2013 16th International Conference on Network-Based Information Systems

Implementation of Web-based Interactive Interface as Software Execution Environment

Hideaki YANAGISAWA, Kayo KONDO
Department of Computer Science and Electronic Engineering
Tokuyama College of Technology
Gakuendai, Shunan, Yamaguchi 745-8585, Japan
yanagisawa@tokuyama.ac.jp, i06kondou@tokuyama.ac.jp

***Abstract*—Cloud computing has become very popular in recent years. Various PaaS systems for collaborative software development and DaaS systems for developing software have been proposed. We also have proposed a PaaS system that delivers both a server-side development and server-side execution environment for character user interface applications that can be executed on the server machine and a server-side development and client-side execution environment for graphical user interface applications that can be executed on the client machine. Our previous system adapted a traditional Web-based interactive interface method that is implemented as internal file access. Therefore, the response time depends on disk access and the system. To improve the response time, we implemented input data flow with instantiation of process management and multithread processing. As a result, the average response time was 4.5 times faster than that for the previous system.**

Keywords: cloud computing, PaaS (Platform as a Service), WebOS, SDCE (Server-side Development and Client-side Execution), SDSE (Server-side Development and Client-side Execution)

I. Introduction

Cloud computing is the delivery of computing facilities as a service rather than a product, using shared resources, software, and information over a network. Cloud computing provides computation, application software, data access, and storage services that do not require end-user knowledge of the physical location or configuration of the systems delivering the services. Cloud computing is categorized as SaaS (Software as a Service), PaaS (Platform as a Service), and IaaS (Infrastructure as a Service). PaaS provides suitable collaborative development environments to implement software as a team.

PaaS systems can deliver Web-based software development environments; however, the applications are typically developed and executed on the same server. Since the interface of a PaaS system is a Web browser, in order to execute graphical user interface (GUI) applications on such a system, the GUI applications need to be translated into JavaScript or a Java servlet. Virtual Network Computing (VNC) is another possible way to develop and execute GUI applications. VNC enables software developers to implement and execute application software on the same server.

In a WebOS environment, GUI applications are customized for the WebOS. As such, portability of the GUI applications is not considered. On the other hand, VNC allows users to control remote hosts. It is also possible to execute GUI applications on the remote host. However, to build a VNC environment, high-performance computers (with a high-performance CPU and vast memory capacity) are required, because the VNC server requests a virtual machine environment for each client. Instead of installing the software development environment on the client machine, VNC server and VNC client settings are required.

It is difficult to develop GUI applications that can run on multiple platforms (i.e., different operating systems (OSs) such as Windows, MacOS, and Linux). To develop applications within a team, a collaborative development environment is required.

We have thus proposed the Server-side Development and Client-side Execution (SDCE) environment for GUI application development and Server-side Development and Server-side Execution (SDSE) environment for CUI application development and/or execution. In the SDCE/SDSE environment, developers implement and test applications on a server, and share the source and/or binary code.

Since PaaS systems assume an online network, if the network is offline, applications on the PaaS system cannot be executed. On the other hand, using the SDCE environment, once a GUI application has been launched on the client side, it is possible to use such an application even if the network is offline, because the GUI application is downloaded to the client machine automatically. These GUI applications are assumed to be written in Java and adapted using Java Web Start technology [4, 5].

Our original SDSE environment did not support server-side execution of application programs requiring interactive I/O. To improve our SDCE environment, we proposed an interactive interface for a Web-based programming and/or execution environment [18]. Using the interactive interface, developers can run application programs that request user data while the application programs are running. However, the response time was sometimes long, and the environment is not user-friendly

Therefore, this paper describes an improvement of the response time for the SDSE environment. The rest of the paper is organized as follows. Section II discusses related work. Section III introduces the SDCE/SDSE environment, while Section IV explains the implementation of traditional (our previous system) and proposed interactive interfaces. We evaluate the proposed interactive I/O environment in Section V, and Section VI concludes the paper.

978-0-7695-5052-7/13 $26.00 © 2013 IEEE
DOI 10.1109/NBiS.2013.61

II. Related work

As related works, we introduce Web-based terminals (WebTTY [6], Anyterm [7], and Ajaxterm [8]). We also introduce related techniques (Ext JS [11], Ext GWT [1], GWT [12], YUI [13]).

