CoRED - Browser-based Collaborative Real-Time Editor for Java Web Applications

Janne Lautamäki, Antti Nieminen, Johannes Koskinen, Timo Aho, and Tommi Mikkonen
Tampere University of Technology
Korkeakoulunkatu 10, FI-33720, Tampere, Finland
{janne.lautamaki, antti.h.nieminen, johannes.koskinen, timo.aho, tommi.mikkonen}@tut.fi

Marc Englund
Vaadin Ltd.
Ruukinkatu 2-4, FI-20540, Turku, Finland
marc.englund@vaadin.com

ABSTRACT

While the users of completed applications are heavily moving from desktop to the web browser, the majority of developers are still working with desktop IDEs such as Eclipse or Visual Studio. In contrast to professional installable IDEs, current web-based code editors are simple text editors with extra features. They usually understand lexical syntax and can do highlighting and indenting, but lack many of the features seen in modern desktop editors. In this paper, we present CoRED, a browser-based collaborative real-time code editor for Java applications. CoRED is a complete Java editor with error checking and automatic code generation capabilities, extended with some features commonly associated with social media. As a proof of the concept, we have extended CoRED to support Java based Vaadin framework for web applications. Moreover, CoRED can be used either as a stand-alone version or as a component of any other software. It is already used as a part of browser based Arvue IDE.

Author Keywords
Development tools, collaboration architectures, Vaadin.

ACM Classification Keywords
D.2.3.c. Program editors. D.3.2.h. Development tools. H.5.3.c. Computer-supported cooperative work. J.8.s. Web site management/development tools.

General Terms
Design, Experimentation.

INTRODUCTION

It is widely recognized that communication problems are a major factor in the delay and failure of software projects [2]. Numerous tools and methods have been proposed to solve issues in different phases of projects, starting from capturing requirements and ending at customer documentation. One of the most promising approaches to communication problems is offered by agile methods that advocate close and frequent communication between the client and the developers. In reality, this is often implemented in the form of a team that shares the same premises, encouraging frequent informal communication.

While the software development community is already struggling with communication issues, the emerging practice of global software engineering is raising even more challenges: Software work is undertaken at geographically separated locations across national boundaries in a coordinated fashion, involving both real time (synchronous) and asynchronous interaction [13]. This emphasizes the need for timely, precise and uniform forms of communication across the planet. Then, since the development takes place at the global scale, also the necessary communication should take place at such scale.

In almost any other field, the recent standard answer to global communication problems has been the World Wide Web, or simply the Web. Indeed, in a relatively short time, the Web has become the platform for all types of applications that enables real-time collaboration in forms and scale that would have been difficult to imagine a few decades ago. Recently, the collaborative capabilities of the Web have been further enriched with a new invention – social media. Facebook, Linkedin and other services enable us to be in contact with the friends, colleagues, and enthusiasts of different topics in real time all over the planet.

While the users of completed applications are heavily moving from desktop to browser, the majority of developers are still working with desktop IDEs such as Eclipse or Visual Studio. At present, most of the available web-based code editors are just text editors with some extra features like code highlighting, indentations and collaboration clued on top and they are not yet as usable as the best of the desktop editors. However, the web editors offer their own possibilities. For example, real-time collaborative editing sits very naturally in the environment. In addition, we get the general web-based application benefits like automatic distribution, installation and updating of applications [9] and independence on the development environment. Steps towards the direction are also proposed in [1, 14].

In this paper, we describe an experiment where the collaborative capabilities of the Web in general and the features of social media in particular are harnessed to help solving some of the communication problems of software development. As a concrete technical contribution, we introduce the browser based editor CoRED[1] (Collaborative Real-time Editor) intended for collaborative real-time editing of Java based source codes. CoRED contains highlighting, indentation, semantic error checking and some code completions. In addition, we present a number of features inspired by social media services, which we believe will be helpful for developing software applications. To the best of our knowledge, the similar set of functionalities is not offered by any of the previously existing web-based code editors.

We have selected Java and Vaadin [6] as the target language and framework, meaning that CoRED is an editor initially aimed for those two. Nevertheless, it can be extended for other environments with reasonable amount of work. The web applications based on Vaadin are implemented just like desktop Java applications. Because of the strong typing and good tool support in Java it is possible to augment CoRED with many features. These features include semantic error checking and code completion. This could just be dreamed of for weakly typed dynamic languages like JavaScript.

The rest of this paper is structured as follows. In the next section, we give an overview to the Vaadin framework, to the Ace editor, and to Java Developer Kit (JDK) which we are using as a base of our work. We also give a brief introduction to Arvue IDE which is going to use our editor as a building block. After this, we describe the technology behind our solutions and discuss the collaborative features. Moreover, we explain how CoRED could be extended for different frameworks. Then, we compare our editor with other code editors available in the web. Towards the end of the paper, we draw some final conclusions.

BACKGROUND

In this section, we give brief introduction to tools we are using in CoRED. We also show Arvue IDE as an example because it is going to be the first actual web application using CoRED. Tools we are using are the Vaadin framework, Ace editor, and Java Developer Kit (JDK). As a

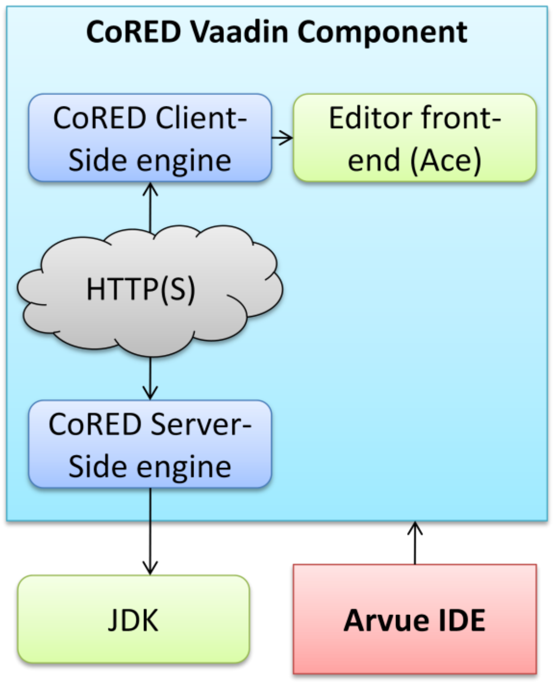


Figure 1. Architecture of CoRED Vaadin component and Arvue IDE using it.

