Restructuring AZDBLab

The Science of Databases

11/21/2009
Version 3.0
Adam Robertson

Table of Contents

Introduction

AZDBLab is a Java application that enables its users to perform experiments on databases and record the results. These results are then used to test new theories relating to databases. It works very well in most cases, but is lacking in others. AZDBLab is a complex system that at times seems impossible to comprehend or add to. The purpose behind this project was to reorganize AZDBLab to allow future developers to understand and expand upon it faster, as well as increasing the amount of useful documentation. Improving the documentation included creating several documentation files and UML diagrams of the code.

UML Editors

One of the first hurdles in restructuring AZDBLab was creating several UML diagrams of the current system. Before this could commence, a UML editor was required, as the existing editor, Violet, couldn't handle large diagrams very well. When a Violet diagram was closed, all of the elements inside that diagram were rearranged. When this happened, Violet usually ended up stacking most of the elements in the diagram on top of one other. The only known workaround for this was to just leave Violet open. Another thing that Violet lacked was the ability to copy and paste multiple elements in or across diagrams. Although these problems didn't occur or weren't much of a hindrance in smaller projects, it was clear that for diagramming a system as large as AZDBLab, another UML editor was required.

When looking for a new UML editor there were several features that were considered important. First and foremost among these was the humble copy and paste. This feature was believed to help save time by allowing similar content to be copied between diagrams, specifically during the phase of the project when the diagrams where being maintained. Another feature deemed useful was the ability to support diagrams based off of preexisting code. There wasn't really a clear picture of what this support might entail, as editors ranged from being able to create entire diagrams from existing code to not supporting existing code at all.

BOUML

The first UML editor which was reviewed was BOUML. It was easy to pick up and use, with one exception. When a class was placed inside a package, the class could not be moved. This problem was later solved via the BOUML forums on source forge. The way to do this was to move the package to the back. This was done by left clicking and selecting move to back. BOUML supported copying, pasting and could generate class objects based on preexisting code as well as code from UML diagrams in several languages, including java. This essentially means that for each class, BOUML would fill in all of the methods and fields so that this class would correspond with the code. However, it was an open source project, developed by a one person. This made using this editor risky, as the owner of the project could decide to drop it and then all of the diagrams created in this editor would be essentially useless.

Netbeans

While looking into BOUML, another UML editor was discovered. Netbeans had a plugin that let its users create UML diagrams. This plugin provided the ability to copy and paste, both inside and across diagrams. Like BOUML, it produced the class objects used in the UML diagrams. However, like Violet, Netbeans had some issues with larger UML diagrams. Firstly it slowed down considerably when the diagrams got larger. Secondly it had some trouble exporting large diagrams to images. The most alarming issue with Netbeans was that it experienced some trouble loading diagrams after saving them. Fortunately this was only an issue on Ubuntu, and this issue was resolved with the Netbeans 6.7.1 release. There was never any trouble with this on Windows.

Eclipse

After looking at Netbeans, it was discovered that Eclipse also had a modeling plugin. This was not given much consideration as at this point the first two UML editors seemed sufficient.

The choice between UML editors boiled down to BOUML, and Netbeans, with Eclipse as a secondary option. BOUML was much faster than Netbeans, and it also had less trouble exporting larger diagrams. Netbeans had more developers on its side, and it also had more

features. The real deciding factor was that Netbeans was supported by a team of developers, while BOUML just had one person working on it. This was considered an issue because if the developer for BOUML stopped supporting it, then all of the work done on the diagrams would be rendered useless, thus Netbeans was chosen as the UML editor for AZDBLab.

UML Diagrams

The next step was deciding exactly how much detail should be in the diagrams. Since there wasn't a best answer for this, it was decided that three diagrams would be created. The least detailed one, level 0, would just show the packages, with none of the classes inside of them. The next diagram, level 1, would have packages and classes, but it wouldn't show any of the methods or fields of the classes. The last diagram, level 2, would show all of the packages, classes, methods, and fields. This design was chosen to ensure that for all circumstances there would be diagram with the appropriate level of information.

To improve readability, all of the packages and classes should be in roughly the same position in each of the diagrams. This would make transitioning from one diagram to another easier. The code in AZDBLab was divided into several parts: Model, View / Control, and Plugins. Therefor it was logical to divide the UML diagrams in a similar fashion. They were divided into three columns, with plugins on the right, the model in the middle, and view / control on the left. There were several packages that didn't fit into any of these categories. These packages were placed in the same location below the three main categories.