The WebTTY package allows any Linux terminal process to be controlled via a text area HTML element on a Web page. The output from the server process is collected on the server side, and is sent to a text area element. Key presses in the text area are collected on the client side and sent to the server process. WebTTY uses AJAX/DHTML patterns to achieve page updates without refreshing the entire page.

Anyterm is a combination of a Web page and a process running on a Web server that provides this access. Anyterm consists of JavaScript on a Web page, an XMLHttpRequest channel on standard ports back to the server, an HTTP proxy such as Apache's mod_proxy, and the Anyterm daemon. The daemon uses a pseudo-terminal to communicate with a shell or other application, and includes terminal emulation. Key presses are detected by the JavaScript, which sends them to the daemon; changes to the emulated screen are sent from the daemon to the JavaScript, which updates the display. Performance is reasonable and SSL can be used to secure the connection.

An easier way of using Anyterm is provided by my.anyterm.org. Instead of installing Anyterm on their own systems, users can log in to my.anyterm.org from a browser and SSH or telnet from there. The site is designed for system administrators and others who require a backup remote access method that works from "Web-only" Internet cafes or from behind corporate firewalls. Anyterm can use almost any Web browser and even works through firewalls. By joining my.anyterm.org, users can access their systems immediately via the Anyterm server without having to install any software. Alternatively, the Anyterm software can be run on a user’s own system.

Ajaxterm is a Web-based terminal that was totally inspired by and works almost exactly like Anyterm, except that it is much easier to install. Ajaxterm is written in Python (with some Ajax JavaScript for the client side). It is very simple to install on Linux, MacOS X, FreeBSD, Solaris, cygwin, and any Unix that runs Python. Compared with Anyterm, there are no partial updates; Ajaxterm updates either the entire screen or nothing, which makes the code simpler and faster. HTTP replies are always gzencoded. When used in 80 × 25 mode, almost all of these are smaller than 1500 bytes (the size of an Ethernet frame) and the current screen is merely replaced with the reply (without any JavaScript string handling). Ajaxterm polls the server for updates with an exponentially growing timeout when the screen has not changed. The timeout is reset as soon as a key is pressed. Since Anyterm blocks a pending request and uses a parallel connection for key presses, the Anyterm approach is better when there are no key presses.

Ext GWT is a Web application framework comprising a Java library for developing rich Internet applications. Ext GWT uses the Google Web Toolkit (GWT) to create Web apps and provides a configurable GUI widget.

The GWT is a development toolkit for building and optimizing complex browser-based applications. Its goal is to enable productive development of high-performance Web applications without the developer having to be an expert in browser quirks, XMLHttpRequest, or JavaScript. GWT allows the user to code in Java, and it then compiles the code into highly optimized cross-browser JavaScript.

The YUI Library is a set of utilities and controls written in JavaScript and CSS for building rich interactive Web applications using techniques such as DOM scripting, DHTML, and AJAX.

We proposed an SDSE/SDCE environment, which is an integrated development environment for developing software on the Web. SDSE/SDCE supports a text editor, command prompt, and file explorer.

III. SDCE/SDSE environment

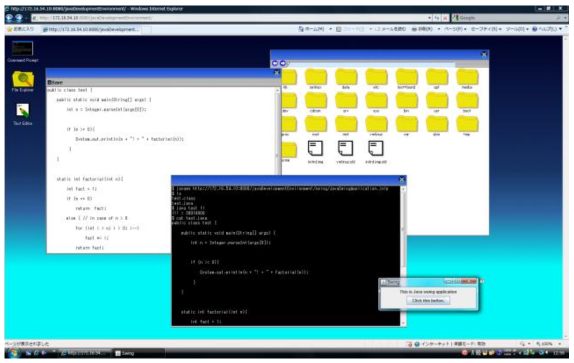
In this section, we introduce the user interface, the SDCE environment for GUI application and the SDSE environment for CUI application. For more details, please refer to [2, 3].

A. *Features and User Interface*

Complicated extensive software is typically developed by a team with many contributors. This applies not only to business applications but also to free software. To implement software in a short period of time, software developers have to collaborate. Thus, collaborative software development tools are necessary. The collaborative software development allows developers to access the same environment, edit the same source code files, and launch applications anytime, anywhere.

