Research at Altaïr[1]

Edited by Philippe Richard

GIP Altaïr
B. P. 105, 78153, Rocquencourt,
France
e-mail: philippe@bdblues.altair.fr

Altaïr is a five year project which began in September of 1986. Its goal is to design an implement a next generation database system. The five year project was divided in two phases: a three year prototyping phase and a two year phase devoted for one part to the development of a product from the prototype and for the other part to a new research effort.

The three year phase ended by the demonstration of the V1 prototype of the O_2 object-oriented database system which has been distributed for experimentation to more than 40 universities and about 13 industrial partners.

The contribution of the Altaïr group to the database research community has been mainly in three areas: data model and database languages, object stores, and programming environments. Furthermore, all these efforts have been integrated in a consistent way into the V1 prototype. The results of this research and development effort are being summarized in a book [BDK91] which is a commented collection of papers, most of them already published in the proceedings of a number of internationally recognized conferences. We briefly survey in the following the O_2 activities.

1 Overview

O_2 is an object-oriented database system which integrates the current database technology with an object-oriented programming paradigm. The O_2 data model merges both *objects* with identity and *values*. Values are manipulated through a set of predefined primitives, and value equality is structural. Objects encapsulate values, and are manipulated through user-defined *methods*. Object equality is identity. Values are built using the set, list and tuple constructors. This feature allows the application programmer to built complex nested data structures. Among our research results, we have clearly established the power of object identity for three critical purposes: *(i)* to represent data structures with sharing and cycles, *(ii)* to manipulate sets and *(iii)* to express database queries. Another important aspect of object-oriented languages is the use of methods for redefining the behavior of objects. In this context, the schema definition merges both structural and behavioral aspects. Thus, providing a consistent view of the schema is a crucial problem in case of schema updates. This problem has been largely studied through two aspects: formalization and use of incremental compilation techniques.

The requirements to design and to support data-intensive applications such as office information systems, engineering CAD/CAM systems and CASE systems impose the integration of a programming language into a database system. The solution taken by O_2 is to support two database programming languages, CO_2 and $BasicO_2$ (built from the C and Basic languages), and a query language for end users, rather than developing a new database programming language from scratch.

In order to efficiently implement the O_2 data model and languages, the system architecture consists of two layers. The first one is concerned with the graphical interface, the languages processor and the schema manager. The second one is the O_2 object manager which implements structural operations on objects, provides support for message passing, and performs associative access operations. The O_2 system has both a workstation and a server component. Problems that have been studied deal with system distribution between the server and the workstations, efficient access to data on disk, clustering of objects and versioning.

In a database application, the code dedicated to man/application communication represents up to 60% of the overall code. Thus, an important contribution of O_2 is its programming environment and its user interface generator. The O_2 programming environment supports a browser for viewing the database schema and objects and a debugger which offers functionalities directly attached to the object-oriented features of O_2. The O_2 user interface generator is a toolkit extension which allows to display and edit multi-media complex objects. It also mixes direct manipulation of data with query facilities.

Evaluation of the V1 prototype has also been one of our activities. This was done by developing and experimenting with real database applications. This was an important source of feedback for improving the functionalities of the languages.

Since September 1989, Altaïr has started its second phase. The Altaïr group has now two goals: the first is to build an industrial strength from the V1 prototype and the second is both to initiate new research directions and to pursue the research effort on O_2. The presentations which follow are the current activities of the research

[1] Altaïr is a consortium funded by IN2 (a Siemens subsidiary), BULL, INRIA (the French National Institute in Computer and Control) and LRI (the Computer Science Research Laboratory of the University of Paris XI).

group at Altaïr. The research directions include data models and languages (Section 2), programming environment and interface (Section 3) and system aspects (Section 4).

References

[BDK91] F. Bancilhon, C. Delobel and P. Kanellakis Eds. *Building an object-oriented database system: the O_2 story.* To be published by Morgan Kaufmann in 1991.

2 Data models and Languages

Participants: G. Barbedette, C. Bauzer-Medeiros, S. Cluet, C. Delobel, C. Lécluse, E. Penny and P. Richard.

