An Adaptation of Our Ada95/O2 Binding to Provide Persistence to the Java Language: Sharing and Handling of Data between Heterogeneous Applications Using Persistence

Thierry Millan, Myriam Lamolle, Frédéric Mulatero

An Adaptation of our Ada95/O2 Binding to Provide Persistence to the Java Language: Sharing And Handling of Data between Heterogeneous Applications using Persistence

Thierry MILLAN, Myriam LAMOLLE

IRIT - CNRS (UMR 5055)
Université Paul Sabatier
118, route de Narbonne
31062, Toulouse Cedex FRANCE
Tel. (+33) (0)5 61 55 86 32
Fax. (+33) (0)5 61 55 62 58
E-mail: (millan, lamolle)@irit.fr

Frédéric MULATERO

MIRA
Département de Mathématiques
Université Toulouse Le Mirail
5, allée Antonio MACHADO
31058 Toulouse Cedex
Tel. (+33) (0)5 61 50 42 20
E-mail: mulatero@univ-tlse2.fr

Abstract. This paper sets out the results of our research relating to persistence in Ada and interoperability between applications written using heterogeneous languages. In this paper, we compare the different features of Ada, Java and O2. By making this comparison, we aim to propose a general framework in order to interconnect different applications using the persistence concept. In addition, we propose an example of the co-operation between Ada and Java using O2. We conclude our paper by comparing our approach with the different approaches proposed by other studies on the same subject.

Keywords. Persistence, Ada 83, Ada 95, Java, Object Oriented Database Management System, O2, interoperability, data environment

Introduction

The aim of our project is to propose an efficient solution to save, restore and exchange data [1]. We use the concept of persistence to save the application's context in an object oriented database and retrieve it from the database. The object oriented database management system (OODBMS) O2 manages the database. We use the interconnection capability provided by O2 to exchange data. A prototype of an interface [2] between Ada and O2 has been developed to validate our studies in this area. This prototype use an O2 database management system version 4.6 and Ada version 95 (gnat compiler 3.10p). It runs on a sparc station ultra 1 with solaris 2.5. Performance tests have been performed and are presented in [1, 2]. In addition, we

extend the results of our previous studies on persistence in order to propose a solution to manage persistent environment with other object oriented languages (e.g. C++, Java). The aim of this generalization is to permit the exchange of data between applications written using heterogeneous languages. The main advantage of this approach is the possibility to delegate certain treatments to applications written in a more appropriate language. For example, it is possible to delegate to a Java application the visualization of the data created by an Ada application. In this way the use of Java makes the realization of an application that accesses data through a network more easy.

This paper proposes a comparative study between the Java language, the Ada language and the O2 OODBMS. This comparison aims to show that the O2 OODBMS could be used as a common platform between Java and Ada in order to exchange data. Besides, O2 respects the recommendations of the Object Database Management Group [3] recommendations. These recommendations provide the features that an OODBMS must respect in order to facilitate interconnection between several OODBMS, or between an OODBMS and applications using the CORBA standard. In this paper, we only present the features concerning the handling of data. However, we do not propose to take into account the persistence of “Ada task” , “Java thread” or “Java AWT classes”. In this paper, we also propose an example that shows the advantages of such co-operation between heterogeneous applications.

In the first part of this paper, we will present our experience concerning persistent environments. We will also provide our definitions of persistence. We will also provide a short presentation of the binding between Ada and O2. In the second part of this paper, we will propose a comparative study of Ada 95, Java and O2. The aim of this comparison is to demonstrate that the Ada 95/O2 binding can be an interesting base for a Java/O2 binding. In the third part of the paper, we will provide an example of how these bindings can be applied to manage the data handled by an airplane simulator. Finally, we will conclude by presenting a quick discussion concerning studies carried out in relation to persistence for the Ada and Java languages. We will also present the advantages of such a binding when used to facilitate data exchange independent of the language that creates and handles data. This exchange can be performed independent of the medium (DBMS, OODBMS, etc.) used as interconnection platform.

Persistence and persistent environment

Our previous work on this subject

The emergence of new applications (computer-assisted design, technical management of documents, Internet, Intranet, etc.) needs to be supported by both database functionality and advanced treatment capacities.

