2402

Declarative Meta Programming to Support Software Development

Workshop Proceedings

ASE 2002, Edinburgh, UK (23/09/2002)

Tom Mens
Programming Technology Lab
Vrije Universiteit Brussel, Belgium
tom.mens@vub.ac.be

Roel Wuyts
Software Composition Group
University of Bern, Switzerland
roel.wuyts@iam.unibe.ch

Kris De Volder
Department of Computer Science
University of British Columbia, Canada
kdvolder@cs.ubc.ca

Kim Mens
Département d'Ingénierie Informatique
Université catholique de Louvain, Belgium
kim.mens@info.ucl.ac.be

A Declarative Persistency Definition Language

Muna Matar, Institute for Continuing Education
Ghent University, Belgium
Koenraad Vandenborre
Inno.com cva Belgium
Ghislain Hoffman, Herman Tromp
Department of Information Technology, Ghent University, Belgium

Abstract

This paper presents the Persistency Definition Language (PDL). PDL is a declarative language that is embedded in Javadoc style tags in order to introduce persistency metadata information in Java classes.

Introduction

Persistency is an object-oriented programming technique that deals with the ability of objects to exist beyond the lifetime of their creator or user. Persistency is usually implemented by preserving the state (attributes) of an object during its lifetime in a database, in most cases possibly in a relational database (RDBMS). The RDBMS technology is robust and widespread.

Designing software to connect an object-oriented software system to a relational database is a tedious task. Object structure and the table based relational technology are two widely differing paradigms. Hence, bridging the gap between the two worlds by mapping objects to database tables is often a task that takes a lot of effort, and is often too pervasive at the different layers of a multi-tiered architecture [2] [3].

Java offers introspection to obtain information about objects and classes at run-time. Unfortunately, introspection, although a powerful feature, is limited to gathering information about class structure and attribute values.

Meta information about classes that is related to making instances of those classes persistent can not be found from class code. Persistency related issues are impossible to declare in Java. This shortcoming of the language lead us to come up with another "declarative language" that helps to identify and declare persistency related information. This language is called the Persistency

Definition Language or simply, **PDL**. PDL integrated with the introspection feature in Java provides a complete persistent description of persistent classes.

Related work

The need to provide auxiliary information for program elements appears to be growing. Information about fields, methods and classes as having particular attributes that indicate they should be processed in special ways by development tools, deployment tools, or run-time libraries is called annotations metadata [5]. The JSR 175, a metadata facility for the Java programming language [4] and the Sun Java Data Objects (JDO) [5] discuss this issue in details.

To describe how persistence related issues and information can be decoupled from class implementation, the Aspect Oriented Programming paradigm (AOP) can be used [7]. See also (http://aspectj.org).

The work of K. Vandenborre and others, described in [6], gives a description of a general methodology which illustrates how persistence can be made orthogonal from the class library by using the AOP.

What AOP presents in terms of solving some issues related to persistency is very valuable. But, even with using AOP, we still need a declarative mechanism by which we can introduce persistency related information into class code which AOP does not provide. That is why the need for PDL was still valid.

PDL

When working with persistency related issues like building a framework to store Java objects, we were confronted with the weak declaration mechanism in Java.

The main problem that we were faced with when using introspection and the reflection API was that declaring many of the persistency issues was not possible in Java. Java is not declarative enough to do so. Java allows only transient and non-transient to be declared and there is no way to incorporating detailed persistence information.

PDL was developed in the context of a storage framework called **PDLF**. The PDLF is an all Java object-relational declarative framework that enables Java objects to be persisted to relational databases. This framework provided persistency description of classes as well as the capability for objects to be stored and retrieved from a relational database. The full description of this storage framework is out of the scope of this paper and can be found in [1].

As mentioned above, PDLF makes Java objects capable of storing and retrieving themselves from a relational database. And since the gap between Java objects and the table based relational structure is very wide, a detailed persistency description of classes and attributes had to be provided somehow to the PDLF to help map objects into relational tables and therefore make objects capable of being made persistent.

An example of one of the persistency issues needed that could not be declared in Java, would be how to identify an object accessor. An Object accessor is an attribute of the object that can be used to access objects with in the database i.e query the database with. Normally an object accessor is an attribute of the class. When an attribute is declared in any Java class, using introspection, certain information about the nature of the attribute and the dynamic accesses to it (e.g. name, type, value, etc.) can be found. This kind of information indicates nothing about a possible use of the attribute in a certain application.

PDL was introduced to cover the shortcomings of Java in terms of persistency aspects. It helps classify persistency aspects as well as provide a persistency description of a class. The main advantage of PDL is to decouple the persistency aspects and description of a class from the class implementation.

PDL is a tagged based extensible language. This means that all declarations are done through using PDL tags. PDL tags are Javadoc style tags that are added to the source code of every persistent class.

PDL can be considered to be a Domain Specific Language (DSL) [8] in its general purpose and form. It is an embedded and a declarative language. Implementing DSLs as embedded languages is a well known technique.

PDL Tags

As described above, PDL is a descriptive tagged based language. It makes use of tags which are Javadoc style tags. Identifying what tags needed came from identifying what persistency aspect were needed and were not possible to cover using Java. After an extensive search and running some tests on some applications, twelve PDL tags were introduced. Those tags are: *@database*, *@table*, *@major*, *@minor*, *@persistent*, *@state*, *@accessor*, *@unique*, *@index*, *@contained*, *@size*, and *@compType*.

PDL tags can be classified as follows:

- versioning tags: which are tags that define the class version. They consist of the *@major* and *@minor* tags.
- mapping tags: which are tags that have to do with mapping classes and attributes to database tables. They consist of the *@persistent*, *@database*,

and *@table* tags.

