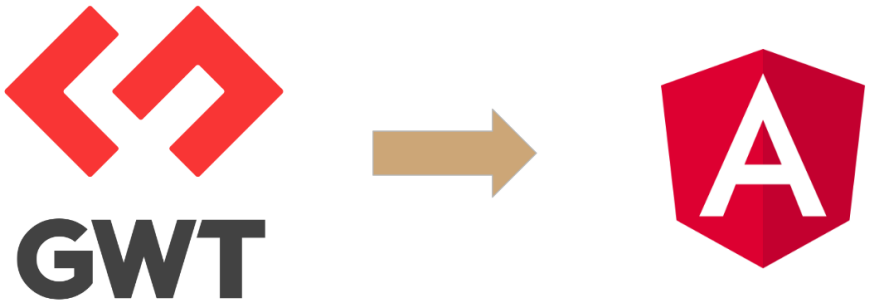
Benoît VERHAEGHE

Rapport PFE



Tuteur : Mme ETIEN

Polytech Lille
Décembre 2017

Contents

1 Cahier des charges **2**
1.1 Contexte 2
1.2 Objectifs 2

2 Résultats **3**
2.1 Recherche bibliographique 3
2.2 Outils d'analyse 3
2.2.1 Décomposition du projet 3
2.2.2 Décomposition d'une application 4
2.2.3 Construction du modèle abstrait 5
2.2.4 Extraction du modèle 6
2.2.5 Analyse des résultats 7

3 Travaux futurs **9**

Bibliographie **10**

1 Cahier des charges

1.1 Contexte

Ce projet s'inscrit dans le contexte d'une collaboration entre l'équipe RMod d'Inria Lille Nord Europe et Berger-Levrault, un éditeur logiciel.

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 4. Le back-end est une application monolithique et doit évoluer vers une architecture de services Web. Le changement de framework graphique est imposé par une nouvelle version de GWT qui n'assurera pas la compatibilité arrière. Le passage à une architecture à services est voulu pour améliorer l'offre commerciale et la rendre plus flexible.

1.2 Objectifs

J'ai comme objectif d'aider Berger-Levrault dans cette migration. Une étape est de modéliser l'application *bac à sable* de Berger-Levrault afin de pouvoir l'analyser. Pour cela, je vais devoir analyser le code source de l'application et discuter avec l'entreprise. Ensuite, je devrai importer le code de l'application dans un logiciel permettant de l'analyser, puis créer le code d'analyse de l'application. La modélisation abstraite d'une application de Berger-Levrault est une étape essentielle car elle assure la réutilisabilité de mon travail pour d'autres logiciels de l'entreprise. Une autrz étape est de faire une recherche bibliographique car elle permettra de démarrer des futures recherches dans le domaine.

2 Résultats

2.1 Recherche bibliographique

La première étape de mon travail a été de le positionner par rapport à la littérature existante.

Il existe trois sous-grands domaines qui auraient pu correspondre.

- Library Migration est le domaine qui parle des changements de librairie dans un langage. Dans mon cas, cela va être un problème lorsque l'on va vouloir retranscrire un comportement défini par une librairie JAVA vers une librairie Angular. Cependant la majorité des articles de ce sous-domaine discute surtout de la migration de librairie d'un langage vers le même langage.
- Monolithic into Microservices discute de la division de sous-ensembles d'applications pour en créer des plus petites. Comme pour la Library Migration, la bibliographie reste majoritairement dans le même langage
- Widget Migration semble être la partie de la litérature qui correspond le mieux à ce que je fais. On discute de la détection des widgets et de comment réécrire ceux-ci dans un autre langage. En particulier, on y retrouve un article de Samir, Stroulia, and Kamel (2007) dans lequel on parle de la migration de Swing vers ajax. Cependant il utilise une analyse dynamique de l'application pour détecter les comportements. Au vu de la taille des applications de Berger-Levrault ainsi que de l'impact de l'utilisation d'une telle stratégie sur les utilisateurs, je ne pourrai pas utiliser la même stratégie. Je pourrai peut être réutiliser le XUL décrit brièvement dans ce document pour représenter le comportement des widgets de mon travail.

2.2 Outils d'analyse

2.2.1 Décomposition du projet

Grâce à la recherche bibliographique, j'ai décidé des différentes grandes étapes que je vais devoir mener pour mon travail.

Une première étape est d'extraire du code source d'une application un modèle abstrait des différents widgets de l'application.