framework, Vaadin is obviously the lowest layer of our architecture. Vaadin is used for communicating over HTTP(S) and for making a separation of concerns between the client and the server [18]. On the client side, we use Ace editor as a front-end, and on the server side JDK as a tool for analyzing the source code. Vaadin also offers us a way for packaging the whole CoRED as a deliverable component as presented in Figure 1. CoRED can be used as a part of Vaadin based application like Arvue IDE or as it is.

Arvue

Arvue[2] is a cloud based IDE and hosting solution for the users who need a simple web-based tool for implementing and publishing Java based Vaadin applications. The basic philosophy of Arvue is to create applications “in the web for the web”. In other words, the goal is to implement web applications in the web and publish them with minimal effort. No other tool except the browser is needed, implying that no installation is necessary in any phase of the development.

Applications are created in the browser-based visual editor that contains both the GUI (see Figure 2) and the code editor. On the GUI side, the user can create a new GUI just by dragging and dropping elements and layouts. The created GUI is then converted to source code and it can be further modified with the code editor (namely CoRED). Arvue is a round trip tool between the GUI and the text

[1]CoRED is available for testing at
http://jlautamaki.virtuallypreinstalled.com/CoRED

[2]Arvue@dev.vaadin.com wiki.
http://dev.vaadin.com/wiki/Arvue

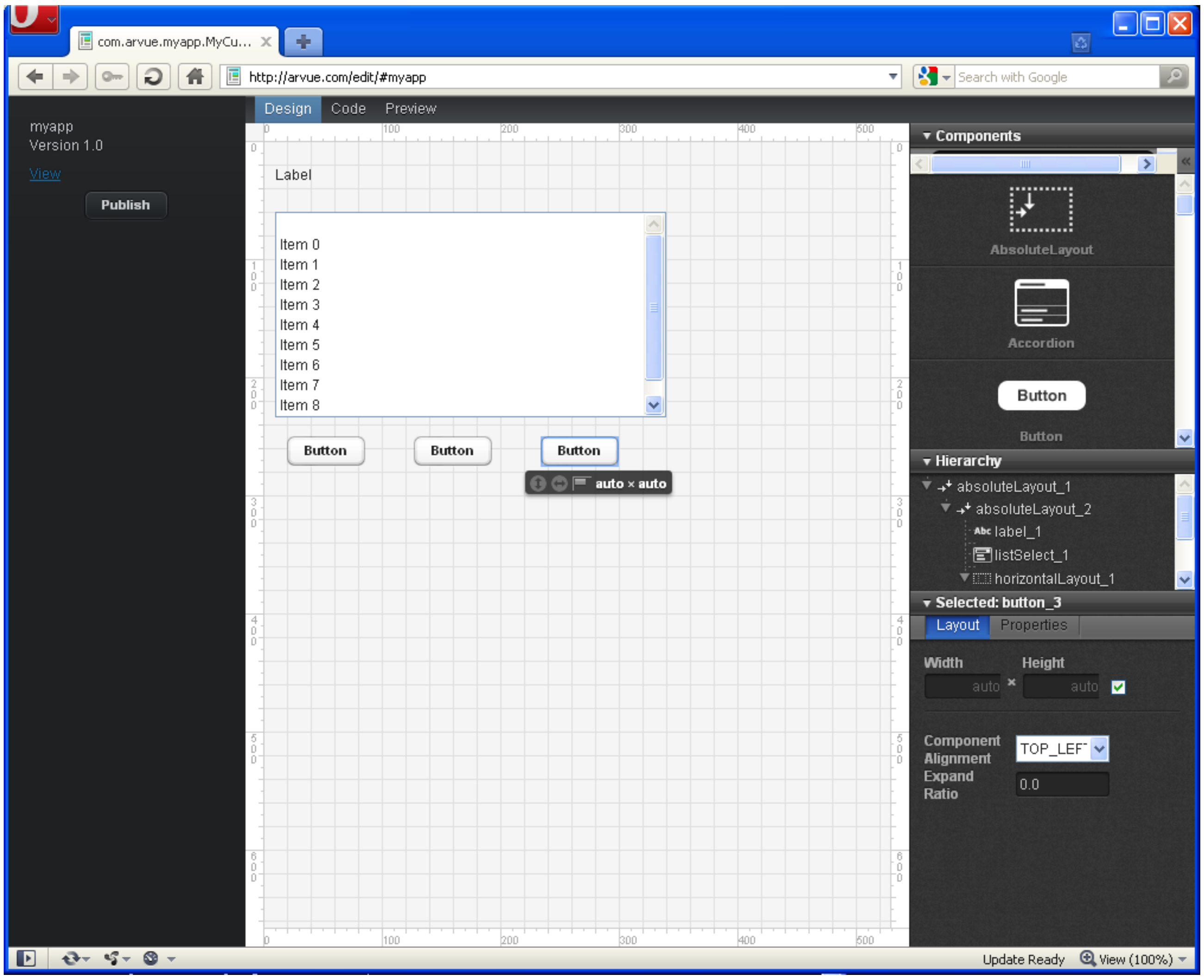


Figure 2. Arvue IDE with the graphical UI editor tab opened.

editor, meaning that same GUI edits can be done from both of the editors. The code editor is also used to implement features behind the GUI. Finally, the new application can effortlessly be deployed to a cloud type environment, which scales up by starting more server instances when needed. Every server is able to host multiple applications, which share many resources while not interfering with each other.

