

# Enhancing Performance in a Persistent Object Store: Clustering Strategies in O<sub>2</sub>

Véronique Benzaken

Claude Delobel

Altair

BP105

78153 Le Chesnay Cedex

E-mail [vero@bdblues.inria.fr](mailto:vero@bdblues.inria.fr) [claudio@bdblues.inria.fr](mailto:claudio@bdblues.inria.fr)

April 26, 1995

## Abstract

We address the problem of clustering complex data on disk to minimize the number of I/O operations in data intensive applications. We first focus on the problems related to the design and implementation of clustering strategies. We then propose a set of clustering strategies as well as an algorithm which implements them for the O<sub>2</sub> system.

## 1 Introduction

New developments, both in the database field and in the programming languages field, have led to the design of new database management systems [Ba88], [Ki89], [Deux90]. These systems have the following characteristics: a complex object model [LR89a], a persistent programming language [AB87], and an object management system [VBD89]. Object management systems have to fulfill the following requirements: (i) efficient management of large amount of (large) objects; (ii) object sharing and versioning; (iii) and usual database functionality such as transaction management, concurrency control and recovery. In this paper, we are interested in how clustering can improve performance of such systems. Consider, for example, a complex object  $o_1$  consisting of two components  $o_2$  and  $o_3$ . If each component object is stored in a different page, we may have to perform up to three disk accesses when we want to retrieve the entire object! A clustering strategy

might suggest how to place these objects as close to one another as possible on disk memory.

We shall describe the clustering strategies adopted in the O<sub>2</sub> system. We propose a set of flexible strategies which rely upon the inheritance hierarchy as well as on the structural information given by the object types. Our strategies are flexible in the sense that we will not try to cluster all the components of an object together; instead, we try to define grouping strategies which take into account the operations performed on the database (i.e., the methods).

We first list the problems which arise when designing clustering strategies. We then describe the strategies that have been implemented in the O<sub>2</sub> system and we show how methods are taken into account and informally give a cost model which allows us to derive an optimal clustering scheme. We detail the implementation of the clustering module and give a set of early measurements which have been performed in the O<sub>2</sub> system in order to evaluate the impact of clustering. Finally, we give some concluding remarks.

## 2 Issues in the Design of a Clustering Algorithm

The objective of a clustering algorithm is to group objects which are co-referenced as close to one another as possible in physical memory. We list here after the critical problems one must answer when designing a clustering algorithm:

- What are the tradeoffs between static versus dynamic clustering?
- What is the input to the clustering algorithm?
- What are the different kinds of clustering algorithms?
- What are the relationships between clustering and architecture configuration?

### 2.1 Static versus Dynamic Clustering.

In the *static* case, clustering is done at the time objects are created and no reorganization is implied when the links between objects are updated. *Dynamic* clustering is done at run time when objects are accessed concurrently and becomes attractive in an environment where the write operations

dominate the read operations. Here the cost of reorganization is identical to the cost of clustering. In a dynamic environment, when the objects are transferred from the disk to the main memory their physical page locations are not necessarily the same after a write operation. In object-oriented database systems object identity and referential integrity to objects are part of the model. If object identity is implemented as a logical reference then a dynamic clustering algorithm is probably better suited.

## 2.2 Input to a Clustering Algorithm

The inputs to a clustering algorithm are usually access patterns of objects, given by the user, providing information about the frequency of accesses from an object to another. The higher the frequency the more effective the grouping has to be. This information can be stored at two different levels: object level or type level. At the object level, the access frequency is stored together with the reference “pointer”. The overhead of this solution is the cost of updating the access frequency every time we navigate through a link between two objects. If recorded at the type level (and therefore recorded within the schema), the interactions between objects are derived from the type of the objects at creation time.

## 2.3 Clustering Algorithms

The problem of clustering can be seen as a graph partitioning problem. The nodes of the graph are the objects, and the edges are the links between objects. This problem is NP-complete. However, as the graph of objects represents the database state, all is needed is an incremental solution where new objects are placed at the “right place”. Most of the algorithms used can be classified as greedy algorithms: they scan the objects according to their links, and try to place them into the same cluster unit. Thus the cost of clustering has no major impact on the overall system. What are the parameters which are used to control a clustering algorithm? We enumerate below the most important steps that must be considered.

