A Method to Achieve Automation in the Development of Web-Based Software Projects

María Consuelo Franky
Department of Systems Engineering
Pontificia Universidad Javeriana
Bogotá, Colombia
lfranky@javeriana.edu.co

Jaime A. Pavlich-Mariscal
Department of Systems Engineering
Pontificia Universidad Javeriana
Bogotá, Colombia
jpavlich@javeriana.edu.co

***Abstract*— This paper proposes a method to achieve a high degree of automation in the development of Web software projects. This method is based on the experience of two consecutive university-industry projects that have received funding from the Colombian government. These projects aim to improve the software development tools of a large-scale software company, applying techniques based on Model Driven Engineering (MDE) and software building tools to achieve a high level of automation in generating new Java Platform, Enterprise Edition (Java EE) projects and in integrating existing components developed by the company. The tools developed in the first project significantly improved the development speed in the company. In the final state of the second (ongoing) project, we expect that MDE transformers will improve flexibility in generating Java EE projects with different architectures and different types of user interfaces, such as JavaServer Faces (JSF) or Java FX2. We believe that the steps performed during those two projects can serve as a guide for other software organizations to effectively automate their development for large scale projects.**

Keywords- Web technologies; Frameworks; Web applications development; Software Reuse; Automatic Software Generation; Model-driven development of Web applications.

I. INTRODUCTION

Competition and market requirements lead to companies developing large software projects to find higher competitiveness through shorter development cycles and lower costs. One way to achieve these goals is through better automation in the development of software projects and also through higher reuse of software components that are useful for multiple projects [1].

This paper proposes a method comprising a series of stages with associated techniques for achieving a high degree of automation and reuse in the development of Web software projects. The steps and techniques described in this paper have been applied to the specific case of Heinsohn Business Technology (HBT) [2], a large-scale Colombian software development company that develops Java EE [3] applications for governmental and financial organizations.

This method was developed as part of two joint projects between the Pontificia Universidad Javeriana [26] (the university of the authors) and HBT, which were funded by the Colombian government. In these projects, we have applied techniques based on MDE [8] and software tools [13] to achieve a high level of automation in generating new Java EE projects and effectively integrating and reusing components developed by the company. As a result, the company has been able to reduce significantly the initial stages of development of Java EE projects.

We are currently working on a second project to further improve these tools and processes. This new project will develop MDE transformers that will increase flexibility in generating Java EE projects with different architectures and different types of user interfaces (JSF [5] or Java FX2[29]).

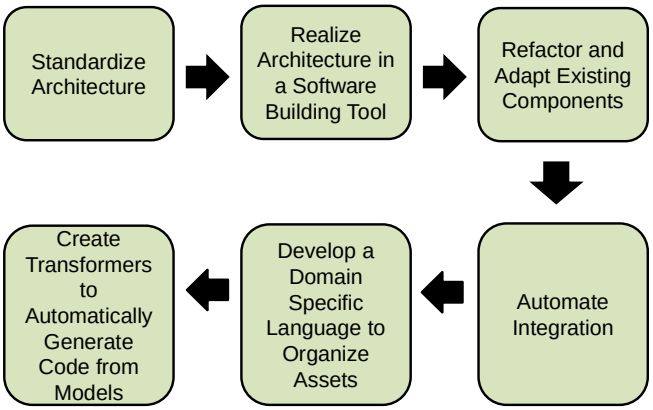

Figure 1. Proposed Method.

Figure 1 is an overview of the proposed method. First, it is necessary to standardize a multilayer architecture for the organization of new projects. Software building tools are used to materialize that architecture when creating the codebase of these projects. It is also necessary to refactor and adapt existing components in the organization, so they can be efficiently integrated into new projects. The next step is to automate the integration of such components into new software projects. A domain specific language (DSL) [27] is developed to create models that effectively reference and organize all of the above assets with the structure and behavior of a web application. Code generators automatically transform those models into working software applications.