Some problems with Netbeans occurred while making the largest UML diagram that were not apparent during the initial testing. Exporting large diagrams to images over a certain amount of pixels didn't work or worked too slowly to be useful. To export it one can simply zoom out, to about 50%, then export the images. However, printing this diagram directly from Netbeans works correctly. Another issue was that moving large packages was prohibitively slow, or with some large packages simply didn't work most of the time. Copying packages sometimes rearranged the contents of the packages, but this was a minor issue. Copying was only used across diagrams; hence the packages would need to be rearranged anyways. In Netbeans, elements in the diagrams are linked to the actual code, so for some diagrams it was necessary to

hide most or all of this information. For several classes, this information which was hidden while saving reappeared after loading the diagram. It could be manually hidden again before the diagram was used, but this behavior would repeat when it was reloaded. This problem appears to have been solved with the last Netbeans update (6.7.1), and several of other problems might have also been solved, such as the issues with exporting large images.

External Libraries

AZDBLab depended on several external libraries; however, these libraries had been accidently deleted. Updates were found for most of the old jars, but there were several for which the updated version didn't match up with the original: `crimson.jar, Jakarta-ant.jar, jaxp.jar,` and `salvo.jar`. Fortunately, after most of the new libraries were found, a copy of the older libraries was discovered. There was some odd behavior with one library, `jaxp`. For one method, `setAttribute`, the code was not implemented! Whenever the method was called, it simply threw an exception that said that the method was not implemented. Looking at the code for `setAttribute` confirmed this. The functionality for this method was the same in both the new and old versions of the library. This functionality is used by the `XMLHelper` class (in `model.experiment`). These errors didn't seem to affect the overall functionality of AZDBLab.

Including external libraries within Eclipse can be done in two basic ways: relative or absolute (file paths). For absolute file paths, Eclipse would search for the file starting from the root directory on the local file system. For example `/home/adam/libs/ant.jar` would include `ant.jar` if the file existed in that location. This is how the libraries in AZDBLab were included. The benefit of this was that the file size on CVS was smaller. However, this configuration was not portable, as the libraries would have to be moved to the same location to be included. This was not always possible because file paths have different starting point in windows, "`C:`", than in UNIX, "`/`", which meant the paths for the libraries would have to be changed when switching between the two. For relative file paths, the libraries would have to reside in the project directory. This would mean that all of the libraries would need to be inside of CVS, making the CVS directory larger. However, since Eclipse would search for the libraries

by starting inside the project directory, it would be able to find them on any system without any changes. When including the new libraries portability was chosen over size, so they were included relatively.

Plugins for AZDBLab were stored as jar files in the plugins folder. Since they depended on AZDBLab classes, and these classes were reorganized, it was necessary to change the code for the plugins. This was a simple matter of changing the import statements to reflect the new location of the classes. Then the new code was jarred and placed into the plugins folder again. Some of the source code for the packages was not updated since this code was not discovered until recently.

Code Changes

The overall structure of AZDBLab was greatly changed during this project. The following is a brief list of these changes. Several packages (`scenario`, `aspect`, `analytic`, and `experimentalSubject`) were moved from the `model` package to the newly created `plugins` package inside the `azdblab` package. The `executor` package was renamed `executable` and moved into the `model` package. The `mediator` and `queryGenerator` packages were moved inside the `query` package. The `resultAnalyzer` package was moved into the `model` package. All of the exceptions were grouped inside the newly created `exception` package (inside the `azdblab` package). The contents of the `dbmsConnector` package were placed inside the `dialogs` package, and the `dbmsConnector` package was deleted. The `runStatusNodes` package was deleted and its contents were moved into the `objectNode` package.

After looking at the `objectNode` class, it became apparent that there were more efficient ways to structure this class. It was an abstract class that all of the nodes on the left side of the display were required to extend. It provided a framework for these nodes to have icons and panels on the right side of the display. All of its subclasses had a method (`getIconResource`) that returned one of two icons, one if it was open, and the other if it was closed, except in one class. This method was turned into a concrete method inside `objectNode` that used two newly created fields in its subclasses. The fields were the strings that were returned (one for open, the

other for closed). This method was simply overridden in the subclass that didn't follow this structure. For another method (`getDataPanel`) some of the subclasses created a new panel every time it was called, while most made one the first time this method was called, and simply returned it on subsequent calls. This functionality was moved into `objectNode`. This abstract method (`getDataPanel`) was made into a concrete one, which set a new protected field (`JPanel dataPanel`) to the result from a new abstract method (`createDataPanel`), if the field was null. It (`getDataPanel)` then returned the new field (`dataPanel`). After making these changes to `objectNode`, its corresponding subclasses were also changed to conform to `objectNode`. During this process some of the strings for the `getIconResource` were accidentally changed. This caused a null pointer error when AZDBLab tried to use these strings to fetch the resources. After some debugging this was discovered and quickly fixed.

