A Fully Integrated Programming Environment for an Object-Oriented Database

Patrick Borras †
O_2 Technology
7 rue du Parc de Clagny
78000 Versailles (France)
Anne Doucet †
LRI - Bat 490
Université Paris Sud
91405 Orsay Cedex (France)
e-mail : anne@lri.lri.fr

Patrick Pfeffer ††
Alcatel Alsthom Recherche
Route de Nozay
91460 Marcoussis (France)
patrick@aar.alcatel-alsthom.fr

Abstract

This paper describes the design and the implementation of OOPE, the graphical programming environment of the prototype version of O_2, an object-oriented database system. One of the distinguishing features of this environment is that it mixes the functionalities of programming environments, of both databases and programming languages. Thus, it facilitates and fastens not only the schema design, but also the development of application programs. Another interesting characteristic is that it is being developed using as much as possible the functionalities provided by the O_2 system, namely the programming language, the database and the graphics functionalities.

Keywords: Object-Oriented Database Management Systems, Programming Environment

1 Introduction

O_2 is an object-oriented database system developed at Altaïr, and presently distributed by O_2Technology. Such a system provides the functionalities of a DBMS, of a programming language and of a programming environment. In this paper, we describe the design and the implementation of OOPE, the programming environment of the O_2 prototype. As a large amount of time while developing a database application is spent in programming activity, a big effort has been made in designing the programming environment.

---

†This research was developed while the authors were working for GIP Altaïr. †This research was developed while the authors were working for GIP Altaïr.

‡The views and conclusions contained in this article are those of the author and should not be interpreted as representing the officials policies, either expressed or implied of Alcatel Alsthom Recherche

Database programming presents two main aspects: the first one consists in building and manipulating the schema, while the second, consisting in writing application programs, is very close to conventional programming. Traditional database system, with the fourth generation languages, favoured the former. They provide facilities for schema manipulations, reports and forms edition, and user interface generation. Application programs are generally written in a host language. The programmer uses then the programming tools provided by the programming environment of the host language. In O_2, there is a unique language for schema design and manipulation and for programming. Therefore, we integrated features specific to schema design (database manipulations) and traditional programming tools in the same environment.

In order to provide ease of use and user-friendliness, OOPE makes an important use of graphics. All information is graphically displayed on the screen, and user interaction is always done via direct manipulation.

The implementation of OOPE fully uses the functionalities provided by the O_2 system. Modelization and implementation are respectively done in O_2 and CO_2, one of the languages developed by Altaïr, the database is used to store and manage the information used by OOPE, and the graphical interface has been built by LOOKS, the interface generator provided by Altaïr.

The paper is organized as follows. After a quick description of the main features of the O_2 system in Section 2, we present in Section 3 the design of OOPE, its functionalities and tools. Section 4 concerns the implementation choices of OOPE. The next section (Section 5) compares OOPE to other database programming environments. Implementation aspects are discussed. Lastly, we conclude in Section 6.

2 Overview of the O_2 Programming System

While designing the O_2 system, a major concern was to increase the power of the language without having to write a completely new language. Therefore, O_2 is an object-oriented layer which is added on top of existing languages (C and Basic). This layer, the O_2 data model, fully described in [LRV88], is used to design the application schema. For the time being, the O_2 system provides two languages, CO_2 and $BasicO_2$ with which the programmer can implement the behavioral part of the application. In this section, we briefly present the O_2 features proposed to build a database schema, and we give the example of a method. More details on the O_2 system can be found in [LR89],[VDB89]and [Da90].

O_2 is object-oriented, which means that information is organized as objects, which encapsulate data and behavior. Objects are instances of classes, which describe the common structure and the common behavior of a set of objects. The structure of a class is defined by a type. O_2 provides atomic types (integer, real, float, char, string, boolean, ...) and three constructors, which are the tuple constructor, the list constructor and the set constructor.

