Designing IDE as a Service

Timo Aho* Adnan Ashraf† Marc Englund‡ Joni Katajamäki** Johannes Koskinen*
Janne Lautamäki* Antti Nieminen* Ivan Porres† Ilkka Turunen**

Tampere University of Technology*
P.O.Box 553, FI-33720 Tampere, Finland
firstname.surname@tut.fi, except antti.h.nieminen@tut.fi

Åbo Akademi University†
Joukahainengatan 3-5, FIN-20520 Turku, Finland
firstname.surname@abo.fi

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

JAMK University of Applied Sciences**
Rajakatu 35, FI-40200 Jyväskylä, Finland
firstname.surname@jamk.fi

Abstract

While the popularity of web applications is growing, most of the software is still developed using desktop tools. Nevertheless, a browser-based development environment could offer a multitude of advantages. For example, only an up-to-date web browser is needed and therefore the user has no need to carry out complex tool installation and update procedures. Furthermore, publishing the applications to the web is easy to offer as a feature of IDE, and since the users are already connected via server, collaborative features are easier to implement. For beginning business, effortless publishing offers new possibilities.

In this paper, we present Arvue, a browser-based tool that enables simple development and publishing of web applications. Arvue applications are created on the browser using an UI designer and an integrated code editor. The applications are stored in a version control system provided by Arvue and they can easily be published to the cloud. Publishing the user-created applications may impose resource usage and security issues, which are also addressed in this paper.

Keywords: Cloud, Integrated development environment

1 Introduction

One of the main trends of software engineering is that the applications are moving from desktop to the web browser. The trend has multiple benefits: Web applications are available globally and can be accessed using any up-to-date web browser. They are also easy and inexpensive to maintain. Updates are distributed automatically just by updating the application on the server. The web, and especially the usage of the cloud [1], makes it possible for anybody to implement applications and then with small effort and relatively low costs to offer it to the global market.

Despite the success of the web as an application platform, the vast majority of software is developed using desktop based integrated development environments (IDEs). However, IDE inside a

browser offers several benefits. The developer does not need to worry about installing, configuring or updating the environment as the latest version available on the cloud is automatically used. Along with IDE, there can be additional developing tools such as a development server or a test harness. Since the developers are connected to the cloud, the created projects can easily be stored in the cloud. In addition, it is possible to implement a set of communication, collaboration and project management tools. Furthermore, it reduces errors if we develop and test applications in the same environment as the actual hosting is done.

In this paper, we describe Arvue[1], that provides the IDE and the hosting as a service. The most visible part of Arvue is a browser-based development environment. The environment contains a visual UI designer and a code editor along with an easy way to preview and publish applications. The applications are stored in a version control system integrated with the IDE. We also identify the potential risks related to user-created applications and give our solutions to monitoring and controlling the resource usage of applications as well as to addressing the security concerns.

The paper is organized as follows. In the next section, we give an overview of the technologies used in our implementation. In Section 3, the overall view of our architecture is presented, as well as more detailed descriptions of the components. Finally, in Section 4 we draw some concluding remarks and discuss possibilities for future work.

2 Background

As already mentioned in the introduction, the benefits of hosting applications in the cloud are evident [1]. The philosophy of cloud computing is to see applications as a service instead of a product. In the cloud, the resources behind the applications can be scaled up and down according to ones needs without owning heavy server infrastructure. In an IDE and hosting service like Arvue, the scaling is naturally a very desirable feature because of unpredictable and changing needs. When selecting the cloud provider, the most typical commercial alternatives are Amazon Web Services[2] and Google App Engine[3]. It is also possible to build cloud of your own by using open-source cloud computing frameworks like OpenNebula[4] an Eucalyptus[5].

A remarkable part of web applications is based on Java programming language. Especially Java Servlet [6] is a fundamental building block for Java based web applications and is used in huge number of tools and frameworks. In this work, for implementation, we use Vaadin [3], that is an open-source framework for developing Java Servlet based web applications. For client side features, Vaadin framework extensively relies on the facilities of Google Web Toolkit[6] (GWT). GWT is an open-source development system that allows the developer to write applications in Java and then compile the source code to JavaScript which can be run on all browsers. When using Vaadin, the developer can implement server side application using the full power and flexibility of Java and Vaadin automatically takes care of the client side.