2.1 Data Models and DBPL

Existing database programming languages provide the application programmer with a rich type system which includes bulk types and record types and a computing power greater than relational systems. However, they do not provide any equivalent of a database schema. Indeed, type systems characterize values domain and are used for the purpose of security and efficiency. Unlike integrity constraints, they do not capture the semantics of extensions, nor of their dynamic properties. In most database programming languages, the constraints relevant to an application are scattered into the application code. This has several disadvantages: the user has no global view of the constraints and they are defined in a procedural way. Moreover, relational schemata are inadequate because they are built on top of a limited type system, using only one constructor, the relation. The problem of defining a schema in the context of a DBPL has been studied in [LR90, LR90a]. The last report proposes a general schema definition for a DBPL, which allows declarative integrity constraints, view definitions, structured schemata and contains a proposal for DBMS interoperability. Moreover, this schema definition is general enough to be used with languages having different type systems or programming paradigms. We are now investigating the problem of managing integrity constraints in this framework. More precisely, we are trying to develop static (compilation) and dynamic (execution) techniques for efficient constraint enforcement inside general transactions.

An integration of the O_2 data model into the Lisp language [Barb90] is currently being done. The language, called LispO_2, has been specified and its interpreter is operational. A complete programming environment is under implementation. A set of schema manipulation primitives has already been implemented and is described in [Barb90a]. A compiler for LispO_2 has also been specified and is under implementation. This compiler will generate C code and will provide numerous optimizations. An object manager is also specified and is under implementation. Its architecture has been designed in order to experiment with various storage techniques for complex objects.

2.2 Query Languages for Object-Oriented Databases

A query language [CDLR89, BCD89] has been designed and implemented for the O_2 system. This language is functional in nature with an SQL-like syntax. It allows the end user to query complex objects according to their interface (methods) and their implementation (structure). An optimizer for this language has been designed and is being implemented. Concurrently, a cost model has been defined to study the performance of the O_2 query optimizer. Several optimization techniques, applicable in an object-oriented environment, are currently under study.

2.3 Active Databases and Views

We have developed a production rule management system which has been integrated into the V1 version of the O_2 system [MP90]. Rules are implemented as objects which are considered to be part of a database schema definition, and can be manipulated at will, either inside a program or interactively, using the graphic user interface of O_2. The events that trigger rules can be either related to time or to message sending. Unlike previous proposals for rules for object-oriented databases, the system takes into account the characteristics of the object-oriented paradigm, allowing inheritance, encapsulation and composition of rules. Finally, previous research on active databases concentrates on support for end users and applications, but ignores development activities; the O_2 system allows the use of rules both for production applications and for debugging.

Views are discussed from a novel perspective [Med89], which combines results from research on database design and user interfaces. The proposed mechanism [MM90] considers that a view is defined as a (virtual) database on top of an existing database, by means of view schema specification; the view is then created by querying the underlying database. The difference between this approach and the traditional relational approach is that the result of a query corresponds to a set of objects which can be browsed and updated; furthermore, updates through views are handled by translating update requests into view schema operations, similar to the approach of views as abstract data types. Present efforts

include employing the rule mechanism for implementing view operations, and improving the performance of the rule system.

2.4 Object-Oriented Design

The goal of this activity is to define an iterative design method driven by the formal characterization of the database schema to be modeled. Our major concern is the integrated formalization of the notion of reusability and of the design process. We are currently working on the comparison of several design methodologies for information systems (IS) based on case studies together with the Universities of Paris I and Grenoble. A state of the art [Pen90] on design methods for information systems which defines an IS comparison pattern has been written.

References

[BCD89] F. Bancilhon, S. Cluet, and C. Delobel. Query languages for object-oriented database systems: the O_2 proposal. In *Proc. DBPL, Salishan Lodge, Oregon*, June 1989.

[Barb90] G. Barbedette. LISPO_2: A Persistent Object-Oriented Lisp. In *Proc. 2nd EDBT Conf.*, Venice, February 1990.

[Barb90a] G. Barbedette. Schema Manipulation in the LISPO_2 Persistent Object-Oriented Language. Research Report 56-90, Altaïr, August 1990.