The remainder of this paper details the proposed method for achieving high automation in the development of Web software projects in a company. Section II describes the initial stage of defining and standardizing a multilayer

 ISBN: 978-1-61208-361-2

architecture for a company. Section III describes the materialization of such architecture through a software building tool (Maven [4]). Section IV describes the refactor of the company reusable components in order to be compatible with the architecture. Section V describes the automated integration of components in Java EE projects. Section VI describes how to incorporate DSL to allow the modeling of web applications (including reusable components) independently of technology. Section VII describes the construction of MDE transformers in order to automatically generate Java EE projects that integrate the reusable components and with different types of user interfaces (JSF or Java FX2). Section VIII analyzes related work, and Section IX presents the conclusions and future work.

II. STANDARDIZING ARCHITECTURE

The company that participated in our two research projects (HBT) develops large-scale software, with a focus on Java EE. During the development of several software applications, HBT determined the necessity of adopting a standard reference architecture to organize the application code.

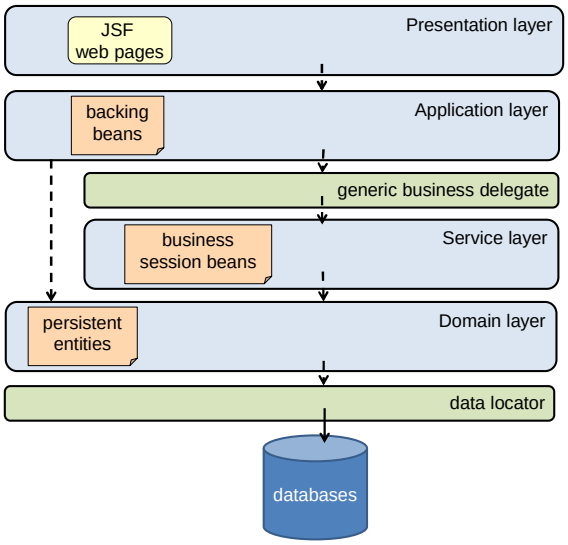

Figure 2. Multilayer Architecture adopted at HBT.

Based on this experience, the first stage of our proposed method is the standardization of the software architecture utilized by the organization in its projects. An important premise for this stage is that the software projects developed by the organization must be of a similar nature and should be effectively addressed by a standard architecture.

Figure 2 depicts the standard architecture adopted by HBT. The architecture effectively separates the application into several decoupled layers. For instance, web pages are supported by backing beans that manage the displayed information. These beans are decoupled from business session beans and entity beans that are in the lower layers.

III. MATERIALIZING THE ARCHITECTURE THROUGH A SOFTWARE BUILDING TOOL

To properly adopt the architecture, it is very important to materialize it in the software building tools that are used to create, integrate, and build software projects. In the case of HBT, since its focus is on Java EE projects, the chosen tool was Maven [4]. Maven is a tool to automate the building lifecycle of a software application, dependency management, and software variants. Maven defines a Project Object Model (POM) file, an Extensible Markup Language (XML) file that stores all of the above information about a software project. A detailed discussion of the reasons for choosing this tool in HBT can be found in [28].

In the context of HBT, Maven was used to materialize the adopted architecture. Each Java EE project is described as a Maven project with the following sub-modules:

- Presentation Layer. JSF [5] web pages realizing CRUD ("Create, Read, Update, and Delete") and business operations.
- Application Layer. Descriptors and backing beans [6] to support the JSF pages.
- Service Layer. Session Beans that realize all of the functionality of use cases.
- Domain Layer. Descriptors and persistent entities [6].
- Persistence Layer. Structured Query Language (SQL) scripts to populate tables with initialization data.

Each of the above sub-modules has a POM file that describes the library dependencies of each sub-module (including dependencies to other sub-modules), the type of artifact that yields after building and packaging (e.g., a Java Archive – JAR – or Web Application Archive – WAR - file [3]), and the identification of each sub-module in a Maven component repository of the organization.

To provide an adequate flexibility in the creation of the codebase of new projects, a useful tool is Maven Archetypes [4], templates based in Maven to instantiate the adopted architecture into new projects that are parameterized by specific design decisions.