When using web servers like Apache Tomcat[7] or Jetty[8], all the served web applications are located on the same Java virtual machine. However, Java lacks many important supportive features to run multiple applications on a single Java virtual machine. To patch this up, a widely adapted

[1] Available at http://www.arvue.com.
[2] http://aws.amazon.com
[3] http://code.google.com/appengine
[4] http://www.opennebula.org
[5] http://www.eucalyptus.com
[6] http://code.google.com/webtoolkit
[7] http://tomcat.apache.org
[8] http://eclipse.org/jetty

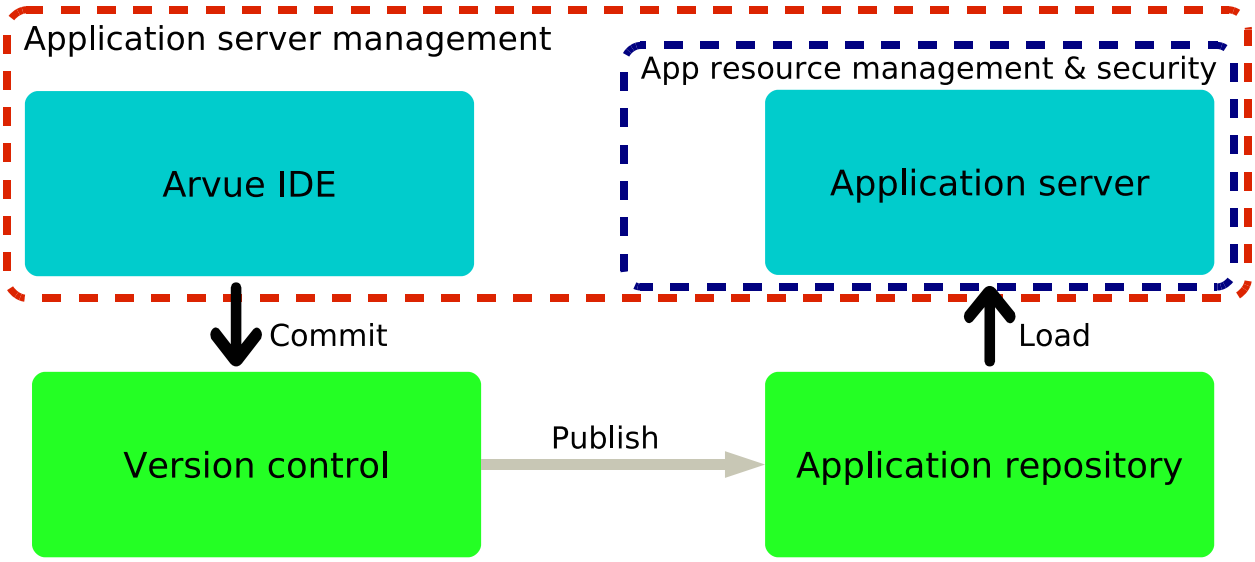

Figure 1. Overall view of Arvue architecture

OSGi[9] specification [13] has been published. OSGi introduces the dynamic component model which allows, e.g., updating, installing and removing components on-the-fly in a single Java virtual machine [4, Section 15.1].

Vaadin framework is compatible with OSGi. As the framework can be configured as an OSGi bundle it is possible for other OSGi bundles to use it directly. In this way, Vaadin web applications can use the Vaadin framework bundle like a dynamic library resulting in very lightweight applications. Thus, the typical size of the Vaadin application bundle file in OSGi is less than 100 kilobytes, and the starting up an application is a light and fast operation.