- Obviously the input information is part of the control and determines a possible cluster of objects.
- Once a cluster has been defined, the algorithm must look for a candidate page to insert this new cluster. Different strategies can be used to find the page : the cluster algorithm may use only the pages available

in the buffer pool, thus avoiding any I/O; the algorithm may search for a page on the disk; alternatively, it may open a new page if it considers that there is no good candidate. If it also splits pages after reorganization, the cost of splitting has to be lower than the cost of searching for the best candidate page. This previous step assumes that the search for a page does not consider the available space into the page.

## 2.4 Clustering and Architecture Configuration

A classification of object servers has been proposed in [DFMV90]. Two basic architectures are retained: the *smart* server architecture and the *dumb* server architecture. In the smart architecture the server understands all the concepts of the object-oriented approach and is decomposed in three layers: I/O, buffer and object layer. The transfer unit between the server and the workstation is the object and thus the effect of clustering is visible only at the server and not on the workstation. Furthermore, clustering can be performed only at the server since it is the only place where the concept of page is understood. Consequently, when a transaction executing on the workstation commits, all the persistent objects have to be returned to the server before applying the clustering algorithm. In the dumb architecture the server is only composed of two layers: I/O and buffer. The concept of object is known to the workstation only. The transfer unit between the server and the workstation is the page and thus if clustering has worked properly, great part of the objects contained within the transferred page will be used without any additional transfers. As the page concept is understood at the workstation, the clustering decisions can be taken at this site.

## 2.5 Options Taken in the O<sub>2</sub> System

The choice made in O<sub>2</sub> to use physical object identifiers was largely motivated by performance reasons; therefore the O<sub>2</sub> clustering strategies we present in this paper are *static*. Nevertheless, we shall show how our solution may be extended in order to dynamically reorganize clusters. These strategies are defined at the *type level* by the database administrator and could be applied in the context of logical identifiers as well. They are independent, from a logical point of view, of the O<sub>2</sub> system: they could be used in any system managing complex objects with inheritance. The *clustering algorithm* which implements them is a greedy tree-pattern-matching algorithm (with no page splitting). Last, our strategies have been implemented

in the context of a *smart* architecture.

### 3 The O<sub>2</sub> Clustering Strategies

The clustering strategies designed in the O<sub>2</sub> system rely on the concept of *placement trees*. The following section describes these trees.

#### 3.1 Placement Trees

Intuitively, the type structure of a class in the O<sub>2</sub> system may be represented as an infinite tree (in case of recursive types) [AK89]. A placement tree is any finite subtree extracted from this infinite tree. A *clustering strategy* for the system classes is a set of placement trees. Given a clustering strategy, each class in the system is the root of at most one placement tree. We have chosen a finite tree for operational reasons. A placement tree for class *c* expresses the way in which components of an instance of *c* are to be clustered. Notice that the definition of a placement tree allows us to handle type recursiveness at the physical level. Figure 1 shows examples of clustering strategies.

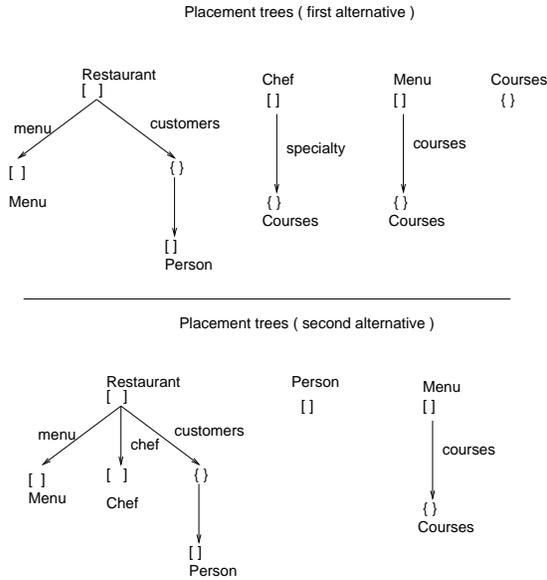


Figure 1: Placement Trees

In the first alternative for the Restaurant class, the placement tree states that `menu` (which is an instance of class `Menu`), the type `customers` and each

