



**Ecole Nationale Supérieure d'Informatique
et d'Analyse des Systèmes**



Mémoire de Projet de Fin d'Études

Pour l'Obtention du Titre

D'Ingénieur d'État en Informatique

Option

Génie Logiciel

Sujet

Mise en place d'une application de gestion délocalisée de commandes Fret SNCF

Soutenu par :

Achraf KARIMI

Sous la direction de :

Pr. Mounia FREDJ, Présidente du Jury : Professeur à l'ENSIAS

Pr. Karim BAINA, Examineur : Professeur à l'ENSIAS

Pr. Ahmed ZELLOU, Encadrant ENSIAS : Professeur à l'ENSIAS

Mr. Saad LAAMOURI, Encadrant Externe : CAPGEMINI

Dédicace

A mes très chers parents,

Nulle expression, nulle dédicace, ne saurait exprimer mon amour éternel et mes sentiments de considération pour les sacrifices que vous avez fait pour mon bien-être. Votre générosité et votre bonté ont toujours été un exemple pour mes frères et moi. Trouvez en ce travail le fruit de votre dévouement et l'expression de ma gratitude et mon profond amour.

A ma petite sœur Najla et mon petit frère Badreddine,

Votre amour et soutien, m'ont toujours donné la force de continuer

A ma très chère famille,

A tous mes ami(e)s de l'école et d'ailleurs,

*Je vous aime tous, et je n'oublierais jamais chaque moment passé avec vous,
Une pensée à Othmane, Yassine, Amine, Adnane, Simo, Soukaina, Ouafae, Jihad, Hicham,
Ilyas, Nabil, Yahya, Younes, Oussama, Mehdi, Mouad, Kaoutar, Rihane, tout le groupe six,
toute la promo 2013 et finalement tous ceux que je n'ai pu cités*

Aux personnes qui m'apprécient,

Un grand Merci

Achraf

Remerciements

Au terme du stage de fin d'études effectué au sein de la société de services Capgemini, je tiens à remercier vivement Mr Ahmed ZELLOU, Professeur à l'ENSIAS, pour son encadrement, son soutien moral, ainsi que pour tous ses conseils instructifs durant toute la période de l'établissement de ce travail.

Il me sera également d'une ingratitude de ne remercier de manière spéciale nos encadrants à Capgemini, je cite Mr. Etienne NASSE et Mr. Saad LAAMOURI mes encadrants, qui ont veillé au bon déroulement de ce projet, Mr. Steeve SAINTMAIN chef de projet et Mme Mounia ZARKAOUI directeur de projet, pour leurs directives précieuses et leurs soutiens.

J'exprime mes profonds respects à tous les membres de l'équipe « flux & commandes » pour la période de travail agréable que j'ai passée à leur côté. Merci donc à Soufiane Mouflih, Younes EL ARIF et Amine RIAD pour leurs hospitalité. À Imane OUBADRISS, Ahmed IDRISSE OUEDRHIRI, Basma LAHLOU, Fatima Zahra et Naoufel LEBHAR pour leur convivialité. À l'ensemble des collaborateurs de Capgemini de Casablanca pour leur sympathie.

Je tiens aussi à remercier tous les membres du jury qui ont porté un intérêt sur mon projet, et qui m'ont fait l'honneur d'accepter de juger le travail.

Un Merci, encore une fois, à toute personne ayant contribué de près ou de loin à la réalisation de ce projet, m'offrant ainsi une occasion pour découvrir le monde du travail, d'apprendre l'esprit de ponctualité et d'équipe, de confirmer les techniques acquises à l'entreprise et de me préparer à affronter la vie active.

Pour tous ceux que j'ai oubliés, je vous dis MERCI.

Résumé

Le présent document est le fruit d'un travail que j'ai effectué au sein de Capgemini Maroc, dans le cadre de projet de fin d'études. Ce projet a pour but de réaliser une application de gestion délocalisée des commandes Fret SNCF, sans passer par les processus métier de son système d'information.

Durant le projet, j'ai eu pour mission dans un premier temps de cerner le sujet et de délimiter le périmètre du projet. Après une analyse de la problématique et des besoins, j'ai constaté que ces derniers dépendaient du développement de l'application. J'ai donc du diviser le projet en modules. Pour chaque module, j'ai suivi le processus suivant : Définition des spécifications, établissement des risques, élaboration d'un modèle conceptuel, mise en œuvre, et test de validation.

Pour bien mener le projet, j'ai décidé de suivre un cycle de développement en spirale, une démarche qui a fait ses preuves dans le domaine des projets informatiques dont les besoins ne sont pas constants.