For the software development environment, we implemented a text editor, command prompt and file explorer. These features are implemented using JavaScript or Java servlets and shell scripts. A text editor, such as vi or Emacs in UNIX and Notepad in Windows, is used to create the source file for the application on the server. The generated files are stored on the server. A command prompt is used to compile source files, launch the developed application, and execute various commands, including “cp” (file copy), “mkdir” (make directory), “mv” (move file), “cd” (change directory), “ps” (process), and “ls” (list files). We use xterm on UNIX and cmd on Windows. The file explorer is used to display a list of subdirectories and/or files in a directory.

Figure 1 shows the user interface, where the file explorer, text editor and command prompt are launched on the browser. The figure also shows that an application program is downloaded and launched on the client machine.



```
public class test {
  public static void main(String[] args) {
    int n = Integer.parseInt(args[0]);

    if (n >= 0){
      System.out.println(n + "! = " + factorial(n));
    }
  }

  static int factorial(int n){
    int fact = 1;
    if (n == 0)
      return fact;
    else { // in case of n > 0
      for (int i = n; i > 0; i--)
        fact *= i;
      return fact;
```

```
$ javaws http://172.16.54.10:8080/javaDevelopmentEnvironment/swing/JavaSwingApplication.jnlp
$ ls
test.class
test.java
$ java test 11
11! = 39916800
$ cat test.java
public class test {
  public static void main(String[] args) {
    int n = Integer.parseInt(args[0]);

    if (n >= 0){
      System.out.println(n + "! = " + factorial(n));
    }
  }

  static int factorial(int n){
    int fact = 1;
```

Figure 1. User interface in a Web browser.

B. SDCE Environment for GUI Application

VNC allows users to launch GUI applications on the server, but this can increase the server's load average. The server load for a Web-based system is less than that for VNC. However, Web-based systems cannot launch GUI applications on the server. We therefore proposed an SDCE environment such that users can develop an application cooperatively on a Web server and then download and execute the developed application. The executions of GUI applications are processed on the client side. The GUI application is downloaded automatically, making it possible to access both the server and client seamlessly. Although the SDCE is a Web-based system, it is possible to launch GUI applications. The execution flow of the GUI application is shown in Fig. 2. The command, input via the command prompt, is analyzed by JavaScript in a Web browser. The JavaScript accesses the Java Network Launching Protocol (JNLP) file. The JNLP file launches the jar file. The application program is automatically downloaded onto the client machine. The application is launched on the client machine.

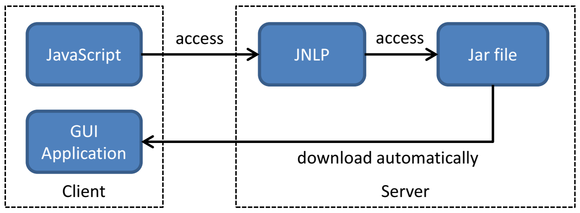
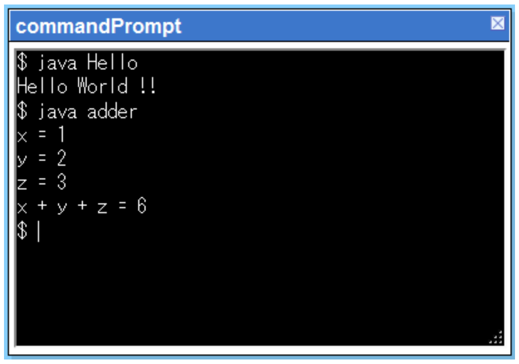


Figure 2. Execution flow of a GUI application.

C. SDSE Environment for CUI Application

Figure 3 shows the command prompt window on a Web browser, in which the "Hello" and "adder" CUI applications are executed. "java" launches the application.

The "Hello" class shows the simple message "Hello World !!". The "adder" class is a simple calculation application, which requests three inputs (x= , y= , z=), calculates "x + y + z", and returns the result. In the following example, "x = ", "y = " and "z = " are outputs of the "adder". "1", "2" and "3" are data input by the user in sequence. The calculation result is then displayed. The detailed implementation is described in section 4.



```
$ java Hello
Hello World !!
$ java adder
x = 1
y = 2
z = 3
x + y + z = 6
$ |
```

Figure 3. Command prompt window in a Web browser.

IV. Implementation of an Interactive Interface

In this section, we describe the implementation of the traditional and proposed interactive interfaces.

A. Traditional interactive interface

Figure 4 shows the data flow on the client side. Client-side programs are written in JavaScript.

