

A Reusable Graphical User Interface for Manipulating Object-Oriented Databases using Java and XML

Suzanne W. Dietrich, Dan Suceava, Chakrapani Cherukuri and Susan D. Urban
Department of Computer Science and Engineering
Arizona State University
Tempe, AZ 85287-5406
{dietrich | s.urban}@asu.edu

Abstract

This paper describes the design and functionality of a graphical user interface (GUI) written in Java Swing that is used to support instructional activities associated with teaching object-oriented database (OODB) concepts. The GUI supports the manipulation of objects in an OODB, assuming the implementation of a specified interface for interacting with an OODB. By using the interface, students can focus on object-oriented design and programming concepts associated with OODB concepts rather than the development of a user interface. Since the GUI uses the Extensible Markup Language (XML) for defining the database schema and data import/export, the use of the GUI provides the added benefit of demonstrating the manner in which XML interacts with database technology.

1 Introduction

Object-oriented concepts are an important component of database education. At Arizona State University, we cover the theoretical and practical aspects of object-oriented databases in our new advanced database concepts course for undergraduates and in our graduate OODB course. Typically, the student is given an assignment to model an enterprise and then implement that enterprise using an object-oriented database product. The goal of the assignment is to illustrate the differences between the object-oriented data model and the relational data model. The implementation of this assignment necessarily requires the development of an interface for manipulating the OODB. Unfortunately, the development of this interface is a time-consuming task, even for a command-line interface.

When we were faced with upgrading our sample implementation for the students from C++ to Java, we decided to address the user interface issue by designing a graphical user interface (GUI) using Java Swing. The students can then reuse the GUI and focus on the more important lessons to be learned from the OODB implementation assignment. The GUI assumes the implementation of a specified interface for interacting with an object-oriented database. We have an implementation of this interface for Objectivity/DB [4], which is the commercial product that we use for demonstrating OODB concepts. Other OODB products can be used with the GUI if an implementation of the expected interface is provided.

The initial implementation of the GUI [5] provided the ability to add, edit, and delete objects and the relationships between objects. Some useful features, however, were lacking from the initial implementation, such as data import, data export, printing the database contents, and support for predefined queries. The second version of the GUI [3] uses the Extensible Markup Language (XML) [6] to support the data import and export capabilities, the Extensible Stylesheet Language (XSL) [7] to display and subsequently print the exported data from a browser, and XML to define the database schema and query interfaces.

The use of the GUI in the OODB implementation assignment allows students to focus on OODB concepts instead of coding a user interface. The GUI also provides additional educational benefits, exposing the students to advanced technologies, such as XML and XSL, and their interaction with database systems.

This paper describes the reusable GUI for the manipulation of OODBs. Section 2 overviews the software architectural design. The user interface is covered briefly in Section 3. Section 4 provides an overview of XML and how XML is used to define the database schema for use by the GUI. Section 5 documents the use of XML for data import and export and discusses the approach to printing exported data using XSL. Section 6 briefly describes the support of predefined queries, and Section 7 concludes the paper with a summary of its contributions.

2 Design

Figure 1 illustrates a high-level package view of the design of the GUI. The `xml.metadata` package is responsible for parsing the XML defining the database schema and verifying the metadata against the Java metadata using the Java Reflection API. The `gui` package interacts with the `xml.metadata` package to retrieve the necessary information for the user interface. The `gui` also calls the `oodb` package, which defines an implementation of an interface for interacting with an underlying OODB product. For example, this `oodb` interface includes behavior to open and close a database, to commit and abort transactions, and to retrieve, add, edit or delete objects. The `xml.impexp` package supports the data import and export features, interacting with both the `xml.metadata` and the `oodb` packages. Figure 1 also shows the interaction of the `oodb` package with the underlying `com.objy` package providing the API for the Objectivity/DB product. The application specific package provides the implementation for the application objects. The `oodb` package assumes that each application object implements an interface that includes specific methods in support of data validation.

