JOXM: Java Object — XML Mapping

Adam Dukovich, Jimmy Hua, Jong Seo Lee, Michael Huffman, and Alex Dekhtyar
Department of Computer Science
California Polytechnic State University
San Luis Obispo CA 93407
{adukovic, jhua, jslee, mhuffman, dekhtyar}@calpoly.edu

Abstract

Java Object XML Mapping (JOXM) is a library that supports the automated persistence and querying of Java objects in a native XML database. The goal of this library is to provide a suitable alternative to standard object-relational mapping (ORM) tools, most notably Hibernate [3]. Unlike techniques such as XML-relational persistence, the storage mechanism provided by JOXM is transparent, allowing the developer to store, retrieve, and query typed Java objects as opposed to plain XML data.

1. Introduction

The rise of the internet has presented a new paradigm of data management that has yet to be satisfactorily addressed by current technology. In a large number of applications, semi-structured data comes over the internet in XML form. The applications parse it into Document Object Model (DOM) and, as necessary, convert DOM trees into internal data models, represented by host language object instances in main memory. Applications then use object-relational mapping (ORM) to serialize and persist the data in a relational database (see Figure 1). Traditionally, use of ORM technologies, such as Hibernate [3], allows software developers to assure persistent storage of important application data, without the need to incorporate database development into the application development process and without the associated costs.

At the same time, we observe that the use of ORM to persist XML data (or semi-structured data that was essentially derived from XML) does not exploit XML's advantages as a way of encoding data. In a scenario described above, ORM software essentially shreds XML documents into RDBMS using techniques that were, generally speaking not designed for semistructured data.

Java Object-XML Mapping (JOXM) was designed to replace ORM in software applications where semi-structured data needs to be persisted. JOXM replaces RDBMS backend with a native XML DBMS. In our view, JOXM combines the benefits of using ORM, such as abstraction and simplicity of the API with no need for database development/administration activities, and the benefits of using native XML DBMS — a more suitable, and potentially more efficient storage mechanism for persisting semi-structured/XML data. While JOXM targets software working with semi-structured information, it is designed to persist *any* serializable Java object, semi-structured or not.

2. Problem Statement

There are three common approaches to handling XML data in an XML database application: ***(i)*** place XML data directly into a relational database (e.g. blob fields); ***(ii)*** convert/shred XML data into an RDBMS; ***(iii)*** store data in a native XML database as indexed content.

At present, no one approach seems to have more favor than the others in industry. The lack of sophisticated and automated tools for storing and processing XML data in databases seems to stifle its adoption in application development, especially when compared with some of the feature-rich tools, such as ORMs like Hibernate, that are available for use on relational databases. There are many ORMs for relational databases; however there has not yet been a successful tool to perform OXM (Object-XML Mapping) that is on par with those existing tools for relational databases. At present, working with XML and taking advantage of XML technology comes at a cost that is similar to that of relational databases without Hibernate. A programmer must learn and thoroughly understand XML, as well as how to use a native XML database and how to query from that database using XPath, XQuery, or XSLT. ORM for XML–OXM–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.

Of these three approaches discussed earlier, our work

978-0-7695-3261-5/08 $25.00 © 2008 IEEE
DOI 10.1109/ICWE.2008.43

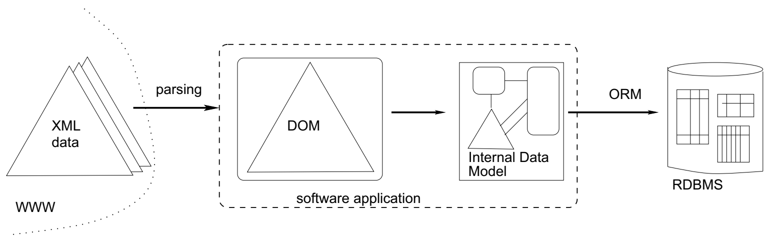

Figure 1. XML data translation to a database.

concentrates on the latter. A *native XML database* defines a logical model for an XML document/data collection, uses native XML index structures as its underlying method of storage, and does not rely on any underlying relational database for storage. Native XML databases use XPath[7] and XQuery [5] as the means of accessing stored XML data, and an emerging XQuery Update [6] standard for data manipulation in the database.

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. Unlike relational databases, which have seen decades of intensive research since the 1970s, truly functional native XML databases have emerged only in the past couple of years. Knowledge of their practicality, performance, and applicability for use in web-based software applications remains severely lacking at the present time.

3 Related Work

In this section, we will briefly discuss some of the prior attempts to perform XML persistence, as well as their strengths and weaknesses.

Hibernate. Hibernate [3] is one of the most popular ORM technologies for interacting with relational databases. Its popularity derives from its simplicity and abstraction over the interactions with the database. Hibernate relieves developers from writing SQL queries and needing to understand the details of converting data from an in-memory object model to the relational database model. Hibernate uses relational databases as its backend. Hibernate uses shredding to persist XML data (represented as Java objects) in the database. The ultimate goal of our project is to determine if using native XML DBMS back end can be more efficient for storage of semistructured data.

