Aurora SDK: A Web Based Integrated Development Environment

Communication Protocols Project

Jingwen Ou
Mahdi Tayarani Najaran
Mushfiqur Rouf

1. Introduction

The goal of this project is to develop a web based Integrated Development Environment (IDE). The client only needs a web browser and an internet connection. By connecting to the server, an IDE looking page is downloaded by the browser that looks like and has the functionality of an IDE environment. The client is able to access previous code stored in the database or create new projects, compile the code and run it. Also, the client is able to download the executable output of the compiler from the server to local disk.

1.1 Context

IDEs are designed to maximize programmers' productivity. They normally achieve this goal by consisting of a source code editor, a compiler and/or interpreter, built-in automation tools, and a debugger [1]. Some modern IDEs even employ plug-in frameworks that support extension to the environment [2], hence meeting various needs of programmers.

1.2 Problems

The startling growing software sizes and hardware consumption (e.g. memory and CPU) of IDEs [3] as well as their plug-ins have gradually become a headache. Moreover, programmers have to ensure that their favorite IDEs and development toolkits (e.g. JDK) are installed and properly configured in their computers before they are able to start working, which takes a substantial amount of time. Even running a properly configured IDE takes a long time to load. It seems like that IDEs will add troubles to programmers these days instead of aiding them. Therefore, some interesting solutions are emerging and can keep the hundreds of megabytes away from disk.

1.3 Propositions

The basic idea is to develop a useful development tool that can be used by any thin client with a web browser. It will make programming on the move, or programming outside of the normal working environment far more convenient and easy. Also, it will provide a plug-in architecture to extend the platform to support more features. However, plug-ins here no longer needs any installation or configuration: appending their URLs to the IDE's setting will automatically trigger additional functionality.

On the other hand, web-based IDE encourages live collaboration between team members, because its structure requires all documents and codes to be modified and saved on the servers, which will release the problems of the implicit code knowledge [4]. The most exciting advantage of web-based IDE is that, with the new generation of smart phones and PDAs that support AJAX and Java [5], programming jobs can be done movably anywhere anytime.

1.4 Related Works

IBM has done an alpha work in solving this problem [6]. It consists of an Eclipse plug-in that handles the actual HTTP server-to-service requests and communicates with Eclipse. After it is downloaded and installed on the server instance of Eclipse, this plug-in extracts the Eclipse User Interface definition (for example, context-based menus, perspectives, and views) and serves it over the Internet following the XML User Interface Language (XUL) standard. The user interactions on the browser are transmitted as events to the Eclipse instance and vice versa. However, it is mostly a continued "screen shot" of the Eclipse UI (transferred by XUL) captured by the browser, which is extremely ineffective and slow in practice (the pages in the browser will keep flashing when new contents are received). Moreover, it does not provide some core functions of IDE such as syntax highlighting or auto completion.

Another example is CodeIDE [7]. It seems exciting in the beginning. However, when writing something more complicated in its editor, it fails miserably. For example, it only supports simple statements in C++ no other than "printf", but not the core concepts of Object-oriented Programming.

1.5 Our Solutions: Aurora SDK

Our implementation of a web based IDE - Aurora SDK, supports full syntax highlighting and full compilation of Java source codes with different plug-in compilers. Two compilers are implemented in Aurora SDK: Eclipse and JDK. Moreover, it can compile source codes into byte codes and run on the server side then give back results. In the first release, downloading the compiled byte codes from the server is made feasible so that users can run the compiled byte codes on their local machine. Moreover, auto completion is also supported in this release.

1.6 Report Organization

The project takes advantage of a powerful tool, Google Web Toolkit (GWT), to develop such an environment. In this report, Section 2 gives a short introduction of this tool. Section 3 goes into design details, whereas, Section 4 has a few comments on the motivations behind such an environment.