Examples of types are:

tuple(name: **string**, age: **integer**);
set(**string**);
list(**integer**);
list(**tuple**(name: **string**, city: **string**)).

```
add class City
        type tuple (name : string,
                    map : bitmap,
                    hotels : set (Hotel))
method Compute_stars(maxprice: integer, minstar: integer)
```

Figure 1: Definition of the class *City*

```
(1)  body Compute_stars(maxprice: integer, minstar: integer) in class City
(2)  co2
(3)  {
(4)  o2 set(name: string, star: integer, price: integer) result;
(5)  o2 Hotel h;
(6)  result=list();
(7)  for (h in self →hotels)
(8)       {
(9)       int p;
(10)      if ((h →stars > minstar) || ((p=[h price]) > maxprice))
(11)        {
(12)        result += set(tuple(name:h→name,
(13)                            star:h→stars,
(14)                            price:p));
(15)      }
(16)      }
(17) display(result);
(18) }
```

Figure 2: Body of the *Compute_stars* method.

The behavior of a class is represented by a set of methods. Their definition is done in O_2 while their body can be written using any of the existing programming languages accepted by O_2. Figure 1 gives an example of a class definition, and Figure 2 shows the body of a method written in CO_2. The O_2 keywords are displayed in bold font. O_2 variables definitions are preceded by the O_2 keyword (lines 4 and 5), while other variables are declared using the C syntax (line 9). The message passing syntax is delimited by "[]" brackets, as shown line 10. The purpose of this method is to build and display an O_2 set, named *result*, in which are collected the name, the number of stars, and the night price of all the hotels of the city for which the number of stars is superior to *minstar* and the night price superior to *maxprice*.

O_2 supports multiple inheritance, and classes are organized into a hierarchy. The inheritance mechanism is based on subtyping. A class, always defined as a subclass of another class, inherits from its superclass(es) the structure and the behavior.

A database application in O_2 consists of programs which represent the different tasks supported by the application. As for methods, the body of a program can be written in any of the O_2 languages. However, the body of a program differs from the

body of a method in that it manages transactions. Furthermore, programs contain calls to methods, but not to other programs, and a program cannot be shared between several applications.

3 Design

Programming environments are essential both for programming languages and for database systems. Their functionalities are however slightly different. Programming environments for database systems, generally called fourth generation languages, arose later. They mostly provide tools for building the schema in a non-procedural way, for creating menus and query forms automatically. Programming environments for traditional programming languages often offer syntax directed editors with many editing and browsing facilities, program verifiers, debugging tools and source code control systems. The main difference between these two approaches comes from the separation of data manipulation and programming languages which exists in database systems. The O_2 system provides a single language for programming and for data manipulation. Its programming environment, OOPE, merges the functionalities of the two approaches in a uniform framework.

The design of OOPE was inspired by the programming environment of Smalltalk-80 [Gol83]. It makes an intensive use of graphics, and has an "all-object" design philosophy. All information is represented by an object, and every action is performed by sending a message to an object.

The care of simplicity led us to represent the information used by the programming environment the same way they are represented in the system, where all information is stored as objects, for bootstraping reasons [VDB89]: the data in the database, the classes, the methods, the applications and the programs. In OOPE, the programmer manipulates all this information as database objects.

This design presents many advantages. First of all, it uses the database functionalities of the O_2 system to store the information managed by OOPE, providing a fast access to it. Secondly, it provides uniformity: the programmer only deals with objects, and every action is performed the same way. The use of a generic display and edition mechanism allows a uniform representation of the objects and of the message passing mechanism. Another advantage concerns the learning time. If everything is represented and manipulated the same way, the only thing the user needs to learn to use OOPE is how to manipulate objects and how to send messages to these objects, which considerably reduces the learning time. Having an object-oriented design, OOPE is easily extensible. A new functionality can be added by defining a new method, or a new object with a method. Finally, this design provides a good integration between the various tools and functionalities: information can be copied, cut and pasted from any object to any other object.

Following the same idea, all the tools (the *Journal*, the *Browser*, the *O_2-shell*, the *Workspace*, the *Debugger*, the *Queries*, and the *Applications*) provided in OOPE are designed with the same philosophy. They all consist of objects stored in the database.