Arvue utilizes the Vaadin framework in its implementation. While editing is done inside a browser, most of the processing is carried out on the server side, as is common in Vaadin based system. For example, the server side of the CoRED utilizes JDK for semantic error checking and it also eases the implementation of code completion. Furthermore, our code editor is implemented as a custom Vaadin component whose client side is a Google Web Toolkit (GWT) [12] widget. Additionally, the client side widget uses JavaScript based Ace editor for basic editor capabilities.

The Vaadin Framework

Vaadin is an open source framework for developing Rich Internet Applications (RIA) using the Java programming language. The Vaadin framework relies extensively on the facilities of GWT [12]. GWT is an open source development system that allows the developer to write AJAX applications [10] in Java and then compile the source code to JavaScript which can be run on all browsers. In the Vaadin framework, GWT is used for compiling the client side engine and for communication between the client and the server.

From the developer perspective, individual Vaadin applications are implemented similarly to Java Standard Edition desktop applications. However, instead of usual UI libraries like AWT, Swing or SWT, the developer has to use the specific set of Vaadin UI components and the framework knows how to use the browser as a view. In addition, new custom made UI components can be implemented. In this case the client side of the customized UI component can be either developed in Java and then compiled with GWT or written directly with JavaScript. The use of any combination of Java and JavaScript is also possible. Thus, this enables us to use readily made JavaScript applications as a part of the Vaadin client side component.

Java Developer Kit (JDK)

Usually, Java applications run on top of a Java Runtime Environment (JRE) and are only compiled using a Java bytecode compiler included in developer kits like the Sun JDK[3]. However, this is not enough in order to develop the code editor applications, which can create other new applications on the fly. Instead, we need to run the application on the developer kit. JDK contains the necessary tools for compiling, executing, debugging and documenting the programs. Furthermore, it includes `tools.jar` package, which contains useful Sun specific APIs for compiling, diagnosing, and parsing the source code.

For our purposes, JDK provides a couple of very useful features: JavaCompiler[4] is an interface for invoking Java compiler from the program. The compiler generates complete error and warning diagnostics during compilation (for example, error messages) and they can be collected using DiagnosticCollector. In addition, by extending ClassLoader and StandardFileManager, the source and destination of compilation can be redirected. Finally, the offered API contains TreePathScanner[5] that can be used for processing source code. The scanner visits all the child tree nodes of the source code and can, thus, be used for finding Classes, Methods, Variables and other kinds of Java structures.

Ace

Ace[6] is an open source code editor written using JavaScript. It can be easily embedded in any web page and it has support for several different programming languages, including Java. Ace offers a lot of basic text editor features such as undo/redo, search/replace, and customization of appearance using themes. It also implements many features important specifically for programmers like syntax highlighting and automatic indentation.

It is easy to extend the behavior of Ace without editing its source code. It is possible, for example, to implement your own keyboard handler. Another feature important to us is that Ace supports markers for showing code errors. Also custom markers like underlining are possible. Moreover, Ace offers information useful in integrating the editor into a broader framework. For example, the position of the cursor in screen coordinates is needed for displaying a suggestion box at a suitable position.

ARCHITECTURE OF CORED

In CoRED, most of the hard work such as checking code errors and generating code suggestions is done on the server side. The client side editor does the interaction with the user. CoRED utilizes the Vaadin framework to tie these two sides together.

Separation of concerns

Both the client side and the server side of CoRED are designed to be easily customizable. Most of the features of the editor, such as error checking or code suggestions are implemented as replaceable and extendable components. A part of the CoRED architecture is presented in Figure 3. The main component, CollaborativeCodeEditor, and its client side counterpart act as glue between the server side components and the front-end editor. For example, when the user needs code suggestions, the main CoRED component requests suggestions from the server and then displays a widget for the user for selecting among the suggestions. The suggestions are generated by server side suggester components, and the selected suggestion is finally sent to the front-end editor.

The front-end editor component is typically a wrapper for a third-party JavaScript code editor. We have implemented a prototype component with three possible choices for the front-end editor: Ace, CodeMirror[7] and Eclipse Orion[8]. Wrapping a JavaScript editor inside a Vaadin GWT component was quite straightforward using JavaScript Native Interface (JSNI)[9] calls. Eventually, we chose Ace as our front-end editor because of its good support for indentation, syntax highlighting and customizable markers among other things.

CoRED has a possibility for flexible component add-ons. For example error checking and code completion components can be added by simply implementing the corresponding interfaces. Next we present the implemented components in detail.

Error Checking

Error checker is an example of component for extending CoRED. For checking errors, there are basically two possible approaches. First, we may use our own or third

---

[3]JDK File Structure for Windows. http://download.oracle.com/javase/6/docs/technotes/tools/windows/jdkfiles.html

[4]Java Compiler (Java Platform SE6). http://download.oracle.com/javase/6/docs/api/javax/tools/JavaCompiler.html

[5]TreePathScanner (Compiler Tree API). http://download.oracle.com/javase/6/docs/jdk/api/javac/tree/com/sun/source/util/TreePathScanner.html

[6]Ace – Ajax.org Cloud9 Editor. http://ace.ajax.org/

[7]CodeMirror. http://codemirror.net/

[8]Orion – Eclipsepedia. http://wiki.eclipse.org/Orion

[9]Coding Basics - JavaScript Native Interface (JSNI). http://code.google.com/webtoolkit/doc/latest/DevGuideCodingBasicsJSNI.html