of the “customers” elements will be grouped with the corresponding instance of Restaurant. This means that, in a single cluster, we shall store the “menu” object, its atomic components, the value `customers`, the elements of the corresponding set and the atomic values which compose them. An instance of Courses may be shared by a Chef or a Menu. Therefore, a Course object may either be clustered with a Chef or with a Menu. Unlike the Orion system [KBC87], such a decision is only taken at run time and is not part of the database schema. The placement tree for the class Person for the second alternative reduces to a single node. Indeed, instances of this class are only composed of atoms and we assume that atomic values are stored with the instance to which they are related. Placement trees are the inputs of the clustering algorithm. They only indicate how objects may be stored, they are not part of the schema (there is neither a class placement tree nor instances of placement trees). The definition of placement trees is orthogonal to the schema, thus data independence is insured by such strategies.

Objects are placed in logical clusters using placement trees associated with the class of which they are instances and according to their identifiers (disk or memory). A logical cluster is composed of a root object and a set of objects which are grouped with it. We assume that the size of a logical cluster is unlimited. The algorithm presented here is a greedy tree pattern-matching algorithm. There exist different alternatives for this algorithm based on either depth first or breadth first traversals of placement trees.

## 4 Implementation

### 4.1 Global Architecture of the System

The setting of the  $O_2$  system [VBD89] consists of a server and a network of workstations. On the server, the object manager is built on top of WiSS [WiSS85]. WiSS provides basic support for persistency, memory management and transactions as such it provides record-structured sequential files, unstructured files, indices and long data items (LDI's). Records may vary in length and are uniquely identified by a RID (Record Identifier) which describes its physical address (i.e a volume number, a page number and a slot number). WiSS directly controls the physical location of pages on disk and does its own buffering. The clustering algorithm executes at the server level while the definition of placement trees is performed at the workstation level. For the sake of brevity, we do not detail the definition language of placement trees, its syntax is very similar to those of  $O_2$  class definitions

[LR89b].

## 4.2 Object Identifiers Allocation

In the  $O_2$  system, objects are uniquely identified and accessed by object identifiers (oid's). For performance purposes, oid's are physical and correspond to a RID. With physical identifiers and a smart workstation/server configuration, a decision has to be taken upon when to assign oid's to newly created objects. Objects are created in the workstation and persistent physical identifiers containing RID's are only delivered on insertion of the corresponding WiSS record. Identifiers are assigned at transaction commit. In addition, this solves the problem of having to change the identifier of a newly created object when the system decides to cluster it afterwards (but before transaction commit) with another "owner" object. The clustering algorithm is responsible for the assignment of physical identifiers, and thus is viewed as a persistent identifiers server.

## 4.3 Logical Clusters and Physical Clusters

In the  $O_2$  system, objects are mapped to records. Therefore, a physical cluster is a set of records. WiSS offers the possibility to perform clustering, since a record can be stored near another one (there is a WiSS primitive to do it). WiSS then tries to store the record in the same or in a neighbor page. We have chosen to map a physical cluster to a page (i.e, a page may contain several clusters). The size of logical clusters was considered unlimited. Of course, the size of a physical cluster is bounded to the page size. Therefore there will be some clusters (larger than a page) which will overlap several (contiguous) pages. Large objects (i.e, with large atomic values) will be stored as WiSS LDI's.

## 4.4 Initial Placement and System Adaptability

Initially, when instances of a given class are created, the placement tree associated with this class generates a first organization on secondary memory. The problem of system adaptability arises in the two following cases:

- Some placement trees have been modified.
- No placement trees are modified but some links between objects have changed.

#### 4.4.1 Placement Tree Modifications

At any time the DBA has the ability to modify placement trees. Neither the clustering module nor the access object module will be affected by such modifications since oid's are physical identifiers. The clustering algorithm will take into account the new grouping scheme. The execution of transactions will not be disturbed. Indeed, only the physical placement of objects is modified (for the newly created objects). This certainly will have some consequences on the performance but not on the behavior of transactions. On the other hand, the physical organization on disk will be modified. The following alternatives thus arise.

- We do not update the old clusters. New placement trees will be taken into account in the future. The objects which have already been placed keep their location on disk.
- We can restructure the database.

