Volume 7, Issue 1, January 2017 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Constructing a Middleware for Rapid Development of Rich Internet Applications in Ruby Using Vaadin Java Framework

Edin Pjanic, Amer Hasanovic, Semir Puskarevic
Faculty of Electrical Engineering, University of Tuzla,
Bosnia and Herzegovina

DOI: 10.23956/ijarcsse/V7I1/01103

Abstract— Web applications are increasingly becoming the most important platform for software applications. New technologies have allowed for development of Rich Internet Applications (RIA). Many frameworks for development of such applications have been developed, mostly in Java programming language. However, dynamic programming languages, such as Ruby, usually have more expressive syntax and do not require compilation so they are suitable for fast application development with short iteration cycles. In this paper we present an approach to constructing a middleware that allows for development of RIA in the Ruby programming language. The middleware is implemented in JRuby, an impementation of Ruby for Java Virtual Machine, and is based on Vaadin, a Java framework for RIA development. The middleware incorporates an adaptation of DataMapper, a Ruby object-relational mapper, that allows for binding the Vaadin UI components to DataMapper models. With this middleware, the entire RIA logic and UI, together with definition of application's data models, can be completely written in JRuby using Ruby's programming style and expressiveness in order to reduce complexity and increase productivity of the RIA application development.

Keywords— rich internet applications, rapid application development, middleware, JRuby, Vaadin

I. INTRODUCTION

Primarily designed as a platform for information delivery, Web has evolved into one of the most popular Internet services. Web 1.0 denotes the first stage of Web, where resources were pulled from the servers and presented within the Web clients in the form of static Web pages linked together via hyperlinks. The only technology available for describing Web page content and design at that time was Hypertext Markup Language (HTML).

Eventually, people realized that information presented in the Web pages could be dynamic and database driven, which led to building Web applications. The first generation Web applications were page-based, generating the HTML on the fly using server-side custom scripts. Examples of those are CGI, JSP, ASP and PHP.

Advances in the client-side technologies, such as JavaScript and Flash, as well as changes in the HTML specification, allowed for creating of more dynamic and interactive Web applications. Web applications of today often present users with the interface similar to the regular desktop applications, with the features such as drag and drop, intelligent grids, trees and other desktop-like user interface (UI) elements. The term Rich Internet Application (RIA) [1][2] is commonly used to refer to Web applications delivered in this form with Web browser as an application platform. Examples of such applications include text processors, spreadsheets, calendars, online games, etc. Those applications mark the era of Dynamic HTML technologies (DHTML).

Development of RIA is difficult. The programmer must have a deep knowledge of several different DHTML technologies, such as HTML, Cascading Style Sheets (CSS), JavaScript programming language, Asynchronous JavaScript and XML (AJAX), programming language for the server side of the application, tools and libraries for database access. Moreover, there is a number of incompatibilities among major browsers. To overcome these problems, various frameworks for the development of RIA are developed. These frameworks help developers concentrate on the application's design and logic without worrying about the underlying technology details and browser incompatibilities. Most frameworks are based on the Java programming language and they translate the application's Java code to the HTML, CSS and JavaScript code that is executed within the user's Web browser.

Java is utilized for several reasons, such as strong data typing, object orientation, platform independence, debugging, robustness and others. Furthermore, the Java programming language is one of the most successful programming languages, if not the most successful one, and a large number of frameworks, tools and libraries are developed in Java and for the Java platform in all aspects of software engineering.

In the recent years, there is a rise of dynamic programming languages [3], such as Ruby, Python and Groovy, whose development productivity outperforms development productivity of Java programming language [4] [5]. Most of them are interpreted languages and don't have a runtime performance that equals the performance of Java. However, some of the dynamic languages are implemented for Java Virtual Machine (JVM) and can utilize Java libraries and frameworks very efficiently. According to Tate [5], they are probably successors to Java.