2. Overview of GWT

Google Web Toolkit (GWT) is a set of development tools, programming utilities, and widgets that enable creating rich Internet applications by writing browser-side code in Java instead of JavaScript. Figure 2.1 provides a visual map of the central aspects of GWT. The tools can be divided into those that are tied to the compiler, and the Java libraries that make up the GWT API.

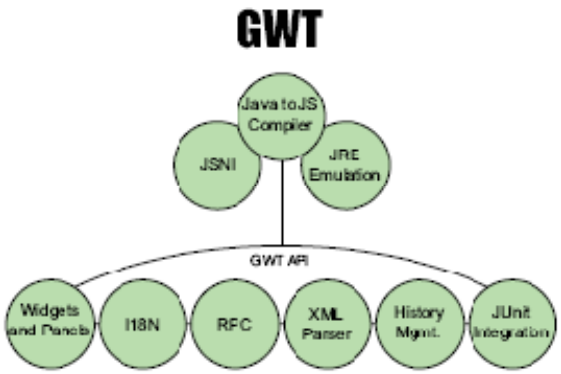

Figure 2.1: Central aspects of GWT

At the core of GWT is a Java-to-JavaScript compiler that produces code capable of running on Internet Explorer, Firefox, Mozilla, Safari, and Opera. The compiler converts the Java syntax to JavaScript, utilizing JavaScript versions of commonly used Java classes like Vector, HashMap, and Date.

The compiler has three style modes that determine what the resulting Java-Script looks like. The default style is obfuscate, in which everything is compressed and nearly impossible to decipher. The next style is pretty, which generates readable JavaScript. The last style is detailed, which produces JavaScript code that looks like the pretty style with the addition of the full class name as part of the JavaScript method name. Another important aspect of the compiler is that it compiles from Java source code, not compiled Java binaries. One last feature to note is that when the code is compiled to JavaScript, it results in a different JavaScript file for each browser type and target locale.

The JavaScript Native Interface (JSNI) allows executing JavaScript from Java as well as executing Java from JavaScript. The JRE Emulation Library is a mapping of Java Runtime Environment (JRE) classes to their JavaScript equivalents.

GWT includes a large library of widgets and panels, making it effortless to build a web application that looks more like a desktop application. The distinction between widgets and panels is that a widget is some sort of control used by a user, and a panel is a container into which controls can be placed.

GWT provides two tools that sit on top of the XMLHttpRequest object. The first is the RequestBuilder class, which is essentially a wrapper around this object. The second tool, GWT-RPC, allows sending and receiving real Java objects between the client and server. Another tool provided by GWT is a set of classes for supporting the JavaScript Object Notation (JSON) message format. JSON is a popular message format known for its simplicity and widespread availability.

GWT also provides support for JUnit, so that programmers don't need to learn another testing framework. The special hosted-mode browser enables developing and debugging in Java without ever needing to deploy the code to a server.

3. Designs

The basic techniques used for developing the web based IDE can be categorized as:

- Pluggable design
- Google Web Toolkit
- Servlet development
- Compiler control
- Database control

3.1 System Structure

In order to develop a flexible structure for the IDE, we referred to the architecture of Eclipse. The architecture used for the IDE can be seen in Figure 3.1. There are four layers in our design:

- Application layer
- Service layer
- Compiler layer
- Persistence layer

We used MVC (Model-View-Controller) architecture to implement our flexible design. In the Application layer, we built GWT widgets to develop the UI. In the Service layer, as a middleware layer, we used GWT Remote Service to communicate between the Application layer and the Compiler layer and the Persistence layer.