For the second alternative a complete study should be performed in order to evaluate at which time we consider that the physical organization is too chaotic. The number of modified placement trees, the number of classes concerned in such modifications might be parameters to be taken into account. In addition to this, for each class whose placement tree has been modified, we can measure the number of instances which have been grouped according to the first PT and to the second one. If the ratio is lower than a given level we can restructure the whole extension of the class. This is an open problem which has not been implemented.

#### 4.4.2 Cluster Reorganization

Different approaches for record clustering and reorganization have been proposed [YLSS85], [SPO89]. Record clustering algorithms usually focus on finding an optimal strategy by rearranging records on pages in the buffer area and writing new pages back to the disk space. Evidently, such approaches are applicable in the case where oid's are logical. We suggest the following solutions to solve the problem of system adaptability in the context of physical oid's. The following cases are taken into account:

- the size of an object increases within a page;
- some logical links are modified.

The first case is automatically managed by WiSS. Indeed if the size of an object increases in such a way that it cannot fit into the page any longer, WiSS allocates a new page and the forwarding pointer allows the system to retrieve the (moved) object.

The second case occurs when among the links which reference an object ( $o$ ), one link is deleted. We adopt the following strategy: when an object which references  $o$  is accessed, thereafter, if there is enough space, we shall re-cluster  $o$  with it (if the corresponding PT allows it). We thus assume that we have full control of the forwarding mechanism of WiSS.

For both alternatives the problem of updating the references arises. As we do not want to manage backward pointers, we shall update, incrementally, only the references for future accesses. Indeed it is possible, at this time, to detect the forwarding pointer and thus to update the new reference. However we need to know when the forwarding pointer could be garbaged. We propose the following solution. First, recall that a reference count of an object is stored in the header of the record containing the object. Assume that the objects have been grouped as in Figure 2. The links ( $o_1, o_4$ ) and

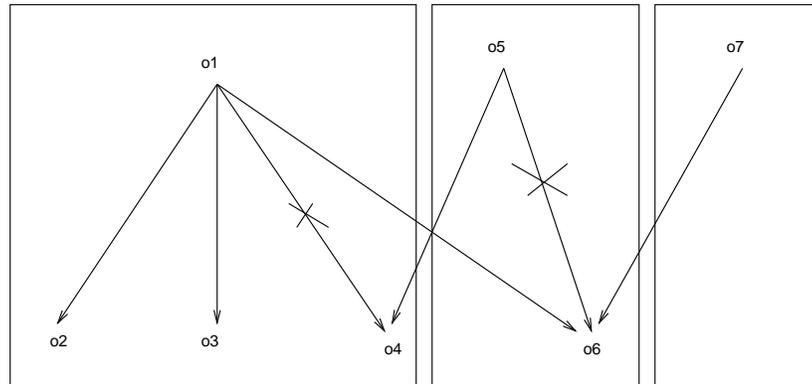


Figure 2: Cluster Reorganization

( $o_5, o_6$ ) are privileged because they correspond to shared objects which have been grouped. Each time such a link is deleted the corresponding object is not at the right place anymore. We are faced with two alternatives:

- 1 the reference count is equal to one (it is the case for  $o_4$ ). When we dereference  $o_4$  through  $o_5$ , if there is enough space in the page of  $o_5$  we re-cluster  $o_4$  with  $o_5$ . If not nothing is done.

2 the reference count is greater than two (it is the case for  $o_6$ ). When we dereference  $o_6$  (through  $o_1$  for example) we shall try to store  $o_6$  close to  $o_1$ . We then replace the (old) record of  $o_6$  with a special forwarding record which contains: the new address of  $o_6$ , a dereferencing count initialized to one (this count will be incremented at each new dereferencing) and the value of the reference count of  $o_6$ . As soon as the dereferencing count is equal to the reference count, the forwarding record will be deleted.

#### 4.5 Performance Evaluation and Measurements

The problem of choosing, for a given class, the “best” placement tree arises. We thus have to take into account the most frequently performed operations against the database. We have detailed a cost function [Be90] which takes into account object faults and main memory waste according to the call frequencies of the methods. It also takes into account set-structured components of an object by assigning an average cardinality to each set and an average number of accessed elements (in the set) per method. This cost function is fully described in [Be90]. With respect to this cost function, one placement tree is optimal. Though our cost model relies on some simplifying assumptions, we have a means to help the database administrator in choosing placement trees.