The database management systems offer high storage and data handling functionality. However, they are not sufficient when complex algorithms are required. The Ada language enables one to write complex algorithms; However, the Ada programmer remains totally in charge of the efficient storage and handling of high volume of data. That is why couplings between databases management systems and programming languages are valuable.

Definitions

Persistence using Ada 83

The method we propose relies on the following principle [5]: **each data to be handled, has to be linked to a typed identifier.** Ada is a strongly typed language, which means all identifiers require a type which is statically set at compile time. In addition, in order to maintain a strong typed system, which is necessary to set up reliable applications, persistent data should be separated neither from the identifier to which it is linked nor from the **type** of this identifier. **Thus, we consider the concept of persistence at the identifier level rather than at the data level.**

In Ada, we define persistence as the property that allows the triplet **(identifier, data, type)** to survive after the program run-time.

In that context, the program data environment [5] is the set composed of the identifiers declared in the program, the identifiers' type and the data linked to these identifiers.

A persistent data environment is a part of the program data environment which survives once the program run-time ends.

The entire environment of a program is not necessarily persistent. A distinction should therefore be made between the **persistent environment** and the **non-persistent (transient) environment**. With such a persistent environment, it becomes possible to re-use part of the program's data environment. It simplifies the design of applications and improves, among other things, the quality of the software.

It is possible to represent the program data environment by an oriented graph structure [5] where identifiers and data are the graph nodes, and where links between them are the graph edges. A persistent environment is a sub-graph defined by the identifiers, all the data linked to the identifiers and the identifiers' type.

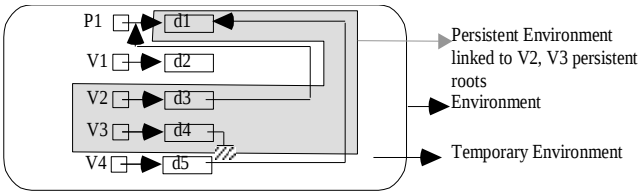

Fig. 1. Persistent environment

Note:

In our definition of a persistent environment, we do not distinguish between data built with the access constructor and the other data. With regard to non access data, we consider that there is only one link between the data and its identifier (see V1 in Figure 1).

Persistence using Ada 95

The object concept is important because it makes the application design easier and it improves the integration between the Ada language and the OODBMS O2.

In Ada, the addition of object concepts allows us to:

- extend types by addition of new attributes and new operations ;
- identify types at run-time ;
- handle values of several specific types and choose an operation at run-time [6].

For this reason, we must improve the rule of "propagation of persistence by reachability" in order to take this new characteristic into account:

– *To be persistent, a data 'd' must be linked to an identifier or a data whose type is the same as, or an ascendant of, the type of the data 'd'.*

To apply this rule, we must re-define persistence through the quadruplet (identifier, type, data, tag of the data):

– *persistence is a property that lets a quadruplet (identifier, type, data, tag of the data) exist after run-time.*

This addition permits the coherence of programs to be retained at the time of the re-use of the environment. Besides, for the persistence instance to exist, all the attributes of a type must be persistent. By way of illustration, we provide the following example: Let a type A derive from a type B and type A uses the types C and D; if persistent instances can exist in type A, persistent instances must also exist in types B, C and D.

Persistent environment properties

In [4], M. Atkinson proposes some properties that a persistent system must respect in order to be transparent for users. These rules are the following:

1. orthogonality to the type system and to creation;
2. propagation of persistence by inheritance or by reachability;
3. behavioral transparency;

Our persistent environment respects the two lasts rules and the orthogonality to creation. However, our persistent environment is not completely orthogonal to the

type system. A special Ada code must be included in the corresponding package to have persistent instance. We ensure orthogonality only for new types' creations. In addition to the rules proposed by Atkinson, we add a fourth rule specially for the Ada language. This rule is the following:

4. integration of persistence into a programming language should not involve any changes in the language.

This rule is essential for standardized languages. Compliance with this rule avoids all the problems that arise when modifications are made to a standardized language (i.e. loss of the standard, ensuring that releases are coherent with new languages releases, etc.).