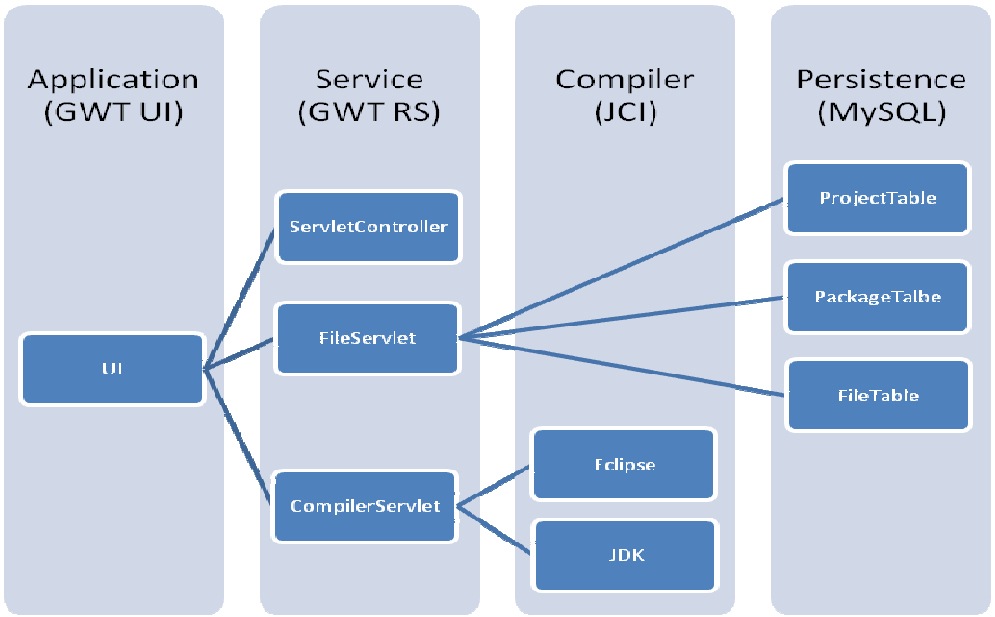

Figure 3.1: IDE Architecture

ServletController is the controller which controls the requests sent by the UI and distributes the requests to FileServlet and CompilerServlet. Two of the layers, Compiler layer and Persistence layer, are at the same level. FileServlet communicates with the database (MySQL) in the Persistent layer. The CompilerServlet communicates with the Java Compiler Interface declared in the Compiler layer. Java Compiler Interface is a pluggable compiler interface that can be plugged in different Java compilers. In our project, two compilers are implemented, Eclipse and JDK. In

future work, we expect that more compilers can be plugged into our web-based IDE, e.g. C compiler.

3.2 User Interface

As depicted in Figure 3.2, we separated the UI into five composites in order to implement a flexible structure:

- Editor Composite
- Explorer Composite
- Information Composite
- Toolbar Composite
- Menubar Composite

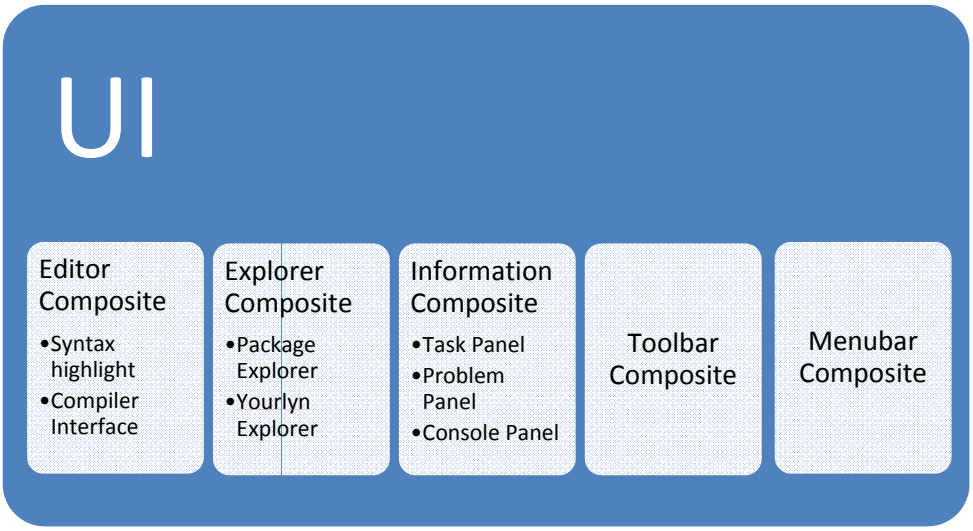

Figure 3.2: User Interface

In the Editor Composite, two sub-composites are implemented: Syntax highlight and Compiler Interface. Syntax highlighting uses match-and-replace mechanism. For example, if the key word "class" is defined in the library (XML), it will be replaced with alternative CSS style "class". Compiler Interface is an interface on the client side that communicates with the Java Compiler Interface in the Compiler layer. It normally acts as a distributor that assigns compiling tasks and also receives messages (compiled results and console information) from the server.