3 Architecture

In this section, we describe the Arvue architecture and components. The overall architecture of Arvue components is represented in Figure 1. Arvue can be divided roughly in four different components:

- Arvue IDE, both graphical UI and collaborative code editor (CoRED) [7], for creating applications. Components are presented in Section 3.1.
- Version controlling services component covered in Section 3.2.
- Arvue Auto Scaler for publishing the application is described in Section 3.3.
- Monitoring resource usages and security of the applications is covered in Section 3.4.

Arvue IDE is going to be run on a dedicated Jetty WebServer as there is, at least for the start, no need for scaling features for the IDE component. The applications created and published using Arvue IDE are hosted on Amazon Web Services cloud to get high scalability.

As can be seen from the figure, Arvue offers all the tools needed for creating, publishing and hosting the applications. Furthermore, the application sources can be downloaded from the system and edited with an external IDE and then upload back to the repository.

[9] OSGi originally stood for Open Services Gateway initiative framework. The longer form is at present seldom used because the specification has moved beyond the original focus.

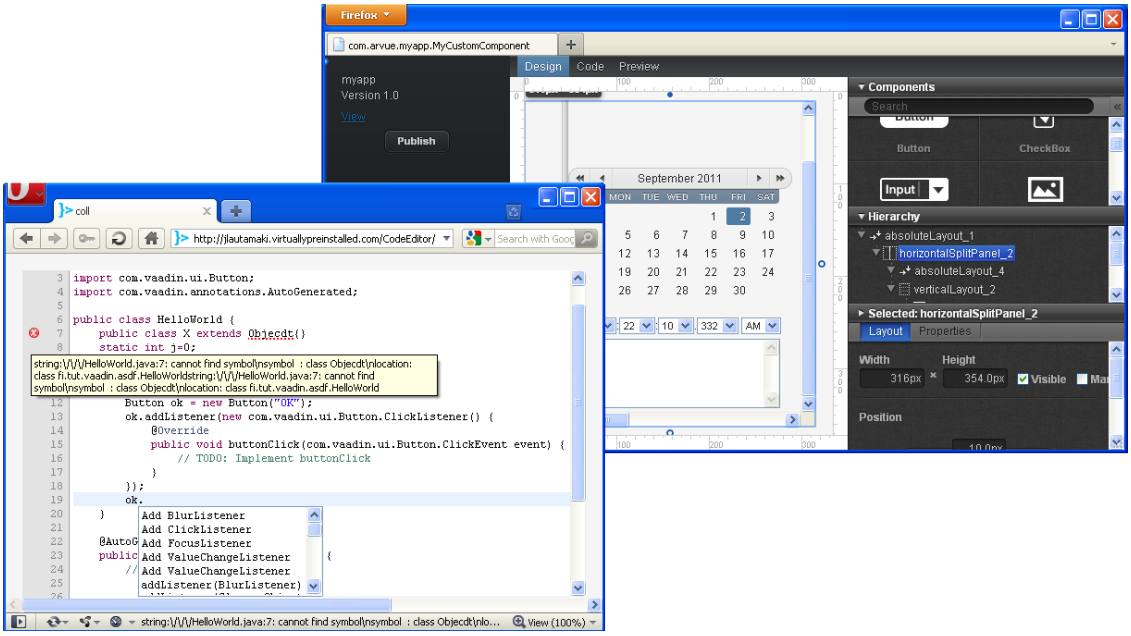

Figure 2. Screenshots on the Arvue IDE. Code editor at left and Graphic UI designer at right.

Let us now go through the components one by one in the following subsections.

3.1 Arvue IDE

Arvue IDE is a browser-based tool for creating Vaadin applications. Furthermore, the applications can be published to the cloud with a single click. Besides being used to create Vaadin applications, the IDE is itself a Vaadin application and could run within the same infrastructure as applications created with it. Arvue IDE contains a visual UI designer and a code editor CoRED (Collaborative Real-time Editor) [7], along with tools for previewing and publishing applications.

In the UI designer presented in Figure 2 (right), the user can create user interfaces by dragging and dropping Vaadin components such as layouts, containers and various kinds of UI elements to the middle panel. The position and size of the elements can be set using a simple mouse-based interface. Further properties of an element, such as its style and whether it is enabled and read-only, can be set by selecting the element and defining its properties in a side panel view. In addition, there is also a hierarchical view of the component structure of the created user interface.