Ce rapport est axé sur quatre grandes parties. La première partie définit le contexte général du projet en présentant l'organisme d'accueil et en définissant le périmètre, ainsi que les objectifs du projet. La deuxième partie porte sur l'étude préliminaire, exposant les grandes fonctionnalités du système, et le situant dans son contexte technique. Dans cette même partie, nous définissons les itérations de développement. La troisième partie est consacrée à l'implémentation des itérations une à une, respectant le processus adopté, et prenant la plus grande partie du rapport. Les outils de mise en œuvre, quant à eux, sont présentés brièvement dans la quatrième partie. Et finalement, une annexe jointe, à la fin, rassemblera les compléments d'information. Elle contient, entre autres, la plupart des interfaces illustrant l'utilisation de l'application.

Mots clés

fret, commande, xml, schéma, générique, flex, mapping

Abstract

The present document summarizes the work done as part of my final project with Capgemini Maroc. The object of the project was creating an application that allows Capgemini, and especially to my work team, to manage locally Fret SNCF orders, without applying the business processes of its information system.

During project analysis, I found that the team's needs depended on application's development. So I had to divide the project into modules. For each module, I followed the process: Definition of specifications and risks, development of a conceptual model, implementation, deployment, testing and validation. In order to well lead the project, I decided to follow a spiral development cycle.

This report focuses on four main parts. The first part defines the contextual environment of the project, presenting Capgemini and defining the objectives. In the second part, we present a preliminary study of the project, outlining the main features of the system and placing it in a technical context. The implementation of iterations is sequentially described in the third part. The fourth part consists on tools' presentation, with which I performed the iterations, respecting the process adopted. Finally, we collected information complements in annex

at the end of this document. It contains, among others, most interfaces illustrating the use of the application.

Key words

fret, command, xml, schema, generic, flex, mapping

Liste des abréviations

Abréviation	Description
AOP	Aspect Oriented Programming
AS	Action Script
BRIC	Base de Regroupement des Informations Commerciales
BSD	Berkeley software distribution License
DAO	Data Access Object
DOM	Document Object Model
EJB	Entreprise Java Bean
IHM	Interface Homme Machine
IOC	Inverse of Control
J2EE	Java Enterprise Edition
MLMC	Multi Lots Multi Clients
REST	REpresentational State Transfer
RIA	Rich Internet Application
SAX	Simple API for XML
SNCF	Société Nationale des Chemins de Fer
SWF	Shockwave Flash
XML	eXtended Markup Language
XSD	Xml Schema Data

Table des figures

Figure 1: <i>Diagramme général de cas d'utilisation</i>	20
Figure 2: <i>Diagramme de Gantt</i>	23
Figure 3: <i>Architecture de bric-MLMC-prgcmd</i>	25
Figure 4: <i>Espace de travail</i>	26
Figure 5: <i>Couches métier de l'application</i>	27
Figure 6: <i>Flux de données entre les trois premières couches</i>	32
Figure 7: <i>Diagramme de classes de GenericMapping</i>	33
Figure 8: <i>Diagramme de classes en construction de la couche Display</i>	35
Figure 9: <i>Liaison entre GenericMapping, BeanServices, et Display</i>	36
Figure 10: <i>Diagramme d'activité de la méthode createDocumentBean</i>	37
Figure 11: <i>Diagramme d'activité de la méthode selectValidGetMethods</i>	38
Figure 12: <i>Résultat d'affichage de la première itération</i>	39
Figure 13: <i>Diagramme de cas d'utilisation concernant la navigation</i>	40
Figure 14: <i>Diagramme de cas d'utilisation concernant la modification</i>	44
Figure 15: <i>Diagramme actualisé de la couche Display</i>	49
Figure 16: <i>Diagramme de séquence pour la méthode "addChild"</i>	50
Figure 17: <i>Changements dans l'interface WindowBean</i>	51
Figure 18: <i>Boutons de modification</i>	51
Figure 19: <i>Options de couleurs</i>	52
Figure 20: <i>Demande de confirmation lors d'une suppression d'élément</i>	52
Figure 21: <i>Diagramme de cas d'utilisation de la troisième itération</i>	53
Figure 22: <i>Diagramme de flux d'information de l'itération 3</i>	54
Figure 23: <i>Diagramme de séquence de la fonctionnalité d'enregistrement</i>	56
Figure 24: <i>Diagramme d'activité de synchronisation</i>	57
Figure 25: <i>Visualisation de la ressource modifiée</i>	58
Figure 26: <i>Diagramme de cas d'utilisation de la gestion des sources de données</i>	59
Figure 27: <i>Diagramme de classes de la quatrième itération</i>	62
Figure 28: <i>Vue générale sur les classes des couches de l'application</i>	63
Figure 29: <i>Diagramme de flux d'information entre les couches de l'application</i>	63
Figure 30: <i>Interface d'accueil de l'application</i>	65
Figure 31: <i>Interface d'affichage des sources de données</i>	65
Figure 32: <i>Interface d'ajout d'une source de données</i>	66
Figure 33: <i>Interface de recherche des ressources</i>	66
Figure 34: <i>Résultat d'une recherche</i>	67
Figure 35: <i>Diagramme de cas d'utilisation de la quatrième itération</i>	68