Our prototype

Principle

We chose to use an object-oriented database management system (O2) because Ada is closer to the object-oriented data model than the relational data model is. Moreover, O2 [9] complies with the object database management group (ODMG) standard [3]. This increases the interoperability between applications.

The Ada/O2 coupling is based on the use of the O2's application program interface (API). The application program interface is a library of functions which allows interaction with the database management system, insofar as the language which interfaces with the system (here, Ada) supports a C language interface.

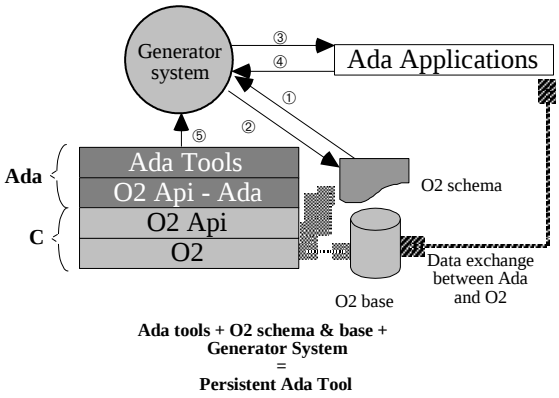

Fig. 2: implementation of Ada persistence

At first, a set of tools has to be designed and implemented in order to connect an Ada application to the O2 database management system (⑤) (see Figure 2). These

tools allow the physical connection to O2, the management of transactions, lists, sets and bags; they also solve the impedance mismatch between Ada and O2 types. It is then necessary to provide programmers with two generator systems. The first generator system generates a set of Ada packages (③) corresponding to a set of O2 classes and to an O2 database (①). The second one generates O2 classes and persistent identifiers (an O2 base) (②) corresponding to a set of Ada packages and to a package containing the database (④).

Communication between the Ada Application and the Database Management System

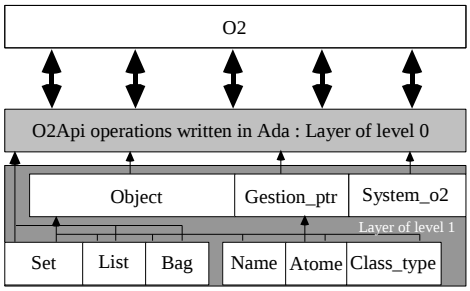

Fig. 3. interface architecture

The architecture [1] set up here (see figure 3) is modular and is composed of two layers. The first one contains all the API operations written in Ada; it represents the exact image of the O2's API written in C. The second layer contains a set of modules providing the services necessary to connect an Ada application with an O2 database. Each module is strongly typed and provides specific services. For example, some modules are used to define sets, lists and bags.

From Ada 95 to Java: adaptation of the persistent environment concept to Java using O2

In the previous part of our paper, we provided a short presentation of the work done in relation to the Ada 83 and 95 languages. Now, we are going to present the similarities and the differences between O2, Ada 95 and Java [7].

At present, new object oriented systems are built around the same concepts (inheritance, polymorphism, abstract class, ...). However, Ada 95, Java and O2 are different because they include specific functionality that represent their valuable specificity. Our aim is to show the possible similarities between an Ada/O2 binding and a Java/O2 binding by reference to the common and specific features of Ada and Java.

In the following two sub-sections, we draw a difference between the functionality that affects the data representation (inheritance, polymorphism, abstract class, ...) and the functionality that does not affect this representation (Ada task, Java thread, Java internet functionality, ...).

Similarities between Ada 95, Java and O2

The following table presents a quick overview of the similarities between the two languages [6, 7, 9] and O2.