Java source code for the designed application UI is automatically generated. The code can be further edited in the code editor illustrated in Figure 2 (left). Using the editor, the user can add functionality to the application by attaching listeners to the UI components or by any other way possible for a Vaadin application. The UI components can also be added and modified in the code editor. The changes created to the code are reflected back to the UI designer, making a round trip between the UI designer and the code editor.

The code editor component offers various features to help the programmer. It uses Ajax.org Cloud9 Editor[10] (Ace) for Java syntax highlighting, automatic indentation and other basic code editor features. Additionally, our code editor checks errors in the user code. Once a while the code is compiled on the server side using Java Development Kit (JDK) compiler. The possible errors are presented in the code editor along with the error descriptions given by the compiler. In addition the editor offers an automatic code completion. For example, when the user types "`myButton`.", a list

[10] http://ace.ajax.org

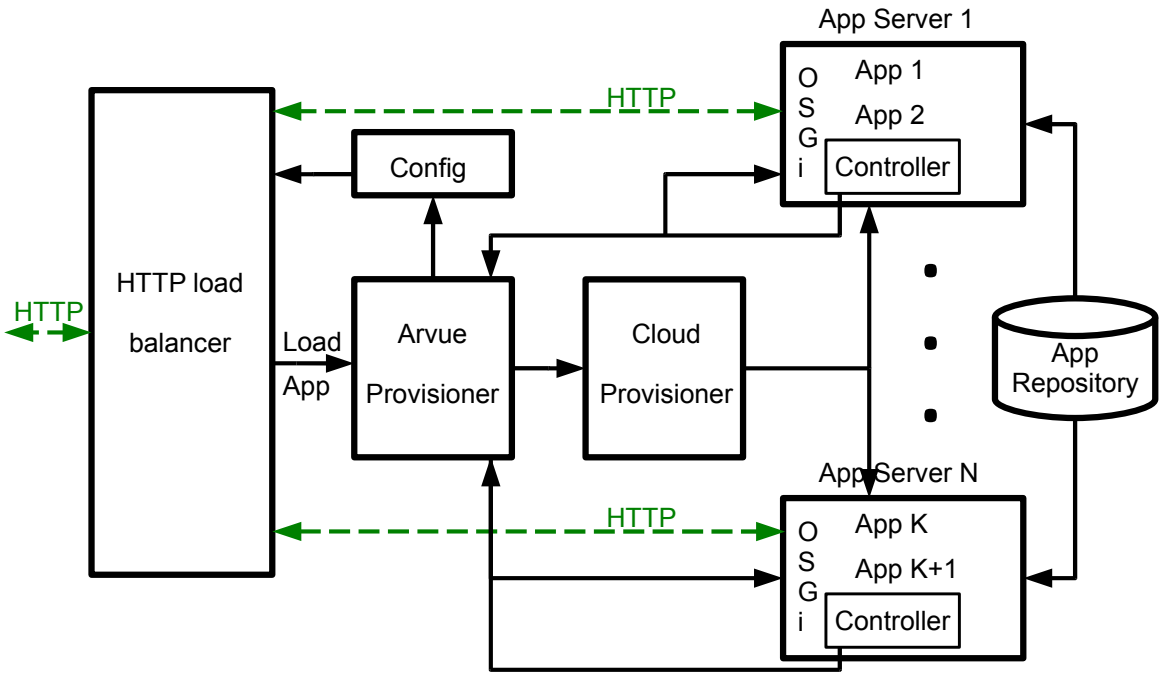

Figure 3. An abstract view of Arvue Auto Scaler.

of the fields and methods of the `myButton` object is shown for the user to select. In addition to the field and method suggestions, there are some Vaadin-specific suggestions. All the suggestions are also generated on the server side using JDK tools. Furthermore, CoRED can work in collaborative mode, meaning that several developers can edit the same source file simultaneously. For further details on the features and implementation of the code editor, we refer to [7].