Frameworks for rapid application development (RAD) of classical Web applications based on the dynamic programming languages are already proven. Those frameworks usually incorporate a number of domain specific languages (DSLs) for even better productivity and expressiveness. Ruby [6] was put on the Web development roadmap thanks to its range of DSLs such as Ruby on Rails and Sinatra Web DSLs as well as DSLs and libraries for object-relational mapping (ORM) such as ActiveRecord, DataMapper and Sequel.

Ruby on Rails [7] is the most popular Ruby framework for Web application development which incorporates a number of DSLs for various purposes. However, although Rails has extensive tools and libraries for AJAX and JavaScript to support Web 2.0 applications development, making a true rich interface would be very difficult. Moreover, there are no established Ruby frameworks that could allow for development of true RIAs.

In this paper we present a middleware for the development of RIAs using the Ruby programming language and its Java implementation, JRuby [8]. We adapt and utilize an existing framework in order to reduce complexity and increase productivity of the RIA application development by utilizing the features that Ruby, as a dynamic language, offers. The same principles presented in this paper could be used for the development of RIA in most any other JVM based dynamic programming language.

The paper is organized as follows. We start with the short presentation of some interesting features of Ruby and JRuby in Section 0. In Section 0 we give an overview of the two different approaches to frameworks for RIA development and outline their basic ideas. A middleware for RIA development in Ruby, based on the Vaadin framework, is presented in Section 0. Finally, we conclude and give an outlook for future work.

II. RUBY AND JRUBY

Java is still the main programming language for the Internet domain. Java is indispensable for writing middleware, the systems software that stands between an application and an operating system. Java's libraries, performance, and portability make it an excellent choice for middleware. However, Java is neither simple nor friendly to very short software development iterations. Dynamic languages, such as Ruby, allow moving from one change in the code to the next without a compile/deploy cycle. More importantly, Ruby has a more expressive syntax, and frameworks based on metaprogramming concepts [9], such as Ruby on Rails, that take a developer to a higher and more productive level of abstraction.

Ruby is a dynamic language that's usually grouped with other scripting languages, such as Smalltalk and Python. It has powerful features, missing in C++ and Java, such as:

- full object orientation, since everything in Ruby is an object
- dynamic typing, and dynamic classes and objects that can change during runtime
- native support for regular expressions and containers
- support for metaprogramming.

Matz's Ruby Interpreter or Ruby MRI, is the reference implementation of Ruby, written in C by Yukihiro Matsumoto. However, this is not the only implementation of Ruby. One especially successful open source implementation, called JRuby, is written for the JVM. While being fully compatible with Ruby MRI, it offers certain advantages. First, since it is implemented on top of the JVM, JRuby threads are mapped to the kernel threads, and unicode strings are automatically supported in JRuby. Second, JRuby can seamlessly interoperate with Java. Meaning, that we can use Java objects as normal Ruby objects, and vice versa. Hence, we can exploit the wealth of Java libraries, using the power of Ruby's flexible syntax.

III. TWO APPROACHES IN RIA DEVELOPMENT

Before we describe the middleware we designed to adapt a Java framework for RIA development in Ruby, we first explain the basic ideas behind RIA development frameworks.

RIA development frameworks can be grouped into two categories. The first category favors a client-centric approach in application design and organization and the second favors a server-centric approach.