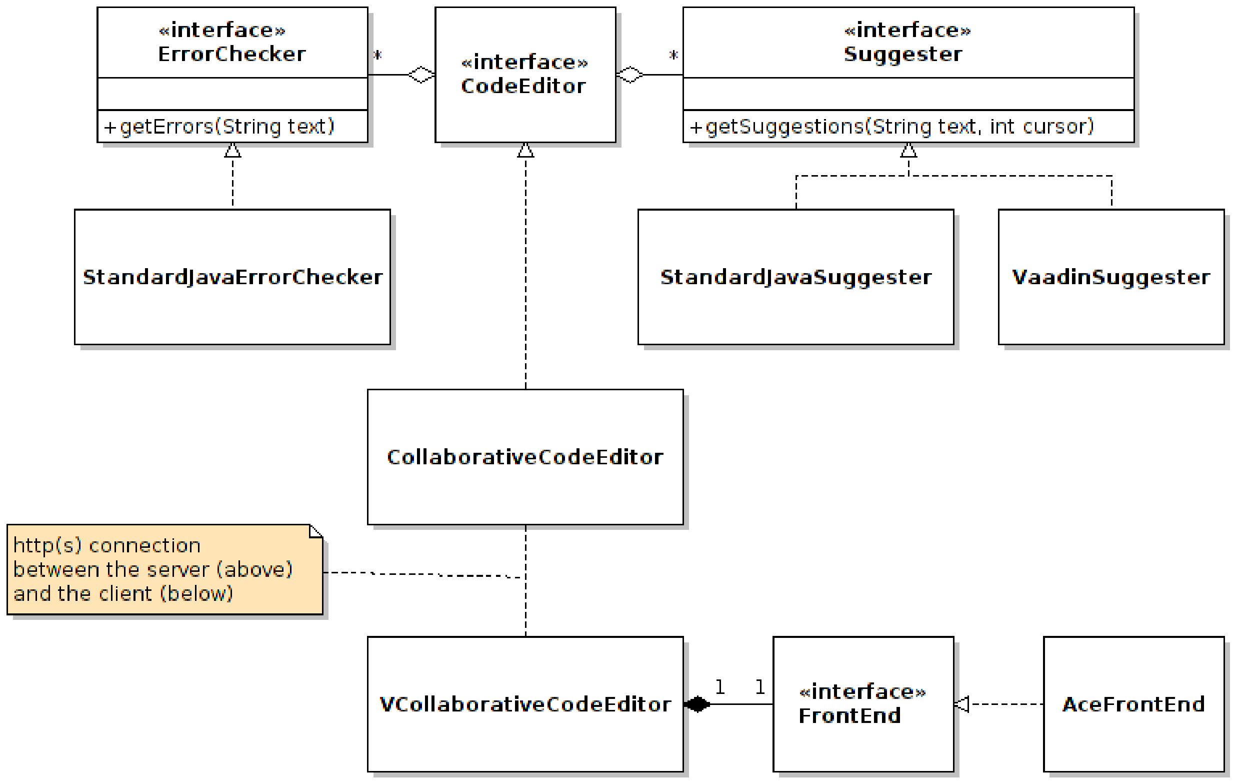


Figure 3. A part of the CoRED class hierarchy.

party library for parsing and error searching. Second, it is possible to compile the code from source to bytecode and then get compiling diagnostics.

For our implementation, we decided to use the latter solution with Java SE Development Kit (JDK). This way we get all-inclusive error and warning diagnostics from JavaCompiler and we do not need to stumble with the parser-compiler incompatibility. However, a basic problem with compiling is its consumption of computing resources. If we use a third party library or some kind of separate incremental parser like in Eclipse SDK for error checking, we could have better changes to tune the balance between efficiency and accuracy. However, we think that the approach is sufficient for our prototype.

In error checking, our basic procedure is to first compile the code and then ask for compiling diagnostics. Error diagnostics offered by JDK contain all the information needed for the message to user: in addition to the actual error message, line and column numbers are available. For efficiency reasons we do not save the files but only compile in memory. Furthermore, we compile only when we assume that the user wants the errors to be checked. In practice we wait for a while after the last edit and compile when user is in an idle state planning the next modifications. No compilation is done during code editing and even during the compilation the user interface is not blocked.