#### 4.6 Early Measurements

The class we have studied is described in Figure 3 and corresponds to the class `Restaurant` slightly modified. The methods defined for `Restaurant` are characterized as follows: method  $m_1$  accesses the `menu` component and the `chef` component of class `Restaurant`,  $m_2$  accesses the component `menu` then the component `course` and the component `chef` of class `Restaurant`,  $m_3$  accesses the `menu` and `course` components and  $m_4$  accesses the `chef` and `specialty` components of class `Restaurant`. The four methods have a call frequency of 25%.

Our experiments took into consideration the variation of the following parameters:

- The placement trees for the class `Restaurant`. Table 2 gives, for each placement tree, the classes it groups.
- The sizes of objects. Table 1 gives the different configurations (A, B,

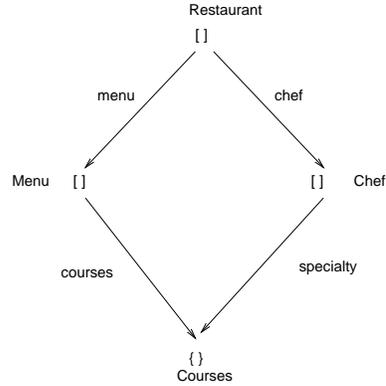


Figure 3: Class Restaurant

C) of measurements. The values reported denote the sizes (in bytes) of the instances of each class. The size of instances of Courses varies from 10 integers to 1000 integers in the set.

- The number of generated objects which varied from 1000 to 10.000.

Configurations	Restaurant	Menu	Chef	Courses
A	42	34	34	128
B	42	34	34	848
C	42	34	34	4048

Table 1: **Configurations of measurements**

The scenario of each experiment is the following. We generated a database according to each placement tree for Restaurant. Each time a database was generated, we executed each method on the whole extension of Restaurant and measured the object faults. We have chosen to penalize an object fault twice as much as main memory waste. The best placement tree according to the cost function is described in Figure 4.

The results for the optimal placement tree are reported in the fourth column of Table 3. The results obtained are the following. In four out of the five sequences, the optimal tree was effectively the best tree (the one which led to fewer object faults). However, the third sequence (configuration C) led to other placement trees as best trees. For this sequence, the simplifying

	Restaurant	Menu	Chef	Courses
1	•	•	•	•
2	•	•		•
3	•		•	•
4	•	•	•	•
5	•	•	•	•
6	•	•	•	
7	•	•		
8	•		•	

Table 2: PT's for Restaurant

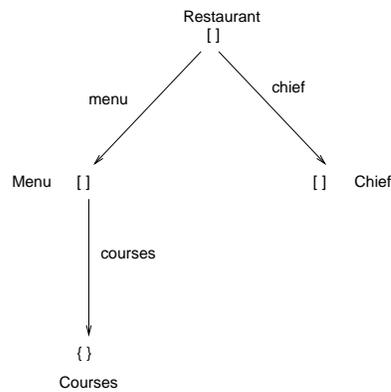


Figure 4: Optimal Placement Tree

		1	2	3	4	5	6	7	8	WC
1000	A	32	27	32	<b>26</b>	32	32	32	32	32
	B	112	65	114	<b>62</b>	112	118	118	118	118
	C	412	414	414	412	412	<i>219</i>	<i>219</i>	<i>219</i>	<i>218</i>
10.000	A	212	<i>145</i>	210	<b>145</b>	212	211	211	211	211
	B	1012	533	1032	<b>512</b>	1012	1078	1078	1078	1078

Table 3: Impact of PT's on the transaction (without sharing)

assumptions of the model were violated. Indeed, each cluster (for the tree n° 4) overlapped three or two pages. Notice that an implementation without any clustering leads in this case to the best results. For the trees n° 6, 7, 8, the number of clusters overlapping several pages was very low (4 for tree n° 6 and zero for the other two). The size of logical clusters is thus a crucial parameter with respect to the benefits of clustering.

