Contracted Persistent Object Programming

Stephanie Balzer

Swiss Federal Institute of Technology Zürich,
CH-8092 Zürich, Switzerland
stephanie.balzer@inf.ethz.ch
http://se.inf.ethz.ch/people/balzer/index.html

Abstract. Enterprise applications, that is large and long-lived applications, require persistence. Conventional approaches to persistence suffer from various deficiencies since they pass a considerable amount of the persistence workload on the programmer. Programmers have to transfer data to and from storage devices and have to provide mappings from the programming data structures to the storage device data structures. Thus, programmers are distracted from modeling the application logic. The invention of orthogonal persistence has provided substantial remedy: the automation of persistence-related operations lets programmers focus on application logic and facilitates reuse. In this paper we introduce contracted persistent object programming, a new approach to persistent programming, that is based on orthogonal persistence, but constrained to the object-oriented case. Contracted persistent object programming further extends orthogonal persistence by Design by Contract and the uniform handling of transient and persistent objects.

1 Introduction

The fundamental artifacts of object-orientation are *classes* and *objects*. Objects, the run-time counterparts of classes, are inherently *transient*. They are created during program execution and vanish after program termination. Enterprise applications [1], that is large and long-lived applications, require *persistent* objects. Since they access the same objects in different program executions, they need objects that survive individual program termination. Persistence support requires thus preserving objects for later retrieval.

As already mentioned in [2] conventional persistence approaches, such as the usage of files or database management systems for storing application data, have many disadvantages. The main problem of those approaches is that programmers are completely aware of the underlying storage devices and have to explicitly deal with them. Thus, a considerable portion of enterprise application code is concerned with transferring data to and from storage devices and, due to the different data models used, with mapping programming language data structures to the storage device data structures.

In conventional persistence approaches, programmers cannot focus on the application logic alone, but are distracted by the persistence-related operations

they have to carry out. Moreover, the coexistence of persistence-related code and application code hinders reuse: it is difficult to reuse existing application code in a different system without modification, and the smooth transition from one storage device to another is utterly impossible.

The concept of *orthogonal persistence* (OP), originally introduced in [2] for procedural programming and then applied to object-oriented programming in [3], has defined a fundamentally different approach to persistent programming. The crucial idea of orthogonal persistence is to automate the management of long-term storage in order that programmers can focus on the application logic.

In this paper we describe the main ideas of our research, namely *contracted persistent object programming* (CPOP). CPOP is based on object-oriented, orthogonal persistence, but pairs it with *Design by Contract* (DBC) [4]. Moreover, CPOP is an attempt to unify the handling of transient and persistent objects by providing the same mechanisms for both of them.

The remainder of this paper is organized as follows: Section 2 provides a short introduction to Design by Contract and an overview of conventional persistence approaches. Section 3 defines orthogonal persistence. Section 4 describes our research area, contracted persistent object programming, and section 5, finally, provides our conclusions.

2 Background

2.1 Design by Contract

Design by Contract (DBC) [4] is a design methodology that enables the specification of mutual obligations between client and supplier classes. *Preconditions* are conditions the client has to fulfill in order that the supplier carries out the required operation properly. *Postconditions* are warranties by the supplier on the quality of the operation execution, provided that the preconditions are met. In addition to preconditions and postconditions, DBC also supports *invariants*. Invariants capture the deeper semantic properties and integrity constraints characterizing the instances of a class.[1]

Contracts capture consistency conditions. In Eiffel, an object-oriented programming language based on DBC (see [5]), contracts are monitored at run-time. Any contract violation signals a constraint violation and consequently results in throwing an exception.

2.2 Conventional Persistence Approaches

Based on [1] we categorize the conventional persistence approaches for object-oriented programming as follows:

[1] To be practical invariants have to hold only at "stable" times, i.e. after object creation and before and after every remote method call.