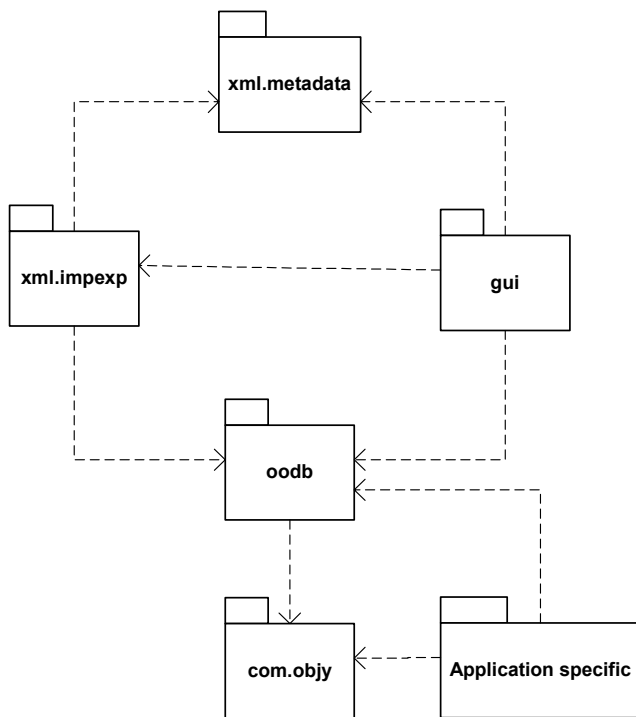


Figure 1 GUI Design

Before the introduction of the GUI, the students worked in groups to develop an application using an OODB product. The students generated a lot of code to implement a command-line interface just to drive the sample application. With the GUI, the class assignment can now be an individual project, where the student can focus on writing the application specific objects and the integrity

constraints that must hold on those objects. Since the GUI needs to know the schema of the underlying database, the student must also specify the object-oriented schema of the database to the GUI using XML. Section 4 describes the format of the XML file for defining the database schema.

3 User Interface

The user interface is implemented using Java Swing. Figure 2 shows the main screen of the GUI, once a database has been opened. The name of the currently opened database is displayed. The GUI displays the classes that are defined in the database, and the displayable attributes for the objects that are members of the selected class. The menu options include Add a new object to the selected class; Edit, Delete or View the selected object; Export DB to export the database instance to a named XML file; Query the database; and Close the database.

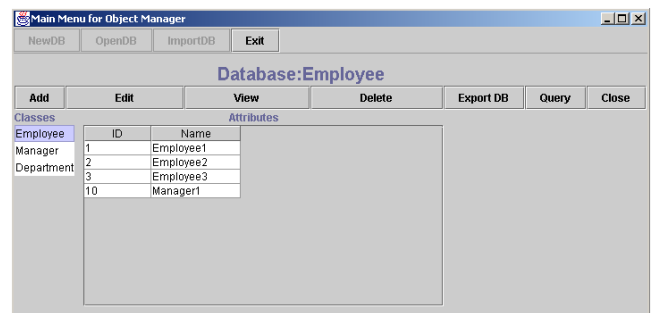


Figure 2 Main Window of the User Interface

When selecting a specific object to add, edit or view, a separate window pops-up, as shown in Figure 3, to specify the values of the attributes and relationships for that object. Note that the GUI displays the value of a single-valued relationship using a combo box. The value of a multi-valued relationship is labeled as those objects that are Related, and those objects that are Not Related. The < and > boxes facilitate the assignment of the multi-valued relationship. When the OK button is clicked, the GUI calls the `validateAddEdit` function that must be defined for every application object.