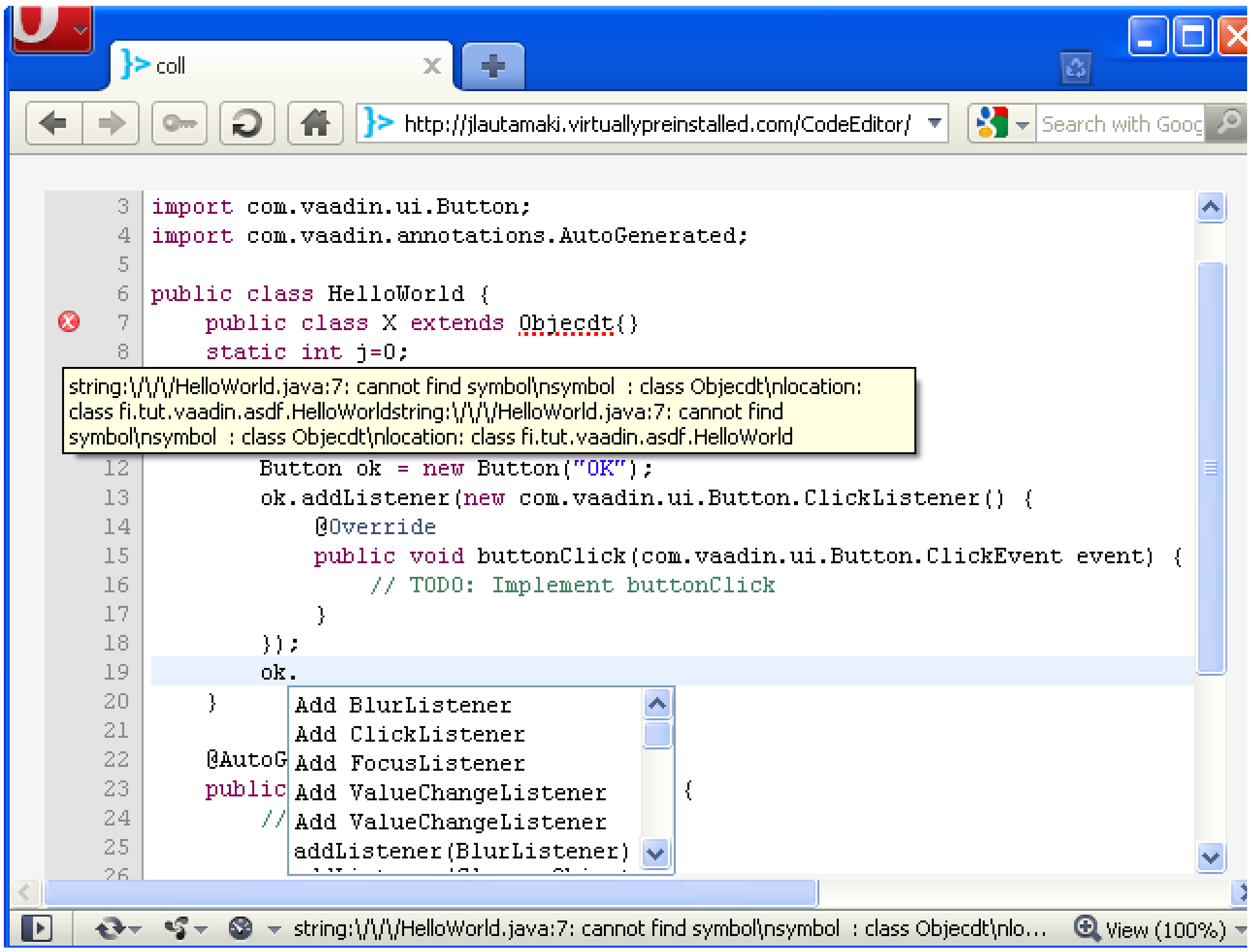
With just a few users, the compiling happens in the blink of an eye and most of the delay is caused by network communication. However, with multiple users, the continuous compiling is an efficiency problem. To make the system more scalable we compile less frequently when the number of users increases. In practice this is implemented with a separate worker thread for compilations.

Suggestions and Code Completions

While editing the source code, the code editor also suggests possible code completions. The suggestions can be invoked in two ways, by using a special key combination or by typing a dot after, e.g., an interface or an object name.

In either case, the server side client analyzes the code with all the implemented suggestion components. In fact, we have implemented two different suggestion components: one for standard Java suggestions and another for Vaadin specific ones. Based on the cursor location in the text the components resolve suitable suggestions. In addition to the actual inserted text, each suggestion includes a visible name of the suggestion and a longer description. These are passed to the editor and then shown to the user as seen in Figure 4.

To help resolving the suggestions, we use tools offered by JDK. With TreePathScanner it is possible to build a tree of classes, methods, variables or other parts of Java syntax. The currently visible variables, their types and methods, and the visibility scopes can be tracked down. To calculate the suggestions, the whole document is scanned with linear time operation. A suggestion may be related to a class originating from an imported package. For resolving those



```
import com.vaadin.ui.Button;
import com.vaadin.annotations.AutoGenerated;

public class HelloWorld {
    public class X extends Objecdt{}
    static int j=0;
```

```
string:\/\/\/HelloWorld.java:7: cannot find symbol\nsymbol  : class Objecdt\nlocation:
class fi.tut.vaadin.asdf.HelloWorldstring:\/\/\/HelloWorld.java:7: cannot find
symbol\nsymbol  : class Objecdt\nlocation: class fi.tut.vaadin.asdf.HelloWorld
```

```
        Button ok = new Button("OK");
        ok.addListener(new com.vaadin.ui.Button.ClickListener() {
            @Override
            public void buttonClick(com.vaadin.ui.Button.ClickEvent event) {
                // TODO: Implement buttonClick
            }
        });
        ok.
    }

    @AutoG
    public                                            {
        //
```

```
Add BlurListener
Add ClickListener
Add FocusListener
Add ValueChangeListener
Add ValueChangeListener
addListener(BlurListener)
```

```
string:\/\/\/HelloWorld.java:7: cannot find symbol\nsymbol  : class Objecdt\nlo...   View (100%)
```

Figure 4. CoRED with generated listener, error tooltip and suggestion box opened (last two cannot normally be opened simultaneously)

suggestions, we load the class and use reflection[10] to ask for its public methods and variables.

Collaborative features

It is natural to consider web applications as multiuser applications. The main requirement for a multiuser application is the shared data [11]. When all the users are connected to the same system it seems only reasonable to assume that in addition to interacting with the system, they are communicating with each other. In our case collaboration means simultaneous editing of code document and some features inspired by social media.

Collaborative editing

For collaborative editing, we decided to use Neil Frasers Differential Synchronization with shadows [5]. It is a robust and convergent collaborative editing algorithm with open source implementations available on various languages, including Java and JavaScript.

