software Engineering Journal *special issue on Object oriented systems, March 1992, Vol. 7, Number 2.*

1 Introduction

Object-oriented computing has emerged in recent years as a major technique in the field of software engineering. This is reflected in the increasing interest in object-oriented languages such as Smalltalk [Goldberg83], C++ [Stroustrup86], Objective-C [Cox86], and Eiffel [Meyer88]. Similarly, object-oriented design [Ormsby91] and, to a lesser extent, object-oriented specification [Cusack90] are becoming influential techniques. As the techniques gain wider exposure, they are applied to an increasing number of application areas.

*Distributed Multimedia Research Group, Computing Department, Lancaster University, Bailrigg, Lancaster LA1 4YR, U.K. (e-mail: gordon@comp.lancs.ac.uk)
†Chorus Systemes, 6 Avenue Gustave Eiffel, St Quentin-en-Yvelines, CEDEX, France (email: rodger@chorus.com)