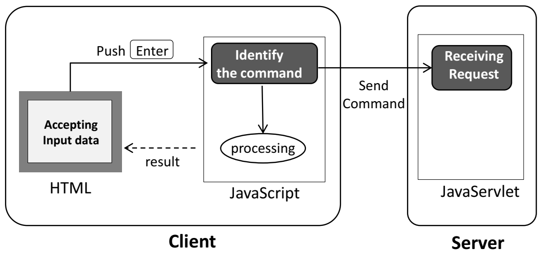


Figure 4. Data flow on the client machine.

Client-side programs wait for a key down event. Client-side programs receive the character string via the command prompt window. Client-side programs split the character string with whitespaces, and analyze the character strings without whitespaces to identify the command. There are two procedures: one for client-side execution (such as "clr-command", which clears the command-prompt window) and the other for server-side execution (such as "ls", "mkdir", compile commands "javac" for the Java language, and "gcc" for the C language). In the case of a server-side command, the client connects to the server and requests execution of the command.

Figure 5 shows the process generation flow on the server. Server-side programs, which are written as Java servlets, receive requests from client-side programs. The server-side programs execute the requests from the clients by creating three threads (RunExProcess, Reader and Writer). RunExProcessThread launches an external program and gets the process ID (PID). ReaderThread watches InputFile, in which input character strings from the client are stored. WriterThread gets the execution results of the external program and writes the execution results to OutputFile.

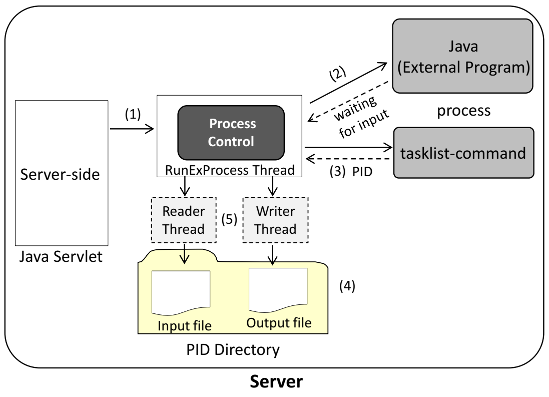


Figure 5. Process generation flow on the server.

Figure 6 shows the data flow between the client and server. The process generation flow on the server is as follows. The server-side program generates RunExProcessThread. RunExProcessThread executes the external program. The server-side program gets the PID of the executed external program and returns the PID to the client. Client-side programs can control the processes on the server using the PID. RunExProcessThread creates a directory named according to the PID, and InputFile and OutputFile. RunExProcessThread generates ReaderThread and WriterThread. ReaderThread gets a character string from InputFile, and passes the character string to the executed external command, which is launched by RunExProcessThread. WriterThread gets the execution results as text from the executed external command, and writes the execution results to OutputFile.

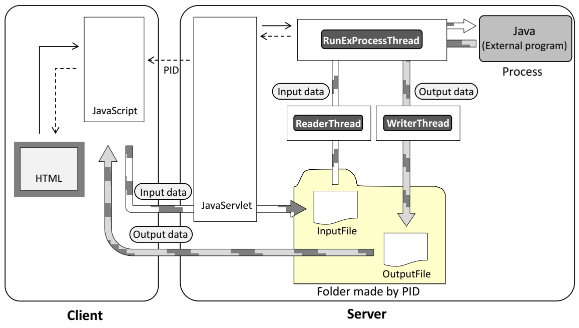


Figure 6. Data flow between client and server.

The data input by the user via the command prompt window in the browser is written to InputFile by the server. The execution results generated by external programs are written to OutputFile. The client gets the updated results and refreshes the results in the command prompt window.

In this method, blocking access using lock file is adapted, because it is necessary to identify user processes with the PID. Moreover, the data input by the user are stored in InputFile. This file access sometimes results in a long response time. Therefore, to improve the response time, we implement an interactive interface with instantiation and multithread attributes.

B. *Proposed Interactive Interface*

Figure 7 shows the proposed interactive interface that is implemented with instantiation and multithread attributes. There are two differences when comparing with the traditional interactive interface.

- Each process is treated as the instance, in which WriterThread is built
- Instead of using InputFile, the data input by the user are stored in the buffer of each process directly.