The basic idea of Differential Synchronization is the following: the server stores the shared document, and each

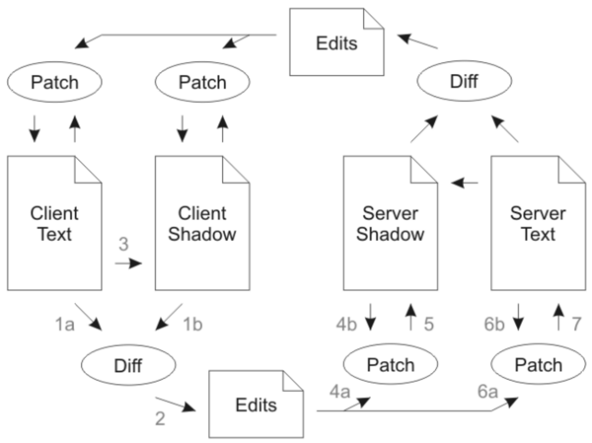
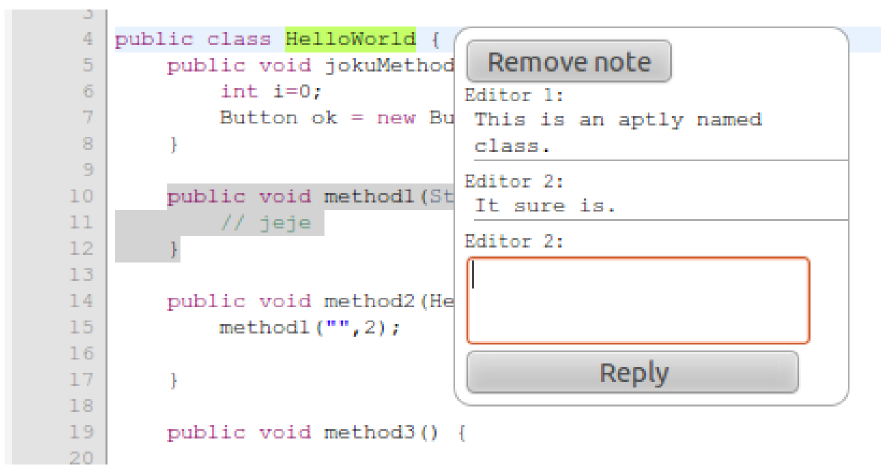


Figure 5. Neil Fraser's Differential synchronization with shadows [11]

[10]Reflection (Java SE Documentation) http://download.oracle.com/javase/6/docs/technotes/guides/reflection



```
public class HelloWorld {
    public void jokuMethod
        int i=0;
        Button ok = new Bu
    }

    public void method1(St
        // jeje
    }

    public void method2(He
        method1("",2);

    }

    public void method3() {
```

Editor 1:
This is an aptly named class.

Editor 2:
It sure is.

Editor 2:

Figure 6 . A note related to “HelloWorld” and a reply in CoRED. The method called “method1” is locked by another editor.

client has a separate shadow copy of the document both on server and client side, along with the copy they are editing (Figure 5). When a client changes its document, the differences between the newly edited document and the latest shadow are calculated using Myer’s algorithm [8]. A patch (made using Bitap matching algorithm [18]) containing the differences is sent to the server. The patch is used both for keeping the server side shadow in sync, and for applying the changes to the shared document. The algorithm is symmetrical in a sense that the changes made to the shared document by other clients, is communicated to the client in the same way as the changes from the client to the server.

Differential synchronization is relatively easy to implement and it meets three basic demands often set for collaborative editing [16]: Firstly, it has high responsiveness. The edits can be done locally and only the differences are delivered to the server. Therefore the local actions are as quick as in a single-user editor. Secondly, it has high concurrency rate and multiple users can simultaneously edit any part of the document. Finally, it is able to hide communication latencies to some extent. Naturally, when latencies grow the conflict rate rises.

In addition to differential synchronization, there are plenty of ways to implement collaborative editing. One approach would be to lock the document or a subsection of the document before editing. A drawback with the locking approach is that a user must wait for the lock to be confirmed by a server, resulting to a loss in usability [15].

Another commonly used technique in collaborative editing is Operational Transform (OT) [3, 4, 7], used in e.g. Ethercodes[11] and Google Docs[12]. In OT, each *operation* (insert, delete, etc.) is recorded and sent to other clients who then transform the operation to take into account all the concurrently executed operations. In practical use, the OT implementations must deal with quite a large number of different editor actions including cut, paste, auto-corrections, auto-indentation, and suggestions, which may make the implementation problematic [5].

Features inspired by social media

In addition to plain collaboration in the form of writing code collaboratively, there are also some features inspired by social media. Perhaps the most obvious feature is the option to write comments in a fashion normally associated with text rather than code, as demonstrated in Figure 6. When some text is selected (“HelloWorld” in the figure), a popup is shown that allows the user to add a note referring to that text. When a user places the cursor on top of an existing note, the note popup is shown. Other users can reply to the original note, thus extending the note to a discussion, as also shown in Figure 6.

The notes reside in their original logical positions even though the document is modified. The start and the end positions of a note are marked with a *marker*. When a user inserts (or deletes) text before a marker, it is moved forward (or backward). The positions of markers are kept in synchronization similarly to the differential synchronization algorithm by sending, along with the text differences, also the position changes of markers from the client to the server and back.

In CoRED, it is also possible to lock portions of the document to be edited only by one editor. For example, if a user wanted to make changes to a specific function without anybody interfering, he/she could lock a function by selecting the function and clicking a "Lock for me" button in a popup that showed up after selecting. After locking, the locked area is shown as grey for other users and they are unable to edit it although they can still add notes to it. Like notes, locks also have a start marker and an end marker which retain their logical positions.

[11] EtherCodes: Online Collaborative Code Editing. http://gigaom.com/collaboration/ethercodes-online-collaborative-code-editing/

[12] Google Docs. http://docs.google.com/

The locking requests are sent to the server, and if no one else had an overlapping lock, the lock is granted. Users are disallowed to do any operations that would modify (or delete) a portion locked by another user. Such actions are already blocked on the client side, but the validity of edits is also checked on the server side. The additional server side check is needed because a lock may have been granted to editor A while editor B was editing the same part of the document. Thus, when B sends its modifications to the server, they are discarded.