Description	Ada 95 functionality	Java functionality	O2 functionality	Affects the data representation?
Inheritance	Tagged type (simple)	(simple)	(multiple)	yes
Parallelism	Task	Thread	-	no
Dynamic data type	Access type	Object	class	yes
Polymorphism	Static and dynamic	dynamic	dynamic	yes
Data typing	Strong	Strong	Strong	yes

In this paper, we do not discuss parallelism. However, in Java, a class can extend the thread class and add new attributes. These attributes can have values that generate a persistent environment consistent with the definition that we have previously provided. In this case, we consider that only these attributes are pertinent.
O2 allows to create values and objects. The values do not have identifiers. It is the values themselves that differentiate between two values. The objects have distinct identifiers. It is the identifiers that differentiate between two objects. In Ada 95, we can define both values and objects. Objects are created using access types. We can consider as values those types that do not use access. In Java, all are capable of being objects. Indeed, we can use the "wrapper class" to encapsulate the primitives types. This feature simplifies the binding because all data structures are identified using a reference. Only one kind of data (object) is implemented.
Ada 95 allows for both static binding and dynamic binding. Java, like O2, only provides for dynamic binding.

Differences between Ada 95, Java and O2

Description	Ada 95 functionality	Java functionality	O2 functionality	Affects the data representation?
Generic	Generic	-	-	yes
Classes' variables and methods	-	static	-	yes
Interface	-	Interface	-	no
Collection	-	-	Set, bag, list	yes
Exception management	Exception,raise	Throw, try, catch	-	yes
Package	Package	Package	-	yes/no

Abstract methods and classes	Abstract	Abstract	-	yes

Our interface between Ada 95 and O2 uses many generic units. This solution is better than using inheritance, because using inheritance can generate type incoherence [8]. This incoherence may affect the behavior of the methods.

Java does not provide collection builders as part of the language. The collections are required to access high volumes of data. These builders are part of the O2 language. In Ada 95, this feature can be efficiently replaced using **generic abstract data types**. In Java, we need to use inheritance [8].

Java enables several classes to be grouped in a package. Packages are not used to implement an abstract data types. In Ada, a package is used to create an abstract data type, to group several abstract types or to specify libraries of sub-programs. In our study, the Java package is **not a pertinent concept** influencing persistent environments. We ignore the Java package's concept in this paper.

O2 does not support abstract classes. Ada and Java support this kind of classes. The Ada 95 and Java compilers checks that no instance of such classes is instantiated. An abstract class can be easily mapped into an O2 class without instance. In our definition of persistence we state that "persistence is a property that lets a quadruplet (identifier, type, data, tag of the data) exist after run-time": **a "type" can be an abstract class, but the "tag of the data" cannot**.

Java and Ada support management of exception, and they consider the exception as data. In O2, we can only save the value of an exception. In this case, we do not take into account the specificity of the exceptions.

Java permits to define the interface of a class. This interface is a set of methods that must be implemented in the classes using this interface. A class can implement several interfaces. This concept does not exist in Ada 95 and O2. In addition, an interface is a set of methods and cannot be used as type of a variable. We consider this concept as **not pertinent** because an interface has no effect on the environment.

Like in C++, developer can declare static variables and methods in Java. These variables and methods are global to all the objects of the class. Only one instance of these variables is present in the system. Static methods permit to handle static variables. Static variables avoid the use of global variables that can generate problems due to side effects. However, the only possibility of storing a persistent environment that includes static data is to define an O2 persistent root for each static variable and to define O2 functions for each static method. In such a case, the developer must find a way of managing the problem due to the side effects. In Ada 95, we can define variables in the package body. These variables can be handled directly by a public sub-program provided in the package specification.

In a word

In this part of our paper, we present the similarities and the differences between Ada 95, Java and O2. This study allows us to compare the concept and functionality

provided by each language. The fundamental concepts of the object languages are covered by Ada 95, O2 and Java. It is easy to define, set, bag and list abstract data types similar to the collection builders present in O2 using inheritance or generic units. The main difficulty of adapting this interface for Java is the transformation of our generic packages into classes using inheritance instead of genericity. The specific functionality (static, abstract, exception, interface) cannot be considered as pertinent to the definition of persistent environment; this is because some of them can be easily simulated in O2.

Example

The example that we present in this paper concerns the development of a airplane simulator. This simulator used the Ada language to implement the treatments of data. It used the O2 OODBMS save the scenarios and the historics of the users.

In this example, we show how to develop a cost efficient solution for the treatment of the data stored in the simulator database (visualization, capture of new scenarios, etc.). We use the Java language to develop these specific applications.

