GWT to Angular: Migration of graphic applications through Models

Benoît Verhaeghe[1,2], Anne Etien[1],
Nicolas Anquetil[1], Stéphane Ducasse[1]
[1]Université de Lille, CNRS, Inria,
Centrale Lille, UMR 9189 – CRIStAL, France

Abderrahmane Seriai[2], Laurent Deruelle[2],
Mustapha Derras[2]
[2]Berger-Levrault, France

***Abstract*—In the context of the graphical application evolution, it happens that the application must switch of implementation language. This evolution should be done by companies which want to stay*Up-to-Date* with the latest technology. It is also an important commercial argument to keep and attract clients. Berger-Levrault is a major IT company which owns the biggest GWT application in the world. The company needs to switch the application implementation of its software from GWT to Angular. Rewriting graphical application is complex and can lead to errors. Furthermore, the transformation from an implementation to another implies modifications of the application structure. Moreover, the migration must not impact the clients of Berger-Levrault. And so the visual aspects of the migrated application have to be the same as the original one. The company has estimated the duration of the migration to X. To reduce the amont of time needed, we've created a tool that semi-automatically does the migration of the application. To achieve this tool, we designed a metamodel to represent a graphical user interface (GUI[1] metamodel). Then we used a three parts application migration strategy and the tools to apply such strategy. Firstly, we instantiated a metamodel from the source code. Then, transform the resulted model to a model that respects GUI metamodel. Finally, we exported the *"graphic"* model to the target code.**

***Index Terms*—Graphical User Interfaces, Industrial case study, Java, Angular, GWT, Migration, Model-Driven Engineering, MDE**

I. Introduction

Desc de BL, et de leurs besoins Berger-Levrault is a major IT company. It uses the framework GWT to develop its graphics applications. GWT doesn't evolve almost any more with only one major release since 2015 whereas there was six major releases between 2010 and 2013. So, Berger-Levrault has decided to migrate all its application from Java using GWT to the Angular technology. The main aim is to avoid to be stuck in an old technology.

Desc des applications de BL (général) et que leurs tailles et tmps de vie est un pb Software of Berger-Levrault are long life applications (more than ten years), they have more than X MLOC and represent X web pages. The lifespan of the program implies that no one in the company has a *"perfect"* knowledge of the applications. This lack of information leads to difficulties for the migration. Indeed, migrating unused code and *hack* that were important in the source language create errors in the target one. This is why such migration is time-consuming and error-prone. Berger-Levrault has estimated the duration of the migration to X.

The migration of an application is not only the transformation of the source code to the target language. It also implies to find the frameworks to use in the target language and to think about the architecture of the applications in the new language. The migration should have a really tiny impact on the client of the company and for the developers of the application. In the case of Berger-Levrault, multiple applications are developed following the same architecture but some other used different framework or language like ReactJS, Aurelia, Laravel or Silverlight. So, it is important to have a solution that can be reused for all the graphic applications with a minimum of modification.

To extract information from the application, it is possible to use a dynamic solution or static one.

Samir *et al.* [1] proposed a dynamic solution to migrate Swing application to Ajax. Their tool runs an instance of the source application in a browser. Then, they are able to detect the widget displayed by the interface and the different actions available. The authors decided to use a dynamic solution to explore the interface of the application. Their contributions are a method and a middleware toolkit for re-architecting Swing application to Ajax one.

Sánchez Ramón *et al.* [2] developed a static solution to reverse engineer RAD (Rapid Application Development) based graphic user interface (GUI). The authors created different metamodel to represent a GUI. The authors used those to metamodels to create a chain of transformation between them which lead to the transformation of the source application the target one. Although we can't reapply the strategy of GUI extraction to our project, because RAD based application is too *simple* compared to GWT based application, the different metamodels and the chain of transformation inspired our work.

We present a new static solution to represent a GUI developed with Java-Swing, a strategy to generate this model, a way to generate the target application whatever the target framework Architecture. Our solution is source and target independent. The tool detects well X% of the widget for a web page. After the migration, X% widgets are well placed and X% are visually identical.

The main contributions of our work are:

[1]Graphical User Interface

- a modular tool that can be reused for other graphics application migration.
- a metamodel to represent a GUI application.
- a migration strategy to migrate graphics application.

In Section II, we give background information on our problem. Then, in Section III, we describe the solution we've proposed. Section IV presents the results and threats to the validity of our works. Finally, we present the related works and conclude in Section V and VII respectively.

II. Problem Description

Berger-Levrault is a major IT company which needs to migrate their GWT-based applications to Angular. The migration of those applications will lead to X days of development. This is due to the X MLOC that composed the software they have to migrate. Our work was tested on the showroom application of Berger-Levrault. This application is used only by the company's developers as an example application of the usage of the widget available for the development.

A. Constraints

The solution we propose must respect some criteria:

- *Source independence*. Our solution should be easy to adapt for all projects written in GWT using Java and for applications not written in Java but describing a GUI.
- *Target independence*. It has to be easy to change the target language without restructuring the metamodel we defined or change any stage of the migration.
- *Modularity*. The migration would be split into many little stages. This offer the possibility to easily extend a step or replace one step by another one without introducing instability.
- *Preserving Architecture*. After the migration, we should find the same architecture between the different component of the GUI (*e.g.* a button which belongs to a panel in the source application still belongs to the same panel in the target application).
- *Layout-preserving visual*. It should not have visual differences between the source application and the target one.