Ensuite, je dois convertir ces différents widgets vers un modèle abstrait qui contient pour chacun le visuel et le comportement du widget en fonction des actions qu'on lui applique.

Enfin, développer un parseur prenant en entrée le modèle abstrait en ressortant le nouveau code source.

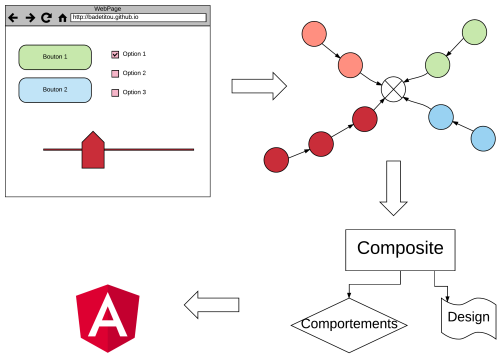


Figure 1: Étapes de mon travail

2.2.2 Décomposition d'une application

Après analyse et discussion avec Berger-Levrault, j'ai pu extraire la structure générale de leurs applications.

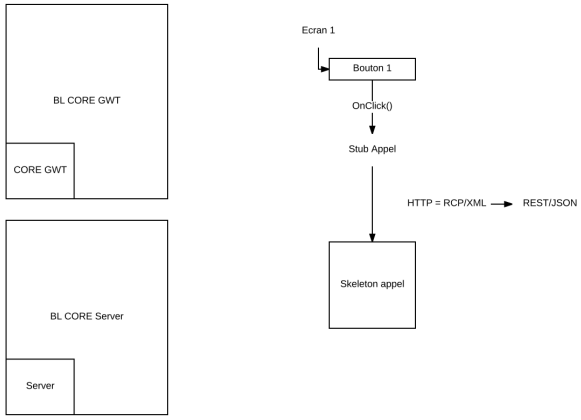


Figure 2: Structure Application Berger-Levrault

Comme le présente la Figure 2, tous les "widgets" provenant de GWT ont été sous-classés par un "widget" Berger-Levrault (ex: BLTextField).

L'application utilise deux fichiers de configuration.

- *application.module.xml* détaille l'ensemble des phases (fenêtres) de l'application.
- *coreincubator.gwt.xml* permet de définir les scripts et les css à utiliser dans l'application. Le fichier décrit aussi des redéfinitions de classes qui se font au moment de la compilation. Ci-dessous, la classe BLUserPreferencesNull sera remplacée à la compilation par BLUser-Preferences. Berger-Levrault utilise ce type de fichier pour définir le comportement de ses logiciels à compilation et non via des options sélectionnables par l'utilisateur.

```
<replace-with class="fr.bl.client.core.refui.base.gwt.BLUserPreferencesNull">
      <when-type-is class="fr.bl.client.core.refui.base.gwt.BLUserPreferences" />
</replace-with>
```

On précise que Berger-Levrault n'utilise pas cette technique pour changer lors de la compilation le visuel de ses widgets, mais cela peut changer leurs comportements.

2.2.2.1 Précision sur les Phases

Une phase contient une ou plusieurs pages métiers. Si elle contient plusieurs pages métiers, celles-ci seront représentées par des sous-onglets dans l'application web.

C'est `Workspace.getPhaseManager()` qui permet de gérer l'ouverture des phases et la commande `ConstantsPhase.Util.get().NOM_DE_LA_PHASe()` permet d'ouvrir une nouvelle phase. Cependant, il faut regarder dans le fichier *application.module.xml* de l'application pour récuperer les vrais liens.

Ex:

```
vpErgo.add(new BLLinkLabel("Internationalisation",
   ConstantsPhase.Util.get().PHASE_MAQUETTE_I18N(),
   null, AbstractDesk.TYPE_TAB_UNIQUE));
```

En cliquant sur le BLLinkLabel, on ouvrira la phase PHASE_MAQUETTE_I18N. Dans le fichier xml, on recherche donc PHASE_MAQUETTE_I18N et on trouve

```
<phase codePhase="PHASE_MAQUETTE_I18N"
  codeValue="GEN_I18N" className="fr.bl.client.coremaquette.PhaseMaquetteI18n"
    raccourci="I18" titre="I18n" description="Exemple d'intertionalisation." />
```

C'est donc la classe fr.bl.client.coremaquette.PhaseMaquetteI18n qui sera réellement cherchée.