Our joint project with HBT created an archetype that includes several profiles that parameterize new projects according to different database engines (Oracle [15], Postgresql [16], MySQL [17], and SQLServer [18]), and different application servers (JBoss [19], Glassfish [20], WebLogic [21], and Websphere [22]).

IV. REFACTORING AND ADAPTATION OF EXISTING COMPONENTS

The next step in the process is to refactor existing software components in the organization, to adapt them to the adopted architecture. Section A explains the existing components at HBT and Section B describes the process to adapt them into the chosen architecture.

 ISBN: 978-1-61208-361-2

A. *Description of company reusable components*

Software development organizations usually create several software components in order to reduce costs and capitalize the knowledge of previous solutions. In the context of HBT, these components address requirements, such as security (originally based on [34]), audit, notifications, batch data processing, text files processing, etc., or improve functionality of existing COTS (commercial off-the-shelf) components [28].

Component reuse may be complex, since a component may span several layers of the architecture, to provide a complete solution to programmers. Components also include several different types of files, each one associated with specific layers in the architecture. For instance, JSF pages in the presentation, session beans for the business logic, persistent entities, SQL scripts to populate databases, etc.

To properly reuse a component in a new project, many of the above elements need to be adapted to the specific project requirements. This task could be made easier by using automation techniques.

B. *Component Refactoring to a Maven Multi-Module Structure*

To properly incorporate existing components into the adopted architecture, it is necessary to convert them to the format of the software build tool. For our project with HBT, the existing component had to be converted to Maven multi-module components [13], to facilitate reuse and integration with the Maven-based architecture. Particularly, the conversion to Maven facilitated the identification of the parts of each component that are associated with each layer of the architecture.

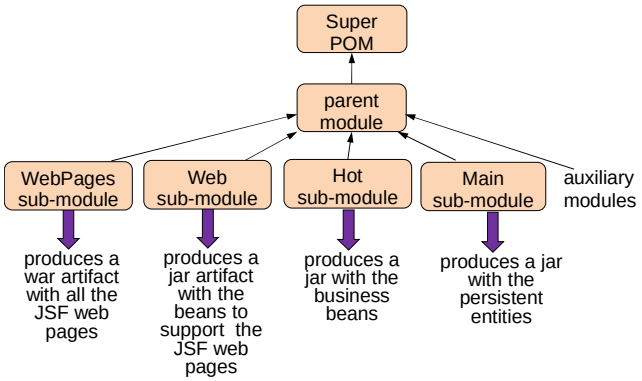

Figure 3. Maven-based Structure of Refactored Components.

Figure 3 depicts the general structure of each refactored component. A refactored component has a root module that contains a sub-module for each layer in the architecture; Figure 3 shows the "Super POM" that is the root module that sets the standards for any Maven hierarchy.

Figure 4 depicts the folder structure of one of the refactored components (in this case, the security component). Each folder corresponds to a Maven sub-module, each one with its own POM file. The pom.xml descriptor at the root folder of the component denotes dependencies on third-party libraries and on other components. These dependencies are inherited by the modules inside the component.

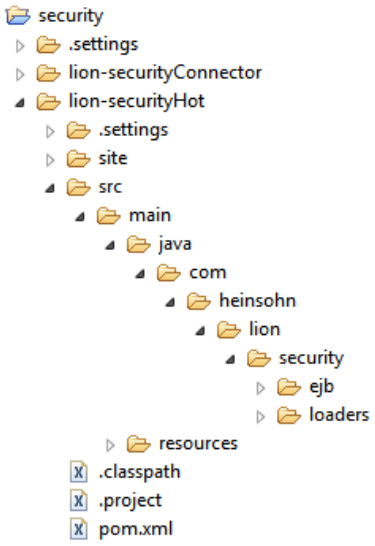

Figure 4. Maven Multi-Module Structure of a Refactored Component.