The class `AZDBLAB` was renamed to `Constants`, as it only contained constants, and three fields. The three fields (`String LAB_USERNAME`, `String LAB_PASSWORD`, and `String LAB_CONNECTSTRING`) were moved from the new class `Constants` to the existing class `Main` (in `azdblab.executable`). This new location was deemed to be a better fit as the fields were used throughout AZDBLab.

An error message was added when the plugins folder was missing, or empty. This was added in the `DBMSConnectorDlg`, in the `loadLabnotebookConnections` method. This error message is a dialog that allows users to exit from AZDBLab or continue.

Project Maintenance

There were several compile time errors that had to be resolved after moving the packages. Eclipse was using an unofficial java library, which didn't include some of the classes required by AZDBLab. Changing this to Java version 6 fixed all of these errors.

Oracle Server Restart

At one point during this project, the entire system stopped working. The problem was that when AZDBLab tried to log in to an Oracle server, where it stored its data, the server didn't respond. When testing the Oracle server directly it was discovered that when an incorrect

username-password combination was given, the server responded with an error message saying so. But when the correct username-password combination was given, it simply didn't respond. Why and how this occurred is still a mystery, but fortunately restarting the Oracle server fixed the problem, and it didn't occur again.

Ant

In AZDBLab the old way to build the jar was by using Ant. Ant is an XML-based build utility. It worked well and was easy to learn and use so this was not changed. However, the `build.xml` file, which is used by Ant to jar AZDBLab, reported an error when it was run. The problem was that Ant was attempting to use an external library which wasn't where Ant expected it to be. Moving the library to the correct location fixed the problem.

CVS

The code for AZDBLab is stored on Lectura in a CVS directory, along with some supporting information about AZDBLab. This data structure caused problems at several points. Sometimes it was not possible to delete files in the current version of the code but keep them in previous versions. This was desirable because keeping records of previous version can be helpful. However it was possible to delete the file from all versions by deleting the file on Lectura. This wasn't an ideal solution, but it was necessary to use this approach on several occasions.

The structure of supporting information was also changed. A directory for the external libraries (`lib`) was added. The scripts in the main directory were moved into a new directory (called `scripts`) along with the directory `obsolete scripts`. All of the documentation was consolidated into a new folder called `doc`. In this folder several sub folders were created to help organize information: `tasks`, `uml`, and `uml_output` (for print outs of the UML).

Additional Documentation

A to-do list was created, and then moved into the tasks folder. This was then split up into several files, one for each task. This was to encourage and facilitate including additional information with each task, and to help organize the tasks. In the doc folder several tutorials were

added. A hints file was created to contain miscellaneous, but highly useful information that could potentially save new developers hours of frustration. A jarring AZDBLab file was created to help new developers build an AZDBLab jar file. This file also contains a very brief Ant tutorial; just enough to understand how build.xml works. A list of CS computers was added, since connecting to these computers was useful and there wasn't a list of them online. A new vs. old libs spreadsheet was to show which new library should be used to replace each old library and whether or not the new library works correctly. A read me file was also added as a place to provide a system overview.

Conclusion

Adding documentation and changing the structure of the code helped streamline AZDBLab, making it both more logical and easier to build upon. This will allow future developers to further improve on AZDBLab more efficiently. Working on this project allowed me to acquire lots of useful skills. It allowed me to learn how to use various tools such as Ant, and the Netbeans UML editor. It also gave me experience working on a large scale project that was tens of thousands of lines long and would be used by other developers.

Attachments

There are two attachments to this thesis.

AZDBLab Presentation

This is a PowerPoint presentation outlining my independent study and thesis. It introduces some basic concepts before explaining how they were used. The presentation doesn't assume its viewers are familiar with any of the concept or tools used for my independent study or thesis.

Level 0 UML Diagram

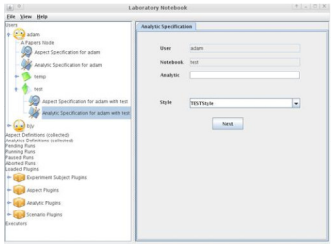
This is a UML diagram of AZDBLab. It only shows the packages. There are three columns in the diagram that are used to organize the packages. The rightmost column is for the GUI, the middle column is for the model, the leftmost column is for the plugins.

