A Programming Language with Natural Persistence

Luc Bläser

Computer Systems Institute, ETH Zurich, Switzerland

blaeser@inf.ethz.ch

Abstract

As data persistence is very poorly supported by current programming systems, we have initiated a research project to improve this situation. The result is the new programming language Persistent Active Oberon, which directly institutionalizes persistence as a fundamental concept and liberates the programmer from writing complicated code for database interactions.

Categories and Subject Descriptors D.3.3 [*Programming Languages*]: Language Contructs and Features Classes and objects, Modules, packages; H.2.4 [*Database Management*]: Systems Concurrency, Object-oriented databases, Transaction processing

General Terms Design, Languages

Keywords Object Persistence, Language Design and Implementation

1. Introduction

Data persistence (the fact that data is durably stored and survives system restarts) is traditionally regarded as a concern that is outsourced from a programming language and provided by a separate system, such as a database, a component serialization framework, or a file system. However, with all of these methods, a programmer has to endure the considerable work of eventually facilitating persistence, be it by programming the necessary interaction with an external system or designing the software for a special persistence framework. While this effort may be acceptable for programs with a simple data topology, persistence support for object-oriented models is however particularly complicated due to the dynamic nature of object graphs, and the associated runtime management of preserving referential integrity. The dilemma of this groundless and strong division between the domain of persistence and the field of programming could be abandoned by institutionalizing persistence in the programming language. Despite numerous previous efforts, [1, 2, 5], no persistent programming language has yet been developed, which really enables persistence without any artificial specific programmer handling. All hitherto persistent languages involve explicit loading/storing of persistent object roots, special transactional management etc. with a dedicated database API. However, with Persistent Active Oberon, we have now designed and implemented a programming language with naturally inbuilt persistence. The language is based on Active Oberon [4, 8, 9] and runs on the AOS operating system [7].

OOPSLA'06 October 22–26, 2006, Portland, Oregon, USA.
ACM 1-59593-491-X/06/0010.

2. Design

In Persistent Active Oberon, data persistence can be derived as an inherent property of the underlying concept of *modules*. Besides representing a static compilation and deployment unit, modules also form singleton instances in the system, maintaining individual program states. A module is automatically loaded by the system, as soon as it is used for the first time (by the user or an importing module). Once loaded (and initialized), the module stays permanently alive and survives all subsequent system restarts. Therefore, the entire state of modules is inherently persistent. Naturally, references also belong to this persistent state and should remain valid at system restart. In other words, modules constitute the persistent roots, implicitly making all transitively reachable objects of the global state persistent. Figure 1 outlines this with the example of a persistent hospital program, together with a possible runtime topology of corresponding instances. Clearly, the module looks identical to a conventional Active Oberon program, i.e. persistence is here indeed seamlessly integrated in the language.

The persistent module significantly changes the notion of program lifetimes: System restarts due to failures, power pauses or maintenance work, no longer cause the abrupt program termination but only interrupt the program execution. In a future system incarnation, the program is then simply resumed and the latest consistent computing state should have survived the system restart. Hence, the execution can be described in terms of atomic transitions from a consistent computing state to the next. In Persistent Active Oberon, such a transition is called *transaction*, which again can be composed of multiple sub-transactions (*nested transactions*). Only the data state, which has been computed by a completely executed top-most transaction, eventually becomes durable. All other data changes are only temporary to a running transaction execution and are discarded on a sudden system restart. In this programming model, transactions are implicitly defined as procedures, describing a semantic higher operation that only changes the program state consistently. Invoked procedures within other procedures are then considered as sub-transactions. The intrinsic activity of an object (the concurrency concept of Active Oberon [4], denoted with the ACTIVE attribute), eventually specifies a sequence of top-most transactions. Figure 2 delineates the use of transactions with the example of a bank system.