Each sub-module yields a small artifact that can be indexed and stored in a Maven repository of the company. This facilitates the integration of new multi-module components in the future.

V. AUTOMATING THE INTEGRATION

Although refactoring components to adapt them to the chosen architecture may reduce development times, there are other tasks that can be performed to further improve this process. Particularly, an adequate automation of the component integration process may speed up the creation of a project codebase.

In the context of our projects with HBT, component integration was automated through a tool called LionWizard. This tool automatically generates a new Java EE codebase utilizing a Maven archetype. The tool further transforms that codebase to automatically integrate all of the components selected by the user, effectively eliminating most of the manual tasks.

Figure 5 is the main window of the wizard. This tool receives as input a series of properties given by the programmer to parameterize the new Java EE project: project folder, Maven project name, group, and version to identify the artifacts of the project, the database engine, the application server to deploy to, etc.

With the above information, LionWizard generates a Maven multi-module Java EE project, based on an archetype.

In addition, LionWizard lets the programmer select specific components to be integrated into the generated codebase. The wizard automatically integrates those components by transforming the POM files of the required sub-modules and root. That transformation incorporates new dependencies to other Maven artifacts and to the sub-

 ISBN: 978-1-61208-361-2

modules of the selected components. The wizard also modifies the required configuration files to effectively integrate the components into the codebase.

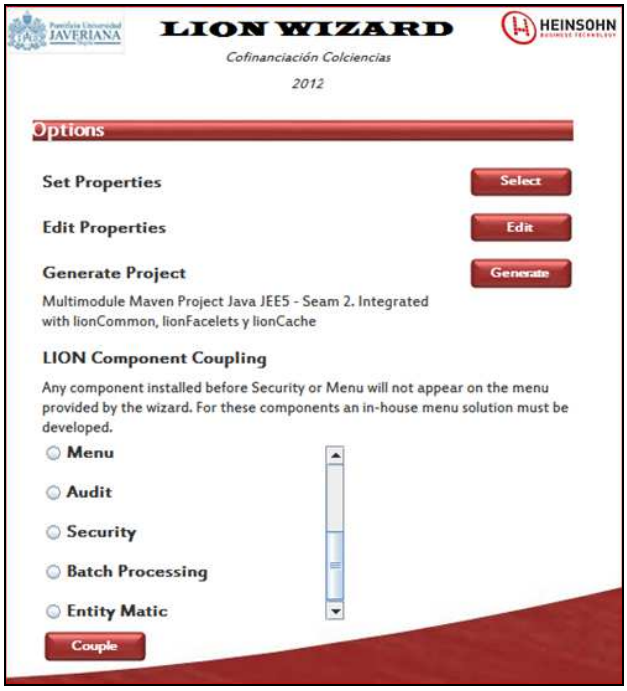

Figure 5. LionWizard's Main Window.

The wizard was designed with sufficient flexibility to seamlessly integrate components that could be developed in the future. To achieve that flexibility, each component has a special configuration file that describes the way to integrate that component into the codebase, originally based on, all of the tasks required to transform the component and the codebase (file modifications, file creations, and properties insertions) to effectively compose them together. Another paper submitted by the authors [28] illustrates the XML configuration file that is used to integrate some components into the codebase of new generated projects.

The automation provided by LionWizard is almost complete. A few tasks are still manual, such as a few SQL script executions and security realms configuration through the console of an application server.

Before the creation of LionWizard, HBT had developed several components. However, their reutilization was hindered by the difficulty to manually integrate them into new codebases. Typically, such integration could take up to three weeks at the beginning of the project. After the creation of the wizard, integration times were reduced to just a few hours, which comprise the time to execute the automatic integration, plus the time to execute a few manual tasks. Another paper submitted by the authors [28] quantifies the benefits of the proposed technology automation through an evaluation of the framework.

VI. DEVELOPING A DOMAIN SPECIFIC LANGUAGE

After executing the previous stages, the integration process of a new codebase can be significantly accelerated. However, the benefits obtained by this automation can be hindered when the codebase evolves during the project, since changes performed to the code may make it harder to automatically integrate new components. Furthermore, as new projects demand the utilization of more recent technologies (e.g., from Java EE 5 to Java EE 6 or 7), make it necessary to modify the wizard to reflect these changes in technology.