In the sequel, in Sections 3.1 and 3.2, we explain how to build an application (i.e we present the functionalities provided by database programming environments) by showing how to create a class, how to define a method and its body for that class, and how to create an application and its programs. Then we present some tools (which are commonly provided by programming environments for languages) intended to help the programmer and to ease his/her work. A more detailed description of OOPE and

of its usage is given in [BDPT90] and in [OOP90].

3.1 Programming a Database Application

Creating an application involves the following three steps:

- First, the programmer generates the schema, which consists of classes. A class in OOPE is represented by a tuple structured object, as shown in Fig. 3. This tuple specifies the name of the class, its type (the internal structure of the objects of this class), its position in the hierarchy (its superclasses and subclasses), and the set of its private and public methods. It also indicates whether the structure of the class is private or public. The *Infos* field contains some additional information, such as the documentation.

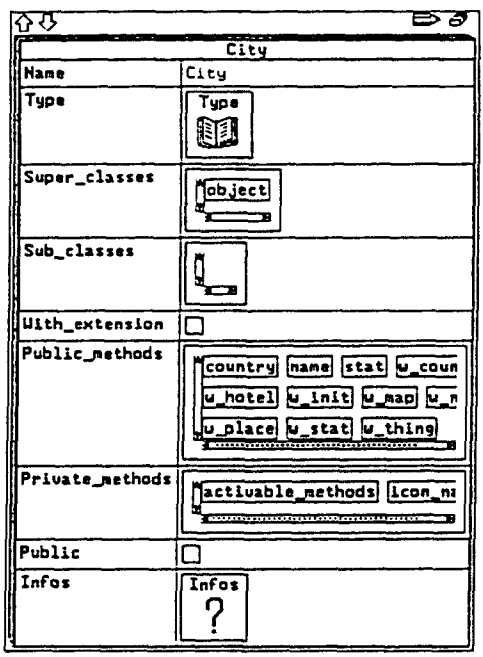

Figure 3: The City class

To be definitively defined, a class must be compiled. This compilation checks if the structural part of the class and the signatures of its methods are correct.

- The next step is the implementation of the methods. A method in OOPE is also represented by a tuple structured object (Fig. 4). It contains the signature of the method, namely its name, its parameters, its result, and the class of its receiver. As the body of a method can be written in any of the O_2 languages, a field indicates which language has been used. Other information (documentation, compiling errors, a compiled or not compiled flag) are gathered in the *Infos* field.

- Finally, once the schema is defined and implemented, the programmer defines and implements the application and its programs. Again, applications and programs are represented as objects of type tuple.

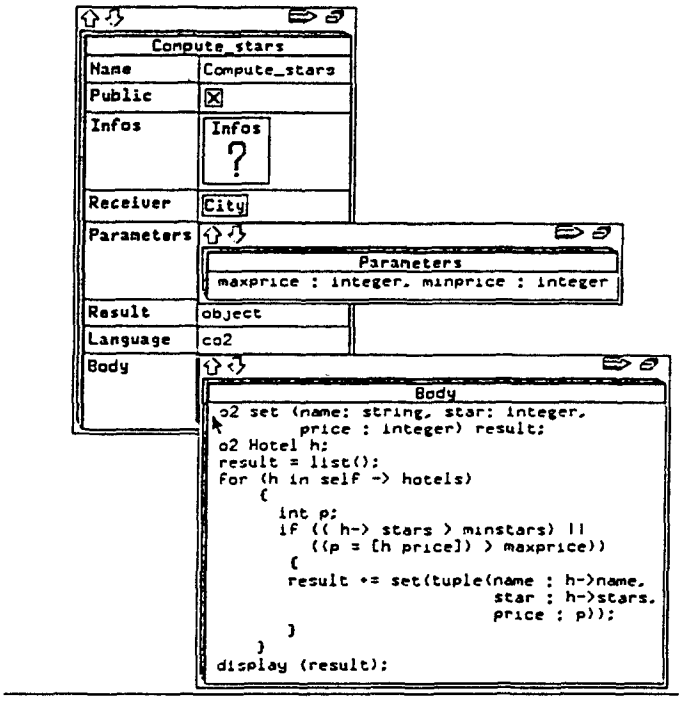

Figure 4: Template for the method *Compute_stars*

These four classes are sufficient to build an entire application. Methods attached to these objects allow to perform the basic functionalities provided by programming environments (compilation, creation of new objects of these classes and deletion of existing instances). It is also possible to run an application, a program or a method.