Performance considerations

The most time-consuming operation on the server side CoRED is the compiling of the edited Java source code. The result of the compilation is needed to present the user with error messages and code suggestions.

In our brief experiments, when running the server on a laptop containing Core 2 Duo processor, the compilation of a typical Java file with a couple of hundreds of lines, took tens of milliseconds. For sufficient user experience, the compilation needs to be done at most once a second, which could be managed on quite modest hardware.

Another heavy operation is applying patches to the collaborative document. The resource consumption of patching depends on the density of edits done on the document. It should be noted that there is a practical usability limit on how many people can edit the same document simultaneously, which is most likely a more constraining factor than the performance of patching. In our brief tests where a few people edited the same document simultaneously, there were no performance issues.

Editing multiple documents can be done on separate instances of CoRED, which could easily be distributed on multiple server machines if needed. Consequently we do not presently consider this as a major performance bottleneck.

CUSTOMIZING THE EDITOR

CoRED can be customized and extended in several ways. As mentioned, it is possible to use new error checkers and suggesters instead of, or in addition to, the existing components. For example, it is possible to use CoRED for another language than Java by implementing custom suggesters and error checkers, and changing the syntax highlighting and indentation of the front-end editor.

Suggestions

In case of Java based frameworks, it is possible to use our Java suggester. The required jar files of the target framework just have to be added to classpath. If the need for special suggestions related to a specific framework arises, it is possible to create custom suggesters by implementing the Suggester interface in Figure 3.

For example, when developing for Vaadin framework, there is often a need to add an anonymous listener for an UI component, such as ClickListener for a button. We have developed a Vaadin specific suggester, which is to be used in addition to the standard Java suggester. It generates empty skeletons for anonymous listeners where applicable. For example, let us assume that there is a Vaadin Button called myButton defined in the scope of the cursor. When the user types “myButton.”, one of the suggestions is to add an anonymous ClickListener. An illustration of the above case can be seen in Figure 4. The anonymous skeletons are created using Java reflection. Similar suggesters can be easily developed for other Java frameworks.

The suggestion feature is not limited to Java language, although we have not implemented any non-Java suggesters. It is possible to develop arbitrary suggesters for any language. However, creating a useful suggester for a dynamically typed language such as JavaScript is more difficult.

Error checking

As our error checker relies on the Java compiler, it can be used with any Java-based framework as long as the correct libraries are in the classpath. The standard Java error checking is most likely sufficient for most Java projects. Although it is possible to create custom error checkers by implementing the ErrorChecker interface (Figure 3).

Once again, things get more difficult with non-Java frameworks. Error checkers can be created for any language but CoRED offers no support for non-Java ones. The Ace editor we use as a front-end, contains a JavaScript error checker that is not a part of the CoRED architecture.

Front-end editor

Ace, the front-end editor used in CoRED, is highly customizable. It offers the possibility to change its appearance and define custom highlighting rules in separate configuration files. Ace already contains the files for the most popular languages such as JavaScript, HTML, XML, PHP, C++. Thus, adapting the front-end to be used for developing for other languages than Java is very simple.

If Ace, for some reason, does not meet our requirements, it is even possible to use another front-end, leaving the rest of CoRED intact. The front-end component must implement the FrontEnd interface (Figure 3). The interface defines the methods needed for the editor, including setting and getting the editor text, changing the cursor position, setting callbacks for changes, displaying error markers, and so on.

RELATED WORK AND COMPARISON

CoRED is not the first in the field of collaborative browser-based code editors. The first collaborative editor was presented as early as 1968 in the demo, retrospectively named as “The Mother of All Demos”[13]. Naturally, the demo in 1968 did not work on the browser. One of the first collaborative editors runnable in a browser (using Java Applets) was REDUCE [17]. The Web 2.0 phenomenon introduced the browser-based collaborative editors to a

[13] Wikipedia – The Mother of All Demos. http://en.wikipedia.org/wiki/The_Mother_of_All_Demos

larger audience. One of the most successful was Writely that later evolved to Google Docs. The step from a text editor to simple code editor is quite short, and currently lots of browser based collaborative code editors are available.

Most of the browser based code editors have some of the same features as CoRED, but none has exactly the same ones. Many of the competing editors like CodeMirror and Ace are aimed for only code editing, not for creating applications. Like in the desktop world, also in the web there are wide variety of different languages and frameworks. Thus, it is virtually impossible to create one IDE that would support the whole project from writing the code from scratch to publishing it for all of the frameworks. As CoRED is a part of Arvue IDE, our goal is exactly the whole process from beginning to deploying to the web. Akshell[14] has the similar kind of framework approach. It allows a developer to implement both the client and the server with JavaScript. The system also takes care of deploying. In addition, Orion project by Eclipse community limits the frameworks that the user can use. Orion is still in a proposal phase but it looks promising.

When comparing the features of other editors, most code editors have code coloring and indenting available for several different programming languages. To the best of our knowledge, there is only one other example of the browser based editor with code completion and error checking. The editor, named WWWorkspace[15], uses Eclipse as a back-end. As our editor has probably some scalability issues because of JDK, one can only imagine the situation with a massive editor like Eclipse. Like CoRED, WWWorkspace is also using strongly typed Java language. Most of other code editor examples are aimed for weakly typed dynamic languages like JavaScript. JavaScript has its own good sides, but it is virtually impossible to do semantic checking for weakly typed interpretive language.

As a further comparison, CoRED can be easily extended to support any Java based framework. As a downside, CoRED is heavily dependent of the server side and it cannot be used in offline mode like, for example, Google Docs.

