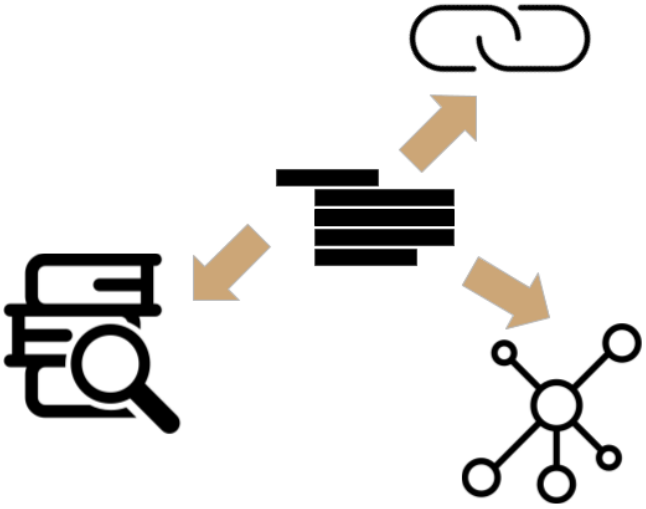
Benoît Verhaeghe

# Étude d'une application GWT



Tuteurs : M. Deruelle, M. Seriai
Mme Etien, M. Anquetil

Berger-Levrault
Inria Lille Nord Europe - RMoD
août 2018

# Table des matières

# 1 Contexte

## 1.1 Contexte général

Mon stage en entreprise est un travail qui s'inscrit dans le contexte d'une collaboration entre l'équipe RMoD d'Inria Lille Nord Europe et Berger-Levrault.

Berger-Levrault invente et développe des solutions pour les administrations et les collectivités locales, pour les établissements d'éducation et de santé publics comme privés, les universités et les entreprises. L'entreprise est implantée en France, au Canada et en Espagne.

J'ai travaillé dans l'équipe recherche et développement de Berger-Levrault à Montpellier. Mes superviseurs entreprises étaient M. Laurent Deruelle et M. Abderrahmane Seriai. Ma superviseure école était Mme Anne Etien. J'ai, dans le cadre de la collaboration entre Berger-Levrault et l'Inria Lille Nord Europe, travaillé aussi avec M. Nicolas Anquetil.

Ce travail est la suite du travail préliminaire que j'ai mené pendant mon Projet de Fin d'Étude à Polytech Lille.

## 1.2 Problématique

Berger-Levrault possède des applications client/serveur qu'elle souhaite rajeunir. En particulier, le front-end est développé en GWT et doit migrer vers Angular 6. Le back-end est une application monolithique et doit évoluer vers une architecture de services Web. Le changement de framework [1] graphique est imposé par l'arrêt du développement de GWT par Google et le problème de compatibilité arrière entre Angular 6 et Angular 1 (AngularJS). Le passage à une architecture à services est aussi souhaité pour améliorer l'offre commerciale et la rendre plus flexible.

Mon travail durant ce stage ne traite que de la migration des applications front-end.

## 1.3 Description du problème

Les applications front-end de Berger-Levrault sont développées en Java en utilisant le framework GWT de Google. Dans l'optique d'homogénéiser le visuel de leurs applications, Berger-Levrault a étendu ce framework. Cette extension s'appelle **BLCore**. Les applications de Berger-Levrault utilisent et/ou étendent **BLCore**, qui lui même utilise et/ou étend **GWT** comme présenté Figure 1.

Les applications de Berger-Levrault sont complexes. Ce sont les plus importantes applications GWT en terme de ligne de code et de classes dans le monde. Elles définissent plusieurs centaines de pages web. Bien qu'une migration complète de l'application en réécrivant l'ensemble du code est possible, c'est une tâche coûteuse et sujette à erreurs. Automatiser tout ou partie de la migration semble donc être la bonne solution, cependant les développeurs ne seront pas formés sur la nouvelle technologie et sur son utilisation dans les nouvelles applications. Une alternative pour contourner ce problème serait de créer des outils facilitant la migration. Les développeurs pourront alors effectuer la migration rapidement et seront formés sur le nouveau langage et sur l'application.

