Transferring Persistence Concepts in Java ODBMSs to AspectJ Based on ODMG Standards

Arno Schmidmeier

Sirius Software GmbH,
Oberhaching

Abstract: *Aspects are abstractions that capture and localise crosscutting type concerns. Although the persistence of aspects has received an increasing interest among researchers in software engineering, basic distinctions between persistent and transient aspects, and their relationship to weaved objects, are lacking clarification. This paper introduces definitions and illustrates how an existing object oriented database management system (ODBMS) can be used as an aspect oriented database management system (ADBMS) based on such definitions and previously established ODMG standards.*

Introduction

Implementation approaches dealing with typical usage scenarios of an object oriented database raise the importance of persistent aspects [1]. The capability to store aspects in an ODBMS requires extending the ODBMS to an ADBMS. This can be achieved by enhancing the persistent object model of an object oriented programming language to that of a general aspect oriented programming language (GAPL), which enhances the first programming language and can be compiled back to it. At the design level, enhancing the persistent object model consists of two major steps:

1. Porting the concept of persistent capable classes to persistent capable aspects
2. Extending the concept of persistence through reachability to encompass aspects.

Once these steps are accomplished, it is relatively easy to write persistent capable aspects. The GAPL compiler enables translating the persistent capable aspects into persistent capable classes, which can then be stored in the ODBMS. As a result, existing commercial ODBMSs can be reused with little or no modifications. The approaches discussed in this article are based on AspectJ, versions 0.8beta1 to beta3 [2], the standards outlined in ODMG 2.0 [3], and compliant database Objectivity 6.0 [4]. A similar implementation of this approach is discussed in [1].

Analysis of the Object and Aspect Models of AspectJ

AspectJ offers, in addition to the java type construct, the aspect type construct. According to the aspect language reference: *"An aspect is a crosscutting type defined by the aspect declaration. The aspect declaration is similar to the class declaration in that it defines a type and an implementation for that type. It differs in that the type and implementation can cut across other types (including those defined by other aspect declarations), and that it may not be directly instantiated with a new expression. Aspects may have one constructor definition, but it must be of a nullary constructor throwing no checked exceptions."* [6]

Instances of an aspect class are called aspect instances[1], which only the AspectJ runtime environment is capable of generating. The aspect class is instantiated based on the aspect signature. However, the standard aspect signature 'of eachJVM()' can be omitted. In this case, one instance is generated inside each Java Virtual Machine where the aspect is used. 'of eachJVM()' realises a kind of a singleton [10] pattern for an aspect.
If a user wants to have more instances of an aspect, the aspect class must be declared using 'of eachobject(PCD[2])', of 'eachcflow(PCD)', or 'of eachcflowbelow(PCD)'. In the first case, a new aspect instance is created for every object associated with the pointcut P. If an aspect class A is defined 'of eachcflow(P)', then one object of type A is created for each flow of control at the join points of pointcut P. If an aspect class A is defined 'of eachcflowbelow(P)', then one object of type A is created for each flow of control below the join points of pointcut P. Except the difference in generating aspect instances, aspect instances and aspects classes behave like objects and classes. Aspect classes not only can extend both java classes and aspects classes, but also can implement interfaces. Aspect instances can be used anywhere a java object is expected. The AspectJ compiler transforms an aspect class into a java class.

Transferring the Persistence Model of Java Classes to Aspects.

For such a transfer to take place, a definition can be made similar to that which is defined for java by the ODMG. Mainly, *persistent capable aspects classes are aspect classes, whose instances can be stored in an aspect oriented database.* All aspect classes are persistent capable if the following conditions are met:

1. The class either implements a specific interface or extends a specific root class.
2. All attributes <u>must be</u> either:
    a. An atomic data type
    b. A persistent capable class
    c. A persistent capable aspect
    d. An atomic data structure
    e. An array consisting only of elements, which fulfil a, b, c, d or e.
    f. or, marked as transient, static or final.