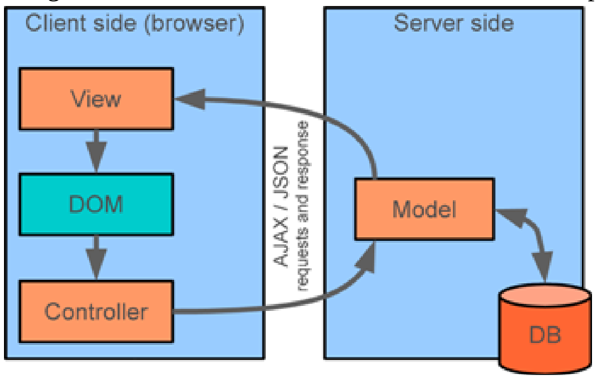


Fig. 1 A client-centric approach in RIA development

Client-centric frameworks place the business logic of the application on the client side (Web browser), as shown in Fig. 1. The client-side of the application is responsible for handling of all UI activities, including event handling, which results in improved UI performance, reduced bandwidth and server load. The server side of the application in this case is in charge for authentication, authorization and database access. Client-centric approach is good in applications that require autonomous or highly responsive UI and do not have frequent database access. The list of types of applications that utilize this approach includes office applications, graphical design applications, games and similar. Google Web Toolkit and Adobe Flex are examples of client-centric frameworks.

In server-centric approach, all application logic and processing is placed on the server side and only a lightweight client is sent to the browser, as shown in Fig. 2. The client side of the framework is responsible for rendering the user interface and forwarding the UI events to the server side. The server side of the framework receives the UI events and handles them appropriately. All application logic, authentication and authorization is in one place on the server side so the application architecture is less complicated. The developer usually writes an application as it is a desktop application. Some of server-centric frameworks are ZK, Vaadin, ICEfaces.

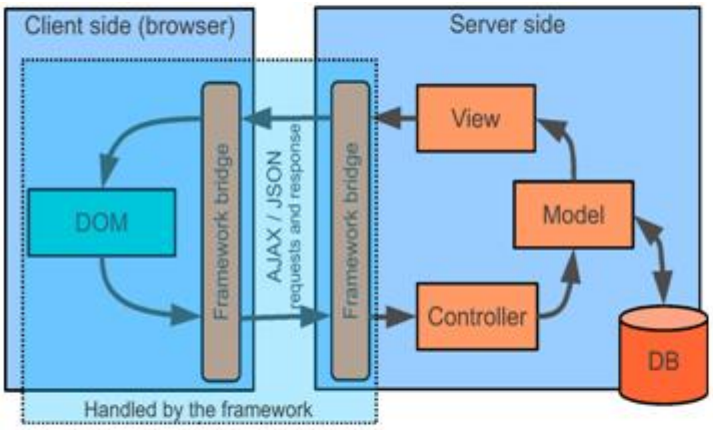


Fig. 2 A server-centric approach in RIA development

Both approaches, client- and server-centric, hide the JavaScript and other complexities from the developer by defining the UI in a custom markup language or programmatically in another programming language such as Java, and automatically generating appropriately optimized corresponding cross-browser compatible code.

IV. INTEGRATION OF RUBY AND JAVA RIA FRAMEWORKS

As we emphasized earlier, the Ruby programming language, and its Java implementation JRuby, is a dynamic and object oriented programming language with expressive syntax and does not require compilation so it is suitable for fast application development with short iteration cycles. Our goal was to make RIA development faster and more productive than the development in the Java programming language.

We evaluated Vaadin - a server-centric framework with component based approach for UI generation. We found Vaadin [10] [11] to be very suitable for integration with Ruby because both the UI and the application logic are programmatically defined with UI component objects. The process of constructing the Web aplications with Vaadin is similar to that of building the desktop aplications with Swing, a standard library for graphical user interface (GUI).

A. Vaadin framework

Vaadin is a server-centric Java framework. Its development began in year 2000, but was open-sourced since 2007. The key idea behind this platform is a development of Web based high quality UI similar to desktop application's UI without using several DHTML technologies but only one programming language, Java. Moreover, the development of a Vaadin application is more like desktop application development using UI component objects. When deployed to a Java application server, those objects are automatically transformed by the framework to the constructs Web browsers understand (HTML, CSS, JavaScript), hence no browser plugins are needed.

The two major parts of Vaadin are the server-side framework and the client-side engine that runs in a Web browser as a JavaScript program. The client-side engine renders the UI using Google Web Toolkit (GWT) and delivers user interaction to the server. The application runs as a Java Servlet session in a Java Servlet container. In Vaadin, application logic is separated from presentation using the so called themes that are defined with CSS.