3.2 Version Control

The code generated by Arvue IDE is saved to a distributed revision control system called Git[11]. Git is meant for managing revisions in a file that has frequent changes possibly by multiple users. The system allows rolling back to previous revisions or assigning creator information for code lines of a file. In practice, Arvue version control system is implemented as an easy-to-use wrapper library built around the JGit[12] that handles all the operations necessary to interact with the repository.

One of the primary features of the version control component is automatic saving. It automatically commits code changes to Arvue Git repository and generates associated to commit messages. Hence, the revisions can be tracked later on. The component also includes features like manual saving and loading of previous work from a specific external repository. This can be done with a simple user interface in the general view of the editor.

3.3 Publishing of Arvue Applications in the Cloud

Arvue Auto Scaler distributes the client application sessions to virtual machine instances in the cloud according to their resource usage. The Auto Scaler consists of a number of subcomponents as shown in Figure 3. In this section, we describe them.

The application servers run on a Virtual Machines (VM) that can be started and terminated on

[11] http://git-scm.com
[12] http://www.eclipse.org/jgit

demand. On each server runs several web applications under OSGi [13] environment. In OSGi environment, the web applications are run as specific components, called OSGi bundles. Bundles can be loaded and unloaded in dynamic fashion and they can communicate with each other.

There are multiple implementations for the OSGi standard [13]: e.g., free open-source Apache Felix[13] and Eclipse Equinox[14] as well as commercial ones like Makewave Knopflerfish[15]. From the open-source implementations, Apache Felix is certified to be compliant with the OSGi specification. Thus, we selected to use this one.

In addition, each application server also runs Controller that monitors, controls, and records application server performance (response time, workload, and resource consumption). Also the performance of individual applications is monitored. The application specific data is generated by Resource Manager which is described in 3.4. The controller sends the data to the Arvue Provisioner as can be seen in Figure 3. Another task of Controller is to control OSGi Felix for loading and unloading of web applications.

The compiled Arvue applications are stored in an application repository. When the balancer receives a request for a web application that is not yet loaded on any of servers, it directs the request to Arvue Provisioner that selects a server and forwards the HTTP request. This causes the server to load the web application from the application repository. After a period of inactivity, the application is unloaded from the server.

The Arvue Provisioner acts as the capacity manager for application servers. Management includes starting and stopping application server instances when there is a need for scaling of the service. While the scaling decisions are made by Arvue Provisioner, the actual lower level tasks are done by Cloud provisioner. In our case, the provisioner is Amazon Web Services cloud, Amazon EC2[16].

All the HTTP requests are routed through a high performance HTTP load balancer and proxy. This way HTTP request load is balanced among the application server instances. As an implementation of the balancer, we use HAProxy[17]. For its functions, the balancer maintains configuration information about application servers that are running and about the web applications executed on each of them.

3.4 Security Issues on the Application Servers

Despite its many merits, OSGi unfortunately does not solve all the security problems attached to multiple public untrusted web application hosting. Some of the vulnerabilities are implied by Java principles [5, 2] and some by the OSGi features [10, 11]. The security issues can be splitted in two categories: the permission control of untrusted OSGi bundles and the more complicated matter of resource usage monitoring and controlling.

Because of the application environment, we are able to solve permission based vulnerabilities moderately simply with the OSGI permissions. The OSGi Security model is based on the Java 2 Specification [12]. The basic idea is that code can be forced to authenticate before it has a permission to execute specific method calls. In our case, authentication by bundle's location is enough. We refer to literature [4, Section 14.7] for more details on the permissions.

For monitoring resource consumption, we have implemented a monitoring component called Resource Manager. In practice, Resource Manager is a simple profiling tool, which is augmented

[13] http://felix.apache.org
[14] http://www.eclipse.org/equinox
[15] http://www.knopflerfish.org
[16] http://aws.amazon.com/ec2
[17] http://haproxy.1wt.eu

with per-application limits such as memory allocation or CPU time used. When one of the limits is reached, Resource Manager tries to interrupt or stop the misbehaving application. In addition to monitoring, it provides resource usage information for Arvue Provisioner via Controller.