All *aspect classes*, which are not persistent capable, are transient *aspect classes*. Instances of transient aspect classes cannot be stored in the aspect oriented database. All aspect instances, which are stored in the persistent storage, are called *persistent aspect instances*. All other aspect instances are called *transient aspect instances*. Transient aspect instances can become persistent aspect instances, like transient objects can become persistent objects, by either storing the aspect directly in the database (e.g. clustering) or achieving persistence through reachability.

Extending the Concept of Persistence through Reachability

It is necessary to extend the concept of persistence through reachability, which takes into account aspect classes, in addition to the existing mapping of java classes drawn by the ODMG [3], [5].

[1] For clarity, we use the term "*aspect class*" for an aspect and the term *aspect instance* for an instance of an aspect class in this paper.
[2] PCD Pointcut discriminator

To be made persistent at the end of a transaction, all transient aspect instances and objects, *must* be:

1. An instance of a persistent capable class or an instance of a persistent capable aspect instance.
2. Directly or indirectly referenced from a persistent aspect instance or from a persistent object.

This concept is called: *persistence through Reachability for aspects and classes (*or in the context of aspects, simply,: *persistence through Reachability.)* A persistent aspect instance remains persistent, till it is removed explicitly from the database, or till it is removed from the database by the database garbage collector. So it is obvious, that the lifecycle of a persistent aspect instance, can easily extend the lifecycle of some Java virtual machines. Additionally, one can use the same aspect instance in several Java virtual machines at the same time.
Persistence through reachability, as just defined, allows an aspect instance to get stored alone without the object instance for which it was bound; and, vice versa, an object gets stored without the aspects, which it was bound to it.

Some usage patterns:
A new instance of an persistent capable aspect class A is generated at any time, when a public method foo() is called in any class. The constructor of the aspect class A stores the aspect instance in the database. One could use explicit techniques, (e.g. clustering or using named roots) or apply persistence through reachability. The aspect instance will be made persistent independent from that fact, if the object that the aspect binds is persistent capable. When an instance of such an aspect is loaded from the database to a different JVM, it is quite clear, that it is not bound to any object anymore, if no reweaving takes place.
A more common usage pattern is, that only instances of transient aspect classes are bound to an object of a persistent capable class. If this object is made persistent, the aspects instances could not be saved.
In some other cases the weaving relationship shall be counted as a reference according the newly established definitions for persistence through reachability. For example, when an object is made persistent all aspect instances bound to this object should be made persistent as well, and vice versa.

Persistence of Weavings

The usage patterns above illustrate the need to further define such patterns and specify which of these patterns are to be supported by the runtime environments of the GAPL and the ADBMS.

Definitions:
Let O is any transient object and A is a transient aspect instance weaved to O.

If the weaving between O and A fulfils the following conditions:

If O is made persistent, then A is automatically made persistent too *and*
If A is made persistent, then O is automatically made persistent too,

…the weaving is considered a *persistent weaving*.

When weaving between O and A meet these conditions:

If O is made persistent, then A is automatically made persistent **<u>or</u>**

If A is made persistent, then O is automatically made persistent.

…the weaving is called a *partial persistent weaving*.

All other weavings are considered *transient weavings*.

Based on further investigations, it is necessary to differentiate the transient weavings even more. If a persistent object or aspect with a transient weaving is loaded into a JVM and the weaves could be re-established, the transient weaving is considered a *rebindable transient weaving*, or in short a *rebindable weaving.* If the reweaving is not initiated by the programmer or by another "user"-aspect, (e.g. from the database runtime or from the runtime of the GAPL), the weaving is called *transient, automatic rebindable weaving* or, in short, *automatic, rebindable weaving*. If a reweaving is not possible we speak from a *lost through persistence weaving.*

it is possible that some weavings can be persistent, some other weavings of the same persistent object might be rebindable transient, and some other are lost through persistence weavings.

Any ODBMS, supporting at least *lost through persistence weaving*s and (partial) *persistent weaving* can be called an ADBMS from an *aspect* point of view.