UI of a Vaadin application is defined programmatically by using server-side UI components. There is a large number of predefined UI components. Furthermore, Vaadin framework provides tools for the creation of custom components. Every server-side component has a client-side counterpart that is rendered by the client-side engine and that user interacts with. All interaction is serialized and sent to the server by the client-side engine as asynchronous AJAX requests. This interaction is received by the terminal adapter. It interprets those messages and converts them to the appropriate events that are associated with the server side UI components. Those events are relayed to the Vaadin application logic that can eventually make changes to the UI components if required. UI components communicate their changes back to the terminal adapter which renders them for the user's browser and sends the changes to the client-side engine.

Vaadin's architecture is depicted in Fig. 3. The same figure depicts the described event propagation and other core concepts.

In addition to the UI model, Vaadin provides a data model that supports binding of data presented in UI components to a separate data source. With that feature, data can be directly updated by UI components, without the need for writing any other code.

User's interaction with UI components are realized using the observer design pattern [12]. For every event that is interesting for the application, a listener object is defined together with a method that will be executed when the event occurs.

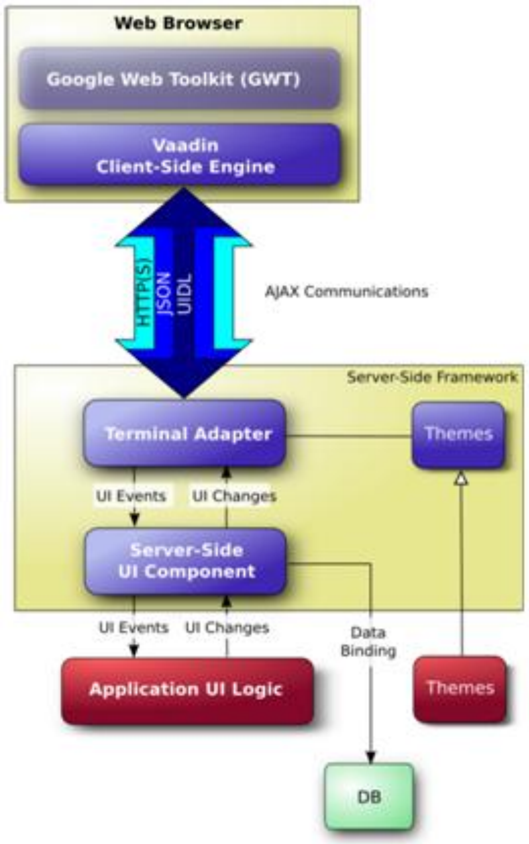


Fig. 3 Vaadin architecture [11]

With the described architecture, the Vaadin framework provides a good opportunity to use Ruby instead of the Java programming language in the development of RIA. For that purpose, we developed a JRuby library that includes the required adaptation of the Vaadin framework in order to use the framework for RIA development in JRuby.

The deployment of JRuby applications interfaced with Vaadin to classic Java Web servers would be complicated. Therefore, we devised a custom approach of running such applications by embedding a small Java application server directly in a JRuby application. The approach is similar to the approach we used to run SIP services with JRuby. The details of this approach are given in [13] and [14].

With this approach the configuration of the application server is done programmatically in JRuby. Therefore, we can create Vaadin UI applications in Ruby and deploy it to the embedded application server as an HTTP servlet.

B. Data binding in Vaadin

An important aspect of every framework is the database access and elegance of dealing with application's data. One of the core concepts of Vaadin framework is a data model that allows for flexible binding of diferent data models with UI components. There are three nested levels of hierarchy in the Vaadin data model: property, item and container, as shown in Fig. 4. In the relational database analogy, they would correspond to field, row and table.