Liste des tableaux

Tableau 1: <i>Description des itérations</i>	21
Tableau 2: <i>Tableau de modélisation</i>	33
Tableau 3: <i>Cas d'utilisation "rechercher par élément"</i>	40
Tableau 4: <i>Cas d'utilisation "rechercher par contenu"</i>	41
Tableau 5: <i>Cas d'utilisation "contrôler la profondeur d'affichage"</i>	41
Tableau 6: <i>Cas d'utilisation "afficher certains types d'éléments"</i>	42
Tableau 7: <i>Cas d'utilisation "remonter à la racine"</i>	42
Tableau 8: <i>Cas d'utilisation "retourner à l'élément précédent"</i>	42
Tableau 9: <i>Cas d'utilisation "extraire un élément"</i>	43
Tableau 10: <i>Cas d'utilisation "plier déplier"</i>	43
Tableau 11: <i>Cas d'utilisation "rétablir la valeur précédente"</i>	44
Tableau 12: <i>Cas d'utilisation "rétablir la valeur suivante"</i>	45
Tableau 13: <i>Cas d'utilisation "éditer la valeur"</i>	45
Tableau 14: <i>Cas d'utilisation "vider le champ"</i>	45
Tableau 15: <i>Cas d'utilisation "supprimer les fils de l'élément"</i>	46
Tableau 16: <i>Cas d'utilisation "créer les fils"</i>	46
Tableau 17: <i>Cas d'utilisation "ajouter un élément"</i>	47
Tableau 18: <i>Cas d'utilisation "supprimer un élément"</i>	47
Tableau 19: <i>Cas d'utilisation "vider la liste"</i>	47
Tableau 20: <i>Tableau des risques de la deuxième itération</i>	48
Tableau 21: <i>Cas d'utilisation "enregistrer la ressource"</i>	53
Tableau 22: <i>Cas d'utilisation "visualiser la ressource modifiée"</i>	54
Tableau 23: <i>Tableau des risques de la troisième itération</i>	55
Tableau 24: <i>Cas d'utilisation "ajouter une source de données"</i>	59
Tableau 25: <i>Cas d'utilisation "modifier une source de données"</i>	59
Tableau 26: <i>Cas d'utilisation "supprimer une source de données"</i>	60
Tableau 27: <i>Cas d'utilisation "rechercher une ressource"</i>	60
Tableau 28: <i>Tableau des risques d'intégration de la quatrième couche</i>	61
Tableau 29: <i>Légende du diagramme de flux d'information des couches de l'application</i>	64
Tableau 30: <i>Cas d'utilisation "afficher les restrictions"</i>	68
Tableau 31: <i>Cas d'utilisation "valider l'ensemble du document"</i>	69
Tableau 32: <i>Tableau des risques d'intégration de la quatrième itération</i>	69

Tables des matières

Liste des abréviations	5
Table des figures.....	6
Liste des tableaux	7
Table des matières	8
Introduction générale.....	10
Chapitre 1	11
1.Contexte générale du projet.....	12
1.1. Présentation de l'organisme d'accueil	12
1.1.1. Capgemini Maroc.....	12
1.1.2. Activités de Capgemini Maroc.....	12
1.1.3. Situation du projet	13
1.1.4. Organisation de l'équipe	13
1.2. Présentation du sujet.....	14
1.2.1. Contexte métier du sujet.....	14
1.2.2. Problématique.....	14
1.2.3. Cahier de charges	15
1.2.4. Méthodologie	16
1.3. Conclusion du chapitre.....	17
Chapitre 2	18
2.Etude préliminaire	19
2.1. Etude fonctionnelle.....	19
2.1.1. Analyse détaillée des clauses fonctionnelles.....	19
2.1.2. Diagramme général de cas d'utilisation	20
2.1.3. Définition des itérations de l'application	21
2.1.4. Planification	22
2.2. Etude technique	24
2.2.1. Analyse détaillée des clauses non fonctionnelles.....	24
2.2.2. Architecture du projet.....	25
2.2.3. Architecture J2EE ?.....	26
2.2.3. Couches métier de l'application.....	27
2.2.4. Conclusion du chapitre.....	29
Chapitre 3	30
3.Mise en œuvre	31
3.1. Itération 1 : L'implémentation de l'affichage d'une ressource.....	31
3.1.1. Spécifications	31
3.1.2. Risques d'intégration.....	31
3.1.3. Conception de l'itération	31