3.2 Software Development Tools

In this section, we present the different programming tools of OOPE. We do not present a sketch of the programming activity with the use of the OOPE, some examples of this use can be found in [BDPT90] and [OOP90]. They are gathered in an object of type set, which is displayed from the beginning to the end of a session. The programmer has thus access to any of these tools, at any time during the programming session. This object, named *Tools*, is shown in Fig. 5.

3.2.1 The Browser

The *Browser* is generally the first tool used by a programmer, when he/she starts a session. Indeed, it allows to navigate through the database, and to access the objects it contains. The name of these objects are displayed on user demand. The programmer only selects a name and the display command to access an object.

The browser also gives the possibility to display the whole hierarchy of the classes, with the class *object* as a root, as shown in Fig. 6.

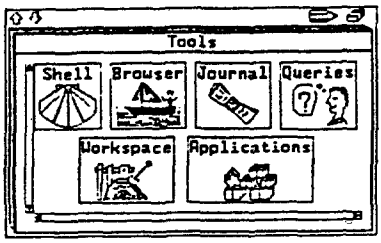

Figure 5: The set of Tools

3.2.2 The O_2-Shell

The O_2-shell is a full screen text editor with a direct interface to the O_2 command interpreter. It is mainly used for two purposes. By giving access to the Unix file system, it allows to read and write Unix files, and thus to import or export new data. The programmer can thus store the definitions of the schema he/she is developing, or bodies used to test parts of the application, in an Unix file, and load them in order to execute them. The O_2-shell also allows to directly execute O_2 instructions in order to test them.

3.2.3 The Journal

The journal allows the programmer to come back to a previous version of an object he/she has modified during the session. This point is important: the journal does not replace a complete versioning system. It is only a simple and fast way to retrieve versions of objects that have just been modified. The journal records critical operations made on OOPE objects, namely classes, methods, programs, applications and named objects. Each time an object is successfully compiled, its version is automatically saved in the journal, with the date. At any time, the programmer can consult the journal, get back a version of an object and restore it. Restoration is manual, and only under user control. The journal only keeps versions of objects during a session. It is initialized to an empty set at the beginning of each session.

3.2.4 The Workspaces

When a programmer builds an application, he/she needs very often to access objects of the database. This is generally done by the browser, as explained above. But, in order to save time, the programmer can store in a workspace all the objects he/she needs, from a session to another. As the workspaces are persistent, the programmer can retrieve at once all the objects he/she was working on during the previous session, avoiding a sometimes tedious navigation through the database. The workspaces, displayed in the *Tools* window, are immediately accessible and can be seen as entry points to the browsing and programming activities. Being a set of objects, the workspace can contain any kind of objects. A programmer can have as many workspaces as he/she wants.

3.2.5 The Debugger

Every programmer knows that programming without bug is utopian. A programming environment providing no debugger is not complete. The O_2 debugger follows the

