JOXM: Java Object XML Mapping

Huffman, M., Hua, J., Lee, J.-S., and A. Dukovich
{mhuffman, jhua, jslee, adukovic}@calpoly.edu
Department of Computer Science
California Polytechnic State University, San Luis Obispo

Introduction

Patrick Connor Linskey and Marc Prud'hommeaux define Object-Relational Mapping (ORM) as "the technique of converting records in a relational database into objects in an object-oriented programming environment." [9] This technique has grown more popular in recent years, allowing programmers easier use of relational databases. Object-relational mapping allows for two-way conversion between relational data and objects. One benefit of this paradigm is that the programmer does not need to know anything about databases in order to take advantage of them—a mere working knowledge of the ORM is sufficient. Other benefits of ORM stated by Douglas Minnaar are a boost in productivity (less code to be written, less code to test and maintain), increased ease of maintenance, and enhanced database independence, can foster Object-Orientated domain thinking, and will help if the application exhibits a typical read-modify-write lifecycle [10]. An ORM does all of these things by abstracting the data layer (in this case the database) and putting it into a framework so that it is much easier for a programmer to work with databases. By abstracting databases into something that is familiar to the programmer, they can be integrated seamlessly into software. Since programmers do not need to learn a completely new concept and can just use an ORM, they would be able to save time by writing less code, and will experience less of a learning curve (depending on ORM).

One of the most popular ORM technologies for interacting with relational databases is Hibernate [1]. Its popularity derives from its simplicity and abstraction of interactions with the database. Hibernate additionally provides its own querying language, the Hibernate Query Language (HQL), that allows querying to be performed on objects instead of relational data. Because of these features, Hibernate has become a very popular method of persistence (the process by which data from objects is stored in a database) and has been used by many software companies such as AT&T, Intuit and others.

XML

XML [11], or the eXtensible Markup Language, is a technology that has seen much development within the past
decade. XML facilitates interoperability between the document-oriented character of HTML and the schema-oriented character of DBMS. XML document has hierarchical structure which contains information about the relationship of data items, self-describing function which is implemented by tagging, and inherent ordering which implement the ordering of the data in document by specifying data items. By such features, users can easily store the attributes and relation of objects in the XML file, execute efficient search and retrieval of XML, and thereby, these days, XML has become a predominant standard format for manipulating information for web services. In our project, we wanted to develop better and effective method to utilize XML as Object data storage -- ORM for XML (or OXM). OXM is very important concept because it allows the developer not to need to know anything about manipulating data between XML and Object. OXM provides an easy way of converting object data to and from the structured XML data. There are native XML database such as eXist that stores the data as XML and allows searching that is unique due to the structure and characteristics of XML. In order to access data in a native database we use XQuery and XPath which is similar to SQL. These two languages allow us to query data not only by the data itself, but also allow us to query the data by the structure and hierarchy.

Problem Statement

Currently, there has yet to be developed and documented a library sufficient to support the automated storage, retrieval and querying of typed Java objects in a native XML database. There are many ORMs for relational databases; however there has not been successful ORMs for XML that is on par with ones for relational databases. At the present, working with XML and taking advantage of XML technology comes at a cost that is similar to that of relational databases. A programmer must learn and understand XML. In addition to this, the programmer must also understand how to use an XML native database and learn how to query from the database using XPath and XQuery. ORM for XML it would give us the benefits of ORM in the world of XML database management. A programmer would be able to take advantage of both the ideas behind ORM and the power of XML and use it in developing software. This, then, would allow us to persist objects to an XML native database.

Previous Attempts At XML Persistence

There have been previous attempts to persist XML data. Hibernate itself has a mechanism to do exactly this [2], and two other products, Hyperjaxb [3][4], and JaxMe [5] provide slightly different ways of accomplishing this goal. They all, however, fall short in similar ways:
They are often tied to a specific platform (e.g. JavaBeans).

They often do not persist the XML data, and merely “shred” the XML to fit in a relational DBMS. This removes many of the advantages of using semi-structured data.

The latter two products both use the technology which was never intended for use with XML databases and does not work well with them.

The last project, JaxMe, is close in scope to our work. However, it has many problems, such as incomplete documentation, an unstable code base, and poor handling of XML persistence in general there is no support for native XML technologies [6, 7].

Because of these deficiencies, a tool that persists XML data to a native database, without the need for programmer intervention, has yet to be successfully implemented.

Implementation

The Java Object XML Mapping (JOXM) library -- which we created for this project -- augments the previously described methods of handling XML data in an application by creating an automated persistence layer to persist Java objects directly to a native XML database. JOXM’s goals are as follows:

- To provide concise APIs for connection to a database, persistence of data to and from a database, and querying of data from a database,
- To allow connection to local databases through the XML:DB API,
- To create a persistence API that abstracts XML data binding (marshaling and unmarshaling) and querying,
- To manage persistence of any Java object into a native XML database, and
- To facilitate support for issuing XPath queries, returning results as typed Java objects.

As previously mentioned JOXM is a general purpose persistence library, with goals similar to those of Hibernate but placed in the context of native XML databases. JOXM aspires to make the technical details of database implementation invisible to users of the code -- as far as users of this library are concerned, they are saving Java objects to a database, which happens to use XML as its storage format. Using JOXM simply entails calling the appropriate methods of our API (add, replace, etc.) and the JOXM code will handle connections, marshalling, unmarshalling, and saving to the embedded database. While the end result will lack some of the functionality of Hibernate -- such as a XML-specific query language similar to the aforementioned HQL -- it will fill an analogous niche
in the XML world to what Hibernate accomplishes in the relational domain.

**Overview**

The JOXM architecture can be broken down into four parts:

1. **Database** - we used eXist [12], an open source native XML database.
2. **XStream** - this component is used for serializing Java objects to XML, and vice versa.
3. **Marshaller** - executes the actual conversion of Java objects to and from eXist’s native XML, with the assistance of XStream.
4. **JOXM** - this was the part we implemented. JOXM includes the code responsible for much of the administrative aspects of the program in general, such as code to create and handle a session with the database.

The JOXM core provides the persistence, connection, and querying APIs that bind an application to the XML persistence model. Instead of the application needing to maintain intimate knowledge about the location or protocols for communicating with the database, it can communicate natively through the JOXM proxy, just as Hibernate abstracts the details of the database from developers.
Conclusion

In this project, we have discussed XML and Object-Relational Mapping. XML is an extremely important and popular technology for storing documents on the web, and ORM is used to store object data within a relational framework. Despite some attempts, there is not yet an effective way of performing Object-XML Mapping (OXM). This has changed with the advent of JOXM. JOXM is a software package that facilitates Object-XML Mapping, and it addresses the needs of applications looking for a generic persistence layer supporting the XML:DB API, XPath, and XQuery standards, and especially those in which the object model is hierarchically complex. It avoids the pitfalls of earlier OXM attempts, such as an inability to handle marshalling and native databases, in ways that are entirely transparent to the programmer. For these users, an invisible persistence mechanism, like JOXM, is an unprecedented and highly useful innovation.

References