3.1.4.	Réalisation et test	36
3.2.	Itération 2 : Les fonctionnalités avancées de l'interface.....	40
3.2.1.	Spécifications	40
3.2.2.	Risques d'intégration.....	48
3.2.3.	Conception de l'itération.....	48
3.2.4.	Réalisation et test	51
3.3.	Itération 3 : Les modifications sur le Bean mappé	53
3.3.1.	Spécifications	53
3.3.2.	Risques d'intégration.....	55
3.3.3.	Conception de l'itération.....	55
3.3.4.	Réalisation et test	57
3.4.	Itération 4 :La gestion des sources de données.....	58
3.4.1.	Spécifications	58
3.4.2.	Risques d'intégration.....	61
3.4.3.	Conception de l'itération.....	61
3.4.4.	Réalisation et test	64
3.5.	Itération 5 :L'implémentation des restrictions.....	68
3.5.1.	Spécifications	68
3.5.2.	Risques d'intégration.....	69
3.5.3.	Conception de l'itération.....	70
3.5.4.	Réalisation et test	71
3.6.	Conclusion du chapitre.....	71
Chapitre 4		72
4.Outils de réalisation.....		73
4.1.	Configuration et gestion de projet	73
4.1.1.	Maven.....	73
4.1.2.	Spring	73
4.1.3.	SVN.....	73
4.2.	Développement et tests	74
4.2.1.	Flex.....	74
4.2.2.	Blaze DS.....	74
4.2.3.	Rest.....	75
4.2.5.	Jaxb	75
4.2.6.	Junit	76
4.3	Conclusion du chapitre	76
Conclusion générale		77
Perspectives		78
Bibliographie		79
Annexe.....		80

Introduction générale

Il y a quelques années, l'informatique était une baguette magique dans la main des professionnels. A coups d'automatisation, on optimisait les processus métiers, on offrait plus de fonctionnalités, on changeait en mieux la façon de faire les choses. Dès lors, des parcs applicatifs ont émergé pour gérer toutes les activités d'une entreprise, et continuaient de croître avec le développement de l'activité professionnelle. Cependant, à force d'user la baguette, les professionnels se sont retrouvés avec des systèmes d'information complexes, composés de plusieurs applications mêlées dont on arrive plus à suivre l'évolution. En conséquence, une incohérence fonctionnelle ou technique peut mettre en péril tout le système de l'entreprise, et sa restitution peut ainsi coûter très cher. La maintenance du patrimoine informatique n'a jamais été cruciale qu'aujourd'hui.

Le présent stage de fin d'étude s'inscrit dans le cadre de la maintenance des applications Fret SNCF prise en charge par Capgemini. À son terme, le projet servira comme support à l'équipe de travail dans ses activités de développement et de test. L'application réalisée permettra, en effet, de gérer localement les commandes de transport Fret SNCF sans passer par les processus métiers du système d'information. Elle fera partie de l'environnement de développement de l'équipe. Plus tard, elle pourra faire l'objet d'un module supplémentaire dans l'application mère de gestion des commandes, livrée en production dans l'environnement client.

Le présent document présente le travail réalisé en stage dans son ensemble. Il comporte quatre parties. La première partie traite le contexte général du projet, elle présente Capgemini en tant que société de services, ses activités et missions. En passant par le pôle flux et commandes auquel je suis affecté. S'en suivra une présentation du sujet et de la gestion de développement en mode spirale. La deuxième partie se concentre sur l'étude préliminaire concernant l'analyse des besoins, ainsi que les cas d'utilisation généraux qui vont définir les cinq itérations de développement en plus de l'architecture technique du projet et les couches logicielles de l'application. La troisième partie couvre les outils utilisés. Et la dernière partie se charge de la mise en œuvre des cinq itérations. Chacune avec sa propre spécification, conception et réalisation.

Une annexe jointe à la fin du rapport rassemble différentes références et compléments de description détaillée relatifs à certains modules du projet. Elle contiendra en particulier toutes les illustrations sur les fonctionnalités IHM de l'application.

Chapitre 1

Contexte général du Projet

Ce chapitre a pour but de situer le projet dans son environnement organisationnel et contextuel. Nous allons voir une petite présentation de Capgemini. Puis nous nous dirigerons vers le pôle flux et commandes du projet SNCF, l'équipe qui m'a accueilli. Ensuite nous allons aborder le vif du sujet.

1. Contexte général du projet

1.1. Présentation de l'organisme d'accueil

1.1.1. Capgemini Maroc

Capgemini est aujourd'hui le leader européen et l'un des leaders mondiaux du consulting et des services informatiques. Le groupe a effectué plus de quarante acquisitions d'entreprises dans le domaine du service informatique lui permettant de diversifier son activité qui regroupe actuellement quatre grands axes à savoir : le consulting, l'intégration des systèmes, le service informatique de proximité et l'outsourcing. Capgemini commence à s'internationaliser à partir des années 70, en investissant et en s'implantant partout dans le monde, afin de répondre aux exigences de ses clients en matière de coûts.