Les phases peuvent être de deux types :

- unique : l'onglet est ouvert une fois, puis actualisé en fonction des demandes
- multiple : un onglet est ouvert à chaque demande

2.2.3 Construction du modèle abstrait

Il a ensuite été possible de créer un modèle d'une application de Berger-Levrault que je vais pouvoir exploiter.

La Figure 3 permet de représenter une application de Berger-Levrault. On y retrouve les Phases et les Pages métiers ainsi que les widgets et leurs actions. C'est ce modèle que j'essaie de visualiser dans mon analyse.

La Figure 4 présente les modèles faisant les liens entre les classes du modèle que j'ai créé et le modèle JAVA.

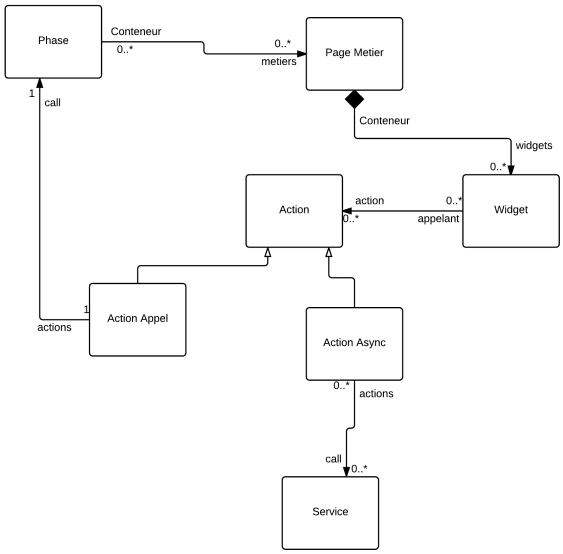


Figure 3: Modèle d'une application de Berger-Levrault V1

2.2.4 Extraction du modèle

Pour faire mon analyse, j'ai utilisé l'outil Moose[1]. Pour le moment je travaille sur la première étape, l'extraction de l'architecture d'une application de Berger-Levrault. L'objectif est de réussir à séparer les différents widgets.

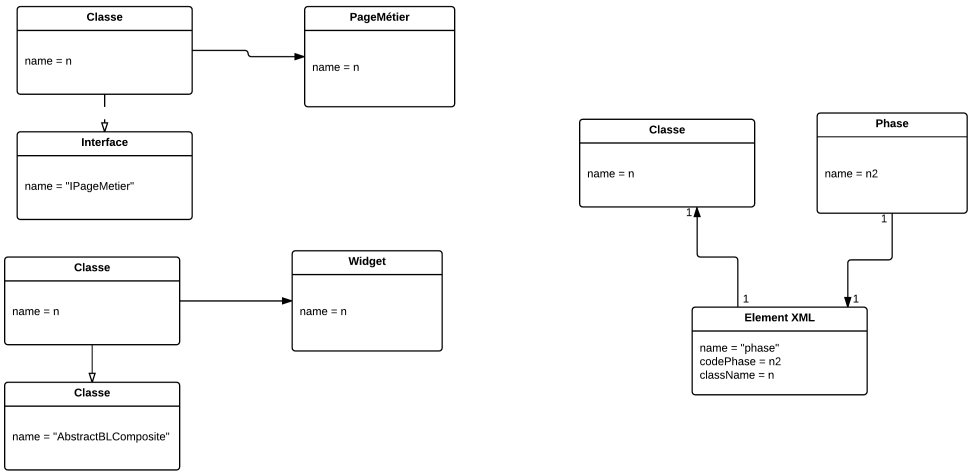


Figure 4: Correspondance Modèle - JAVA

La première étape est la génération d'un fichier *.mse*. Pour cela j'ai utilisé l'outil VervaineJ. Ce logiciel me permet de créer le fichier *.mse* depuis le code source d'une application (sans avoir besoin de le compiler). J'avais avant cela essayé d'utiliser l'outil jdt2famix. Mais cet outil a

[1]Moose est une plateforme pour l'analyse de logiciels et données.

besoin du code source compilé de l'application. Cependant les applications de Berger-Levraut sont composées de plusieurs petits projets. Cette structure de projet rend l'utilisation de jdt2famix impossible pour le moment.

2.2.5 Analyse des résultats

La page d'accueil type d'une application de Berger-Levrault est présentée Figure 5. Comme nous pouvons le voir, il y a beaucoup de liens qui existent depuis cette phase vers d'autres. Ce sont tous les liens présents sur la gauche de la page Web. Cette page web est générée avec de nombreux Widgets *BLLinkLabel* comme on peut le voir dans la code Figure 6. Les Widgets sont donc naturellement recodés en des liens HyperTextes.