REFINEMENT OF AZDB LAB

Adam Robertson

What AZDB Lab is

- Performs operations on databases
- Records the results
- Run experiments of databases
  - Automated
- Used to test new theories



Outline

- Independent Study
  - Goals
  - Adding R
  - Adding Gnuplot
  - UML Diagrams of changes
- Honors Thesis

The R Language

- Started as S at Bell Laboratories
- Statistics-based programming language
  - Operations
  - Analysis
  - Graphing
- True programming language

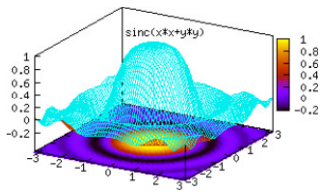
```
BoxDens=function(data, npts = 200., x = c(0.,
    add = TRUE, col = 11., border=FALSE,colli
{
    dens <- density(data, n = npts)
    dx <- dens$x
    dy <- dens$y
    if(add == FALSE)
        plot(0., 0., axes = F, main = "", xli
            ylab = "")
    if(orientation == "paysage") {
        dx2 <- (dx - min(dx))/(max(dx) - min(
            x[1.]
        dy2 <- (dy - min(dy))/(max(dy) - min(
            y[1.]
        seqbelow <- rep(y[1.], length(dx))
        if(Fill == T)
            confshade(dx2, seqbelow, dy2, col
        if (border==TRUE) points(dx2, dy2, ty
```

Adding R

- Communicating Between R and Java
  - Run R from the command line
    - Java executor class
  - Specify input + output files
    - With command line options
- Writing the input file
  - Store data as an R variable
  - Write commands to perform on the data

Gnuplot

- Graphing utility
- 2D / 3D



Adding Gnuplot

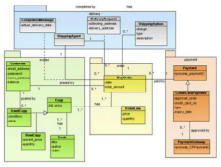
- Communicating between Gnuplot and Java
  - Similar to R
- Writing the input file
  - Data file
  - Operations file
- Gnuplot Settings
  - Mapping for options to values

Outline

- Independent Study
- Honors Thesis
  - Restructuring AZDB Lab
    - UML Tools
    - Design Patterns
    - Current UML Diagram
  - Building AZDB Lab

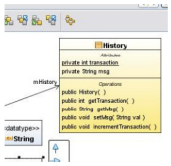
UML

- Diagrams of code
- Compact Visual Representation
- Offers different levels of information
- Different types of diagrams
  - Class
  - Object
  - Activity



UML Tools

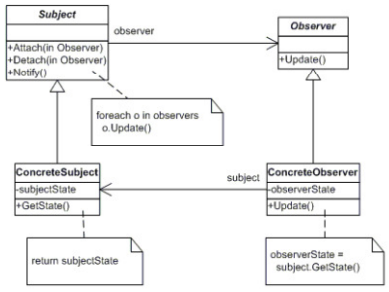
- Violet
  - Used to model AZDB Lab in the past
  - Tons of quirks
- Bo UML
  - Maintained by one person
- Netbeans UML
  - Used to model AZDB Lab currently
- Eclipse Modeling



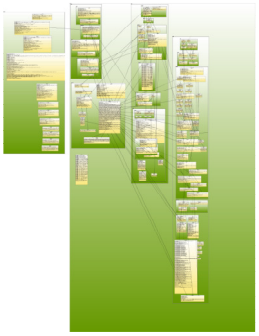
Design Patterns (Gang of Four)

- Simple solutions to general problems
- Increases organization / abstraction
- Creation
  - Singleton – One instance of a class
- Behavioral
  - Observer – Notify other classes of changes
- Structural
  - Iterator – Give information without giving the underlying data structure

Observer



UML of AZDB Lab



Outline

- Independent Study
- Honors Thesis
  - Restructuring AZDB Lab
  - Building AZDB Lab
    - Ant
    - XML
    - Sample Build.xml

Ant

- Similar to make file
  - But with less quirks
- Used to automatically build projects
  - Saves time vs. compiling individual files
  - Used to group smaller tasks
- Build utility
- Uses XML to specific functionality

XML

- Represents data textually
- Looks like HTML
  - Tags (Open / Close)
  - Uses '<' and '>' as separators