Installée depuis 2007 au Maroc, Capgemini Maroc est la seule filiale africaine de Capgemini TS, elle compte aujourd'hui plus de 600 collaborateurs. Le développement de cette filiale marocaine s'inscrit dans le système mondial de production du Groupe baptisé Right shore. Capgemini Maroc participe aujourd'hui à des projets importants de développement et de maintenance de systèmes d'information pour le compte d'une dizaine d'acteurs francophones majeurs notamment dans le secteur public, l'énergie, les services financiers et les télécommunications. En s'implantant à CasaNearshore, Capgemini répond à la stratégie d'industrialisation des grands groupes français qui recherchent des partenaires capables de les accompagner sur l'ensemble de leurs projets informatiques [CapPre]

.

1.1.2. Activités de Capgemini Maroc

L'activité principale de la filiale marocaine est d'accueillir l'offshoring d'une partie d'un projet d'intégration de services, pour le compte de sociétés clientes françaises. Il s'agit donc d'accompagner le client dans la transformation et la restructuration de leurs systèmes d'information. Ceci, par le développement de solutions technologiques innovantes et adaptées aux besoins parfois spécifiques du secteur d'activité dans lequel le client exerce. D'autres activités viennent enrichir la panoplie des services que propose Capgemini Maroc en plus de l'offshoring, notamment le consulting et le développement des solutions pour le marché local marocain [CapPre]

1.1.3. Situation du projet

Le projet s'inscrit dans le cadre du système d'information de Fret SNCF, qui est constitué d'un ensemble de grandes applications, conçues pour répondre chacune à un ou plusieurs processus métier : contractualisation, commande, transport, valorisation, et facturation. Et ce, dans un contexte limité à un seul type de service proposé par SNCF.

Parmi ces applications, bric-MLMC est l'application qui, de son nom, gère le service Multi-Lots Multi-Clients. Elle couvre entièrement l'étape de Commande, et répond en partie aux étapes de Contractualisation, Transport et Valorisation. Bric-MLMC ne gère pas la facturation du service, il fait appel aux services d'une autre application appropriée. Pour chacune de ces étapes, bric-MLMC compte une application consacrée. L'application bric-MLMC-prgcmd gère l'étape de commande pour le service Multi-Lots Multi-Clients, et est ainsi, le contexte le plus bas niveau du projet de fin d'étude.

1.1.4. Organisation de l'équipe

L'équipe accueillante compte 10 personnes, divisée entre le centre de service de Lille (3 personnes) et celui de Casablanca (7 personnes). Les sept personnes à Casablanca forment le pôle flux et commande. Il est en charge de deux projets, MLMC et DEF (DEplacement Ferroviaire). Ces projets ont pour axe principal la gestion des commandes.

Selon les règles en vigueur, une personne dans l'équipe assure le rôle d'encadrant en suivant de près le travail du stagiaire, et une autre assure le rôle de référent technique. Le stagiaire quant à lui, doit se conformer aux règles de gestion interne de ressources humaines et les règles de gestion des équipes. Ainsi il doit revenir vers le Team Leader pour la définition des tâches à remplir sur le management visuel, un grand tableau notant les activités de chaque membre de l'équipe. Et vers le directeur de projet en cas de demande de congé ou d'absence. Le directeur de projet ne fait pas partie de l'équipe, mais avec le team leader, assurent le management de toute l'équipe. L'encadrant et le référent technique, quant à eux, assurent le support dans le travail du stagiaire.

1.2. Présentation du sujet

1.2.1. Contexte métier du sujet

MLMC est un service de transport destiné principalement aux industriels. Le service a été reconçu pour répondre au mieux aux besoins des clients. Auparavant, lorsqu'un client venait demander un service de transport, SNCF mettait à sa disposition un certain nombre de wagons, et sa commande était acheminée individuellement. Le nombre de wagons dont le client avait besoin ne dépassait pas la moitié de wagons d'un train complet¹. En conséquence, la puissance de la locomotive n'était pas bien exploitée, la disponibilité de la locomotive était réduite et le prix du service se faisait cher pour le client.

Aujourd'hui MLMC répond à la satisfaction du client en se basant sur un nouveau principe. Celui de choisir plusieurs lots dont les trajets se rejoignent en un sous trajet commun, délimité par une gare de départ et une gare d'arrivée. Puis les rassembler dans la gare de départ. Ensuite les transporter en un seul grand lot jusqu'à la gare d'arrivée. Et enfin, expédier chacun vers sa propre destination finale [MLMCPre].

1.2.2. Problématique

Le service qui est devenu en soi plus complexe, propose trois types de services (M1, M2 et M3). Pour chaque type, le client ouvre une commande spécifique, et est tenu de l'enrichir par ses prévisions de trafic sur une unité de temps. Ces informations doivent être saisies continuellement avant chaque trimestre, mois ou semaine selon le type du service.

L'application bric-MLMC-prgcmd est conçue pour supporter ce processus de commande. La base de données mise à disposition stocke les ressources sous format XML, pour des raisons d'interopérabilité. Cependant, la structure complexe des commandes fait que les traitements sur la ressource XML soient très gourmands, réduisant considérablement les performances de l'application, aussi bien que le travail de maintenance effectuée par l'équipe de travail. En effet, l'équipe ne dispose pas d'outil pour traiter directement les ressources. Chaque correction ou vérification de ressource incombe à la personne :

- d'accéder manuellement à la base de données,
- de charger le contenu de la ressource dans un éditeur de texte,
- de parcourir la ressource à la main pour vérifier ou modifier,
- d'enregistrer manuellement dans la base de données.

