Orthogonal to the Java Imperative *

Suad Alagić, Jose Solorzano, and David Gitchell

Department of Computer Science
Wichita State University
Wichita, KS 67260-0083, USA
alagic@cs.twsu.edu

Abstract. Three nontrivial limitations of the existing Java™ technology are considered from the viewpoint of object-oriented database technology. The limitations are: lack of support for orthogonal persistence, lack of parametric (and in fact bounded and F-bounded) polymorphism and lack of an assertion (constraint) language. These limitations are overcome by leaving Java as it is, and developing a declarative (query in particular) component of the Java technology. This declarative language is implemented on top of the Java Virtual Machine, extended with orthogonal and transitive persistence. The model of persistence also features complex name space management.

Keywords: Declarative languages, orthogonal persistence, F-bounded polymorphism, Java Virtual Machine.

1 Introduction

In spite of the fact that Java™ [17] is a product of good language design, it exhibits several serious limitations with respect to object-oriented database management. A perfectly valid justification is that it was probably never conceived as a database programming language. Yet, some of these database-critical features are important for object-oriented programming in general.

The first such limitation is, of course, the lack of support for persistence. Persistence is a key feature of object-oriented database technology [10].

The second limitation is the absence of parametric polymorphism. This limitation is particularly critical in database management because most database technologies rely heavily on collections. Static typing of collections in Java is impossible, and so is static typing of queries. Awkward type casts and extensive dynamic type checks are required, affecting both efficiency and reliability.

The third limitation is the lack of an assertion language. Such declarative capabilities for expressing preconditions, postconditions and class invariants are important for object-oriented programming in general ([24], [22], [25]). Moreover, constraints and other declarative language features (including queries) are critical for any viable database technology.

* This material is based upon work supported in part by the U.S. Army Research Office under grant no. DAAH04-96-1-0192.
Several projects are under way, extending the Java technology with persistence [11], parametric polymorphism ([27], [9]) and even queries [15].

The projects extending the language with parametric polymorphism necessarily require changing the language, but they do not necessarily change the Java Virtual Machine.

Java OQL [15] does not change the language, but supporting queries as methods and as objects requires run-time reflection [3] where the standard notion of static typing is extended to dynamic compilation. In addition, persistence capabilities are required in the underlying architecture.

PJama (formerly PJAVA) [11] provides orthogonal persistence without changing the language, but it requires an extension of the Java Virtual Machine. The Java Virtual Machine is specified in [23].

The approach that we take in extending the Java technology with database capabilities is different from any of the above. We do not change the language in spite of its limitations. We leave Java as it is. But we develop a declarative language with all the desired features discussed above and make it fit into the existing Java technology. An extension of the Java Virtual Machine is still required in order to support a sophisticated and orthogonal model of persistence in order to implement this declarative language.

The declarative language $MyT$ presented in this paper makes it possible to express preconditions, postconditions and class invariants. This addresses the problem of the lack of assertions in Java. But the constraint language goes further in allowing declarative specification of methods.

The constraint language has its limitations in expressiveness ([2], [4]). Just like data languages, it is not computationally complete. But it covers a variety of non-trivial applications [6] and it is integrated into the Java programming environment by allowing references to Java classes.

On the other hand, Java classes can access persistent objects and their classes, which are created by $MyT$. Java classes can also naturally create persistent classes and objects without any references to $MyT$ facilities. This is done by simply relying on PJama, and possibly on our name space management extension.

Java is not a persistent programming language. $MyT$ features an orthogonal model of persistence and transitive persistence (persistence by reachability). Unlike all other approaches, persistence capabilities in this model are associated with the root (top) class. This way all classes are persistence capable. The model is naturally based on reachability. In addition, this model of persistence features hierarchical name space management, extending the existing Java mechanism based on packages. Such complex name space management is lacking in persistent supporting Java extensions, such as PJama [11].

Collection classes are critical for object-oriented database technologies. Typing generic collection classes in Java has well-known problems. The best we can do is to specify generic collection classes whose elements are of type $Object$. Accessing such collections requires type casts, which means dynamic type checks.