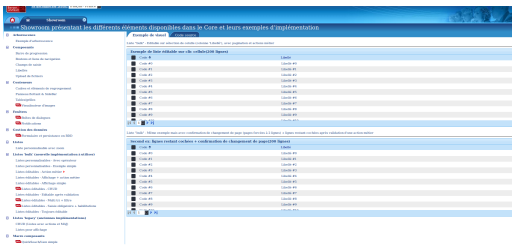


Figure 5: Page d'accueil bac à sable

```
@Override
public void buildPageUi(final Object object) {

    // ########## Demos composants d'ergonomie ##########
    BLFieldSetPanel fspErgo = new BLFieldSetPanel("Ergonomie");
    BLVerticalPanel vpErgo = new BLVerticalPanel();
    vpErgo.add(new BLLinkLabel("Phase avec dashboard", ConstantsPhase.Util.get().PHASE_INCUBATOR_SAMPLE_DASHBOARD(), nul
    vpErgo.add(new BLLinkLabel("Phase avec bandeau usager, panneau flottant et sidebar", ConstantsPhase.Util.get().PHASE
    vpErgo.add(new BLLinkLabel("Internationalisation", ConstantsPhase.Util.get().PHASE_MAQUETTE_I18N(), null, AbstractDe
    vpErgo.add(new BLLinkLabel("Exemple de zoom dans liste", ConstantsPhase.Util.get().PHASE_INCUBATOR_SAMPLE_ZOOMABLE_L
    vpErgo.add(new BLLinkLabel("Exemple de liste éditable (CRUD)", ConstantsPhase.Util.get().PHASE_INCUBATOR_SAMPLE_EDIT
    vpErgo.add(new BLLinkLabel("Visualisateur d'images", ConstantsPhase.Util.get().PHASE_INCUBATOR_SAMPLE_VISU_IMAGES(),
    fspErgo.setWidget(vpErgo);

    // ########## Demos composants standards ##########
    BLFieldSetPanel fspDemos = new BLFieldSetPanel("Composants");
    BLVerticalPanel vpDemos = new BLVerticalPanel();
    vpDemos.add(new BLLinkLabel("Exemples de listes", ConstantsPhase.Util.get().PHASE_INCUBATOR_SAMPLE_LIST(), null, Abs
    vpDemos.add(new BLLinkLabel("Exemples de listes BI", ConstantsPhase.Util.get().PHASE_TEST_LISTE_BI(), null, Abstract
    vpDemos.add(new BLLinkLabel("Exemples de listes éditables", ConstantsPhase.Util.get().PHASE_INCUBATOR_SAMPLE_EDITABL
    vpDemos.add(new BLLinkLabel("Exemples de listes éditables paginées", ConstantsPhase.Util.get().PHASE_INCUBATOR_SAMPL
    vpDemos.add(new BLLinkLabel("Exemples de listes éditables avec restrictions", ConstantsPhase.Util.get().PHASE_INCUBA
    vpDemos.add(new BLLinkLabel("Exemples de listes personnalisables", ConstantsPhase.Util.get().PHASE_INCUBATOR_SAMPLE_
    vpDemos.add(new BLLinkLabel("Exemples de composants", ConstantsPhase.Util.get().PHASE_INCUBATOR_SAMPLE_INPUT(), null
    vpDemos.add(new BLLinkLabel("Exemples de navigations", ConstantsPhase.Util.get().PHASE_INCUBATOR_SAMPLE_NAV(), null,
    vpDemos.add(new BLLinkLabel("Exemples de macros composants", ConstantsPhase.Util.get().PHASE_INCUBATOR_SAMPLE_MACRO(
    vpDemos.add(new BLLinkLabel("Déclencher artificiellement une erreur (démonstration dé-obfuscation de trace client)",
```

Figure 6: Page d'accueil code JAVA

J'ai travaillé sur le modèle Moose pour rechercher grâce à une analyse statique du code les différents éléments représentés dans le modèle Figure 3. En particulier, j'ai cherché à retrouver une phase avec de nombreux liens vers d'autres (comme j'ai trouvé juste avant).