Our second joint project with HBT is performing a further step in automation by relying on Model Driven Engineering (MDE) [8] to utilize models as the main development artifact in a software Project.

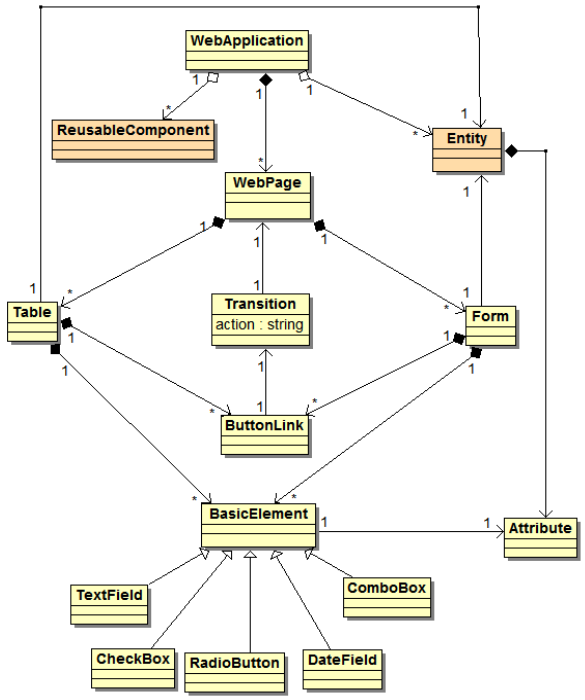

Figure 6. Fragment of the DSL's Meta-Model.

MDE aims to utilize models for every stage in the software engineering process (requirements, design, etc.). In particular, MDE's intent is to utilize transformation tools to automatically generate the source code from models. As a result, a set of models of an application can be used to generate software for different platforms and frameworks. An adequate utilization of MDE may facilitate the evolution and seamless integration of components across the entire development lifecycle and also provide a degree of independence over the technology used to implement the software applications.

In the context of our project with HBT, ongoing work is creating a DSL [27] with an aim to model concepts [32] associated to the structure and behavior of large-scale web applications, while adhering to the adopted multilayer architecture (see Section II). In addition, this language aims to provide clear abstractions of the available components and

 ISBN: 978-1-61208-361-2

their integration mechanisms. The implementation of that language is based on Eclipse EMF [9] and Xtext [33].

Figure 6 depicts a Unified Modeling Language (UML) diagram with the main elements of the language's meta-model, such as the following:

- A web application is composed of multiple web pages and works on multiple persistent entities.
- A web page can contain multiple forms and tables.
- Each form or each table may contain different widgets (basic elements) to depict attributes of the entities.
- Each form or table may also contain links and buttons to connect to other pages after executing some business action.
- Abstractions to model reusable components and their mechanisms of integration.

VII. CREATE MDE TRANSFORMERS

The last stage towards automation is the creation of transformers that can automatically generate application code and integrate components from the information provided by the models specified by the DSL.

In our experience, to create effective transformers it is necessary to capitalize good practices of previous development efforts and also in the previous stages of the proposed method, so that the generated code can include the best strategies for implementation.

Our ongoing work with HBT is creating a transformer that generates Java EE code with the following elements:

- A Maven multi-module web application codebase with the same structure as described in the previous sections of this paper.
- Automatic integration of the existing components. To achieve this, the generator utilizes the information provided by the models (based in the DSL) and the automatic integration code of LionWizard.
- CRUD operations and user interfaces for each entity of the application
- A set of JSF web pages, their backing beans, and a configuration file that describes the page flow (faces-conFigurexml) based in the transitions expressed in the model.

The transformer is based on EMF [9] and Acceleo [7]. Future work is to create transformers for other user interfaces different from JSF, such as Java FX2 [29]. In all of those cases, the models will stay independent of the specific implementation technology.