Hyperjaxb. Hibernate's XML-relational mapping technology is not meant to manipulate or transform (update) XML. Similarly, Hyperjaxb [8] stores XML data by shredding it into relational databases. Hyperjaxb implements the Java Architecture for XML Binding (JAXB) [9] specification and provides a mechanism to marshal and unmarshal XML content in and out of a relational database. However, this technology relies on compile-time code generation and adherence to the JavaBean object model. Worse yet, like its counterpart in Hibernate, neither of these approaches allows the abstraction of storage model (XML) from the object model (Java objects).

JaxMe. JaxMe [2] supports object-mapping to a native XML database. It uses the JAXB specification by marshaling and unmarshaling XML data to an XML database. JaxMe requires pre-known schema definitions to generate JAXB-enabled objects, which ultimately requires more work by the developer to create these definitions. In contrast, we used an annotation-based approach of defining these schemas. In our view, the latter approach saves time and effort. JaxMe was never intended for use with XML databases and it fails to abstract the persistence layer from the object model. Additionally, JaxMe appears to have been abandoned by its development team, and its reliability and technical support are questionable.

JNXD. Of all the tools we considered, JNXD [10] comes closest to JOXM. The JNXD framework takes java objects, serializes them into XML, and persists them to a native XML database, just as JOXM does. JNXD integrates an array of technologies and APIs similar to JOXM: XStream, eXist, and XML:DB API. It is our understanding, that JNXD and JOXM had been under development in parallel over the past year.

XML:DB API The XML:DB API [12] is a standard for providing a way of gaining access to data in an XML database. XML:DB can be viewed as being the XML equivalent to ODBC (for relational databases), and its use is promoted by the XML:DB Initiative. The XML:DB API specification is currently in a working draft and defines two levels of conformance, core level 0 and core level 1. The core level 0 is the base API which provides for the concepts of resources and services. Core level 1 consists of the specification for XPath querying services. Natively, the XML:DB API specification does not allow for querying typed objects. Rather, the specification provides return types of queries in the form of DOM, SAX, and text as well as binary content.

4. Implementation

The Java Object XML Mapping (JOXM) library 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 is an implementation of object-XML mapping (OXM) for the Java language. JOXM's goals are the following: ***(i)*** concise connection, persistence, and querying APIs; ***(ii)*** connection to local databases through the XML:DB API; ***(iii)*** persistence API that abstracts XML data binding (marshaling and unmarshaling) and querying; ***(iv)*** persistence of any Java object into a native XML database; ***(v)*** support for issuing XPath queries, returning results as typed Java objects.

JOXM is a general purpose persistence library, with goals similar to those of Hibernate but placed in the context of native XML databases. JOXM provides a high-level abstraction of Java object persistence, with a "hands-off" approach in implementing a persistence layer. Knowledge of the connection protocol, storage format, and XML querying (XPath/XQuery) syntax is not required. Application developers using JOXM to persist Java objects are simply making a few method calls which, in turn, save Java objects to an XML database.

4.1. Overview

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. JOXM connects to a local/embedded database using the XML:DB API, ensuring that the implementation of the XML database need only to adhere to the XML:DB specification. This layer of abstraction allows the application to swap the back-end database at any time. Besides support for connection management and transparent persistence, JOXM allows developers to issue XPath queries across its interface, with automated type conversion provided by default.

4.2. JOXM Architecture

JOXM. JOXM library is composed of several modular components, as illustrated in Figure 2. The JOXM library has one main package, edu.cp.joxm, composing two subpackages: edu.cp.joxm.annotations and edu.cp.joxm.aspects. The main package contains four classes designed to provide the core interfaces: (1) connection, (2) persistence and (3) querying.