Framework

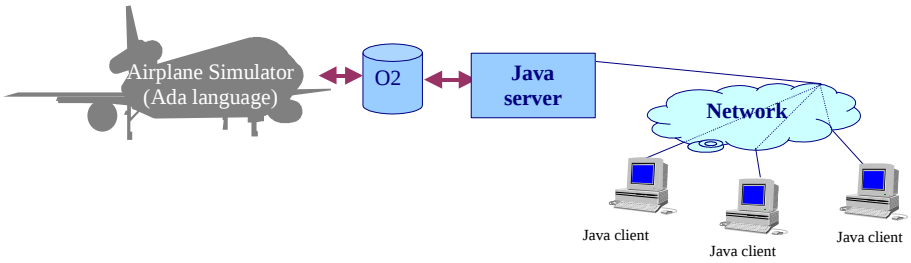

Fig. 4. General framework of the system

The Java server manages the transactions between distant clients. It can decrypt and/or encrypt data, control access and verify the scenario proposed.
The Java client can decrypt and/or encrypt information provide by the Java server. It also provides statistical tools to treat the data provided by the Java server. The main advantage of this application is that the Java client can be easily implemented into different platforms without any need for recompilation. The realization of the Java client and server is relatively easy because Java provides specialized tools which facilitate design and realization of the client-server applications. However, applications written with Java are slower than applications written with Ada. In addition, Ada is a standardized language.

Example of code

Ada code

```
package SCENARIO is
          type T_SCENARIO is record
            ...;
          end record;
          type C_SCENARIO is access T_SCENARIO;
          ... -- Operations for the type C_SCENARIO
end SCENARIO;
with SCENARIO, SET...;
package PERSISTENT_ENV is
          type SET_SCENARIO is limited private;
          S1 : SET_SCENARIO; -- Persistent Data
          procedure COMMIT();
          procedure ABORT();
private
          package PKG_SET_SCENARIO is new SET(...);--A1
          subtype SET_SENARIO is PKG_SET_SCENARIO.T_SET;
end PERSISTENT_ENV ;
```

Java code

```
class SCENARIO {...;
} //end SCENARIO
package MYTOOL.PERSISTENT; //J1
          import MYTOOL.PERSISTENT_TOOLS.SET;
          class SET_SCENARIO extend SET {
            void put (SCENARIO SCN) {//J2
              super.put(SCN);};
            ...};
          public final class PERSISTENT_ENV { //J3
            static SET_SCENARIO S1; //J4 Persistent Data

            PERSISTENT_ENV(){...}; //J5
            public static void COMMIT() {...}; //J6
            public static void ABORT()  {...};
            ....} //end PERSISTENT_ENV
// End package
```

O2 code

```
class SCENARIO type tuple {...};
...;
end;
name SET_SCENARIO : set (SCENARIO); //O1
```

Discussion

We use an instantiation of the generic package "Set" (***A1***) to simulate the set constructor provided by the O2 language (***O1***).
Java does not provide generic classes . Thus, we create a class "Set " that uses the class "Object" as element. We use a Java package (***J1***) that contains the specialization of the class "Set" to "SET_SCENARIO". The re-definition of the "put" method (***J2***) could be useful to avoid type mismatch when a programmer inserts values into the SET_SCENARIO. Indeed, only the class "PERSISTENT_ENV" can be exported outside the package.
We declare the class "PERSISTENT_ENV" as "final" to avoid its specialization (***J3***) of this class. The persistent roots are declared "static", because there is only one persistent environment in an application. "Static" guaranties that only one occurrence of the persistent environment exists, even if there are several instantiations of the class "PERSISTENT_ENV" (***J4***, ***J6***). A constructor is inserted into the "PERSISTENT_ENV" class to initiate the persistent environment. In the Ada language, this initialization (***J5***) is achieved by adding code into the package body. Indeed, it is possible to run a piece of code when the generic package is instantiated.

In this example, we see that there are several differences between the different implementations. The general software framework is similar but the implementation requires a special effort. It is important to carry out studies concerning solutions using design methods (OMT, UML) to palliate this effort.