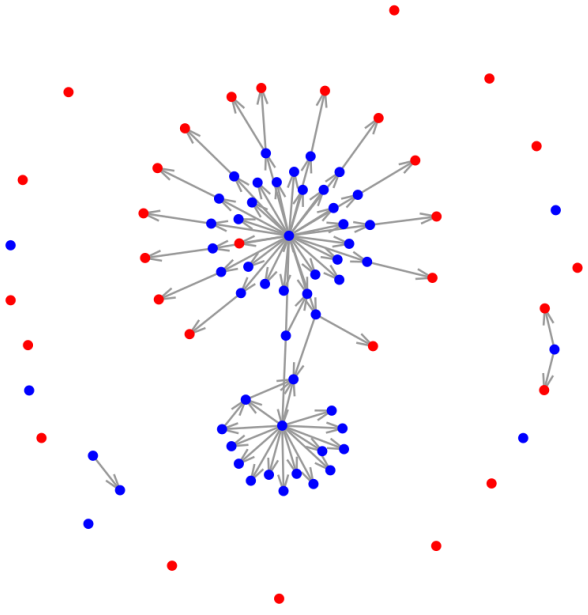


Figure 7: Relation entre les phases

La Figure 7 est une sortie que j'ai réussi à obtenir représentant l'application de Berger-Levrault. Les cercles bleus représentent les différentes phases. Les cercles rouges représentent les pages métiers. Les arcs, la possibilité d'aller d'une phase à une autre grâce à un widget s'ils vont d'une phase à une autre. Si un arc va d'une phase à une page métier, c'est l'utilisation d'une page métier par une phase qui est représentée.

On repère aisement deux HUB (phases qui ont des liens vers beaucoup d'autres phases). En utilisant Moose et le code source de l'application, j'en ai déduit que c'était les *home* que nous avons détectés pendant l'analyse du code source.

Nous pouvons aussi voir des phases seules, ce sont soit des phases de tests qui n'ont donc réellement aucun lien avec le reste de l'application, soit des phases utilitaires qui sont utilisées dans des phases abstraites que je ne peux pas actuellement exploiter avec Moose car il me manque des sources de l'application.

3 Travaux futurs

La suite logique après avoir trouvé les phases et les pages métiers est de trouver les widgets contenus dans chaque page métier. Il faudra ensuite trouver l'ensemble des actions impossibles et réussir à les attribuer. Il sera intéressant de définir une carte grâce à Moose qui contiendra tous ces éléments et nous permettra de naviguer entre les éléments.

Appronfondir la recherche bibliographique sur la widget migration semble aussi être la bonne chose à faire afin de faire un point sur ce qui existe déjà dans la migration de widget. Je pourrai sûrement m'inspirer du travail qui a déjà été fait et ajouter mon travail dans ce domaine.

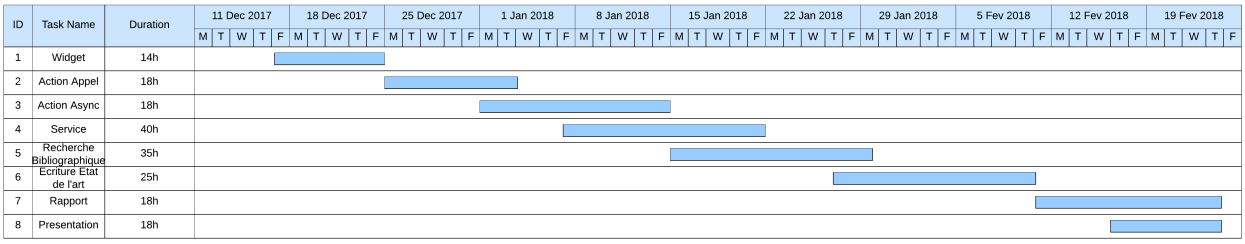


Figure 8: Prévision du travail futur

Comme le présente le Gantt Figure 8, la recherche de widget ne devrait pas être compliquée. C'est la recherche des Services (et comment les définir dans le modèle) ainsi que la recherche bibliographique et l'écriture de l'état de l'art qui devrait me prendre le plus de temps. En effet, la définition des Services paraît moins évidente et la recherche bibliographique ainsi que l'état de l'art sont les tâches pour lesquelles j'ai le moins de connaissance. Une phase d'apprentissage rallongera donc le temps que peuvent prendre ces tâches pour un professionnel du domaine.

Bibliographie

Samir, Hani, Eleni Stroulia, and Amr Kamel. 2007. “Swing2script: Migration of Java-Swing Applications to Ajax Web Applications.” In *Reverse Engineering, 2007. WCRE 2007. 14th Working Conference on*, 179–88. IEEE.