JGraph Adapter Notes

Table of Contents

Introduction

This example demonstrates how to implement a custom model for a transactional backend (such as a database). It takes into account the command history (ie it is notified on all changes *including* undo and redo) and hooks into the graph model to notify the backend of all changes. It is possible for the backend to *not* accept certain changes.

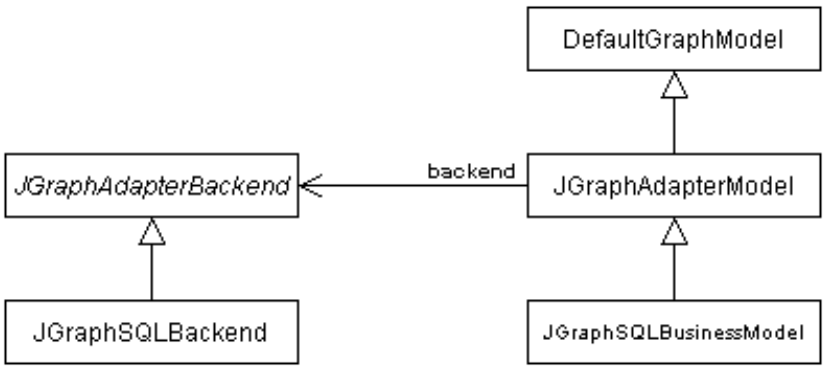
In order to take full advantage of this example you must download a JDBC driver. The default database is HSQLDB, which may be downloaded from http://hsqldb.sourceforge.net [http://hsqldb.sourceforge.net]. Note that the example also runs without a database connection, but the query window will not produce results without a database.

To enable the HSQLDB database, you must uncomment the following lines in JGraphAdapterExample.main:

```
Class.forName("org.hsqldb.jdbcDriver");
conn = DriverManager.getConnection("jdbc:hsqldb:" + backendFilename, "sa", "");
```

Participants

Figure 1. Adapter Participants



The main class is the JGraphAdapterModel, a DefaultGraphModel extension which will be used as the graph model. The JGraphAdapterModel has a reference to a backend which implements the JGraphAdapterBackend interface. The interface provides the methods which the graph model requires to keep the business model in sync.

In the example, the JGraphSQLBusinessModel acts as the graph model and the JGraphSQLBackend is in charge of updating the database based on the notifications that the business model sends to the backend.

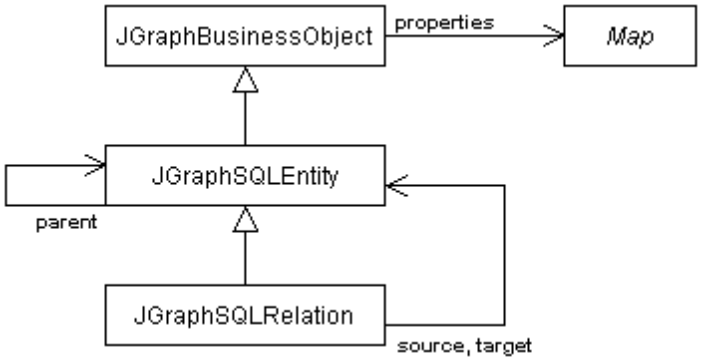
Validation

All methods (except for the commit and rollback methods which are used to mark transaction boundaries) in the JGraphAdapterBackend have a validate boolean parameter. If this parameter is true then the backend is expected not to perform the actual changes, but to check whether the changes are valid and throw an exception if they are not.

If an exception is thrown during a transaction (a non-validating invocation-sequence), then the rollback method is invoked. Otherwise, after all invocations in the sequence, the commit method is invoked.

Business Objects

Figure 2. Adapter Business Objects



The user objects in the example application are JGraphBusinessObjects. A JGraphBusinessObject is an object with an arbitrary number of properties stored in a hashtable. The object has two subclasses: The JGraphSQLEntity is used to represent vertices and groups, and the JGraphSQLRelation represents edges in the business model. (Note that the parent-child relation and the source and target of the edge is stored not only in the graph model, but also in the business model and that the ports are created on the fly, one per vertex/group.)

The mapping from business objects to cells is implemented in the JGraphAdapterModel. While it is possible to map multiple business objects to one cell, it is *not* possible to map one business object to multiple cells. The mapping of multiple objects to one cell may be used to compose information from various sources (aka backends). For example one could implement a backend for an LDAP server and one for a project database, and compose a Person-cell out of the data from both systems. To implement such a setup, you'd have to implement a multicast-backend which is composed out of a set of other backends and manages the invocation of their methods.