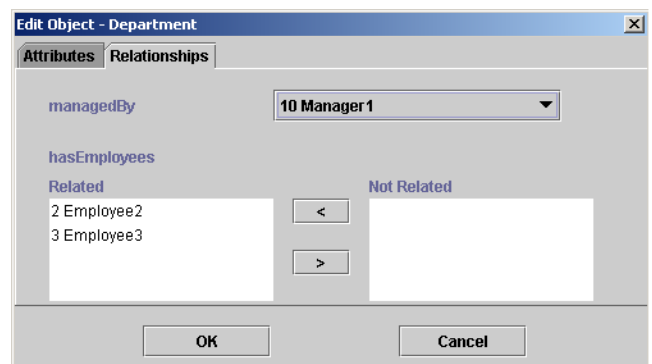


Figure 3 Attributes and Relationships Window

4 Data Definition Using XML

XML forms the basis of the data definition for the GUI. XML is a standard adopted by the World Wide Web Consortium (W3C). XML is similar to HTML, consisting of textual data scoped by tags. HTML has fixed tags that describe the presentation of the text. XML maintains the simplicity of HTML but describes the content of the textual data, allowing for the definition of new tags and the nesting of matching tags (to arbitrary depth). The component of XML consisting of the matched tags and the text enclosed within the matched tags is called an *element*. The following example XML element describes an employee, having a name and an identification number:

```
<employee>
  <name>John Smith</name>
  <id>123456789</id>
</employee>
```

A Document Type Descriptor (DTD) can be used to define the expected structure of an XML document. The DTD syntax annotates the expected structure using symbols to indicate the number of occurrences of the element, where + indicates one or more occurrences, * indicates 0 or more occurrences and ? indicates an optional element specification (0 or 1). If no symbol is specified, then the element must appear exactly once. Figure 4 gives the DTD for the GUI's DDL specification in XML. The schema definition for the GUI consists of one or more class elements, which include descriptions for class information, attributes and relationships. Some tag names for the DDL are loosely-based on the ODMG's Object Definition Language [2].

```
<!ELEMENT Schema (Class)+>
<!ELEMENT Class (ClassInfo, Attribute*, Relationship*)>
<!ELEMENT ClassInfo(name, extends?, isAbstract?, keys)>
<!ELEMENT keys (key+)>
<!ELEMENT Attribute (label, name, type,
  getMethodName, setMethodName?,
  isDisplayable, isVirtual?)>
<!ELEMENT Relationship (label, name, relType,
  getMethodName, setMethodName?,
  addMethodName?, removeMethodName?,
  getValuesMethodName)>
<!ATTLIST relType values(single|multiple) #REQUIRED>
<!ATTLIST relType inverse CDATA #IMPLIED>
```

Figure 4 Data Definition DTD

Each ClassInfo element provides the name of the class, whether it extends another class and whether the class is an abstract class. These elements contain only textual data, which is defined in a DTD as:

```
<ELEMENT name (#PCDATA)>
```

PCDATA means Parsed Character Data, which is designed to facilitate the exchange of data in many languages. To save space in the DTD description of Figure 3, the simple PCDATA elements are not shown.

An Attribute element describes an attribute of a class. Some of the elements describing an attribute are used only by the GUI and other elements are used for data import and export. The label element indicates the label that the GUI uses when displaying the attribute. The name and type elements are straightforward, specifying the name and type of the attribute. Since an attribute value must be retrieved and assigned, the getMethodName and setMethodName elements specify the names of methods to get and set the attribute value, respectively. The isDisplayable element indicates whether the attribute is included in the abbreviated object display in the GUI.

An attribute may be virtual as indicated by the isVirtual element nested within the attribute element. There are situations in the GUI where it is useful to display information that is not explicitly stored in an object. For example, an object may contain a reference to another object, which is not directly displayable. A virtual attribute provides a mechanism by which a method can be provided to dereference an object reference and display its value.

A Relationship element describes an association between classes. A relationship has a label for the GUI, a name and type. XML elements may include attributes, which are indicated by the ATTLIST tag. The relType element has a required descriptive attribute called values to indicate whether the relationship has single or multiple values. The inverse of a relationship is also specified using an attribute of the relType element. The inverse is of type CDATA, which is a string, and #IMPLIED indicates that the attribute is optional. A relationship also has methods for retrieving and setting its value, which is declared by the getMethodName and setMethodName elements. The optional addMethodName and removeMethodName elements specify the methods to add and remove objects for multi-valued relationships. The getValuesMethodName retrieves all possible values of the type given by the relType element.