VIII. RELATED WORK

Maven [13] is widely used to manage the building process in software projects. The evidence of this is the high amount of projects stored in public Maven repositories [12]. The common way to utilize Maven is to automate the compilation and packaging process. Our contribution is the utilization of Maven and its enhancement with additional programs to automate the component integration process, based on a standardized architecture.

There are some tools based in Maven that create codebases from archetypes and generate CRUD operations for entities, such as JBoss Forge [23] and AppFuse [24]. However, these tools do not address the automatic integration of complex components.

There are several strategies for automatic reuse and integration in software projects. Some of the strategies are code generation and program transformation, software product lines, and web services (see a description of these strategies in [28]). Our approach can be classified as a simpler, scope-bounded code transformation approach. Because of its simplicity, it is more maintainable while being adequate for the integration of components into the codebase of software projects.

There are several tools to apply MDE principles for the development of web-based applications, such as Web Ratio [31], Magic [10], and Integranova M.E.S. However, these environments have a high licensing cost, which makes adoption difficult. Moreover, they lack the flexibility to seamlessly adapt their language and transformers to the specific architectures required by a software development organization. The MDE environment that we are building is based on open source software tools that allow HBT to further enrich the DSL modeling language and add new transformers for other technologies.

IX. CONCLUSION AND FUTURE WORK

This paper proposed a method to achieve automation in the development of large web-based applications and the integration of existing components in an organization. This proposal is based on the experience of executing two joint projects with HBT in order to automate their development tools.

The increment in productivity obtained by the first project was promising, since it significantly reduced the times to create new codebases. The ongoing work is expected to yield similar improvements.

Overall, these joint projects have improved the synergy between the University and a software development organization. Future work is to enhance the MDE tools to incorporate additional transformers to other technologies, such as .NET and mobile applications, platforms for which HBT also have components that could be reused. In addition, all of the above will be integrated into the existing product line framework that is being developed in a parallel project [14].

X. ACKNOWLEDGMENTS

Contract/grant sponsor: This article is part of the projects Lion and Lion2, executed by the SIDRe research group of the Pontificia Universidad Javeriana and Heinsohn Business Technology, co-financed by Colciencias (Colombian Administrative Department of Science, Technology and Innovation).

We thank Colciencias [25] for funding the projects described in this paper. We also thank all of the other participants in both joint projects with HBT:

- From HBT: Alvaro Javier Infante, María Catalina Acero, Leonardo Giral, Angee Zambrano, Cristián Fernández, Rubén Darío Betancur, Carlos Díaz, Jorge Camargo.
- From the Pontificia Universidad Javeriana: Andrea Barraza-Urbina, Luisa Barrera, John Carlos Olarte, Francisco Mora.

REFERENCES

[1] C. Larman, "Agile and Iterative Development: A Manager's Guide", Addison-Wesley Professional, 2003.

[2] Heinsohn Business Technology, URL http://www.heinsohn.com.co 01.03.2014.

[3] Oracle, "Java EE at a Glance", URL http://www.oracle.com/technetwork/java/javaee/overview/index.html 01.03.2014.

[4] Foundation A. Maven, "Introduction to Archetypes", URL http://maven.apache.org/guides/introduction/introduction-to-archetypes.html 01.03.2014.

[5] D. Geary and CS. Horstmann, "Core JavaServer Faces", Prentice Hall 3 edn., 2010.

[6] M. Keith and M. Schincariol, "Pro EJB 3: Java Persistence API", Apress, 2006.

[7] Obeo, "Acceleo", URL http://www.eclipse.org/acceleo/ 01.03.2014.

[8] S. Kent, "Model driven engineering", Springer Berlin / Heidelberg, 2002, p. 286–298, URL http://www.springerlink.com/content/9vuqb4hp8fyg2adv/abstract/ 01.03.2014.

[9] The Eclipse Foundation, "Eclipse modeling framework (EMF)", URL http://www.eclipse.org/modeling/emf/ 01.03.2014.