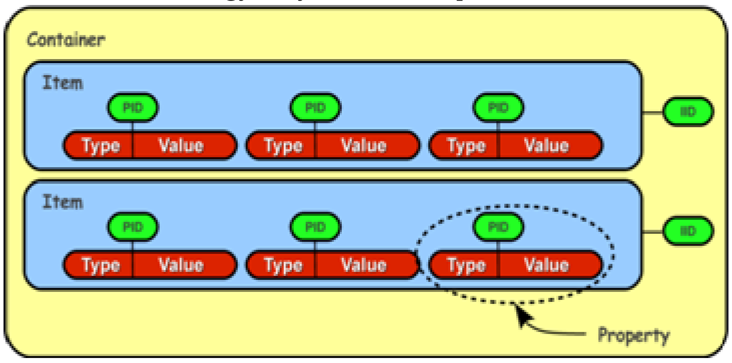


Fig. 4 Vaadin data model

The data model is realized as a set of interfaces in the \verb+com.vaadin.data+ package of the Vaadin library. The package contains the \verb+Property+, \verb+Item+ and \verb+Container+ interfaces, as well as other specialized interfaces and classes for realization of this data model. The Vaadin data model can be used in almost all UI components in order to implement data binding. Additionally, more than one component can be bound to the same data source that allows for implementation of various viewer-editor patterns. Furthermore, captions for most UI components can be set automatically to properties names of the container they are bound to, which is another useful feature of the Vaadin library.

There are several ORM frameworks available in Ruby with expressive and self explanatory syntax. The most popular Ruby ORM frameworks are ActiveRecord and DataMapper. ActiveRecord is an ORM primarily developed and used for Ruby on Rails applications. DataMapper is a Ruby ORM that is fast and thread-safe [15]. The DataMapper class diagram is depicted in Fig. 5.

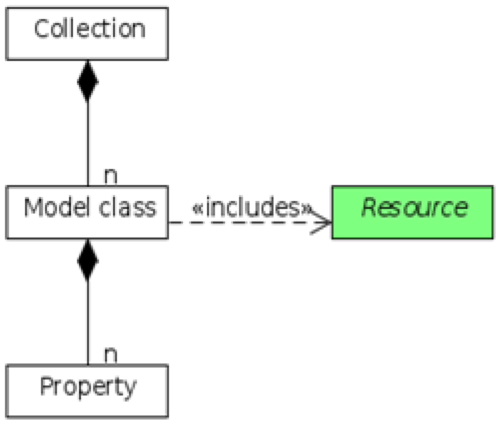


Fig. 5 DataMapper class diagram

An instance of a DataMapper model is a Resource. In a relational database, this is a row. A resource contains a collection of Properties, that corresponds to fields. All resources from a database query belong to the same Collection object. In DataMapper, there is no need to use SQL language for database access. All data manipulation is performed directly in Ruby.

The following example is a definition of a DataMapper model. It is self explanatory and demonstrates the expressiveness of the DataMapper model syntax.

```
class Person
  include DataMapper::Resource
  property :id,           Serial
  property :first_name,     String
  property :last_name,      String
  property :science,        String
  property :year_of_birth,  Integer
end
```

The model from the previous code corresponds to the People table in the database. The properties of the model correspond to the table's fields with the same names. It is neither mandatory for the model to have properties' names the same as the corresponding fields, nor the model has to have the same number of properties as number of the table's fields. These parameters, as many other in the model, are configurable.

For a detailed explanation of DataMapper, the reader is advised to look at the DataMapper's website [15].

In order to give developers flexibility, productivity and other benefits of both Ruby and Vaadin, we developed an interface between the Vaadin data model and Ruby DataMapper framework. The iterface is constructed by adapting the DataMapper module. We created the two new classes in the DataMapper module and extended the Resource module so that they implement the Container, Item and Property interfaces of the Vaadin API, as depicted in the class diagram in Fig. 6.

The described interface allows for binding Vaadin UI components to the data fetched by utilizing the DataMapper ORM.