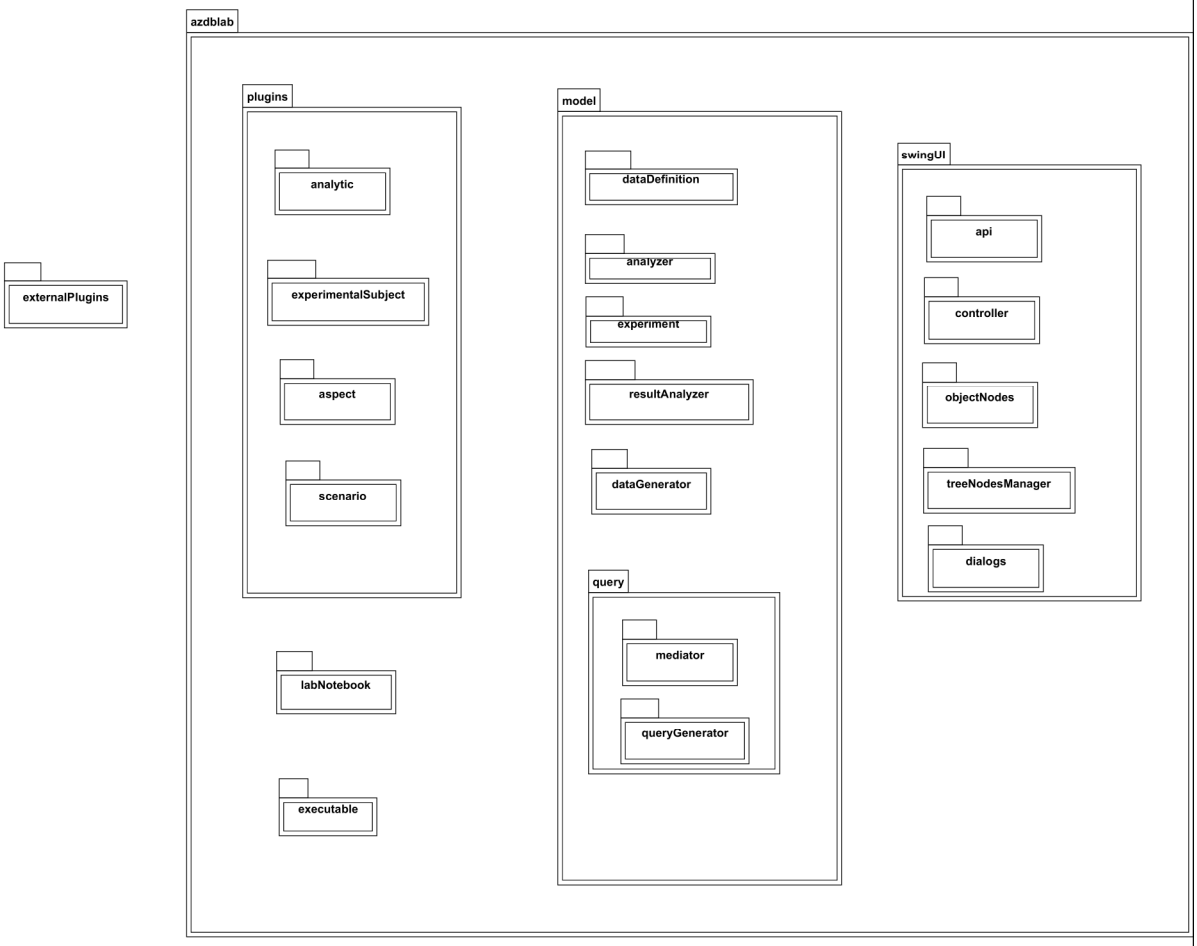
```
<?xml version="1.0"?>
<note>
   <to>Adam</to>
   <from>Adam</from>
   <heading>Reminder</heading>
   <body>Test Tomorrow!</body>
</note>
```

Sample Build.xml

```
<project name="Sample" >
        <!-- set global properties for this build -->
        <property name="src" location="src"/>
        <property name="build" location="build"/>

<target name="init">
        <!-- Create the build directory structure used by compile -->
        <echo>  This prints on the console </echo>
        <mkdir dir="${build}"/>
</target>

<target name="compile" depends="init"  >
        <!-- Compile the java code from ${src} into ${build} -->
        <javac srcdir="${src}" destdir="${build}"/>
</target>

</project>
```

Conclusion

- Provided an interface between R / Gnuplot and java
- Experienced a programming on a large scale project
- Help optimize + design future additions to AZDB Lab
- Learned how to use Ant
  - I plan to use this in the future


azdblab
plugins
analytic
experimentalSubject
aspect
scenario
labNotebook
executable
model
dataDefinition
analyzer
experiment
resultAnalyzer
dataGenerator
query
mediator
queryGenerator
swingUI
api
controller
objectNodes
treeNodesManager
dialogs
externalPlugins