However, we achieved our objective to provide the DBA with a tool to help the choice of the most adequate placement tree. When a cluster fits into a page, the model predicts the placement tree which leads to fewer object faults. When a placement tree leads to clusters whose average size is greater than the page size, such a tree must be rejected. The point is now to find a means in order to select the best placement tree among the remaining ones. We thus shall design, in future work, a cost model which will take into account the sizes of objects.

While the evaluation performed is sound, it could have been more thorough. Further measurements are currently performed in order to evaluate the benefits of clustering [BDH90]. We use a standard object-oriented benchmark (the Tektronix Hypermodel Benchmark) as the basis of the evaluation.

## 5 Related Work and Conclusions

Most of the prototypes already developed [ZH87], [KBC87], [At85], [MSOP86], [HK89], attempt to perform clustering. In all such systems the segment or the page is the clustering unit and clustering is done when the object is created. The approach adopted in Orion [KBC87] defines the unit of clustering as the composite (exclusive/dependent) object. This is done at the schema level. We think instead that the user should not need to worry about physical issues: the clustering information has to be transparent from an user point of view. Furthermore, if the concept of cluster (via the notion of composite object) is part of the schema, when we want to modify the placement in secondary storage we have to perform a schema update! Data independence is not insured anymore.

In [CK89], it is shown that clustering exploits structural information such as composition hierarchy and inheritance properties. A simulation model has been developed which gives a comparative evaluation of different clustering strategies. The clustering algorithm studied in [CK89] chooses an initial placement for newly created objects which depends on the most frequently used composition links between instances.

The grouping strategies presented in the Encore system [ZH87] seem flexible and powerful (in particular the clustering-by-value feature). In Encore, the following clustering rules may be chosen: (i) ability to store one object per segment if it is large, (ii) ability to store components of a complex object together, (iii) ability to group all the instances of a given class in the same segment, (iv) grouping by value properties (e.g. all blue cars will be stored together). The power of these strategies relies on the fact that, in Encore, objects are shared by means of copies. However, in an environment where update operations are frequently performed, managing such copies seems penalizing with respect to performance.

The approach adopted in the PRIMA project [HMMS87] uses the concept of *type-molecule* as a clustering mechanism for complex objects [SS89]. This concept allows the description of flexible clustering strategies. The main drawback of this approach is that some objects have to be duplicated between several complex objects and that clustering is update dependent.

Design decisions about the clustering are under the control of the Database Administrator. Nevertheless, to help the Database Administrator in taking his decision we have established a link between the access patterns of the methods of a class and the “optimal” placement tree for this class. Our technique is similar of the one proposed by [Sc77] for the hierarchical data model. The idea of Schkolnick is to partition a hierarchical tree structure into subtrees and to find the best partition according to some access patterns inside the hierarchical structure. Our strategy follows the same idea, but differs in the following: (i) the data model is not the same (objects can be shared, inheritance relationship), (ii) clustering is done at commit time as well as persistency, (iii) clustering is transparent to the programmer, thus placement trees can be modified at any time without affecting programs.

Placement trees provide a mechanism for the database administrator to specify exactly how the instances of the classes of a database’s class hierarchy are to be clustered on disk. By separating the clustering mechanism from the schema, the database administrator is provided a significantly more powerful mechanism than if clustering was associated with one particular type constructor. In addition, this separation makes it possible for the database administrator to tune the overall performance of the system by modifying the placement trees used for a particular database. Since this can be accomplished without modifying the schema or any applications, data independence is insured.