In the Explorer Composite, two sub-composites are implemented: Package Explorer and Yourlyn Explorer. Package Explorer is an explorer for navigating elements such as project, package and class and their relations. It is tightly connected to the database in the Persistent layer and is organized by FileServlet in the Service layer. Yourlyn is an experimental implementation of Eclipse's plug-in Mylyn [8]. It can trace the events triggered by users then calculate their Degree of Interests.

In the Information Composite, three sub-composites are implemented: Task Panel, Problem Panel and Console Panel. Task Panel displays the tasks that the users are currently doing. It also captures the events triggered by the users. Problem Panel displays the compiled problems sent by the server. It is related to the Compiler Interface in the Editor Composite. Console Panel displays the console information of the running program. It is also related to Compiler Interface which can also run class files besides compiling Java files.

3.3 Editor:

The editor is implemented as a pluggable structure. We separated the Editor into two parts: UI and Servlet. In the UI part, different syntax highlighting libraries can be plugged into the editor. In the Servlet part, different compilers can be plugged into the editor (Figure 3.3).

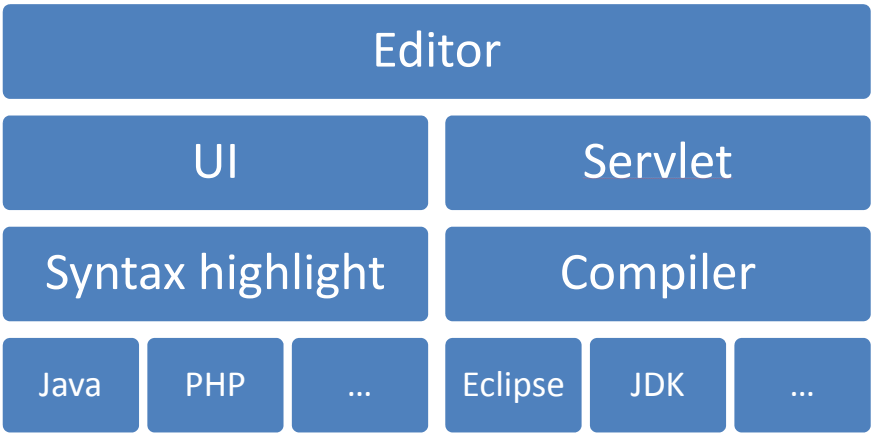

Figure 3.3: Editor

3.4 Database

Complex applications deal with huge amount of data. Managing that huge data is a challenging task. In the current world of highly distributed applications, data has to be stored in a highly distributed nature for efficiency, redundancy and other issues.

In our model implementation, instead of going into a novel distributed data management system, we used a common database management system – MySQL. We understand that even the distributed database systems will fail to manage the huge amount of data web based applications have to deal with.

To respect this fact, in our implementation we decoupled the database manager from the other parts of the code to present the obvious fact that we could use some distributed data management system too.

3.4.1 Design Pattern

The goals of our data manager module design were:

1. The library should be easy to use from other parts of the code
2. The library should decouple the data access from the programming logic
3. Redundant coding in the database manager should be reduced.

To meet the first two criteria, we selected the Data Access Object (DAO) design pattern [9]. DAO pattern dictates to create an interface that does not expose the detail of the database allowing the reduction of coupling in the code. The data access layer and the logical part of the code need not know anything about one another. DAO pattern can successfully keep the functions of these two parts separately.