Experiences

Sirius Software's Research and Development is currently using Objectivity 6.0 [4] with AspectJ (version 0.8beta1 through 0.8beta3) for the ADBMS, and its GAPL based on the concepts demonstrated in this article. Neither were the AspectJ compiler, nor the AspectJ and Objectivity runtimes changed. *Partial persistent weavings*, *lost through persistence weaving*s and *automatic rebindable transient weavings* are supported out of the box and heavily used.
In the last case, the class is persistent capable, while the aspect class is transient and from a 'of eachJVM' type.
An automatic rebindable weaving for transient aspect classes of the type 'of eachobject()' can be realized by modifying the AspectJ compiler. Currently, the AspectJ compiler does not allow the user to directly invoke the automatically generated methods responsible for initiating the (re)weaving.
Therefore, if rebinding is necessary for transient aspects of type 'of eachObject()' the Java Reflection API is used to bypass these restrictions.
The combination of Objectivity and AspectJ version 0.8beta3 does not currently support rebindable weavings where the aspect class is persistent capable and the class is transient.

It was further discovered in a real world example, that no aspect class of the type 'of eachcflow()' and 'of eachcflowbelow()' is currently stored in the database. Moreover, no aspect instances of the type 'of eachJVM()' which is stored in the database. However, these transient aspect instances are often used in rebindable weavings. Aspects instances of the type 'of eachobject()' are quite often stored in persistent database, most of which are stored through *persistent weavings.*

Sirius Software is due to release a proof of concept in the near future (as well as detailed website postings [7]) that will further substantiate, based on experiences, the feasibility of ADBMSs through existing commercial ODBMSs [8], [9].

Conclusions

Concrete definitions of persistent and transient aspects are required to establish and realize the concept of persistence *of aspects* in object-oriented programming. The definition in this article proved to be a solid foundation for a common wording of Sirius developers and architects in discussing and designing persistency in aspects.
The common wording is a prerequisite for pattern hatching in such an environment. It is still important to examine which patterns of persistent aspects are needed, as well as, the types of rebindable weavings that are really required to support these patterns, when applied in real world projects. The weaving relationship between an aspect and an object does not necessarily establish, or require, persistency. A dynamic weaving support from a GAPL can further propogate the use of persistent aspects. In fact, the Java Mapping of the ODMG can be easily extended to Java based GAPLs like AspectJ and closing the gap between existing commercial ODBMSs and future ADBMSs.

Biography

Arno Schmidmeier (arno.schmidmeier@sirius-eos.com) is the Chief Scientist at Sirius Software GmbH. Prior to his current position he architected the EOS ® SLM Solution. He is the technical representative of Sirius Software in the TMF. He is also an independent member of the Java Specification Request 0090 'OSS Quality of Service API'.

References

[1] "On to Aspect Persistence", Awais Rashid, Proceedings of Second International Symposium on Generative and Component-based Software Engineering GCSE 2000 (part of Proceedings of NetObjectDays2000), pp. 453-463 (Also to appear in post symposium proceedings published by Springer-Verlag)
[2] AspectJ Home Page, http://aspectj.org/, Xerox PARC, USA
[3] Cattell, R. G. G., et al., "The Object Database Standard: ODMG 2.0", Morgan Kaufmann, c1997
[4] Objectivity Home Page, http://www.objectivity.com/, Objectivity inc. Mountain View, USA
[5] Cattel, R.G.G., et al, "The Object Database Standard: ODMG 3.0", Morgan Kaufmann, 2000
[6] Gregor Kiczales, Erik Hilsdale, et al, "Language Semantics", http://aspectj.org/doc/primer/ref/semantics.html,
[6] "Jasmine 1.21 Documentation", Computer Associates International, Inc., Fujitsu Limited, c1996-98
[7] Sirius Research AOD Page, http://www.sirius-eos.com/
[8] Versant Home Page, http://www.versant.com/, Fremont CA, USA
[9] Fast Objects by Poet, http://www.fastobjects.com/
[10] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995
[11] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, William G. Griswold. An Overview of AspectJ. To appear in ECOOP 2001, 2001