[CDLR89] S. Cluet, C. Delobel, C. Lécluse, and P. Richard. Reloop, an algebra based query language for an object-oriented database system. In *Proc. DOOD*, Kyoto, Japan, December 1989.

[LR89a] C. Lécluse and P. Richard. A Uniform Way of Manipulating Objects and Structured Values in Object-Oriented Databases. In *International Workshop on Database Programming Languages*, Oregon Coast, USA, June 1989.

[LR89b] C. Lécluse and P. Richard. Modeling Complex Structures in Object-Oriented Databases. In *ACM Int. Symp. on PODS*, Philadelphia, USA, March 1989.

[LR89c] C. Lécluse and P. Richard. The O_2 Database Programming Language. In *proc. 15th VLDB Conference*, Amsterdam, The Nederlands, August 1989.

[LR90] C. Lécluse and P. Richard. Data Abstraction, Bulk Data and Relations in Database Programming Languages. Research Report 44-90, Altaïr, June 1990.

[LR90a] C. Lécluse and P. Richard. Data Base Schemas and Types Systems for DBPLs, A Definition and Its Applications. Research Report 55-90, Altaïr, June 1990.

[MM90] J.-C. Mamou and C. B. Medeiros. Interactive Manipulation of Object-oriented Views. Research Report to appear, Altaïr, 1990.

[Med89] C. Medeiros. Views as a mechanism for adding types to a database. In *Proc. XV Latin American Conference on Informatics*, pages 21–28, July 1989.

[MP90] C. B. Medeiros and Patrick Pfeffer. A Mechanism for Managing Rules in an Object-oriented Data base. Research Report to appear, Altaïr, 1990.

[MM90a] C. B. Medeiros and J.-C. Mamou. Visoes em sistemas orientados a objetos: problemas e implementação. In *Proceedings, XVI Conferencia Latinoamericana de Informatica*, 92-103, September 1990.

[Pen90] E. Penny. New Insights on Information System Design Methods. Research Report to appear, Altaïr, 1990.

3 Programming Environment and Interface

Participants: A. Doucet, J. M. Larchevêque, J. C. Mamou and P. Pfeffer.

3.1 Programming Environment

In most database application development environments, the definition of the schema and the development of the application programs are two separate activities, each of them having their own environment. This presents some drawbacks, namely, the lack of homogeneity, the need to learn two different environments and the difficulty to switch from one to the other.

We have designed and implemented a programming environment [BDPT90, BDP90] for O_2, named OOPE, which integrates both the definition of the schema and the programming activities in the same environment. It contains graphical schema editors as well as classical programming tools such as a browser, workspaces, a journal, a debugger.

OOPE is implemented on top of the O_2 system, using all the functionalities provided by O_2. In particular, it makes intensive use of the underlying database to store and manage the information it handles.

We aim to extend the programming environment with more general tools intended to assist the programmer during the development of applications. The whole

process of an application development involves the requirements, functional and detailed specifications, the programs, the tests and their reports, and various manuals. All these elements are related by relationships, such as "generates, derives, implements, tests, ...". These relationships may be used to check consistency and completeness of the elements, and to help the programmer during the maintenance phase. They can also provide the programmer with a development framework. A specification of such a system and its use for assisting the programmer during the application development is under way.

3.2 Diagnostic tools

A debugger is an important part of a programming environment. As traditional debugging techniques do not match the requirements of O_2, we defined an entirely new paradigm for debugging support [DP89, DP90, Pfe90]. The program under the control of the debugger builds a database which reflects its internal state. Static information (symbol tables) and dynamic information (execution stack) is managed by O_2. The O_2 debugger is fully integrated into the programming environment, and is designed and implemented in O_2 and in C. The O_2 debugger also provides an event/trigger mechanism, which automatically informs the programmer about possible mistakes.

The event/trigger mechanism is implemented using a CO_2 interpreter coupled with the debugger. Other uses of this interpreter, such as conditional breakpoints and interactive modification of the compiled code, are under study.