To meet the third criterion, we implemented our data manager code using the Reflection built into Java. With the help of this approach:

1. We could build a data manager that provides an Object Oriented view of the data. Now, for each table in the database, we had a class (bean) to hold the data. Loading, saving and finding – can be done using these classes only.
2. Adding more tables in the database translated to writing one more class with a set of attributes (which can even be done in an automated manner, as does the framework named Ruby on Rails). Once we have the new class definition, no load/store DML (Data Manipulate Language) code is needed since it is automatically handled by the single generic implementation written using the Reflection technology.

We implemented our database in MySQL. MySQL is not particularly famous for distributed database management. However, only standard data definition language was used, so we believe our database will be compatible with any other RDBMS.

3.4.2 Implementation Details

The class diagram of the data access module can be found in Figure 3.4.

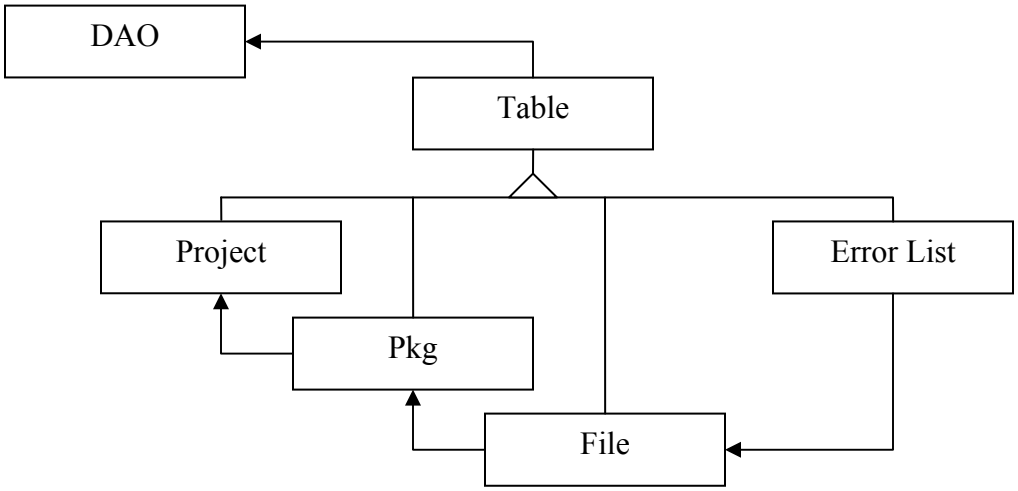

Figure 3.4: Data Access Module

The DAO class is a singleton class; it manages the connection with the database system.
The Table class is the base class that defines the generic data manipulation operations using Java reflection. The operations that we defined are:

1. Load an object (row)
2. Save (update or insert) an object (row)
3. Find a set of objects based on a conjunctive set of search criteria
4. Delete an object (row)
5. Delete a set of objects based on a conjunctive set of search criteria
6. Load all
7. Delete all

All other implementation classes are shadows of the database tables. These classes extend from the Table class, thus inheriting the data manipulation operations. Only the fields are to be defined with proper data type compatible with the type used to declare the fields in the database.

4. Discussions

While there currently exists popular IDEs such as Eclipse, however, there seems not to be a huge demand for web based IDEs. We state against this belief, claiming that web based IDEs offer functionality not provided by traditional IDEs.

Aside from including features of all web based applications, such as thin clients and any time/any place services, web based IDEs have other unique features. The compiling on the server side can be done on powerful and separate processors, saving the client all the trouble. Also, the client does not need to have all SDKs installed to be able to use them.

As all these feature are also provided by a remote desktop connection (RDC), but the ability to share a project between multiple users, allowing them to develop different parts of a project simultaneously without extra coordination gives web based IDEs an edge over RDCs. However, RDC can be ineffective in transmissions and inconvenient to use. What's more, we need to have another remote computer that we have access to and an RDC tool to make the connection happen.

