Schema Modifications in the LISPO₂ Persistent Object-Oriented Language

Gilles Barbedette
Altair
BP105
78153 Le Chesnay Cedex
email: gilles@bd blues.altair.fr

Abstract
This paper addresses the issue of schema evolution in LISPO₂, a persistent object-oriented language. It introduces the schema modifications supported by the LISPO₂ programming environment and presents the potential inconsistencies resulting from these modifications at the schema, method and object levels. Furthermore, it describes how the environment efficiently detects such inconsistencies using a database representing the schema definition. Moreover for correct modifications, it presents how this database is used to update the schema, to trigger method recompilations and to restructure objects using a semi-lazy evolution policy.

1 Introduction

Advanced application domains such as Computer-Aided Software Engineering or Office Automation require both modeling power to represent and manipulate complex objects (e.g. programs, documents or rules) and persistence facilities to store and share these objects between application executions. These new applications led to the development of object-oriented database systems (e.g. [Banc88], [Bane87a]) and persistent programming languages (e.g. [Agra89], [Atki81]). The former augment database systems with expressive power while the latter extend programming languages with persistence. Following this same trend, we developed LISPO₂ [Barb90], a language extending Lisp with the O₂ object-oriented data model [Léc89a] and orthogonal persistence. However, a language, alone, does not create a productive environment. The programmer needs tools which deal with the interactive design and implementation of applications. Recognizing the iterative nature of software development, as in [Booc90] and [Goss90], the LISPO₂ programming environment supports an “evolutionary prototyping” development process where design and implementation are not seen as sequential stages but as interleaved ones. In this process, the prototype iteratively evolves from its first version to the final product as the programmer gains experience with the application and refines its design and implementation. In order to support this mode of development, the programming environment has to facilitate the modification of the design in order to incorporate the results of previous experiments. To fulfill this requirement, the LISPO₂
programming environment provides a mechanism for class modification which enables the programmer to change class definitions on the fly, even though some objects have been previously created and some methods compiled. In such a situation, classical environments [Meye88], [Stro86] require exiting the environment, modifying class definitions, recompiling relevant classes and methods and reloading them. Moreover, the test database has to be regenerated. In contrast, the LISPO2 environment checks the consistency of the modifications with respect to the static semantics of the language. Furthermore, it assists the developer in understanding the effects of his/her modification by pointing out the affected methods, and it triggers their recompilation. Finally, it updates objects automatically in order to meet their new class definitions.

The remainder of the paper is organized as follows. Section 2 introduces the features of the LISPO2 language necessary for understanding the rest of the paper. Section 3 presents a taxonomy of the schema modifications supported by the programming environment. The next three sections address their repercussions respectively on schema definition, methods and existing objects. Moreover, they detail the implementation choices that we made to efficiently detect inconsistencies in the schema, to trigger method recompliations and to restructure objects after a schema modification. This is followed in Section 7 by a comparison with other related approaches. Finally, we conclude by summarizing the innovative features of the environment and by indicating future plans.

2 Overview of the LISPO2 Language

This section briefly introduces the features of the LISPO2 language relevant to the issue of schema evolution. For a more detailed presentation of LISPO2, the reader is referred to [Barb90].

- Classes, Types, Operations and Methods.

LISPO2 is a class-based object-oriented language. A class defines the structure and the behavior of a set of objects called its instances. The structure of an object is defined by a type. A type in LISPO2 is either an atomic type (e.g. integer or float) or a complex type built from other types and classes using the tuple, set and list type constructors. Tuple types are used to model aggregation. Set types represent homogeneous collections without duplicates while list types support indexable homogeneous collections. The behavior of an object is defined by a set of operations. A class definition introduces only the specification of operations, called signatures. A signature includes the name of the operation, the type/class of its arguments (if any) and the type/class of its result. The implementation of an operation is defined by a method. Separating the specification of an operation from its implementation allows the programmer to work with a partially implemented application (no method associated with an operation) or to explore alternative implementations of the same operation (several methods