The first part of our work aims to validate a generic software framework independent of the language features. A second part of our work must generalize the study presented in the third part of this paper "from Ada 95 to Java" for other languages. The last part of our work seeks to propose different modules according to the language features. For example, we must take into account the fact that languages support parameterized types while others do not.

Conclusion

Other studies have been carried out in relation to persistence. Up to now, these studies have related to the persistence of the Java language [10, 12]. However, an Australian team has presented a solution for managing persistent Ada object [11]. [10, 11, 12] propose solutions 100% pure Java or 100% pure Ada. These solutions provide an outstanding result with no type mismatch. However, these solutions are very restrictive and do not provide solutions for exchange data. Thus, interconnection between heterogeneous applications is difficult.
The Java solutions allow the class schema to be saved with the persistent environment. This feature avoids problems of incoherence between applications class schema and the corresponding persistent environment class schema when classes are modified. This solution implies a verification of the coherence at each run-time. The solution that we propose in this paper provides the same feature but we do not provide

for any verification. This verification will be implemented later. In addition, the solution [12] can be easily connected with the ObjectStore database. In this case, we will have also a coupling between a high level language and a database which complies with the ODMG recommendations.
The first prototype has shown the feasibility of a such coupling. The test we performed has shown that performance is good when the number of transient data handled is greater than the number of persistent data handled. At present, we are carrying out a study to generalize the previous work to all languages supporting abstract data type implementation. After this study, we will perform tests concerning the performances of our system, when we distribute data and treatment through a network. Indeed, we want to perform tests in relation to a system in which the treatments are distributed through heterogeneous applications. In addition, we are thinking of replacing the O2 OODBMS by ObjectStore in order to carry out study and which will allow us to compare the Ada/O2 binding and the Ada/ObjectStore binding.

Acknowledgments

The authors would like to thank Julian Cockain for reviewing this paper and for suggesting a number of helpful linguistic improvements.

References

[1] **T. Millan**
Ada et les Systèmes Orientés objets: les Environnement Persistants au Travers d'un Système de Gestion de Bases de Données Orienté Objets
University Paul Sabatier Thesis; 14 September 1995 – Toulouse (France)

[2] **T. Millan, P. Bazex**
Ada/O2 Coupling: A solution for an Efficient Management of Persistence in Ada 83 - Reliable Software Technologies - Ada-Europe'96 - Lecture Notes in Computer Science 1088; Springer-Verlag - Page 396-412; 10-14 June 1996 - Montreux (Switzerland)

[3] **R. Cattel**
ODMG 93: The Object Database Management Group - Edition Morgan Kaufmann, 1994

[4] **M. Atkinson & O. Peter Buneman**
Type and Persistence in Database Programming Languages - ACM Computing Surveys - vol. 19, n° 2 - June 1987

[5] **T. Accart Hardin T. & V. Donzeau-Gouge Viguié**
Conception et outils de programmation. Le style fonctionnel, le style impératif avec CAML et Ada - InterEditions, 1992

[6] **J. Barnes**
Programming in Ada 95 - Addison-Wesley Publishing Company, Inc., 1996

[7] **P. Niemeyer, J. Peck**
Exploring Java - Edition O'Reilly & Associates, Inc., 1996

[8] **B. Meyer**

Conception et Programmation par Objet : pour un Logiciel de Qualité -
Edition InterEditions, 1990

[9] **O2-Technology**
The O2 User Manuel version 4.6

[10] **M. J. Oudshoorn, S. C. Crawley**
Beyond Ada95: The Addition of Persistence and its Consequences -
Reliable Software Technologies - Ada-Europe'96 - Lecture Notes in Computer Science 1088; Springer-Verlag - Page 342-356; 10-14 June 1996 - Montreux (Switzerland)

[11] **M. Atkinson; L. Daynes; M.-J. Jordan; T. Printezis; S. Spence**
An Orthogonally Persistent Java - ACM SIGMOD Record, December 1996

[12] **G. Landis; C. Lamb; T. Blackman; S. Haradhvala; M. Noyes; D. Weinred**
ObjectStore PSE: a persistent Storage Engine for Java
Object Design, Inc. (Internet document)