FUTURE WORK

Since we are still working with the editor, it is obvious that numerous directions to future work exist. To begin with, introducing support for other phases of development work, such as requirements or project management, would be natural extensions. Then, also these tasks could be turned collaborative as well as more closely linked with development activities. Currently, we assume that these features come from environments that use CoRED as a component.

[14]Akshell. http://www.akshell.com/

[15]WWWorkspace. http://www.willryan.co.uk/WWWorkspace/

Another obvious direction for future work is to perform the series of usability studies in order to find out how programmers wish to use the facilities of the system. This could take place in the form of a coding camp, where students would take part in the experiment, and provide feedback on the system. Based on the results, we can then further refine the implementation and focus on parts that provide most support for the actual development work. In addition, the results of such studies could also help us to identify more potentially useful features that are commonplace in social media but not widely applied in software development.

Finally, in order to gain experiences from a larger user base, we are going to publish CoRED in the open Vaadin Directory[16] as a re-usable add-on for all the Vaadin Community. CoRED can be used as a component of other Vaadin projects or as a stand-alone application. It will also be used as a part of the IDE called Arvue. Currently, Arvue is in an early alpha stage. The other components are going to be the graphical designer for generating user interfaces and the capability to save applications to Git version management. In addition the developed applications can be published directly to the offered cloud. Arvue is mostly designed for creating small Vaadin applications and for testing purposes, but there is no obvious reason why it could not be used more generally.

CONCLUSION

In this paper, we described the browser based editor CoRED intended for collaborative real-time editing of the Java based source codes. We extended the system to offer some Vaadin framework specific features for developing the web applications in the web. As editing features, CoRED contains highlighting, indentation, semantic error checking and code completion. As a combination, this set of features is unique when compared with other browser based code editors. CoRED is going to be published in the Vaadin Directory as a re-usable add-on. It will also be used as a part of the web based IDE called Arvue.

CoRED is built to be modular and many of its parts can be replaced or extended. In this paper, we gave a small summary of what kind of modifications are needed to extend CoRED to different frameworks. In the case of Java based frameworks, just small additions for the suggestion component and error checker are needed. However, for non-Java frameworks more laborious modifications are needed and the benefit of using CoRED is smaller. Fortunately, communication from client to server and collaborative features should work without problems in frameworks of all kind.

As a further contribution, we discussed on how the web applications should be developed and made a small comparison between our editor and other web based code

[16]Directory – vaadin.com. http://vaadin.com/directory

editors available. As a piece of future work it would be interesting to do more experimentation on the usability as well as the scalability of our system.

REFERENCES

1. Begel, A., DeLine, R., and Zimmermann, T. Social media for software engineering. In *Proceedings of the Workshop on Future of Software Engineering Research*, pp. 33–38. Santa Fe, NM, USA, 2010.
2. Curtis, B., Krasner, H., and Iscoe, N. A Field Study of the Software Design Process for Large Systems. Communications of the ACM 31(11), pp. 1268–1287. 1988.
3. Davis, A. H., Sun, C., and Lu, J. Generalizing operational transformation to the standard general markup language. In *Proceedings of the 2002 ACM conference on Computer supported cooperative work*, pp. 58–67. New York, NY, USA, 2002.
4. Ellis, C. A., and Gibbs, S. J. Concurrency control in groupware systems. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, pp. 399–407. Portland, OR, USA, 1989.
5. Fraser, N. Differential Synchronization. In *Proceedings of the 2009 ACM Symposium on Document Engineering*, pp. 13–20. New York, NY, USA, 2009.
6. Grönroos, M. Book of Vaadin. Uniprint. 2011.
7. Ignat, C-L., and Norrie M. C. Customizable collaborative editor relying on treeOPT algorithm. In *Proceedings of the eighth conference on European Conference on Computer Supported Cooperative Work*, pp. 315–334. Norwell, MA, USA, 2003.
8. Myers, E. W. An O(ND) difference algorithm and its variations. Algorithmica 1(1), pp. 251–266. 1986.
9. O'Reilly, T. What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software. O’Reilly. 2005.
10. Paulson, L. D. Building rich web applications with Ajax. Computer 38(10), pp. 14–17. 2005.
11. Patterson, J. F., Hill, R. D., Rohall, S. L., and Meeks S. W. Rendezvous: an architecture for synchronous multi-user applications. In *Proceedings of the 1990 ACM conference on Computer-supported cooperative work*, pp. 317–328. New York, NY, USA, 1990.
12. Perry, Bruce W. Google Web Toolkit for Ajax. O'Reilly Short Cuts. O'Reilly, 2007.
13. Sahay, S. Global Software Alliances: The Challenge of 'Standardization'. Scandinavian Journal of Information Systems 15, pp. 3–21. 2003.
14. Storey, M-A., Treude, C., van Deursen, A, and Cheng, L-T. The impact of social media on software engineering practices and tools. In *Proceedings of the Workshop on Future of Software Engineering Research*, pp. 359–364. Santa Fe, NM, USA, 2010.
15. Sun. C. Optional and Responsive Fine-Grain Locking in Internet-Based Collaborative Systems. IEEE Transactions on Parallel and Distributed Systems 13(9), pp. 994–1008. 2002.
16. Sun, C., Jia, X., Zhang, Y., Yang, Y., and Chen, D. Achieving Convergence, Causality-Preservation, and Intention-Preservation in Real-Time Cooperative Editing Systems. ACM Transactions on Computer-Human Interaction 5(1), pp. 63–108. 1998.
17. Sun, C., Jia, X., Zhang, Y., Yang, Y., and Zhang, Y. REDUCE: a prototypical cooperative editing system. In *Proceedings of the Seventh International Conference on Human-Computer Interaction*, pp. 89–92. 1997.
18. Sun, W. Manbed, U. Fast text searching with errors. Communications of the ACM, 35(10), pp. 83–91. 1992.