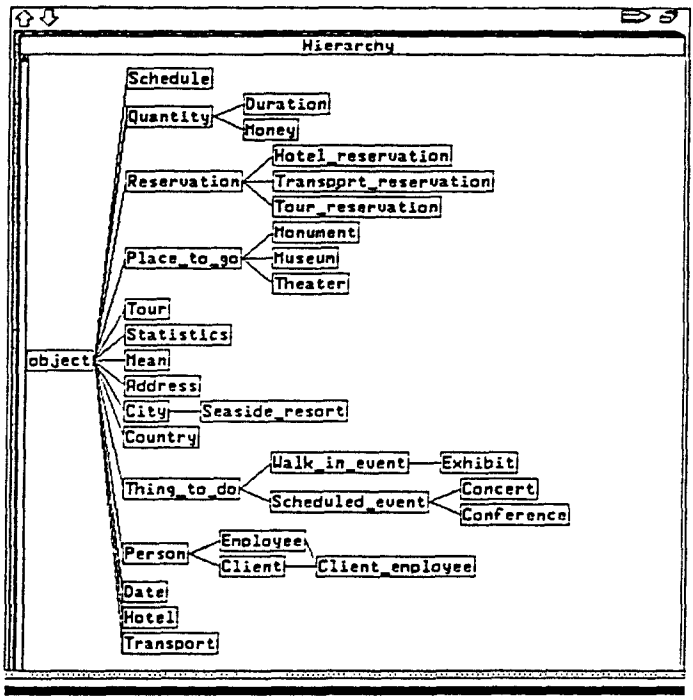

Figure 6: The class hierarchy

object-oriented principles of the OOPE. The debugger paradigm is the following: a program under the control of the debugger is aware of this control and will modify its internal behavior in consequence. The program itself constructs a database of O_2 objects that reflects its internal state at each step of the computation. Composed of four O_2 objects, the O_2 debugger is fully integrated in OOPE. It behaves as any other tool and does not necessitate the learning of a new command language. The four objects of the debugger are (i) an editor to display the source of the programs or methods, (ii) symbol tables, containing information about a method (types and values of the variables, for example), (iii) an execution stack, composed of the symbol tables of the methods called during the execution of a program, and (iv) an execution manager providing a set of commands the user can perform. The execution manager is the tool used to control the execution of a program.

If the programmer detects an abnormal behavior of a program he may execute it under the control of the O_2 debugger, which provides the programmer an interactive control on the program execution and on the message passing, the possibility to edit the values of variables and to execute methods and functions external to the program. To be used in a efficient way, the O_2 debugger supports late binding. The interested reader can find further information in [DP89] and [DP90].

3.2.6 Miscellaneous

In addition to these tools, OOPE gives an access to the O_2 Query language. The *Queries* object, which belongs to the *Tools* object, is of type set(query). It contains a set of existing queries which can be displayed and executed. It also allows the user to create and edit new queries.

The *Tools* also contain an icon called *Applications*, which contains all the applications known by the system. The user can launch an application by opening the corresponding icon and sending a *run* message to it.

4 Implementation Aspects

Programming environments handle a lot of data. The idea of using a database to store and manage these data is common now [Ber87], because of all the services to manage a large amount of data (such as data sharing and integrity, fast access to the data, protection against failure) it provides, comparing to traditional file systems. Our choice of an object-oriented design for OOPE was obviously influenced by the perspective of using the O_2 database system.

In this section, we describe how we implemented OOPE, using the various functionalities (not only the database functionalities, but also those of the language and the interface) provided by the O_2 system.

- The database functionalities

  The main feature of a database is the capacity to store a large amount of data, and to provide a fast and convenient access to it. OOPE manipulates a lot of data, which amount is potentially increasing. All the objects needed to build an application, namely the classes, the methods, the applications and the programs, are stored in the database, as well as the objects they are composed of (the *information* object, the *documentation*, the *errors*), and all the objects composing the different tools. This architecture allows a noticeable performance gain. For instance, retrieving the source of a method is done in a constant time, whatever the number of methods stored in the database (which is not always the case in relational systems where the retrieving time is generally proportional to the size of the relations).

  Using a database to store the programming objects also allows a convenient retrieval: it can be done either by navigation through the objects in the database or by using the query language, which provides high level retrieval operations.

  Another important facility offered by database systems is data sharing, which is not provided by traditional file systems. In programming environments, where objects are often shared among others, this functionality avoids important redundancy and management of this redundancy. For example, a software component can be shared by requirements, documentation, product, milestone.

  Furthermore, database systems offer data integrity, which guarantees a consistent database, and protection against system failures. A good recovery mechanism is of primary importance, because it prevents the programmer from loosing his/her data due to system failures.

  Data independence is also an interesting feature of databases. It allows the tools to remain independent from each other, and to view only the part of the database they need.