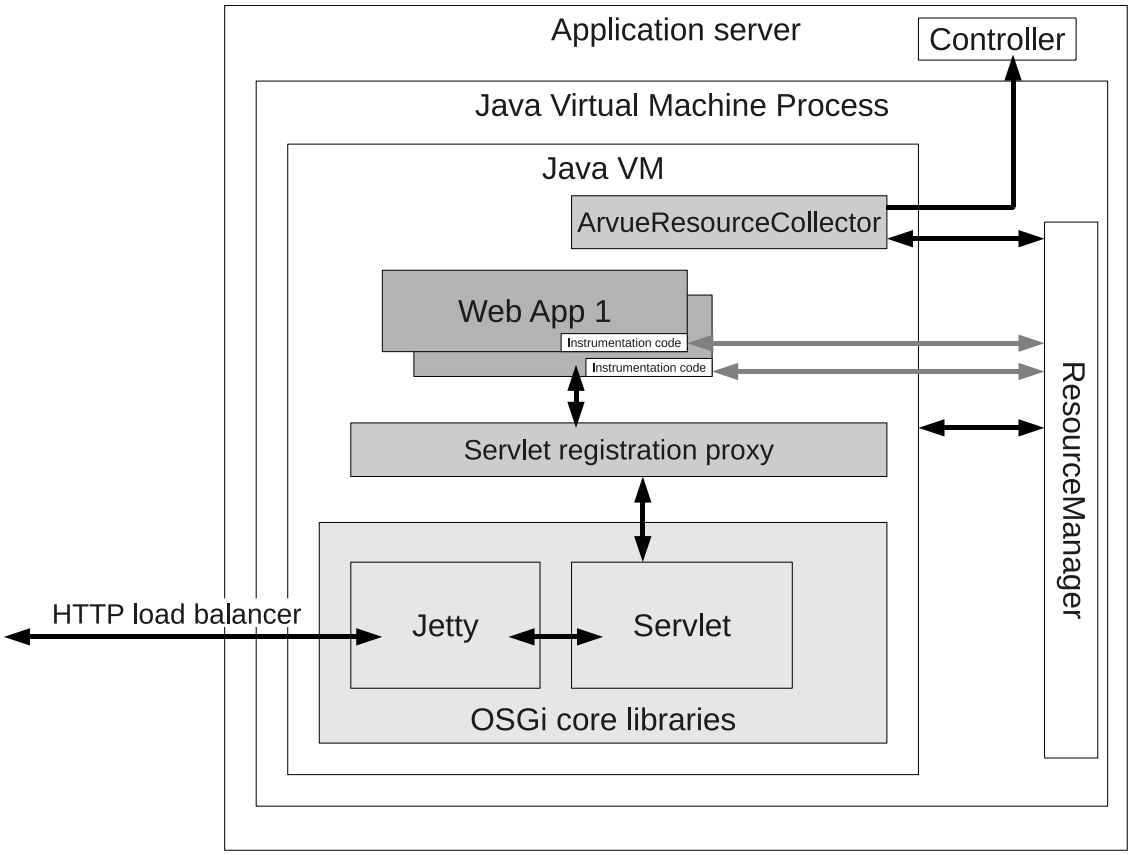

Figure 4. Resource Manager component

Resource Manager contains three different components as shown in Figure 4: the dynamic servlet registration bundle with a service call proxy to add monitoring aspect to web applications, Resource Manager for actual profiling tasks, and ArvueResourceCollector to capture the resource usage data from Resource Manager component.

Resource Manager is based on the Java Virtual Machine Tool Interface (JVMTI) [9] that provides a way to inspect the state of applications running in the Java virtual machine. Resource Manager is hooked to the Java virtual machine so that when the virtual machine starts a new thread, Resource Manager is called by the JVMTI. Similarly, it is called when threads are stopped. Memory allocations can be easily covered by instrumenting the code to call Resource Manager via Java Native Interface (JNI) [8] whenever a new object is allocated. The instrumentation is carried out when the class file loaded, so no preliminary changes to the code are required. To track the release of the memory JVMTI callbacks are used. CPU time is monitored with a separated thread that collects periodically (like once a second) the used CPU time on all the active threads.