3.3 Incremental Development in a Compiled Environment

To reconcile the imperatives of generating optimized code, detecting errors before program execution, and providing a degree of flexibility and interactivity comparable to that found in interpretive environments, a reflection is conducted on *(i)* very high-level features of DBPLs contributing to source code reusability and malleability, *(ii)* compile-time optimization of these features, and *(iii)* incremental compilation and lazy optimization. Results to date include an optimal incremental parser [Lar90a] and the global design of a Language-Based Editor with the capacity of generating and incrementally updating optimized code [Lar90b]. Current research focuses on incremental data-flow analysis methods applicable to static resolution of method calls, early detection of integrity constraint violations, and static optimizations of storage management.

3.4 User Interface

Today, many object-oriented or complex object database systems provide data models that allow programmers to model the application concepts in terms of complex objects. A complete management by the programmer of the display of the information presentation is tedious, complex, and very time-consuming.

The LOOKS system [PBL*90] provides the O_2 programmer with high level primitives to get object or value presentations. These primitives generate generic presentations, i.e. presentations built from generic display algorithms.

However, the information encapsulated in an object is not always fit for display. Usually, the programmer wants to show only a restricted view of the object and hide the superfluous or private information. Objects or values model real world concepts but the user interface designer wants to show a view of these concepts and not a representation of the modeling he used. This led us to propose an alternative to generic presentations: the ToonMaker system [Mam90]. By enhancing the generic presentations approach based on structured editors through the introduction of a customization capability, this system offers a framework to automatically present any database object while giving programmers the means to change drastically these presentations. A prototype of this system is now achieved which allows the designer of the user interface to build a new presentation, starting from the generic one, by means of graphical constructors, used as building blocks. The construction is performed inside an interface editor that shows dynamically the changes on the presentation. The originality of this work is the embedding of object-oriented database management system concepts inside usual user interface concepts.

References

[BDPT90] P. Borras, A. Doucet, P. Pfeffer, and D. Tallot. OOPE : The O_2 Programming Environment. In *Proceedings of the 6th PRC BD3*, France, September 1990.

[BDP90] P. Borras, A. Doucet, and P. Pfeffer. Using an OODBMS to Implement a Programming Environment, Experience and Lessons. In *Object-Oriented Program Development Environments Workshop, ECOOP/OOPSLA '90*, Ottawa, Canada, October 1990.

[DP89] A. Doucet and P. Pfeffer. A Debugger for O_2, an Object-Oriented Database Language. In *Proceedings of the First International Conference on Technology of Object-*

Oriented Languages and Systems, pages 559 – 571, CNIT Paris - La Défense, France, November 1989.

[DP90] A. Doucet and P. Pfeffer. Using a Database to Implement a Debugger. In *IFIP : Conference on Database Semantics*, North-Holland Elsevier, July 1990.

[Lar90a] J. M. Larchevêque. Using an LALR compiler compiler to generate incremental parsers. In *Proceedings of the Third International Conference on Compiler Compilers, Schwerin (GDR)*, Lecture Notes in Computer Science, Springer-Verlag, October 1990.

[Lar90b] J. M. Larchevêque. Incremental compilation in the O_2 object-oriented database system. In P. Kannelakis F. Bancilhon, C. Delobel, editors, *Building an object-oriented database system, the Story of O_2*, to appear, Morgan Kaufmann, 1991.

[Mam90] J.-C. Mamou. ToonMaker: An Approach to Create Presentations For An Object-Oriented Database System. Research Report to appear, Altaïr, 1990.

[PBL*90] D. Plateau, P. Borras, D. Leveque, J.C. Mamou, and D. Tallot. Building user interfaces with the LOOKS hyper-object system. In *Proc. Eurographics workshop on object-oriented graphics*, 1990.

[Pfe90] P. Pfeffer. *Débogage en environnement orienté objet persistant.* PhD thesis, Université de Paris-sud, July 1990.

4 System Aspects

Participants: V. Benzaken, G. Bernard, C. Delobel, G. Jomier and D. Stève.