The different tools of OOPE make an intensive use of the database, increasing their performance. The journal and the workspace for instance, which are both of type set, take advantage of the efficient set management provided by O_2. But, the debugger is probably the tool which takes the most benefits from the database. The implementation of the debugger and its advantages are fully described in [DP90]. We only mention here the main features of its design and their benefits. Instead of having one unique symbol table for a whole program, the debugger stores its information in several small symbol tables, represented by objects, one per method. This allows a considerable performance gain, by decreasing the access time to the symbol information, as well as a space gain: the process size of the O_2 debugger is much smaller than process sizes of other traditional debuggers, such as GDB [Sta86] or DBX [AM86]. Other functionalities of the debugger also improved their performance: breakpoints are gathered in a list structured object, and detecting if a breakpoint has been set on a given line is equivalent to the access time of an object in main memory. It is clear that the use of the database has been very positive for the implementation of the debugger.

- The use of the O_2 language

  The use of the O_2 database induces the use of the O_2 languages, to define the schema and to write the methods.

  Programming with O_2 provides all the advantages of the object-oriented programming.

  First of all, the modelization power of the language allowed a very simple and easy specification of our objects. The possibility, for instance, to define sets of heterogeneous objects was very useful for the workspaces. Indeed, workspaces can contain any kind of objects. It is not possible to know in advance which objects the user wants to keep. The object *Workspace* must be able to contain any kind of classes. Using a set of objects to modelize it, the programming of this tool became very simple and fast, while it could have been quite complex in a relational database.

  An important feature of the O_2 system is extensibility. The addition of a new tool or of a new functionality to an object does not require modifications of existing tools, which avoids tedious recompilations. For instance, the last tool added to OOPE was the Journal. Its development was made independently of the other components of OOPE. Its integration in OOPE has been done without any modification of the other tools. Likewise, it is possible to remove one of the tools without modifying the other tools. Adding a new functionality to a tool can be done by simply writing a new method for this tool, and does not affect at all any other tool. Object-oriented programming allows a complete integration of the tools in the programming environment. Their behavior is similar, all interactions are done through selection of items in menus, and by editing objects. There is no special command language because of this uniform interaction way.

  Object-oriented languages are well suited for reusability. In order to avoid frequent rewriting of pieces of code already written, O_2 provides a tool box containing a predefined set of classes, objects and values that the programmer can use. For example, the tool box contains all classes and methods used to

handle dates and currency. It also provides all functionalities to implement dialogs in an application. For more details, the reader can read [Ara90], which fully describes this tool box.

- The use of LOOKS

  The user interface of OOPE is entirely written using LOOKS [LOO89]. LOOKS is a user interface generator supporting the graphic and interactive manipulation of O_2 values and objects. It provides a set of 14 primitives which allow to create, remove, edit and save, maintain the consistency of the presentations of any O_2 value or object.

  Using LOOKS to build a user interface simplifies a lot the programming activity. The complete management of a user interface is done in a very simple way using the set of primitives it offers. For instance, only two primitives are needed to display a presentation of an object on the screen, as shown in the following example:

```
class Address
      type tuple (number : integer,
                  street : string,
                  city : City,
                  zip_code : string,
                  country : Country)
```

An object o, belonging to the class *City* is displayed, in a generic fashion, with the primitives **present** and **map** of LOOKS as follows:

```
p = present (o, EDIT, LEVEL_ONE, NESTED_WINDOW);
map (p, SCREEN, COORDINATE, NO_PERSIST, 10, 10);
```

present creates a presentation for the object *o* and **map** displays this presentation as shown in Figure 7.

The parameters of **present** and **map** control the amount of information displayed, its editability and its placement on the screen.

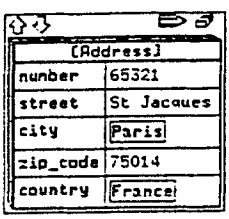

Figure 7: The class *Address*

Moreover, LOOKS offers a generic display and editing mechanism which allows a uniform representation of the objects and the message passing mechanism. This standardization of the object behavior reduces the learning time of OOPE and its tools.