- retrieval tags: which are tags used in queries. They include the *@accessor* tag.
- internal state tags: which are tags that describe the internal structure of objects. They consist of the *@state*, *@size*, *@contained*, and *@compType*.
- table design tags: which are tags that help in designing table columns. They include the *@unique* and *@index* tags.

Since defining new needed tags is the core of PDL, we can say that PDL is closely related to XML. In XML one can define his/her own set of tags. PDL lets one identify persistent aspects of classes using meaningful tags and it lets one add information (meta-data) about each aspect. PDL is also flexible in the sense that new needed tags can be defined and added. We have to keep in mind that PDL was developed to add persistency meta information to classes so any new added tags must be relevant to the purpose of finding PDL.

A detailed description of each of the PDL tags can be found in [1].

Using PDL Tags

PDL tags are added to the source file of any persistent class. They are embedded within a Javadoc comment. And since PDL tags are of Javadoc style, they are situated preceding attributes they provide persistency aspects to. Those tags act like trigger points in the source code. In addition to that, those tags provide a structure that can be processed by a special tool.

Example 1 below illustrates the way PDL tags are inserted into the source code of any persistent class.

Example 1:

```
package MyApplication;

    /**
     * @database "Company"
     * @table "Employee"
     * @major 01
     * @minor 00
     */
public class Employee {
      public static ClassVersion classVersion = new ClassVersion("01","00");

    /**
     * @persistent
```

```
     * @accessor
     * @index
     * @unique
     */
    private Name empName;

    /**
     * @persistent
     * @contained
     */
    private Address address;

    /**
     * @persistent
     * @accessor
     * @index
     * @size 10
     */
    private ByteField jobTitle;

    //constructor
    public Employee(){
    }

//other constructors and methods go here
}
```

Processing PDL Tags

To parse the PDL tags in the source files, a tool (PDL processor) has been developed. This tool uses a special Javadoc doclet. The tool takes a source file with PDL tags in it and produces an XML file. This XML file contains a persistency description of the persistent class.

It must be clear here that Javadoc and XML are used in the PDLF context merely as tools. XML as the universal format for structured documents and data on the Web is modular, easy to read and most important easy to parse. Through parsing an XML document an application can make the data available in different formats.

The PDL processor mentioned above makes use of an XML parser which helps generate SQL code that is used by the PDLF for different purposes such as creating database tables and storing and retrieving objects from those tables.

Running Javadoc on the source code presented above in example 1 with the special doclet (PDL processor) will produce the following XML file.

```
<?xml version='1.0'?>
```

```
<!DOCTYPE ClassLibrary >
 <classDescriptor Class="MyApplication.Employee">

   <classVersion major="01" minor="00">
   </classVersion>

   <db database="Company " table="Employee">
   </db>

 <pAttribute accessor="true" index="true" unique="true" contained="false">
      <attributeOfClass>
        MyApplication.Employee
      </attributeOfClass>
      <attributeName>
        empName
      </attributeName>
 </pAttribute>

 <pAttribute accessor="false" index="false" unique="false" contained="true">
      <attributeOfClass>
        MyApplication.Employee
      </attributeOfClass>
      <attributeName>
        address
      </attributeName>
 </pAttribute>

 <pAttribute accessor="true" index="true" unique="false" contained="false">
      <attributeOfClass>
        MyApplication.Employee
      </attributeOfClass>
      <attributeName>
        jobTitle
      </attributeName>
      <size size="10">
      </size>
 </pAttribute>

</classDescriptor>
```

Obtaining the XML file was one of two stages any persistent class needed to go through to be able to register with the PDLF. When a class registers with the PDLF, instances of this class can be made persistent in addition to the fact that a complete persistence description of that class is added to the central repository of the PDLF. This repository is a metadata container of persistent classes and their mappings to database tables. This repository is heavily used when reading and writing objects to database tables.

Conclusion

What we have presented in this paper is an extensible langauge, PDL, which can be used to declare persistency metadata information in classes. Due to the extensibility nature of this language, it can be also used to introduce other meta information about classes beside persistency class information.

It is important to note here that PDL emphases the idea of decoupling persistency issues from class implementation issues. It provides a single point of reference to persistency related information within a persistent class.

Providing this information using Javadoc style tags made it easy for us to present a persistent description of classes and process it using many of the available tools and technologies like XML.

References

[1] Muna Matar. A methodology for object persistence in Java based on a declarative strategy, Ph.d Thesis., Ghent Univesity, 2001.

[2] Peter Kroha. Objects and Databases. McGraw-Hill Publishing Company, 1993.

[3] S. Agarwal, A. Keller., and R. Jensen., Bridging object-oriented programming and relational databases. 1993

[4] Java Community Process, JSR 175, www.jcp.org/jsr/detail/175.prt

[5] Java Data Objects, JSR 12. Craig Russell. Sun Microsystems Inc.

[6] K. Vandenborre., M. Matar., and G. Hoffamn. Orthogonal persistence using Aspect Oriented Programming. The proceedings of the first AOSD workshop on Aspects, Components and Patterns for infrastructure software, April 2002.

[7] G. Kiczales, J. Lamping, A.Mendhekar, C. Maeda, C. Lopes, J. Loingtier, and J. Irwin. Aspect-Oriented Programing. The proceddings of the European Conference on Object-Oriented Programing, June 1997.

[8] A. van Deursen, P. Klint, J. Visser. Domain-Specific Languages: An Annotated Bibliography. ACM SIGPLAN Notices, June 2000.