[10] No Magic, "MagicDraw", URL http://www.nomagic.com/products/magicdraw.html 01.03.2014.

[11] Integranova, "Integranova M.E.S.", URL http://www.integranova.com/integranova-m-e-s/ 01.03.2014.

[12] Foundation A. Maven, "The Central Repository", URL http://search.maven.org/#browse%7C47 01.03.2014.

[13] Sonatype Company, "Maven: The Definitive Guide", O'Reilly Media, 2008.

[14] C. Parra, L. Giral, A. Infante, and C. Cortés, "Extractive SPL adoption using multi-level variability modeling", Proceedings of the 16th International Software Product Line Conference - Volume 2, SPLC '12, ACM:New York, NY, USA, 2012, p. 99–106, URL http://doi.acm.org/10.1145/2364412.2364429 01.03.2014.

[15] Oracle, "Oracle Database: Introducing Oracle Database 12c: Plug into the Cloud", URL http://www.oracle.com/us/products/database/overview/index.html 01.03.2014.

[16] Group PGD, "PostgreSQL", URL http://www.postgresql.org/ 01.03.2014.

[17] Oracle, "MySQL", URL http://www.mysql.com/ 01.03.2014.

[18] Microsoft, "Microsoft SQL server", URL http://www.microsoft.com/en-us/sqlserver/default.aspx 01.03.2014.

[19] JBoss Community, "JBoss application server 7" , URL http://www.jboss.org/jbossas 01.03.2014.

[20] Oracle, "GlassFish", open source application server - project kenai, URL https://glassfish.java.net/ 01.03.2014.

[21] Oracle, "Oracle WebLogic Server", URL http://www.oracle.com/technetwork/middleware/weblogic/overview/index.html 01.03.2014.

[22] IBM software, "WebSphere software", URL http://www-01.ibm.com/software/websphere/ 01.03.2014.

[23] JBoss, "JBoss Forge", URL http://forge.jboss.org/ 01.03.2014.

[24] Atlassian, "AppFuse", URL http://appfuse.org/display/APF/Home 01.03.2014.

[25] Colciencias: Departamento administrativo de Ciencia, Tecnología e Innovación,. URL http://www.colciencias.gov.co/ 01.03.2014.

[26] Pontificia Universidad Javeriana, Department of Systems Engineering, URL http://puj-portal.javeriana.edu.co/portal/page/portal/Facultad%20de%20Ingenieria/dpto_sist_presentacion 01.03.2014.

[27] M. Fowler, "Domain Specific Languages", Addison-Wesley Professional, Firts Edit., 2011, p. 413.

[28] M. C. Franky et al., "Achieving Software Reuse and Integration in a Large-scale Software Development Company: Practical Experience of the Lion Project", submitted to IET Software in July- 2013.

[29] J. Weaver, G. Weiqi, S. Chin, D. Iverson , and J. Vos, "Pro JavaFX 2: A Definitive Guide to Rich Clients with Java Technology", Apress, 2012.

[30] G. Booch, J. Rumbaugh, and I, Jacobson, "The Unified Modeling Language User Guide", Addison-Wesley, 2006.

[31] WebRatio, "The New Business-IT Equation", URL http://www.webratio.com/ 01.03.2014.

[32] S. Kelly and J.-P. Tolvanen, Domain-Specific Modeling: Enabling Full Code Generation, 1st ed. Wiley-IEEE Computer Society Pr, 2008.

[33] M. Eysholdt and H. Behrens, "Xtext: implement your language faster than the quick and dirty way," in Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion, 2010, pp. 307–309.

[34] M. C. Franky and V. M. Toro, "CincoSecurity: Automating the Security of Java EE Applications with Fine-Grained Roles and Security Profiles", International Journal On Advances in Security (IARIA Journal), vol. no 3&4, 2011, URL http://www.thinkmind.org/index.php?view=article&articleid=sec_v4_n34_2011_10 01.03.2014.

 ISBN: 978-1-61208-361-2