The source code of OOPE is composed of 15000 lines of CO_2 code. It consists for the given time of 75 classes and 320 methods (excluding the inherited methods). OOPE is a quite complex O_2 application. Its success in terms of functionalities and performance validates our implementation choices. Indeed, the use of the O_2 system to implement the programming environment was very positive, for all the reasons previously given. In addition to the services provided by the database, it allowed a gain in performance, a gain in development time, and the use of a generic display and edition mechanism. Furthermore the use of CO_2 provided much flexibility in the design of the schema.

5 Related Work

In this paper, two aspects were considered: the design and the implementation of OOPE. We first compare the design of OOPE with other related work, and then discuss our implementation choices.

For the time being, not much work has been done on programming environments for OODBMS (Object-oriented database management systems), mostly because they are still recent. Among the existing systems (Orion [KBC+88], Ontos [TAD90], Gemstone [MS87], Iris [FBC+87]), only Gemstone uses a programming environment (which is the Smalltalk-80 [Gol83] programming environment) comparable to OOPE. Smalltalk-80 has a very powerful and complete programming environment, emphasing visual accesses to objects. As in O_2, commands are mostly performed through menus. The Smalltalk-80 browser is a sophisticated and efficient tool, but does not allow a graphical display of the class hierarchy, as it is done in O_2.

Being more related to programming environments for languages than for databases, OOPE is mostly to be compared with programming environments for languages. Languages such as C++ or Objective-C provide efficient and complete programming environments inherited from C. But they do not reflect the object-oriented aspects of the language. Moreover, the tools they provide, particularly browsers and debuggers, have poor performance, principally because they are based on Unix files scanning.

Previous approaches for implementing software engineering or programming environments tended to use conventional file system to store their data. To manage it efficiently, additional management components had to be developed. These tools usually are built to deal with particular problems and do not always fit well together or with all the other requirements. In principle, database systems can help to manage this data, by offering functionalities to efficiently store and manage a large amount of data. Indeed, several attempts [Pen86] have been done to use a database, mostly using the relational or the entity-relationship model. But these experiences were not completely satisfying, partly because of the absence of some functionalities (such as versioning, long and nested transactions), but also because of the lack of a good data model. Software development applications involve large and complex objects, which are often related by various relationships, and which need to be manipulated by special operators (able for instance to operate on syntax trees, flow graphs,...) which are usually not provided by DBMS.

The object-oriented model is best suited to represent complex data, but up to now, experiences in using object-oriented database systems to handle the data manipulated by software development systems are quite rare, because available products appeared only recently.

Another approach, taken by Damokles [DGL86] and Cactis [HK88], consists to ex-

tend database systems to support the data management needs of software engineering. They provide features such as complex objects representation, software versions, long transactions. These two interesting experiences are intended to support large software engineering environments rather than database application development environments as done by OOPE, which does not address the "programming in the large" issues.

6 Conclusion

In this paper, we described OOPE, the interactive programming environment of O_2. It provides both the functionalities of database and of languages programming environments, and is one of the few existing programming environment for object-oriented database system. Due to its homogeneity, OOPE is easy to use and to understand. It has a nice interface, which allows simple manipulations of objects on the screen and avoids to learn a new command language.

OOPE is a quite complex O_2 application. It has been developed using as much as possible the various possibilities provided by the O_2 system, namely, the database to store the objects it manipulates, the language to implement the functionalities, and the interface generator to build the interface. OOPE is used inside the Altaïr group and is in Beta-test in several French industries and universities. Its success, in terms of performance and of functionalities validates our design and implementation choices.

OOPE was designed for a prototype version of O_2. O_2 is a now a product, commercialised by O_2 Technology. Its programming environment, named O_2 Tools is an improved and completed version of OOPE. It integrates new functionalities, such as schema update functions, transactions and multi-base functionalities. The user interface is more convenient and provides customization facilities. The browser is divided into five specialized sub-browsers and will provide a much faster and convenient access to the classes, methods, named objects, applications and programs. Other services, such as a test generation tool, a versioning system for the objects, the classes and the methods, and a complete documentation tool are planned to be added to the programming environment.