La complexité de la migration des applications de Berger-Levrault réside dans leurs tailles mais aussi et surtout dans le changement de langage. En effet, les applications sont développées intégralement en Java

1. Framework : ensemble cohérent de composants logiciels structurels, qui sert à créer les fondations ainsi que les grandes lignes de tout ou d'une partie d'un logiciel (architecture).

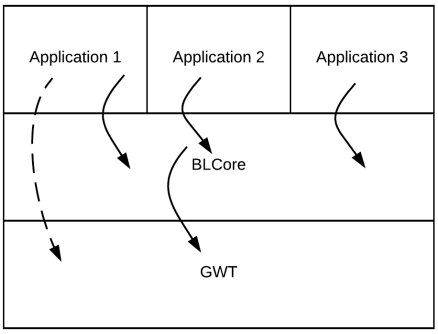



FIGURE 1 – Structure application

tout comme le framework GWT. Or, pour utiliser Angular 6, le programme doit être écrit en TypeScript. La question qui se pose est de savoir quel couche de la Figure 1 nous devons migrer et comment ?

En plus des difficultés techniques inherent à un tel projet, une entreprise comme Berger-Levrault a aussi des contraintes provenant de leurs développeurs et de leurs clients. Parmi ces contraintes, la migration devra, entre autre, conserver l'architecture sur laquelle se base les applications de Berger-Levrault et ne pas perturber les clients de Berger-Levrault du point de vue visuel des applications et comportementales.

Mon objectif est donc de trouver des solutions pour aider à la migration des applications de Berger-Levrault, de les évaluer et d'appliquer la migration évaluer la plus satisfaisantes vis-à-vis des contraintes posées. Pour cela, j'ai dans un premier temps étudié la structure d'une application de Berger-Levrault. Puis, j'ai défini avec un expert Angular l'architecture attendu pour les applications post-migration. Ensuite, j'ai définie une stratégie pour faire la migration en m'inspirant d'une étude de l'état de l'art que j'ai mené. Enfin, j'ai commencé le développement d'une suite d'outil permettant de mettre en application la stratégie définie et l'évaluer.

J'ai effectué cette étude sur l'application *bac-à-sable* de Berger-Levrault. Cette dernière permet aux employés de Berger-Levrault de consulter les éléments disponibles depuis BLCore. Bien que plus petite que les applications en production, elle contient tout de même plusieurs centaines de classes. Cependant, j'ai régulièrement vérifier que mon travail pouvait s'appliquer sur les projets plus important de Berger-Levrault.

# 2 Etat de l'art

# 3 Description du problème

Dans le contexte de l'évolution des applications de Berger-Levrault, l'entreprise a estimé le transformation du code à X jours de développement. Ceci s'explique majoritairement par les X MLOC [2] utilisés pour les logiciels. Un de mes objectifs à Berger-Levrault est de définir une stratégie de migration qui réduit le temps nécessaire pour la transformation des programmes.

## 3.1 Contraintes

Berger-Levrault étant une importante entreprise dans le domaine de l'édition de logiciel, elle a des contraintes spécifiques vis-à-vis d'un outil de migration. En effet, la solution logiciel que j'ai produit doit respecter les contraintes suivantes :