- *Object Serialization*: Mechanism for encoding respectively decoding object graphs into and from binary representations. The mechanism serializes the root of the graph as well as all objects transitively referenced by the root object. Object serialization does not preserve previously common sub-structures, does only provide navigational access to the serialized objects starting from the root object, and does not scale very well. It is consequently only suited for a limited number of cases, such as remote method invocation, where sharing of sub-structures is undesired. Object serialization is a valid complement to a persistence mechanism, but not a replacement thereof.
- *Relational Database Interface*: Two-tiered architecture consisting of an object-oriented programming language and a relational database management system. Programmers access the underlying database using a well-defined Application Programming Interface (API). The API offers methods for connecting to databases and methods for storing, updating, and retrieving objects. Java provides two complementary APIs: JDBC (Java Database Connectivity) for dynamic SQL operations and SQLJ for statically checked embedded SQL statements (see [6]). Although relational database interfaces provide the persistence facilities an enterprise application needs, they inherently suffer from the impedance mismatch between the object model of the programming language and the relational model of the database. The impedance mismatch will force the programmer to maintain a complex mapping between the two incompatible data structures.
- *Object Database Interface*: Object database interfaces do not suffer from the impedance mismatch as relational interfaces do. Apart from the easy mapping, however, object database interfaces provide persistence-related operations, such as for deletion or transaction control, that rather defeat persistence independence (see section 3).
- *Persistence Frameworks*: Persistence frameworks provide a huge selection of persistence facilities, such as access to a wide variety of heterogeneous data sources in case of Java Data Objects (JDO) or distributed persistence in case of Enterprise Java Beans (EJB). Unfortunately, they both do not comply with persistence independence (see section 3) [1].

3 Orthogonal Persistence

As defined in [2], [7], [3], [8], [1] persistent programming languages foster programmer productivity by unbanning them from dealing with persistence issues and by promoting reuse. Persistent programming languages comply with the following *orthogonal persistence* (OP) principles:

- *Orthogonality*: All data must have the same rights to persistence, irrespective of their type, size, or any other property. This principle keeps the programmer from providing persistence support by hand for those data types the language lacks persistence support. Such handwork is undesirable since it distract the programmer from his actual task, namely implementing the application logic.

- *Transitivity*: Whenever data is stored everything that is needed to use that data correctly has to be stored as well. This principle prevents dangling references. Moreover, it assures that stored objects can be correctly interpreted upon retrieval since the principle applies to objects and their classes likewise.
- *Persistence Independence*: The semantics of a programming language must not change when introducing persistence except for the fact that data may outlive a single program execution. This principle promotes reuse; code written for a transient context can be easily reused for a persistent context and vice versa.

4 Contracted Persistent Object Programming

4.1 Definition

For the remainder of this paper we provide the following definitions:

Definition 1. *Object persistence is the ability to make run-time objects of an object-oriented program survive program termination*[2]*.*

Definition 2. *Contracted persistent object programming (CPOP) achieves object persistence. CPOP complies with the principles of orthogonal persistence (see section 3) and uses Design by Contract (DBC) as its driving force. CPOP also unifies the handling of transient and persistent objects by providing the same mechanisms for both of them.*

4.2 Consequences of Orthogonality

The orthogonality of contracted persistent object programming has the following consequences:

- *Language Completeness*: The programming language must provide the full range of persistence mechanisms enterprise applications need. Those mechanisms have to be based on object-orientation and have to be consistent with the concepts of the programming language.
- *Transparency*: The persistence mechanisms of the object-oriented programming language must not reveal by what means persistence is achieved. This obviously requires a lot of automation in mapping the persistence mechanisms of the programming language to those provided by the actual underlying storage devices.

[2] This definition slightly differs from the persistence definition provided in [3] in that we define persistence in relation to the program execution and restrict it to the object-oriented case.

4.3 Contracts

Database management systems provide, in addition to type checking, consistency constraint checking. Conventional programming languages, however, only provide type checking. We believe that enterprise applications need the provision of consistency constraint definition and checking at the language level. Since Design by Contract enables the specification and checking of consistency conditions, we require a persistent programming language to support DBC.