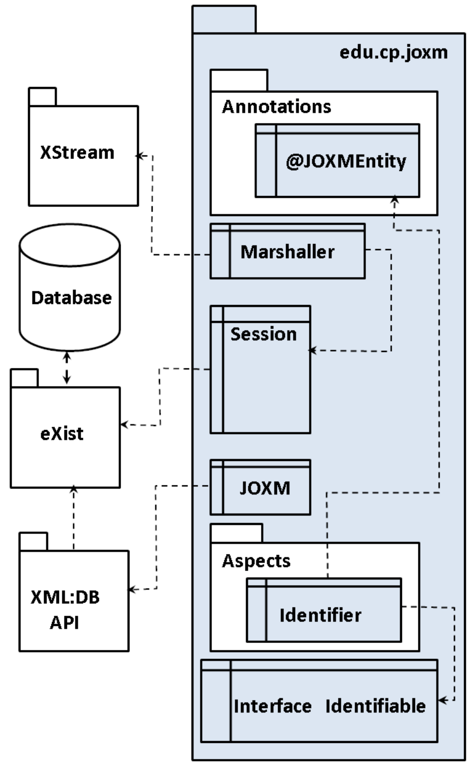

Figure 2. JOXM Architecture

eXist. The JOXM library, in its current version, stores its data in an eXist native XML database [11]. eXist provides an open-source, Java-based, native XML database that integrates several popular XML technologies, including XQuery and XUpdate. eXist also supports both embedded and server modes, although only the former is supported by the current version of JOXM. eXist conforms to the XML:DB API, using the Xindice [4] implementation. Within JOXM, eXist can be swapped for another database just as long as it adheres to the XML:DB specification.

XStream XStream [1] is a Java library for serializing objects to and from XML. Unlike the JAXB interface which requires objects to be JavaBeans, XStream allows for the marshaling and unmarshaling of any Java object. XStream relies heavily upon Java annotations to provide the metadata in order to influence and configure the conversion to and from XML. Such annotations include hints as to aliases of fields (alternate names), structure (whether attributes or elements are used), and how collections are stored. By default, the conversion to and from XML is accomplished through the use of reflection, meaning that private and internal data will be serialized, not just its public interface (as is the case in JavaBeans). Consequently, XStream is a light-weight library that requires little configuration or customization to use it ”out of the box”. In the context of JOXM, XStream is used by the Marshaller class, which acts as an proxy between XStream and the Session class. XStream is used for the purpose of serializing objects from Java to XML, and vice versa. Further details of how this operates in practice can be in [1]. Below is an example of an annotated class in order to use JOXM.

```
import edu.cp.joxm.annotations.JOXMEntity;
@JOXMEntity
public class CalEvent extends Event {
   /**
    * @param title
    * @param description
    * @param date
    * @param hasTime
    * @param allDay
    */
   public CalEvent(String title,
                   String description,
                   Date date) {
      super(title, description, date);
   } }
```

5. Conclusions and Future Work

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 prior attempts, there has not yet been an effective way of performing Object-XML Mapping (OXM)–that is, until now. 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.

We have designed JOXM to be a light-weight replacement for ORM for software applications which need to persist large quantities of semi-structured data/XML data stored as Java objects. JOXM uses a native XML backend, and provides simple ORM-style API for storage and retrieval of objects, which releives software developers from the need to work with the database back end directly.

JOXM is a work in progress. The current version can be found at http://wiki.csc.calpoly.edu/joxm. As with many research projects, much more work would need to be completed. Our current goal is to test how JOXM perfoms vs. Hibernate for applications using XML data. We have modified an open source calendar application to work with JOXM. The next steps involve building a version of the same application using Hibernate and running performance testing.

In addition, we plan to work on extending the features of JOXM in the following directions:

- Allow users to issue XQuery queries to the database;
- Build an XPath interpreter for the Hibernate Query Language (HQL) used by [3];
- Implement annotation-based specifications and customizations;
- Implement custom specification for deferred (lazy) or immediate (greedy) evaluation;
- Provice more control of the marshaling and unmarshaling contexts (such as storing inlining data as attributes);
- Provide support for the XQuery Update Facility (XQUF) (once specification is approved).

References

[1] Xstream development team. xstream - about xstream. URL http://xstream.codehaus.org.

[2] Apache software foundation. welcome to jaxme 2. URL http://ws.apache.org/jaxme, 2004.

[3] Red hat middleware. hibernate - relational persistence for java and .net. URL http://www.hibernate.org, 2006.

[4] Apache software foundation. apache xindice. URL http://xml.apache.org/xindice, 2007.

[5] S. Boag, D. Chamberlin, M. Fernandez, D. Florescu, J. Robie, and J. Simeon. XQuery 1.0: An XML Query Language. W3C Recommendation, Jan. 2007.

[6] D. Chamberlin, D. Florescu, and J. Robie. XQuery update facility. *W3C Working Draft*, 8, 2006.

[7] J. Clark, S. DeRose, et al. XML Path Language (XPath) Version 1.0. *W3C Recommendation*, 16:1999, 1999.

[8] CollabNet. Hyperjaxb3 - relational persistence for jaxb objects. URL https://hyperjaxb3.dev.java.net, 2007.

[9] CollabNet. Jaxb: Jaxb reference implementation. URL https://jaxb.dev.java.net, 2007.

[10] R. M. Gama, R. d. O. Lopes, and V. F. d. Castro. Java persistence framework for xml databases. URL https://jnxd.dev.java.net/, 2007.

[11] W. Meier. eXist: An Open Source Native XML Database. *Web, Web-Services, and Database Systems: Node 2002 Web-And Database-Related Workshops, Erfurt, Germany, October 7-10, 2002: Revised Papers*, 2003.

[12] K. Staken. XML database API draft, 2003.