LISPO₂: a Persistent Object-Oriented Lisp

Gilles Barbedette
Altair
BP105
78153 Le Chesnay Cedex

Abstract
Large and complex design applications such as CASE, office automation and knowledge-based applications require sophisticated development environments supporting modeling, persistence and evolution of complex objects. This paper presents a brief overview of a persistent object-oriented LISP named LISPO₂ combining both language and database facilities. LISPO₂ is a Lisp-based environment providing the O₂ object-oriented data model extended with the notions of constructor and exception. Persistence is achieved through the use of a persistent name space. These features centered around an interpreter facilitate quick prototyping of applications.

1 Introduction
In order to support new application domains such as CASE, office automation or knowledge bases, a development environment has to provide both modeling power and database facilities. The former requires the representation and manipulation of complex objects (programs, documents, rules...), while the latter requires sharing and persistence between program invocations. Databases and programming languages have tried separately to cope with these types of applications. Traditional database systems support persistence, but fail to model objects either in their structural or behavioral complexity. For instance, first normal form relational systems do not deal with structural complexity. Moreover, to express complex behavior, application programmers have to use a query language embedded in a programming language. They are thus faced with the well known "impedance mismatch" problem [Cope84]: they have to learn two different languages and map continuously their different models.

In contrast, programming languages, in particular object-oriented languages [Gold83] [Stro86] [Meye88], offer powerful features such as encapsulation, inheritance and exception handling that ease the design, implementation and evolution of modular and robust applications. However they support only a limited form of persistence through the use of files. Therefore the programmer has to flatten the highly interconnected in-core data structure onto the linear format of files. These error-prone conversions interfere with the application logic, decreasing programmer productivity. Recently, efforts to integrate database and programming languages have come either from database people producing object-oriented databases [Banc88], [Bane87], [Andr87], and [Maie86] or from programming language people producing persistent languages [Schm77], [Atki81], and [Alba85]. These efforts focus on eliminating the major bottleneck to programmer productivity in such systems: the impedance mismatch.

LISPO₂ belongs in this trend. It is a Lisp-based language devoted to supporting incremental and
interactive design of applications. It provides the programmer with object orientation to cope with complex design and persistence facilities to deal with data lifetime.

LISP02 is developed within the Altair project whose objective is to build a multi-language object-oriented database system called O2. This system provides a data model [Lécl89a] with which the programmer can design an application schema and, for the time being, two languages, namely BASICO2 and CO2 [Lécl89b], used to implement the behavioral part of the application. However LISP02 should be distinguished from BASICO2 and CO2. The latter are intended to be used in an industrial environment while the former is designed to experiment with object orientation and persistence. Hence, although it retains the advantages of the O2 data model, LISP02 is autonomous and is developed following the LISP philosophy using bootstrapping techniques, i.e. almost all the system is implemented using the language. Hence, it offers the flexible architecture needed in an experimentation environment.

The remainder of the paper is organized as follows. Section 2 introduces the object-oriented features of the language. Section 3 is devoted to the integration of persistence in the language. This is followed, in Section 4, by a brief outline of the implementation. Section 5 compares LISP02 with other related approaches. Finally, we conclude by indicating future plans.

2 Object-Oriented Features

Although several object-oriented languages [Stro86], [Gold83], [Meye88] and databases [Care88], [Andr87], [Bane87], [Banc88] have appeared, there is no universal consensus on the term “object-oriented” [Wegn87]. Its meaning ranges from “prototype based” [Lieb86] to “class based with inheritance” [Stro86]. LISP02 belongs to the family of languages where the term object-oriented implies the conjunction of encapsulation, class and inheritance. This section gives an informal presentation of how these notions are defined and used in the LISP02 language. Additional features such as tailored object creation and exception handling are also described. We illustrate them by means of a simple example. This application concerns the management of a flying club whose members have a passion for antique airplanes.

2.1 Objects and Classes, Values and Types

To manage the complexity of applications, any modern programming language provides the programmer with both abstract and concrete representation of data. Data abstraction allows the programmer to model and organize information of the real world naturally in term of its functionality without concentrating on its implementation. In contrast, concrete representation is generally used to structure the implementation of data.

LISP02 is an object-oriented language and thus offers data abstraction through objects. Objects encapsulate a state via an operational interface. Object manipulation is achieved through operations and not by directly accessing its internal state.

On the other hand, LISP02 also supports unencapsulated data. These are values. They are used both at an implementation level to represent object states and at a modeling level whenever data abstraction is not useful. Thus, an object can be seen as a capsule built around a value. While objects are manipulated by operations, values are manipulated by operators. Figure 1 shows how we can represent the parts of an airplane as objects. It presents a wheel object whose encapsulated value is an aggregate containing the name of the part, the set of parts of which it is a component.