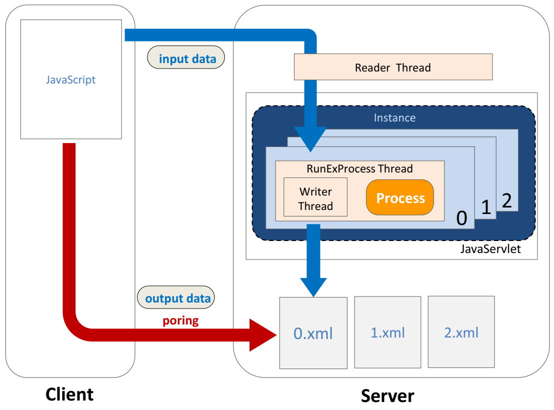


Figure 7. Instantiation and multithread processes

The numbers 0, 1 and 2 in Fig. 7 are the instance numbers, and 0.xml, 1.xml and 2.xml are the respective files into which execution results of each process are written. The client accesses the server with a user id, command prompt id and instance id. The client accesses the N.xml (N is the instance number) file, in which the execution results of the external program are stored. The command prompt window of the Web browser is refreshed with N.xml in which the execution results are stored.

V. EVALUATION

To evaluate our system, we measured the process generation and response times. Table I describes the evaluation environment. Firebug [17] was used to measure all response times. The server and client were connected by a hub.

Table I. Evaluation environment

	Client	Server
OS	Windows XP	Windows 7
CPU	Intel Core2 Duo 2.10 GHz	Intel Core i3 2.93 GHz
Memory	2 GB	1 GB
Network I/F	1000BASE-T	1000BASE-T

A. *Traditional Interactive Interface*

We repeated the experiment 10 times for the traditional interactive interface and measured the process generation and response times. The process generation time includes the time required to create the directory, InputFile and OutputFile for the generated process. Table II gives the response time and process generation time and Fig. 8 shows the relation between process generation and response times, while Fig. 10 shows the data transfer times. Figure 9 presents the data transfer time. The average process generation time is 407.7 ms and the median process generation time is 253.7 ms.

The minimum response time is 228.0 ms (for this time, the process generation time is 192.7 ms) and the maximum response time is 1080.0 ms (for this time, the process generation time is 958.8 ms). The average response time is 470.8 ms and the median response time is 307.5 ms.

Table II. Response and process generation times of the traditional system

	Response (ms)	Process Generation (ms)
1st	309	258
2nd	396	360
3rd	924	778
4th	281	248
5th	260	227
6th	675	622
7th	306	224
8th	228	193
9th	249	209
10th	1080	959

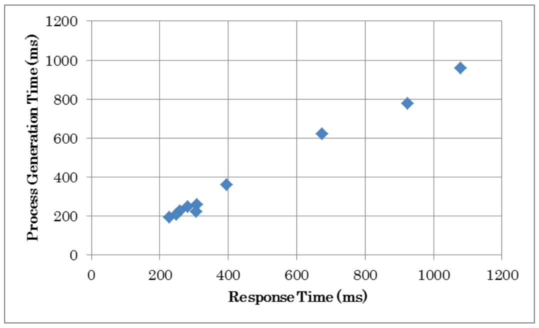


Figure 8. Process generation and response times.

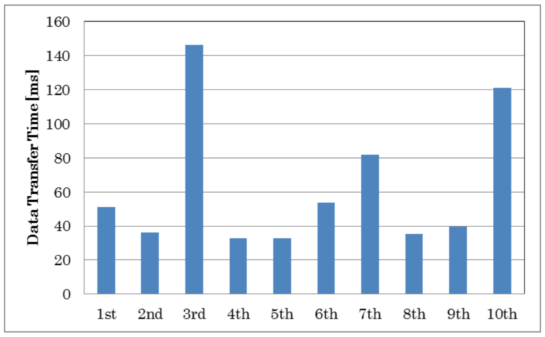


Figure 9. Data transfer times.

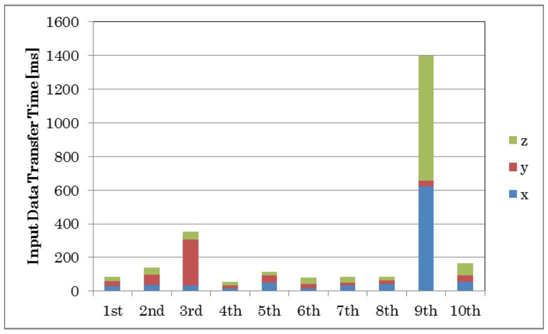


Figure 10. Response times for interactive I/O.