To demostrate the data binding feature of Vaadin UI components with DataMapper we created an application that results with UI depicted in Fig. 7. The UI consists of a table at the left-hand side of the window and a form at the right-hand side of the window. The table contains data about people from a database. When a user of the application selects a person in the table, that person's data are loaded to form's UI components. If the user makes any change in the selected person's data by editing a value in any form's component, the change is automatically reflected in the table and written to the database.

A JRuby code that results in a Web browser UI with the described functionality is shown in the following listing which, together with Person model defined in the previous listing, represent the entire application code.

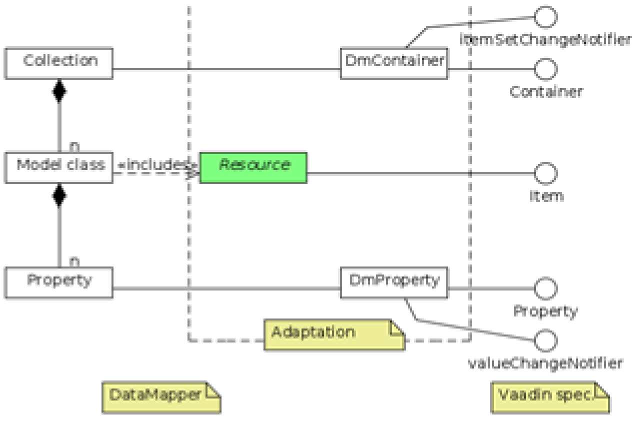
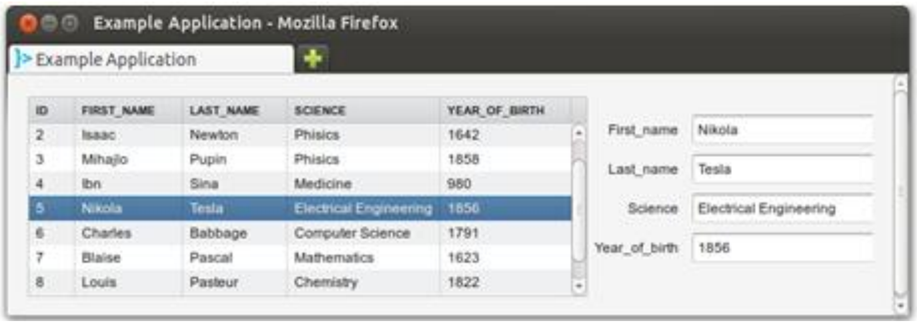


Fig. 6 Class diagram of the developed interface between DataMapper and the Vaadin data model

Since this container is implemented according to the Vaadin data model, it can be used for binding Vaadin UI controls to it.

The following code of a simple JRuby-Vaaddin application demonstrates the use of our container and construction of user interface together with definition of an event listener. A screenshot of the resulting application's UI is depicted in Fig. 7.

```
class ExampleApp < Application
  def init()
    main = Window.new 'Example Application'
    set_main_window  main

    dm_cont = DataMapper::DmContainer.new(Person.all)

    horizontal = HorizontalLayout.new

     table = Table.new
     table.container_data_source = dm_cont
     table.selectable = true
     table.immediate = true
     table.page_length = 7

     form = Form.new
     form.immediate = true

     horizontal.spacing = true
     horizontal.add_component table
     horizontal.add_component form

     main.add_component horizontal

     listener = PropertyValueChangeListener.new do |event|
        form.item_data_source = table.get_item(table.value)
        form.visible = (table.value != nil)
     end
     table.add_listener listener
   end
end
```



ID	FIRST_NAME	LAST_NAME	SCIENCE	YEAR_OF_BIRTH
2	Isaac	Newton	Phisics	1642
3	Mihajlo	Pupin	Phisics	1858
4	Ibn	Sina	Medicine	980
5	Nikola	Tesla	Electrical Engineering	1856
6	Charles	Babbage	Computer Science	1791
7	Blaise	Pascal	Mathematics	1623
8	Louis	Pasteur	Chemistry	1822

First_name	Nikola
Last_name	Tesla
Science	Electrical Engineering
Year_of_birth	1856

Fig. 7 UI of the demo application

Most of the above source code, from line 8 to 23, is for constructing the described UI. The UI components display the data about people from an external database by utilizing the data binding feature. In order to bind the components to the data in the database we first fetch the data about all people from the database in line 6 by using the Person DataMapper model described earlier.