## References

- [AB87] M. Atkinson, P. Buneman, "Types and Persistence in Database Programming Languages." *ACM Computing Surveys*, June 1987.
- [AK89] S. Abiteboul, P. Kannelakis, "Object Identity as a Query Language Primitive" *In Proc Int Conf ACM Sigmod* Portland 1989.
- [At85] T. Atwood, "An Object Oriented DBMS for Design Support Applications" *in proc IEEE COMPINT* Montreal 85.
- [Ba88] F. Bancilhon, "Object-Oriented Database Systems" *Proceedings of the ACM PODS conference*, pp 152-162, 1988.
- [BDH90] V. Benzaken, C. Delobel, G. Harrus, "Measuring Performance of Clustering Strategies: the CluB-0 Benchmark" *Altair internal report* 90.
- [Be90] V. Benzaken, "An Evaluation Model of Clustering Strategies in the O<sub>2</sub> Object-Oriented Database System" *to appear in proc. of the 3rd ICDT conf.* Paris, December 1990.
- [CK89] E. E. Chang, R. H. Katz, "Exploiting Inheritance and Structure Semantics for Effective Clustering and Buffering in an Object Oriented DBMS" *in proc ACM SIGMOD conference* Portland 1989.
- [Deux90] O. Deux et al., "The Story of O<sub>2</sub>, *IEEE Transactions on Data and Knowledge Engineering*, March 90.
- [DFMV90] D. DeWitt, P. Fattersack, D. Maier, F. Velez, "A Study of Three Alternatives Workstation/Server Architectures for Object-Oriented Database Systems" *to appear in the proc., of the VLDB'90 conf.* Brisbane, Australia, August 90.
- [GR83] A. Goldberg, D. Robson, *Smalltalk 80 : The language and its implementation*, Addison Wesley, 1983.
- [HK89] S. Hudson, R. King, "Cactis: A Self-Adaptative, Concurrent Implementation of an Object-Oriented Database Management System." *ACM TODS*, vol 14 n° 3 September 89.

- [HMMS87] T. Harder, K. Meyer-Wegner, K. Mitschang, A. Sikeler, "PRIMA: A DBMS prototype supporting engineering applications" *Proceedings of the VLDB'87 conference*, Brighton 87.
- [K\*87] J. Banerjee et al. "Data Model Issues for Object Oriented Applications" *ACM TOIS*, Vol 5 no 1, Jan 87.
- [KBC87] W. Kim, J. Banerjee, H. T. Chou. "Composite Object Support in an Object-Oriented Database System", *OOPSLA '87 Proceedings*.
- [KBG89] W. Kim, E. Bertino, J. Garza, "Composite Objects Revisited" in *Proc ACM Sigmod conf*, Portland 1989.
- [KCF87] S. Khoshafian, M. J. Carey, P. Franklin, "Storage Management for Persistent Complex Objects" *Private communication* 1987.
- [Ki89] W. Kim, "Object-Oriented Concepts, databases, and Applications", *ACM Press, Addison-Wesley*, 1989.
- [LR89a] C. Lécluse, P. Richard "Modeling Complex Structures in Object-Oriented Databases." *Proceedings of the ACM PODS conference*, Philadelphia 89.
- [LR89b] C. Lécluse, P. Richard "The  $O_2$  Database Programming Language" in *Proc., of the VLDB'89 conf.*, Amsterdam, August 89.
- [MSOP86] D. Maier, J. Stein, A. Otis, A. Purdy, "Development of an Object Oriented DBMS" *Technical Report CS/E-86-005* 1986.
- [Sc77] M. Schkolnick, "A Clustering Algorithm for Hierarchical Structures", *ACM Transactions on Database Systems*, Vol 2, n<sup>o</sup>. 1, March 77.
- [SPO89] P. Scheuermann, Y. C. Park, E. Omiecinski, "Heuristic reorganization of clustered files" *Proceedings of the 3rd FODO conference*, Paris 89.
- [SS89] H. Schoning, A. Sikeler, "Cluster mechanisms supporting the dynamic construction of complex objects" *Proceedings of the 3rd FODO conference*, Paris 89.

- [St84] J. Stamos, "Static Grouping of Small Objects to Enhance Performance of a Paged Virtual Memory", *ACM Transactions on Computer Systems*, Vol 2 no 2 May 84.
- [VBD89] F. Velez, G. Bernard, V. Darnis, "The  $O_2$  Object Manager : an Overview", *Proceedings of the VLDB 89 conference*, Amsterdam 1989.
- [WiSS85] H. T. Chou, D. J. Dewitt, R. H. Katz, A. C. Klug. Design and implementation of the wisconsin storage system. *Software - Practice and Experience*, 15(10), October 1985.
- [YLSS85] C. Yu, K. Lam, M. Siu, C. Suen, "Adaptative record clustering", *ACM TODS*, vol 10 no2, 1985.
- [ZH87] M. Hornick, S. Zdonik, "A Shared Segmented Memory System for an Object-Oriented Database", *ACM TOIS*, vol 5 no1 January 87.