4.1 Remote Method Execution

The V1 version of the O_2 system is multi-machine. This means that a particular machine (the "server") acts as a data repository, while the other machines (the "workstations") have no part of the database on their local disk, if any. The server and the workstations exchange messages via a local area network (an Ethernet in V1) [VBD89]. The architecture chosen for V1 is that of an *object server*, where the unit of transfer between the server and the workstation is an object. In this architecture, the server understands the concept of an object and is capable of applying methods to objects. Whereas the server is mainly concerned with data access and management, and workstations with data processing, the ability of executing methods on the server may lead to response time improvements. For instance, suppose that the programmer is writing a selection on a large set with a low selectivity and a simple selection criterion, and that the objects of the set have not already been referenced by the application. The objects of the set are thus not loaded in object memory on the workstation. Executing the selection on the workstation would imply downloading all the objects of the set on the workstation before applying the selection on them. It may be more efficient to transfer execution on the server machine, because only the selected objects will have to be moved.

In the V1 prototype, the distributed architecture is visible to the application programmer (but not to the end user). The programmer is aware of the existence of two machines (the server and the workstation) and may explicitly specify on which of the machines a message passing expression is to be executed. We are now investigating an automated process where the system itself dynamically chooses the site of application of the method. Criteria could be the current workload of the server and/or the workstation, the number of objects to transfer from a site to another to preserve the execution context, and the size of the receiver object.

4.2 Using Idle Workstations

In the V1 prototype, O_2 runs as two processes, one on the server and one on the user workstation. If the user starts several O_2 applications simultaneously, or if an O_2 application is to be executed simultaneously with one or more cpu-intensive application (large compilations, for instance), it may be worthwhile to take advantage of the inactivity of other workstations in the network by submitting some processing to them. We are now investigating this possibility. A number of load sharing policies have been proposed in the literature. However, the O_2 context is particular, because an O_2 application is both interactive and cpu-intensive, and workstations are owned by users who should have absolute priority on the use of their machine [BSS90]. Of course, the user should not be aware of the underlying migration mechanism. The system will decide to run some process on the appropriate workstation, and interactivity will be preserved in a transparent way. The mechanism that starts a process on a remote machine and handles interactivity has been designed and implemented. We are currently designing the transfer policy (should this new process be executed remotely?), the location policy (is there any available workstation now?), and the information policy (how do we get the information that such workstation is available, and what does that mean?).

4.3 Versions

People using databases allowing versions of objects rapidly get lost among versions because DBMSs do not have tools to present to users the versions of different objects that go together. Thus, consistency is an important problem of version management.

To solve this problem which makes the classical implementation of versions inapplicable to databases containing a large number of objects with various schemata of versioning, we propose a completely new approach to versions, the database version approach [CJ90].

In this approach, a multiversion database, i.e., a database where objects may appear in several versions, is modeled as a set of logically independent database versions. Each database version contains one version of each object present in the database (its value may be "does not exist"). Thus the definition of consistency of a database version is the same as consistency in database without versions, i.e., monoversion database.

Multiversion database users work in a database version exactly in the same way as in a classical monoversion database, thus each version of an object is seen in a context of versions consistent with it. Moreover, users may perform transactions deriving new database versions from existing ones. To take into account database versions, the classical definition of transaction for monoversion database is extended and a transaction is defined as a process that takes a set of database versions each one from a consistent state to another consistent state.

Logically, database versions are independent, but physically they may share versions of objects. The main advantages of the database version approach are:

> The concepts of consistency and transaction in multiversion databases are defined consistently and as an extension of their definition in monoversion databases.
>
> Using the database version approach, versioning is orthogonal to data modeling, concurrency control and access authorization.
>
> The versioning of complex objects does not lead to any particular problem compared to the versioning of simple objects.

A prototype [CKW90] implements a version manager and validates the database version approach. Work is in progress on numerous interesting consequences of this new concept on different aspects of a DBMS and on applications using it.

4.4 Clustering

The issue of performance of objects stores is critical for object-oriented database systems. Our research in this direction is based on clustering aspects. The term clustering refers to how objects are grouped into larger units of storage. We have developed clustering strategies which are implemented in the V1 version of O_2 [Ben90a, BD90]. Clustering strategies rely on composition hierarchy of classes as well as on inheritance and are described by the database administrator by means of placement trees. A opposed to other clustering proposals, ours provide data independence. In order to provide the database administrator with a means to select the best strategy, we also designed an evaluation model [Ben90b]. This model takes into account access patterns of classes and allows to derive an optimal strategy in linear time. Early measurements have been performed in the system which validate in part the cost model. We are currently performing more thorough measurements based on the Tektronics Hypermodel Benchmark. Last, a clustering benchmark is currently being defined [BDH90].