- *Indépendance du langage source.* La solution doit être facile à adapter pour tout projets utilisant GWT ainsi que d'autre logiciel n'étant pas écrit en JAVA mais possédant une interface utilisateur. Cette contrainte doit être respecté pour pouvoir être réutiliser sur différents projet. Dans le cas de Berger-Levrault, cela permet aux développeurs d'appliquer la solution sur d'autres logiciels qu'ils veulent migrer.
- *Indépendance du langage cible.* Il doit être facile de changer le langage cible de la migration sans devoir restructurer ou développer l'implémentation de la stratégie de migration. Cette contrainte garantie que la solution peut être utilisée quelque soit l'architecture du langage cible. Dans notre cas, nous migrons des applications de GWT vers Angular. Angular a vu 6 versions majeurs sortir depuis 2016. Grâce à l'indépendance du langage cible, la stratégie de migration reste valide quelque soit la version du langage cible.
- *Approche modulaire.* La migration doit être divisée en petite étapes. Cela permet de facilement remplacer une étapes ou de l'étendre sans introduire d'instabilité. Cette contrainte est essentielle pour les entreprises qui désirent avoir un contrôle fin du processus de migration. L'approche modulaire permet entre autres aux entreprises de modifier l'implementation de la stratégie pour respecter leurs contraintes spécifiques.
- *Préservation de l'architecture.* Après la migration, nous devons retrouver la même architecture entre les différents composants de l'interface graphique (*c.-à-d.* un bouton qui appartenait à un panel dans l'application source appartiendra au même panel dans l'application cible). Cette contrainte permet de faciliter le travail de comprehension de l'application cible par les développeurs. En effet, ils vont retrouver la même architecture qu'ils avaient dans l'application source.
- *Préservation du visuel.* Il ne doit pas y avoir de différence visuelle entre l'application source et l'application cible. Cette contrainte est particulièrement importante pour les logiciels commerciaux. En effet, les utilisateurs de l'application ne doivent pas être perturbés par la migration.

Une dernière contrainte inhérent aux entreprises est la possibilité pour les équipes de développement de continuer la maintenance des applications pendant le développement de la stratégie de migration et la migration elle même.

## 3.2 Comparaison de GWT et Angular

2. MLOC : Million lines of code

Tableau 1: Comparaison des architectures de GWT et Angular.

| | GWT | Angular |
|---|---|---|
| page web | Une classe Java | Un fichier TypeScript et un fichier HTML |
| style pour une page web | Inclue dans le fichier Java | un fichier CSS optionnel |
| Nombre de fichier de configuration | Un fichier de configuration | Quatre fichiers plus deux par sous-projets |

Dans le cas de ce projet, le langage de programmation source et cible ont deux architectures différentes. Les différences sont syntaxical, semantical et architectural. Pour la migration d'application GWT vers Angular, les fichier *.java* sont séparé en plusieurs fichiers Angular.

Comme présenté dans le Tableau 1, la séparation des fichiers Java en fichier Angular se fait à trois endroits, les fichiers de configuration, les fichiers définissant la page web et les fichiers de style.

Avec le Framework GWT, un seul fichier est nécessaire pour représenter une page web. L'ensemble de la page web peut donc être contenu dans ce fichier, il reste toutefois possible de créer d'autre fichier pour séparé les différents éléments de la page web. Le fichier java contient les différents widgets de la page web, leurs positions les uns par rapport aux autres et leurs organisations hiérarchique. Dans le cas de widget sur lesquels une action peut être exécuté (comme un bouton), c'est dans ce même fichier qu'est contenu le code à executer lorsque l'action est réalisée. En Angular, on crée une hiérarchie de fichier correspondant à un *sous-projet* pour chaque page web. Ce sous-projet contient plusieurs fichiers dont un fichier HTML qui contient les widgets de la page web et leurs organisations, et un fichier TypeScript contenant le code à exécuter quand une action ce produit sur les widgets du fichier HTML. On a donc une séparation de un fichier GWT vers deux fichiers Angular pour représenter les widgets et le code qui leurs est associé.

Pour le style visuel d'une page web, dans le cas de GWT, il y a un fichier CSS commun à toutes les pages webs et des modifications qui sont appliquées directement dans le fichier java de la page web. Ces modifications peuvent porter sur la couleur ou les dimensions. En Angular, on retrouve le même fichier CSS général pour tout le projet, cependant c'est un fichier CSS que l'on doit créer par sous-projet qui va définir le visuel des éléments de la page web. Il y a donc création d'un fichier supplémentaire en Angular par rapport à GWT.