¹ Un train complet se constitue d'une locomotive et d'une vingtaine de wagons

Sans rappeler le temps et la patience perdus à cause de cette manipulation. Ce processus encoure des risques éminents d'erreurs pour les raisons suivantes :

- le positionnement difficile dans une ressource de taille très grande,
- la complexité du contenu de la ressource,
- l'utilisation d'outils externes pour la récupération et l'édition de la ressource,
- la modification de la ressource sans validation avec le schéma.

Chose qui a poussé l'équipe à exprimer un grand besoin d'automatisation de la procédure.

1.2.3. Cahier de charges

Un cahier de charge préliminaire a été établi à l'issue d'une réunion tenue avec l'encadrant sur les besoins primaires. L'application doit répondre aux charges suivantes :

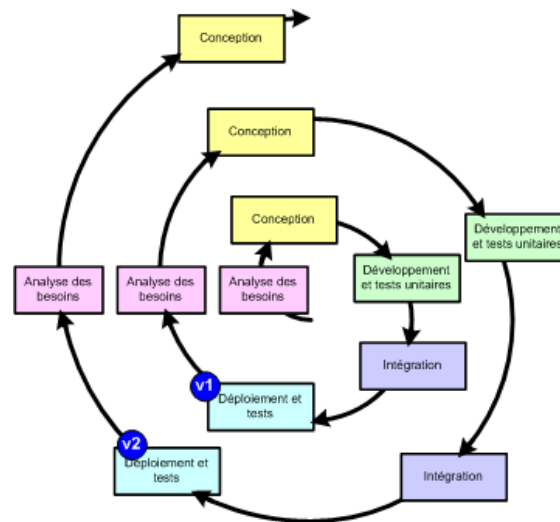
- Afficher le contenu d'une ressource, formaté selon le type,
- Modifier le contenu d'une ressource,
- Enregistrer les modifications,
- Gérer les sources de données,

La ressource dans ce contexte est soit une Précommande, Commande, ou Répartition, dont la période couverte est respectivement, le trimestre, le mois, et la semaine. Ce cahier de charges décrit essentiellement les directives d'ordre général auquel nous nous sommes référés pour dégager les aspects fonctionnels de l'application. À constater que les spécifications ne donnent pas plus de détails sur les fonctionnalités elles mêmes. En vérité, un retour rapide sur les aspects fonctionnels nous montrerait que certains aspects dépendent logiquement d'autres : Pour pouvoir modifier, il faut d'abord visualiser. Pour pouvoir visualiser, il faut d'abord spécifier la source de donnée. Ainsi, la description des fonctionnalités relatives à un aspect dépendent plus ou moins des fonctionnalités de l'aspect prédécesseur. C'est pour cette raison qu'il est difficile de spécifier les fonctionnalités à cette phase de développement. Une autre raison serait que les besoins exprimés par l'équipe soient susceptibles de changer en fonction du développement des fonctionnalités : de nouveaux besoins pourraient apparaître, ceux-ci pourrait apporter de nouvelles fonctionnalités ou rendre obsolètes certaines d'autres déjà implémentées. Ceci a été déterminant dans le choix de la méthodologie de développement.

1.2.4. Méthodologie

Face à un besoin flexible et non détaillé, nous avons décidé d'adopter la méthode spirale comme méthode de développement. Elle sied parfaitement à la situation, où le client précise ses besoins au fur et à mesure que le produit progresse. La méthode s'appuie sur une succession de cycles dont chacun se déroule en quatre phases :

- détermination des objectifs, des alternatives et des contraintes ;
- analyse des risques, évaluation des alternatives et conception;
- développement et vérification de la solution retenue ;
- revue des résultats et vérification du cycle suivant.



Le cycle de vie en spirale est un modèle générique de cycle de vie évolutif qui a été proposé par Barry W. Boehm en 1984. Ce modèle, axé sur la maîtrise et la réduction des risques, est davantage un cadre de travail guidant la construction d'une démarche spécifique de projet, plutôt qu'une démarche formalisée.

Le cycle de vie en spirale est applicable pour de projets internes à l'entreprise. Les maîtrises d'ouvrage peuvent voir dans le cycle de vie en spirale une méthode de conduite de programme, dont chaque boucle serait un sous projet. Chaque boucle de spirale permet :

- D'identifier les objectifs propres de la boucle,
- Les moyens alternatifs pour atteindre les objectifs,
- Les contraintes de chaque alternative,

Elle donne lieu au choix d'une alternative, validé par un prototype le cas échéant, et à l'exécution de l'alternative choisie. A l'issue de la boucle, une revue des produits et des résultats fournit une évaluation qui sert d'entrée pour la boucle suivante [Spirale].