References

[BD90] V. Benzaken and C. Delobel. Enhancing Performance in a Persistent Object Store: Clustering Strategies in O_2. In *Proc. 4th Int. Workshop on Persistent Object Systems Design, Implementation and Use*, Martha-Vineyard, Mass., September 1990.

[BDH90] V. Benzaken, C. Delobel and G. Harrus. Measuring Performance of Clustering Strategies: the CluB-0 Benchmark. Research Report to appear, Altaïr, 1990.

[Ben90a] V. Benzaken. Regroupement d'Objets sur Disque dans un Système de Bases de Données Orienté-Objet. PhD thesis, Université de Paris-sud, 1990.

[Ben90b] V. Benzaken. An Evaluation Model of Clustering Strategies in the O_2 Object-Oriented Database System. In *Proc. 3rd ICDT Conference*, Paris, December 1990.

[BSS90] G. Bernard, D. Stève and M. Simatic. Placement et Migration de Processus dans les Systèmes Répartis Faiblement Couplés. Research Report to appear, Altaïr, 1990.

[CJ90] W. Cellary and G. Jomier. Consistency of Versions in Object-Oriented Databases. In *Proc. 16th VLDB Conference*, Brisbane, Australia, August 1990.

[CKW90] W. Cellary, T. Koszlajda and W. Wieczerzycki. Database Version Manager Prototype. Rapport Contrat Gip Altaïr, July 1990.

[VBD89] F. Velez, G. Bernard, and V. Darnis. The O_2 Object Manager: an Overview. In *Proc. 15th*

VLDB Conference, Amsterdam, The Netherlands, August 1989.

5 Esprit participation

Altaïr is involved as a participant in the Esprit Basic Research Action FIDE[2]. This action groups European teams which are pioneers in the area of database programming languages and persistent languages.

In this project, the group works on type systems for database programming languages. The overall problem is to integrate the notions of a type system as defined in programming languages and as in database schemata. An essential requirement for the type system is to support generic application software without losing the security provided by a typing scheme. Among the questions addressed by this action are: What are the necessary base types? What type constructors should be provided? Is data type completeness feasible? What are the notations for type definitions? What is the semantics of the type system? The type system will need to be sufficiently rich to naturally accommodate the information system data, e.g. directly or indirectly providing matrices and procedures, abstract data types, polymorphic types, bulk data types, set types, etc. It also needs to allow definition of application-level types.

Our contribution to the programming environment topic considers the step-wise refinement from a specification language to a very high-level implementation language, namely Quest [Car89]. It consists in defining efficient and incremental code generation strategies for very high-level features of Quest, notably subtype polymorphism and recursive types

Altaïr is also involved in the Esprit II project ITHACA[3] [ANM* 90] started in 1989. The objective is to create a platform for the development of information and production control systems, using an object-oriented approach to develop applications.

Altaïr is involved in the Kernel Group of this project in charge of developing a persistent programming and storage environment. We are working on the following topics: data model, kernel architecture, trigger mechanisms, multilevel transactions.

References

[ANM* 90] M. Ader, O. Nierstrasz, S. McMahon, G. Müller, A-K. Pröfrock. The ITHACA Technology: A Landscape for Object-Oriented Application Development. In *Proceedings of the ESPRIT Conference*, October 1990.

[Car89] L. Cardelli. Typeful Programming. Digital Research Report, No 45, May 1989.

6 Altaïr Organization

Altaïr is managed by F. Bancilhon. The research activities are under the responsibility of C. Delobel and are organized into three groups. The data models and languages group is led by P. Richard, the programming environment group by A. Doucet, and the system group by G. Bernard. The research group consists of 15 people and the development group consists of 25 people.

[2]Project 3070, Formally Integrated Data Environment.

[3]Project no 2121, Integrated Toolkit for Highly Advanced Computer Applications.