We also measured response times for interactive I/O. We repeated the experiment 10 times. Figure 11 gives the results of the experiments. From Fig. 11, adder.class finished in a short period of time except in the 3rd and 9th experiments.

B. *Proposed Interactive interface*

We again repeated the experiment 10 times for the proposed interactive interface and measured the process generation and response times. The process generation time includes the time required to create the directory and OutputFile for the generated process. Table III gives the response time and process generation time. Figure 11 compares the performances of the traditional and proposed interactive interfaces and the relation between process generation and response times. For the proposed interactive interface, the average process generation time is 22.8 ms and the median process generation time is 26.5 ms. The average response time is 103.5 ms and the median response time is 71.5 ms. It is thus seen that the average response time is 4.5 times faster than that of the traditional system.

Table III. Response and process generation times of the proposed system

	Response (ms)	Process Generation (ms)
1st	262	26
2nd	164	2
3rd	139	14
4th	110	26
5th	74	40
6th	69	28
7th	65	27
8th	65	25
9th	54	28
10th	33	11

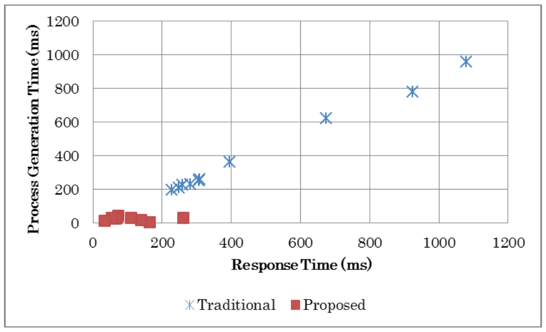


Figure 11. Data transfer times.

VI. Conclusion

We described an improved interactive interface for a Web-based programming and/or execution environment. Using interactive I/O, developers can test their implemented programs more freely.

In our original system, generated processes are identified by a PID, whereas they are identified as the instance in our proposed system. Employing the proposed system, improved response times are possible.

As future works, we intend to evaluate our system performance with many users accessing the system at the same time. In addition, we aim to implement necessary features for a Web-based collaborative development environment.

Moreover, we would like to extend the SDSE/SDCE environment to an educational system for learning a programming language. Our SDSE/SDCE environment is suited to an educational environment, owing to the simplicity of its interface. However, to achieve this, we need to implement features of a learning management system.

References

[1] Ext GWT, http://www.sencha.com/products/extgwt/

[2] H. Yanagisawa, "Evaluation of a Web-based Programming Environment", in Proc. of 2012 15th International Conference on Network-Based Information Systems Workshop, CD-ROM, pp. 633-638, 2012.

[3] H. Yanagisawa, "Web-based Environment for GUI Application Development", in Proc. of 2012 23th International Conference on Advanced Information Networking and Applications Workshops, CD-ROM, pp. 993-998, 2012

[4] Java Web Start Technology, http://www.oracle.com/technetwork/java/javase/javawebstart/index.html

[5] Java™ Web Start Guide, http://docs.oracle.com/javase/7/docs/technotes/guides/javaws/developersguide/contents.html

[6] WebTTY, http://testape.com/webtty_sample.html

[7] Anyterm, http://anyterm.org/

[8] Ajaxterm, http://antony.lesuisse.org/software/ajaxterm/

[9] Amy Editor, http://www.amyeditor.com/

[10] ACE, http://ace.ajax.org/#nav=about

[11] Ext JS, http://www.sencha.com/products/extjs/

[12] GWT, http://code.google.com/intl/webtoolkit/webtoolkit/

[13] YUI, http://developer.yahoo.com/yui/

[14] Java™ Network Launching Protocol & API Specification (JSR-56) Version 7.0, http://download.oracle.com/otndocs/jcp/jnlp-7-mrel-eval-spec

[15] Deploy full-featured applications with Java Web Start, http://www.techrepublic.com/article/deploy-full-featured-applications-with-java-web-start/6120125

[16] JNLP File Syntax, http://docs.oracle.com/javase/7/docs/technotes/guides/javaws/developersguide/syntax.html

[17] Firebug, http://getfirebug.com/

[18] H. Yanagisawa and K. Kondo, "Interactive Interface for Web-based Programming Environment", in Proc. of 2013 27th International Conference on Advanced Information Networking and Applications Workshops, pp. 168 - 173, 2013.