The usage of a contract-based, object-oriented programming language such as Eiffel [5] would further facilitate run-time consistency constraint checking, which persistent programming obviously requires [7]. Thus, contracted persistent object programming unites the power of statically typing with run-time consistency checking.

4.4 Uniformity

We believe that persistent programming should strive for as much uniformity as possible in the handling of transient and persistent objects. Consequently, for every mechanism persistent objects require we should ask ourselves whether that mechanism might make sense for transient objects too.

We have found that property-based querying as well as transaction handling have to be available for both transient and persistent objects.

Property-Based Querying Persistent programming typically encompasses not only the storing of data but also their retrieval. We believe, that to be practical, retrieval mechanisms have to be property-based. Consequently, CPOP has to provide property-based querying facilities that enable programmers to specify object sets by indicating the properties all objects within that set have to meet.

Property-based querying makes also sense for transient objects. The provision of such a mechanism would relieve programmers from maintaining references to objects they keep accessing[3]. Property-based querying facilities further could replace existing container data structures such as arrays or hash tables.

Transaction and Recovery As mentioned in [7] the database and programming language communities have different approaches to concurrency control: Whereas the programming languages predominantly use synchronization mechanisms, the database management systems concentrate on parallel mechanisms based on transactions, which are aborted in case of conflicts.

Transactions exhibit the so called ACID properties, that is atomicity, consistency preservation, isolation, and durability [9]. We believe that transient concurrent programming basically needs the same properties, except for the last one,

[3] It must be noted that such an approach requires adapting garbage collection: not being referenced does no longer imply “garbage”.

namely durability. Thus, the provision of a transactional framework would not only foster the handling of persistent objects but also the handling of transient concurrent objects (see [10]). Contracted persistent object programming consequently has to provide a transactional framework that exhibits ACID properties with the option to switch durability on or off.

5 Conclusions

In this paper we have introduced the main ideas of our research, namely contracted persistent object programming (CPOP). CPOP is based on orthogonal persistence (OP) constrained to the object-oriented case and extends OP by Design by Contract (DBC) and the uniform handling of transient and persistent objects. We expect the introduction of DBC and the uniform provision of property-based querying and transaction facilities to be the main contributions to the field of persistent programming.

References

1. Atkinson, M.P.: Persistence and java - a balancing act. In Dittrich, K., Guerrini, G., Merlo, I., Oliva, M., Rodrguez, M., eds.: Objects and Databases. Volume 1944 of Lecture Notes in Computer Science., Springer-Verlag GmbH (2000) 1–31
2. Atkinson, M.P., Bailey, P.J., Chisholm, K., Cockshott, W.P., Morrison, R.: An approach to persistent programming. Comput. J. **26** (1983) 360–365
3. Atkinson, M.P., Daynès, L., Jordan, M.J., Printezis, T., Spence, S.: An orthogonally persistent Java. SIGMOD Record **25** (1996) 68–75
4. Meyer, B.: Object-Oriented Software Construction. Second edn. Prentice Hall Professional Technical Reference (1997)
5. Meyer, B.: Eiffel: The Language. Prentice Hall Professional Technical Reference (1991)
6. Clossman, G., Shaw, P., Hapner, M., Klein, J., Pledereder, R., Becker, B.: Java and relational databases: SQLJ (tutorial). In: SIGMOD Conference. Volume 27., ACM Press (1998) 500
7. Atkinson, M.P., Morrison, R.: Orthogonally persistent object systems. VLDB J. **4** (1995) 319–401
8. Atkinson, M.P., Jordan, M.J.: Providing orthogonal persistence for Java (extended abstract). In Jul, E., ed.: ECOOP. Volume 1445 of Lecture Notes in Computer Science., Springer-Verlag GmbH (1998) 383–395
9. Elmasri, R., Navathe, S.B.: Fundamentals of Database Systems, 2nd Edition. Second edn. Benjamin/Cummings (1994)
10. Chair of Software Engineering of ETH Zurich: Simple Concurrent Object-Oriented Programming (SCOOP). http://se.inf.ethz.ch/research/scoop.html (2005)