Now, when we have our DataMapper container object defined, we bind the table to the container object in line 11. When the user selects a row in the table, we want to enable editing the corresponding person's data. For this purpose, in lines 25 to 28, we define an appropriate event listener.

Definition of event listeners in Ruby is much more elegant, compact and clear than the equivalent code in Java. An event listener is defined in Ruby by utilizing a block of code, while the same process in Java would require definition of an anonymous class with its methods and all static datatyping, which is much less elegant.

V. CONCLUSIONS

We described an approach to constructing a middleware for building RIA in the Ruby programming language. The middleware is designed as an interface from JRuby with Vaadin, a Java RIA framework. The middleware includes the adaptation of the DataMapper ORM in order to be used for data binding with Vaadin UI components. With this middleware, the entire RIA logic and UI, together with a definition of application's data models, can be completely written in JRuby using Ruby's programming style and expressiveness.

The entire source code of the middleware described in this paper, together with the demo application and the simple instructions for running the code in JRuby is available on the GitHub repository [16].

This middleware is a work in progress. Future work would include the development of a Ruby gem [17] based on this middleware and scalability studies in a cloud environment.

REFERENCES

[1] P. Fraternali, G. Rossi, and F. Sánchez-Figueroa, “Rich internet applications,” *IEEE Internet Computing*, vol. 14, no. 3, pp. 9–12, 2010.

[2] J. Allaire, “Macromedia Flash MX – A next-generation rich client,” Technical report, Macromedia, 2002.

[3] L. Tratt, “Chapter 5 Dynamically Typed Languages,” *Advances in Computers*, vol. 77, pp. 149 – 184, Elsevier, 2009.

[4] L. D. Paulson, “Developers Shift to Dynamic Programming Languages,” *Computer*, vol. 40, no. 2, pp. 12–15, 2007.

[5] B. Tate, *Beyond Java*, O’Reilly Media, 2005.

[6] D. Flanagan and Y. Matsumoto, *The Ruby Programming Language*, O’Reilly Media, 1 ed., 2008.

[7] M. Bachle and P. Kirchberg, “Ruby on Rails,” *IEEE Software*, vol. 24, pp. 105–108, 2007.

[8] (2017) The JRuby website,[Online]. Available: http://jruby.org/

[9] P. Perrotta, *Metaprogramming Ruby*, Pragmatic Bookshelf, 1st ed., 2010.

[10] (2016) The Vaadin website, [Online]. Available: http://vaadin.com/

[11] M. Grönroos. (2016) Book of Vaadin 6 homepage, [Online]. Available: http://vaadin.com/book/vaadin6/

[12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*, Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1994.

[13] E. Pjanić and A. Hasanović, “Exploring Ruby and Java Interoperability for Building Converged Web and SIP Applications,” *International Journal of Digital Information and Wireless Communications (IJDIWC)*, vol. 1, no. 3, pp. 597–610, 2011.

[14] E. Pjanić and A. Hasanović, “A JRuby Infrastructure for Converged Web and SIP Applications,” in *Digital Information Processing and Communications*, ser. Communications in Computer and Information Science, pp. 72-84. Springer-Verlag Berlin Heidelberg, 2011, vol. 188.

[15] (2016) The DataMapper website, [Online]. Available: http://datamapper.org/

[16] (2017) The RIA Middleware Prototipe homepage on Github, [Online]. Available: http://github.com/edictlab/RIAdemo/

[17] D. Berube, *Practical Ruby Gems*. Berkely, CA, USA: Apress, 2007.