1.3. Conclusion

A l'issue de ce premier chapitre, nous avons pris connaissance du périmètre du projet, qui demeure dans un contexte assez large, ne faisant du projet qu'un petit maillon de toute une chaîne, sans que ça en amoindrisse l'importance. Nous avons aussi consulté le cahier de charge, qui sera l'objet d'étude du chapitre suivant. Et finalement, nous avons vu le cycle de vie adopté pour mener à bien la réalisation de l'application.

Chapitre 2

Etude préliminaire

Dans ce chapitre, nous allons effectuer les premières études du projet. Une analyse détaillée du cahier de charge est nécessaire dans cette phase. Cela nous donnera une vision plus claire sur les itérations de développement .En plus d'une étude technique sur l'architecture générale du projet et les couches métier.

2. Etude préliminaire

2.1. Etude fonctionnelle

2.1.1. Analyse détaillée des clauses fonctionnelles

Nous allons procéder à une analyse détaillée des clauses fonctionnelles du cahier de charge, en vue d'en comprendre le besoin réel de l'équipe. Chaque spécification analysée donnerait lieu à des spécifications plus précises sur le développement de l'application.

Afficher le contenu d'une ressource

Les ressources dans la base de données sont de trois types, Précommande, Commande et Répartition. Chacune admet une structure différente. Le besoin d'afficher le contenu de la ressource pourrait ne pas être aussi simple qu'il paraît, si on prend en considération que cette structure peut changer dans les futures maintenances. En réalité, puisqu'on parle de contenu XML, chaque type de ressource possède son schéma XSD. Dès lors on se rend compte de l'obligation de prendre ce schéma dans l'affichage de la ressource, car cette spécification n'exprime pas le besoin d'un affichage simple de contenu, mais d'un affichage élaboré capable de cacher sa complexité.

Modifier le contenu d'une ressource

L'action de modification est généralement critique. Elle l'est encore plus lorsqu'il s'agit de respecter la structure, le type, et les restrictions sur ce dernier. Le recours au schéma XSD est une évidence. Néanmoins l'effet de conformité avec les règles du schéma doivent apparaître au moment de modification. Chose qui permettra à l'utilisateur de savoir immédiatement l'état de la modification qu'il vient d'effectuer.

Enregistrer les modifications

Dans cette spécification, on sort du contexte de la ressource. L'enregistrement se rapporte plus à la source de données qu'autre chose, ce qui implique directement la prise en compte des différents types de sources de données. On comprend, alors, que l'enregistrement doit être

repensé pour chaque type. Cependant, la difficulté ne résiderait pas dans leur nombre, mais dans la manière d'enregistrer propre à chacun. Il devient donc nécessaire de bien spécifier les caractéristiques et points de distinction entre sources.

Gérer les sources de données

L'application devra traiter des ressources provenant de plusieurs endroits différents. Et ce n'est pas juste une question de localisation, plusieurs variables peuvent distinguer une source de données et ses interactions avec l'application. C'est pour cette raison, que la gestion doit adopter un aspect générique capable de s'adapter avec la diversité des sources.

2.1.2. Diagramme général de cas d'utilisation

Avec l'analyse détaillée du cahier de charges, nous avons maintenant tous les éléments nécessaires pour situer l'application dans son cadre fonctionnel. Nous avons également projeté la lumière sur les interactions avec l'utilisateur, à qui elle va apporter un grand support dans ses tâches de test et de maintenance. Nous rappelons que la méthode choisie est la méthode spirale, avant de nous lancer dans les itérations, nous devons d'abord définir les points d'entrée. Ces points d'entrée sont présentés dans le diagramme général de cas d'utilisation de la figure 1.