5. Summaries

In this project we developed a web based IDE using GWT. Using a simple browser to connect to the server, most effort was made to provide the standard functionality of Eclipse for our IDE. Figure 5.1 is a screenshot of the environment.

Future work of this project includes allowing multiple users to work on a single project and the ability to debug code.

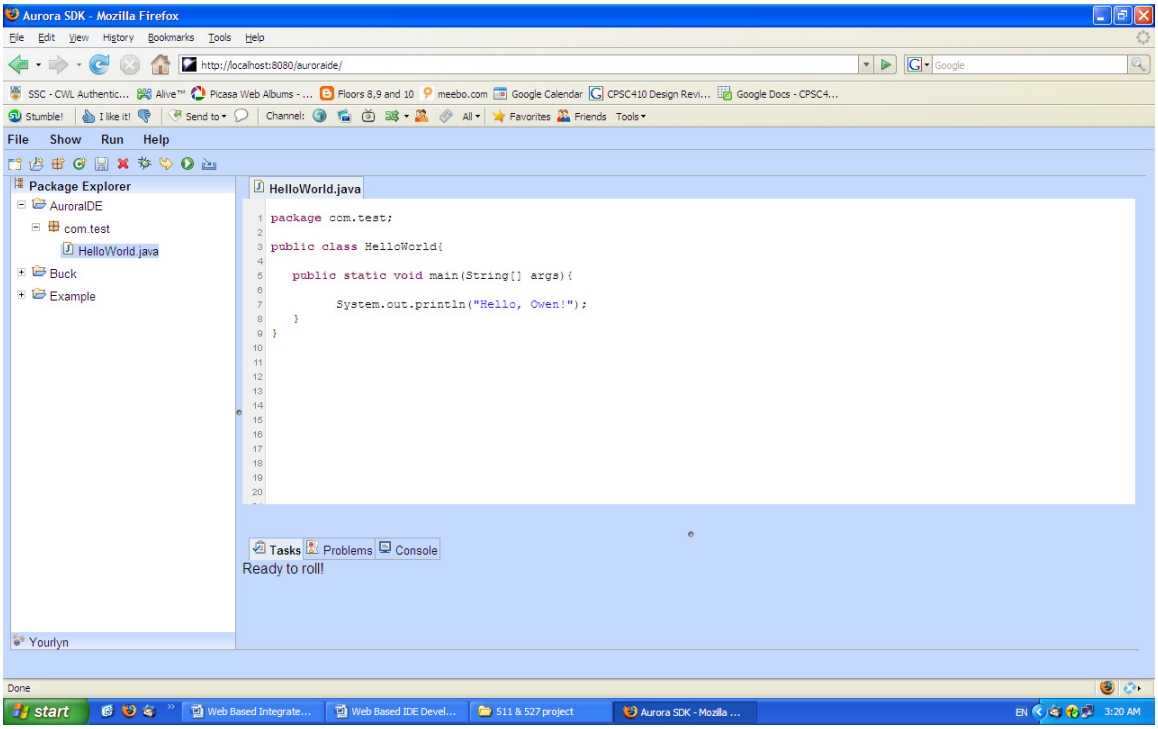

Figure 5.1: Screenshot of the IDE

6. References

`

[1] Wikipedia. Integrated development environment.
http://en.wikipedia.org/wiki/Integrated_development_environment
[2] Plug-in Development Environment, Eclipse.
http://www.eclipse.org/pde/
[3] The current Eclipse 3.3 SDK is about 300MB (which includes the JDK)
[4] Maintaining mental models: A study of Developer Work Habits, LaToza, Venolia, DeLine, ICSE 2006
[5] Opera Widgets
http://www.opera.com/products/mobile/widgets/
[6] Eclifox, IBM
http://www.alphaworks.ibm.com/tech/eclifox
[7] CodeIDE
http://www.codeide.com/
[8] Mylyn
http://www.eclipse.org/mylyn/
[9] Data Access Object
http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html