4 Concluding Remarks

In this paper, we described a web based integrated development environment and hosting service called Arvue. With the introduced system it is fast and easy even for the beginner without any specific tools installed on the computer to create and publish Vaadin [3] based web applications.

The most visible part of Arvue is a browser-based IDE. The IDE contains a visual UI designer and

a code editor along with an easy way to preview and publish applications. The applications are stored in an integrated version control system and can be published to cloud for anybody to access with a web browser. We also identified the challenges related to running user-created applications and gave our solutions for monitoring and controlling the resource usage of applications and for scaling the system up and down. Decisions concerning the security problems of user-created web applications are also illustrated.

Most of the browser-based development tools have some of the features of Arvue, but none has exactly the same. The goal of Arvue is to take the process from beginning to publishing and hosting to the web. In contrast, most of the current tools are just source code editors, not complete IDEs with hosting.

Nevertheless, Akshell[18] and Cloud9 IDE[19] have a similar kind of approach as Arvue. Both examples allow a developer to implement both the client and the server with JavaScript. Cloud9 IDE also supports some other languages. Both of them also offer hosting services as a part of the deal. As another interesting related work, the Eclipse community has started the project called Orion[20] that has somewhat similar goals with Arvue. None of the mentioned IDEs contain a visual user interface designer available in Arvue IDE.

Since Arvue is still in the early proposal phase, we have numerous directions for future improvement. The collaborative capabilities of the tool have to be improved; in addition to those already implemented in the editor component as described in [7] we should have more tools for project management and for architectural design. Another obvious direction for future work is to perform a series of usability studies in order to find out how programmers wish to use the system. Based on the results, we can further refine the implementation and focus on the parts that provide most support for the actual development work.

References

1. Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. A view of cloud computing. Communications of the ACM 53, 4 (2010), 50–58.
2. Geoffray, N., Thomas, G., Muller, G., Parrend, P., Frénot, S., and Folliot, B. I-JVM: a Java virtual machine for component isolation in OSGi. In International Conference on Dependable Systems and Networks (DSN 09) (Los Alamitos, CA, 2009), IEEE Computer Society.
3. Grönroos, M. Book of Vaadin, 4th ed. Uniprint, Turku, Finland, 2011.
4. Hall, R. S., Pauls, K., McCulloch, S., and Savage, D. OSGi in action: Creating modular applications in Java. Manning Publications, Greenwich, CT, 2010.
5. Java Community Process. Java specification request 121: Application isolation API specification, 2006. Version 2.7, final.
6. Java Community Process. Java specification request 315: Java servlet 3.0 specification, 2009. Version 3.0, final.
7. Lautamäki, J., Nieminen, A., Koskinen, J., Aho, T., Mikkonen, T., and Englund, M. CoRED—Browser-based collaborative real-time editor for Java web applications. In Proceedings of the Fifteenth ACM Conference on Computer Supported Cooperative Work (CSCW 12) (2012), ACM. Submitted.
8. Oracle. Java native interface specification, 2006. Version 6.0.
9. Oracle. JVM tool interface, 2006. Version 1.2.1.

[18] http://www.akshell.com
[19] http://cloud9ide.com
[20] http://wiki.eclipse.org/Orion

10. Parrend, P. Software Security Models for Service-Oriented Programming (SOP) Platforms. PhD thesis, Institut National des Sciences Appliquées de Lyon, France, 2008.
11. Parrend, P., and Frénot, S. Java components vulnerabilities: An experimental classification targeted at the OSGi platform. Tech. Rep. 6231, Institut National de Recherche en Informatique et en Automatique, Le Chesnay Cedex, France, 2007.
12. Sun Microsystems. Java 2 security architecture, 2002. Version 1.2.
13. The OSGi Alliance. OSGi service platform: Core specification, 2009. Release 4, version 4.2.