The development team has to continue the maintenance of the source application. This is a constraint inherent to industrial companies.

B. GWT and Angular Comparison

The source language and the target language can have two different architectures. Their difference can be syntactical, semantical and also architectural. In the case of the migration of the application from Java using GWT to Angular, the .java file will be separated into multiple Angular files.

With GWT, the source program is composed with one class per web page. So, it is possible to have only one file that represents one web page. In this class, we find the widgets' architecture and organization. It contains layout information such as *VerticalPanel*, *HorizontalPanel*, *FlexPanel*, etc. and style information by using setter call to widgets to change the height, width, color, etc. The .java file contains also the business code link to the widgets. GWT demand one configuration file to declare all the web page's Java class to determine the URL path to access to the web pages.

TABLE I
Comparison of GWT architecture and Angular one

Differences	Java using GWT	Angular
number of configuration file	one configuration file	four configuration files plus two files per subprojects
web page	one java class	one Typescript file plus one HTML file
style for a web page	include in java class	one optional CSS file

With Angular, those pieces of information are dispatched in multiple files. First of all, Angular is a Single Page framework based on Component, which means there will be only one web page and we will navigate into this web page. A component is a sub-part of a web page. It can be a complex widget like a calendar, a part of a webpage like a sidebar or a complete webpage. A component can contain other components, so the component representing a webpage can contain the sidebar component, and this last contains the calendar. Only the reached web page content is loaded. At the root of the Angular project, it has the main configuration files. Those files explicit the frameworks that will be used in the application. Then for each web page, we will find the following structure. A folder that contains at least an HTML file, a TypeScript file. It can also contain a CSS file, and some configuration file. The configuration files contain specific needed of the webpage. It can be compared to the *import* call in Java. The CSS file contains the specific style for this webpage. The TypeScript file contains the behavioral business logic of the webpage. The HTML file contains the architecture of the webpage and the layout information. The layout information is expressed by the class attribute defined in a general CSS file at the root of the project.

C. Migration strategy solutions

There are different solutions to migrate an application. All the solutions have to conclude with the respect of the two major constraints, the respect of the target architecture and the visual preservation.

- *Manual Migration*. This strategy is the re-development of the all application without using semi-automatic migration tool. This strategy offers the possibility to easily fix some error of the old application and to redesign the application following architecture principle of the target language.
- *Rule Engine*. Using a rule engine to migrate the totality or part of an application has already been applied in some project. We have to define and create rules to use this solution. It is possible migration is not completed by the strategy, so the developers will have to end the process of migration by some manual work. Using a rule engine can

be efficient but it implies to not be source independent nor target independent.

- *Model-Driven Engineering (MDE)*. Model-Driven Engineering implies the development of metamodels to do the migration. This strategy follows all the constraint we defined. It is also possible to an automatic or a semi-automatic migration with MDE strategy. This last implies to do some manual work to complete the migration processus.

Because we need to conform our solution to the constraint of this kind of migration and of the company, we developed a migration tool that uses Model-Driven Engineering strategy.

III. Proposed Solution

A. metamodel

B. Import

C. Export

IV. Discussion

Critères d'évaluation

- évaluation avec les développeurs de la solution qu'on leur propose
- Nombre de phase correctement détecté
  - Nombre de widget correctement détecté (on peut compter à la main sur 20 pages web) ou via comparaison de DOM
  - Nombre de widget correctement placé (dans l'architecture sûrement pas 100% mais pas mal)
- Nombre de widget visuellement correctement placé (notion d'attribut, de panel).. Ici on aura de mauvais résultat qui proviennent de l'analyse statique et non dynamique (qui est une contrainte forte pour nous)

V. Related Works

VI. Future Works

VII. Conclusion

In this paper, we exposed a solution to the problem of visual preservation and respect of the target architecture during the migration of an application. Our work has shown encouraging results with the case study of Berger-Levrault. We migrated complex and simple web pages from the applications of Berger-Levrault. The tool currently migrates correctly X% of a web page. To do so, we used a strategy based on a metamodel. We presented the strategy and its implementation in Pharo. Our solution allows us to migrate a web page following the target architecture and with a correct widget hierarchy. The visual aspect of the rendered application could differ from the original one because of the layout applied to the first one. This work can be used as a support to improve the migration of Graphical User Interface. The migration of the business code of the original application is a feature that can help developers. It would be also interesting to improve the way the layout is represented or to define which kind tool, on top of the model, can help the developers with the migration or their daily development.

Acknowledgements

This work was supported by Ministry of Higher Education and Research, Nord-Pas de Calais Regional Council, CPER Nord-Pas de Calais/FEDER DATA Advanced data science and technologies 2015-2020.

References

[1] Hani Samir, Eleni Stroulia, and Amr Kamel. Swing2script: Migration of java-swing applications to ajax web applications. In *Reverse Engineering, 2007. WCRE 2007. 14th Working Conference on*, pages 179–188. IEEE, 2007.

[2] Óscar Sánchez Ramón, Jesús Sánchez Cuadrado, and Jesús García Molina. Model-driven reverse engineering of legacy graphical user interfaces. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*, pages 147–186. ACM, 2014.