Pour les fichiers de configurations, GWT n'a besoin que d'un fichier de configuration qui défini les fichiers java correspondant à une page web et les URL que l'on devra utiliser pour y accéder. En Angular, il y a deux fichiers de configuration générales. Le premier, *module*, explicite les différentes page web accessible dans l'application ainsi que les services distant et les composants graphiques (widgets) utilisable dans l'application. Le second, de *routing*, définit pour les différentes pages web de l'application leurs chemins d'accès.

### 3.3 Stratégies de migration

Il existe plusieurs manières d'effectuer la migration d'une application. Toute les solutions doivent respecter les contraintes définis Section 3.1.

— *Migration manuelle.* Cette stratégie correspond au re-développement complet des applications sans l'utilisation d'outils aidant à la migration. La migration manuelle permet de facilement corriger les potentielles erreurs de l'application d'origine et de re-concevoir l'application cible en suivant les

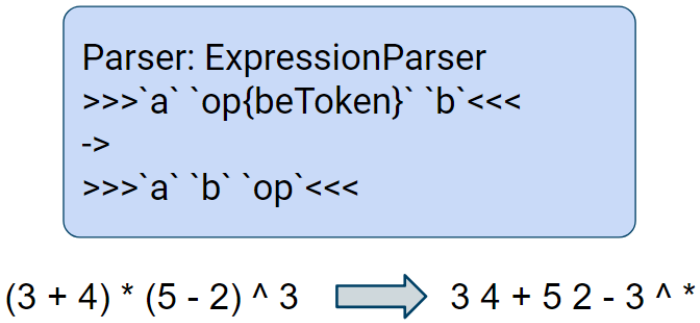



FIGURE 2 – Exemple de règle de transformation

precepts du langage cible.

— *Utilisation d'un moteur de règle.* L'utilisation d'un moteur de règle pour migrer partiellement ou en totalité une application a déjà été appliqué sur d'autres projet [1]–[3]. Pour utiliser cette stratégie, nous devons définir et créer des règles qui prennent en entré le code source et qui produisent le code pour l'application migrée. La Figure 2 montre un exemple de règle de transformation. Dans ce cas, elle permet de changer la position des opérateurs dans une expression mathématique source. L'opérateur est maintenant en suffixe de l'expression. Il est possible que la migration ne soit pas complète. Dans ce cas, les développeurs devront finir le processus de migration avec du travail manuelle. L'utilisation d'un moteur de règle, bien qu'efficace, implique une solution qui n'est ni indépendante de la source, ni indépendante de la cible de la migration.
— *Migration dirigée par les modèles.* La migration dirigée par les modèles implique le développement de méta-modèles pour effectuer. La stratégie respecte l'ensemble des contraintes que nous avons défini. Une migration semi-automatique ou complètent automatique est envisageable avec cette stratégie de migration. Comme pour l'utilisation d'un moteur de règle, dans le cas d'une migration semi-automatique, il peut y avoir du travail manuel à effectuer pour compléter la migration.

# 4 Mise en place de la migration par via les modèles

# 5 Modélisation

# 6 Discussion

# 7 Conclusion

## Bibliographie

[1] J. Brant, D. Roberts, B. Plendl, and J. Prince, “Extreme maintenance : Transforming delphi into c,” in *Software maintenance (icsm), 2010 ieee international conference on*, 2010, pp. 1–8.

[2] S. I. Feldman, “A fortran to c converter,” in *ACM sigplan fortran forum*, 1990, vol. 9, pp. 21–22.

[3] R. W. Grosse-Kunstleve, T. C. Terwilliger, N. K. Sauter, and P. D. Adams, “Automatic fortran to c++ conversion with fable,” *Source code for biology and medicine*, vol. 7, no. 1, p. 5, 2012.