Figure 5 provides a UML diagram of an abbreviated EMPLOYEE-MANAGER-DEPARTMENT example, which will be used to illustrate the data definition features. A sample instance of the XML data defining the employee class for this example is given in Figure 6. An employee has attributes id, name and salary, and a single-valued worksFor relationship having type Department.

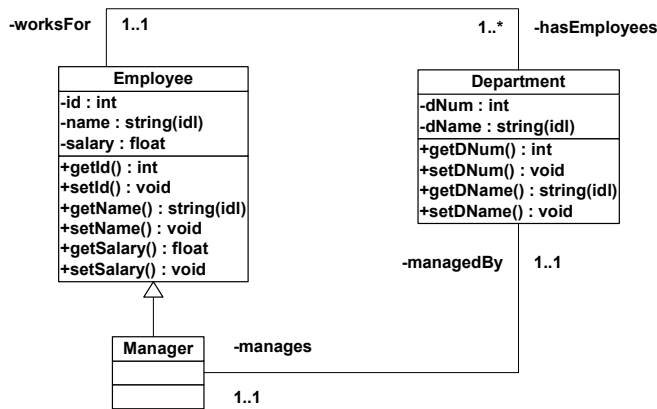


Figure 5 Abbreviated Employee Example

The GUI uses the metadata from the result of parsing the XML schema definition to provide an interactive environment to browse the database contents, create new objects, edit existing objects and delete objects. The latest version of the GUI also provides support for data import and export.

5 Import and Export Using XML

The import and export capabilities are important features of the GUI. An instance of the data can be given to students to standardize the contents of the database to facilitate the assessment of the assignment. A predefined scenario can be scripted and the database contents at the end of the execution of the scenario can be exported to XML. The exported data can then be displayed in a browser and printed to verify the results.

Figure 7 provides the DTD for the data import and export capability, describing a generic Data element consisting of any number of Objects. An object is described by its class, its oid, and the values of its attributes and relationships. An Object element has a required oid attribute, which provides a unique id to reference the object. Both attributes and relationships have a name and a value. The value of a relationship is a collection of object references, which must be specified as the required value of the objRefs attribute of the EMPTY relValues element. The objRefs value is a space separated list of (one or more) references to oids. For relationships that have a null value, the export utility will not include a Relationship tag in the output XML, and upon subsequent import, will initialize the relationship value to null by default.

Figure 8 gives the XML for an instance of an employee object that conforms to the data DTD specification of Figure 7. The employee with id 1 named Employee1 earns 50000 and works in the department that is uniquely identified by the object reference "O4". Since the relValues is an EMPTY element, XML allows an abbreviated matching tag.

```

<Schema>
  <Class>
    <ClassInfo>
      <name>Employee</name>
      <isAbstract>false</isAbstract>
      <keys><key>id</key></keys>
    </ClassInfo>
    <Attribute>
      <label>ID</label>
      <name>id</name>
      <type>int</type>
      <getMethodName>getId</getMethodName>
      <setMethodName>setId</setMethodName>
      <isDisplayable>true</isDisplayable>
    </Attribute>
    <Attribute>
      <label>Name</label>
      <name>name</name>
      <type>java.lang.String</type>
      <getMethodName>getName</getMethodName>
      <setMethodName>setName</setMethodName>
      <isDisplayable>true</isDisplayable>
    </Attribute>
    <Attribute>
      <label>Salary</label>
      <name>salary</name>
      <type>double</type>
      <getMethodName>getSalary</getMethodName>
      <setMethodName>setSalary</setMethodName>
      <isDisplayable>false</isDisplayable>
    </Attribute>
    <Relationship>
      <label>worksFor</label>
      <name>worksFor</name>
      <relType values="single" inverse="hasEmployees">
        Department</relType>
      <getMethodName>getWorksIn</getMethodName>
      <setMethodName>setWorksIn</setMethodName>
      <getValuesMethodName>getWorksInValues
        </getValuesMethodName>
      </Relationship>
    </Class>
    .....
  </Schema>
  