7 Authorship and Acknowledgements

OOPE was designed by the authors with the collaboration of Jean-Claude Mamou. Didier Tallot and Jean-Claude Mamou helped in the implementation of OOPE. This paper benefits from the careful reading of our colleagues from Altaïr, in particular Gilles Barbedette and Didier Plateau.

References

[AM86] E. Adams and S. S. Muchnick. Dbxtool: A Window-Based Symbolic Debugger for Sun Workstations. *Software, Practice and Experience*, 16 (7), July 1986.

[Ara90] G. Arango. Self-Explained Toolboxes: a Practical Approach to Reusability. In *TOOLS 90*, Paris, France, June 1990.

[BDPT90] P. Borras, A. Doucet, P. Pfeffer, and D. Tallot. OOPE : The O_2 Programming Environment. In *Proceedings of the 6th PRC BD3* , Montpellier, FRANCE, September 26-28 1990.

[Ber87] Philip A. Bernstein. Database System Support for Software Engineering - An Extended Abstract - . In *Proceedings of the 9 th International Conference on Software Engineering*, pages 166 –178, Monterey, California, March 1987.

[Da90] O. Deux and al. The Story of O_2. *Special Issue of IEEE Transactions on Knowledge and Data Engineering*, March 1990.

[DGL86] K. R. Dittrich, W. Gotthard, and P. C. Lockemann. DAMOKLES - A Database System for Software Engineering Environments. In *Proceedings of the IFIP Workshop on Advanced Programming Environments*, June 1986.

[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.

[FBC+87] D. H. Fishman, D. Beech, H. P. Cate, E. C. Chow, T. Connors, J. W. Davis, N. Derrett, C. G. Hoch, W. Kent, P. Lyngbaek, B. Mahbod, M. A. Neimat, T. A. Ryan, and M. C. Shan. Iris: An Object-Oriented Database Management System . *ACM Transactions on Office Information Systems*, 5(1), 1987.

[Gol83] A. Goldberg. *Smalltalk-80 : The Interactive Programming Environment.* Addison-Wesley, 1983.

[HK88] Scott E. Hudson and Roger King. The Cactis Project: Database Support for Software Environments . *IEEE Transactions on Software Engineering* , 14(6):709 – 719, June 1988.

[KBC+88] W. Kim, J. Banerjee, H. T. Chou, J. F. Garza, and D. Woelk. Composite Object Support in an Object-Oriented Database System . In *Proceedings of the ACM SIGMOD Int. Conf.*, Chicago, USA, May 1988.

[LOO89] Gip Altair, BP 105, 78153 Le Chesnay. *LOOKS users manual*, 1989.

[LR89] C. Lécluse and P. Richard. The O_2 Database Programming Language. In *Int. Conf. on Very Large Databases*, The Nederlands, August 1989. ACM.

[LRV88] C. Lécluse, P. Richard, and F. Vélez. O_2, an Object-Oriented Data Model. In *ACM SIGMOD*, pages 424 – 434, 1988.

[MS87] D. Maier and J. Stein. Development and Implementation of an Object-Oriented DBMS . In B. Shriver and P. Wegner, editors, *Research Directions in Object-Oriented Programming* , pages 355 – 392. MIT Press, Cambridge, MA, 1987.

[OOP90] *OOPE: The Object-Oriented Programming Environment.* Gip Altair, BP105, 78153 LE CHESNAY Cedex, FRANCE, Version 1.0, Released 15 December 1989. Printing revision 1.1 edition, 9 January 1990.

[Pen86] Maria H. Penedo. Prototyping a Project Master Data Base for Software Engineering Environment. In *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, pages 1 – 11, December 1986.

[Sta86] R. Stallman. The Gnu Debugger . Technical report, Free Software Foundation, Inc., 675 Mass. Avenue, Cambridge, MA, 02139, USA, 1986.

[TAD90] C. Harris T. Andrews and J. Duhl. The Ontos Object Database. Technical report, Ontologic Inc, Burlington MA, 01803, 1990.

[VDB89] Fernando Vélez, Vineeta Darnis, and Guy Bernard. The O_2 Object Manager, an Overview. In *Int. Conf. on Very Large Databases*, The Nederlands, August 1989. ACM.