3. Implementation

The implementation of Persistent Active Oberon comprises a compiler, a runtime system with a *persistent storage* infrastructure and a *main memory caching* mechanism, as well as a *schema evolution* facility, to manage multiple versions of the same module and to migrate the persistent data to new versions. An incremental garbage collector enables efficient and complete reclamation of non-persistent objects, using the persistent mature object space algorithm [6] with a new extension for the support of simultaneous object caching [3].

```
MODULE Hospital;
 TYPE
  Patient = OBJECT
    VAR name: ARRAY 32 OF CHAR; age: INTEGER;
        doctor: Doctor;
  END Patient;
  Doctor = OBJECT (* … *) END Doctor;
  PatientList = OBJECT (* … *) END PatientList;
 VAR patients: PatientList;
END Hospital.
```

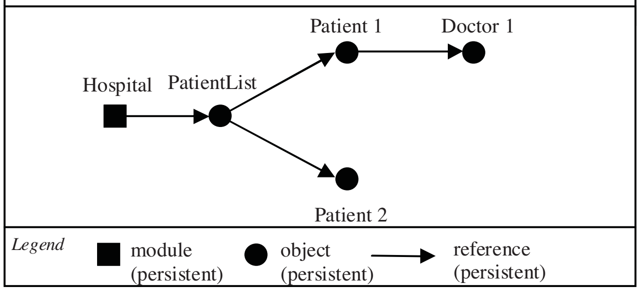

Figure 1. A persistent program

```
Account = OBJECT
 VAR balance: REAL;
 (* an implicit transaction *)
 PROCEDURE Withdraw(amount: REAL): BOOLEAN;
 BEGIN
  IF balance >= amount THEN DEC(balance, amount); RETURN TRUE
  ELSE RETURN FALSE END
 END Withdraw;
 PROCEDURE Deposit(amount: REAL);
 BEGIN INC(balance, amount)
 END Deposit;
END Account;
PROCEDURE Transfer(sender, receiver: Account; amount: REAL);
VAR success: BOOLEAN;
BEGIN
 success := sender.Withdraw(amount); (* sub-transaction *)
 IF success THEN receiver.Deposit(amount) END
END Transfer;
BankManager = OBJECT
VAR a, b: Account; success: BOOLEAN;
BEGIN {ACTIVE}
 success := Transfer(a, b, 1000); (* top-most tranaction *)
 ReportStatus(success)
END BankManager;
```

Figure 2. Implicit transactions

Performance measurements have shown that the persistent runtime system is certainly competitive to the scalability and efficiency of a classical database solution [3].

The system also permits safe interoperability between persistent and conventional non-persistent modules, such that one can flexibly decide where persistence is needed and the implied higher runtime costs are justified. For this purpose, references leading from a persistent to a conventional module are flagged with the TRANSIENT attribute and are safely reset to NIL on system restart.

4. Conclusion

The example of Persistent Active Oberon shows that data persistence can be supported as a naturally inbuilt concept of a programming language, significantly easing the use of persistent data in a program. The source code and a running version of the system are available at [3].

Acknowledgments

Many thanks go to Prof. Dr. Jürg Gutknecht, Dr. Thomas Frey, and Raphael Güntensperger for their helpful support during this work.

References

[1] M. P. Atkinson, P. J. Bailey, K. J. Chisholm, et al. PS-Algol: A Language for Persistent Programming. Australian National Computer Conference, Sept. 1983.

[2] M. P. Atkinson, L. Daynès, M. J. Jordan, et al. An Orthogonally Persistent Java. SIGMOD Record, 25(4):68-75, Dec. 1996.

[3] L. Bläser. The Persistent Active Object System. ETH Zurich, 2004. http://www.bluebottle.ethz.ch/Persistent

[4] J. Gutknecht. Do the Fish Really Need Remote Control? A Proposal for Self-Active Objects in Oberon. Joint Modular Languages Conference (JMLC), March 1997.

[5] A. L. Hosking and J. Chen. PM3: An Orthogonally Persistent Systems Programming Language Design, Implementation, Performance. Very Large Database Conference (VLDB), Sept. 1999.

[6] J. E. B. Moss, D. S. Munro, and R. L. Hudson. PMOS: A Complete and Coarse-Grained Incremental Garbage Collector for Persistent Object Stores. Intl. Workshop on Persistent Object Systems (POS), May 1996.

[7] P. J. Muller. The Active Object System. Design and Multiprocessor Implementation. PhD Thesis 14755, ETH Zurich, 2002. http://www.bluebottle.ethz.ch.

[8] N. Wirth and J. Gutknecht. The Oberon System. Software - Practice and Experience, 19(9): 857-893, Sept. 1989.

[9] N. Wirth. The Programming Language Oberon. Software - Practice and Experience, 18(7): 671-690, July 1988.