```

Figure 6 Employee class definition in XML

```

<!ELEMENT Data (Object)*>
<!ELEMENT Object (objClass, Attribute+, Relationship)*>
<!ATTLIST Object oid ID #REQUIRED>
<!ELEMENT Attribute (attrName, attrValue)>
<!ELEMENT Relationship(relName,relValues)>
<!ELEMENT relValues EMPTY>
<!ATTLIST relValues objRefs IDREFS #REQUIRED>

```

Figure 7 Data DTD

The GUI exports the data to an XML format using the Data DTD given in Figure 7. The exported data can be used to later import into another copy of the database, as students will use a standardized database instance for their assignment. The exported data can also be displayed in a browser using XSL, which provides templates for the translation of XML documents. In this case, the XSL specifies templates to transform the XML into HTML for presentation in a browser. The browser's print capability can then be used to print the exported data.

XML and XSL are based on a semi-structured data model [1]. Because of its similarity to HTML, XML has an intuitive meaning and is relatively straightforward to understand. However, XSL is more complex, providing mechanisms to recursively apply templates to the nodes of the semi-structured data to provide a presentation of the XML data in HTML. The students will be provided with sample XSL templates for printing exported data.

6 Querying the Database

XML is also used to integrate the specification of predefined queries that the GUI can invoke. The XML defines a query with a label to include in the GUI and the name of the method to call to answer the query, given by the queryMethodName element. The queryResultType element specifies the result type of the query, and the displayable attributes for that class are shown in a query result window. The following is a sample query specification in XML:

```

<query>
  <label>Find Computer Science Employees</label>
  <queryMethodName>getCSEmps</queryMethodName>
  <queryResultType>Employee</queryResultType>
</query>

```

7 Summary

This paper has presented a reusable GUI for the manipulation of objects stored in an OODB. Students have successfully reused the GUI in a class assignment that explores the Objectivity/DB object-oriented database product. The GUI will eventually be made available for educational use as part of the dissemination of the curriculum materials developed for a grant sponsored by the National Science Foundation to develop a second database course for undergraduates.

```

<Data>
  <Object oid="O1">
    <objClass>Employee</objClass>
    <Attribute>
      <attrName>id</attrName>
      <attrValue>1</attrValue>
    </Attribute>
    <Attribute>
      <attrName>name</attrName>
      <attrValue>Employee1</attrValue>
    </Attribute>
    <Attribute>
      <attrName>salary</attrName>
      <attrValue>50000.0</attrValue>
    </Attribute>
    <Relationship>
      <relName>worksFor</relName>
      <relValues objRefs="O4"/>
    </Relationship>
  </Object>
</Data>

```

Figure 8 Data Instance for Employee Example

Acknowledgements

This work was partially supported by the National Science Foundation's DUE CCLI-EMD program (DUE-9980417).

References

- [1] Abiteboul, S., Buneman, P. and Suciu, D., *Data on the Web: From Relations to Semistructured Data and XML*, Morgan Kaufmann, 2000.
- [2] Cattell, R. G. G. *The Object Database Standard: ODMG 3.0*, Morgan Kaufmann, 2000.
- [3] Cherukuri, C., *Integrating XML with a Graphical Object Manager for Object-Oriented Databases*, MCS Project, Arizona State University, December 2000.
- [4] Objectivity/DB Version 5.1, Objectivity, Inc., Mountain View, CA.
- [5] Suceava, D., *An Object Manager for Object-Oriented Databases*, MCS Project, Arizona State University, December 1999.
- [6] The World Wide Web Consortium (W3C)'s XML web page, 2000. <http://www.w3.org/XML/>
- [7] W3C's XSL web page, 2000. <http://www.w3.org/Style/XSL/>