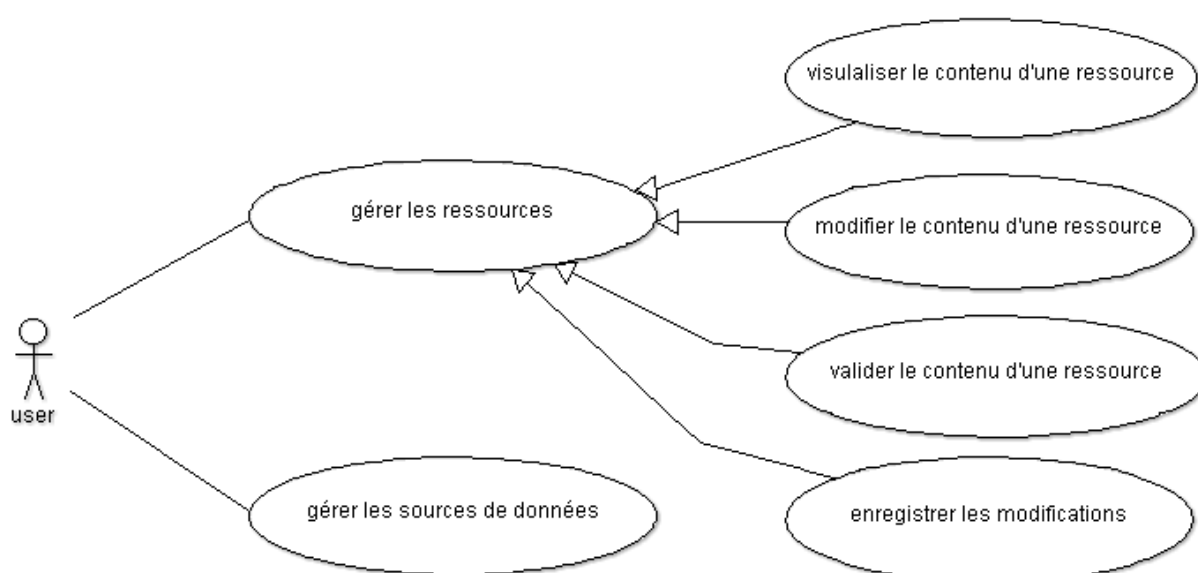


Figure 1: Diagramme général de cas d'utilisation

Sans surprise, nous retrouvons presque les mêmes aspects fonctionnels. L'ajout de la « validation du contenu de la ressource » comme point d'entrée est justifié par son indépendance avec la modification. L'utilisateur pourrait choisir de valider le contenu d'une ressource sans qu'il l'ait modifié au préalable. Aussi, il est à noter que la « gestion des sources de données » n'a pas été développée dans le diagramme, car elle sera implémentée au cours d'une seule itération, là où l'implémentation de la « gestion des ressources » s'étalera sur plusieurs.

2.1.3. Définition des itérations de l'application

Les cinq points d'entrée représentent les aspects fonctionnels des itérations. Chaque itération devra par la suite raffiner son point d'entrée en des fonctionnalités unitaires dont l'implémentation s'accomplira suivant les étapes de développement classiques.

Les itérations que nous avons définies dans le développement de l'application sont cinq, classées ici par ordre de réalisation :

1. Une itération pour l'affichage d'une ressource
2. Une itération pour le développement de l'interface et des interactions
3. Une itération pour l'enregistrement dans la ressource
4. Une itération pour la gestion des sources de données
5. Une itération pour implémenter la validation avec les restrictions

Une description de chaque itération est donnée par le tableau 1:

Tableau 1: *Description des itérations*

Itération	Description
1	Cette itération aura pour but de préparer une représentation du contenu de la ressource à l'interface, de façon à ce qu'elle puisse donner une marge de manœuvre pour les fonctionnalités à offrir.
2	Cette itération implémentera toutes les interactions de navigation et d'édition dans l'interface. Les modifications effectuées ne seront appliquées que sur le modèle local représentant la ressource.
3	Cette itération couvrira l'application, depuis l'interface, des modifications sur le modèle distant représentant la ressource.

4	Cette itération ajoutera la gestion des différents types de sources de données. Et implémentera la fonctionnalité de recherche de ressource dans chaque source, complétant ainsi les processus métier de l'application.
5	Cette itération sera autour de la validation des modifications dans l'interface. Cette validation se basera sur les restrictions du schème XSD qui devront accompagner les représentations de la ressource dans ses transformations.

Le choix de l'ordre est en partie influencé par les dépendances qui lient les itérations. Mais selon ces dépendances, un ordre normal serait « 4 », « 1 », « 2 », « 5 », « 3 ». Ceci s'explique par la priorité basse qu'a donnée l'équipe à l'itération « 5 », ce qui l'a repoussé en dernier. Et pour l'itération « 4 », nous n'avions pas accès aux sources de données dans les débuts de développement. Nous ne voulions pas entamer une itération sans être sûrs de pouvoir la terminer. De plus que, l'itération « 1 » implémentera le socle de l'application puisqu'elle couvre une bonne partie du processus de traitement de la ressource.

Concernant le raffinement dans chaque itération, il sera décrit, quand c'est possible, par des cas d'utilisation. Ensuite, une partie sur les risques d'intégration sera impérative avant toute conception, afin de déterminer les fonctionnalités dont l'implémentation pourrait entrer en conflit avec l'implémentation d'une fonctionnalité déjà existante dans le système, et prendre ce risque en considération. La conception, quant à elle, est une étape primordiale dans le cycle de vie d'une itération. Sa présentation dans le rapport doit être autant claire et pertinente. À cet effet, dans chaque itération, un diagramme de classe accompagné au besoin d'un diagramme de séquences, ou d'activité, mettront le point sur la composition et le fonctionnement du système.

2.1.4. Planification

La planification du projet est parmi les phases en aval. Elle consiste à prévoir le déroulement du projet tout au long des phases constituant le cycle de développement. Conformément au processus de développement adopté. Cependant, à défaut de connaissance des itérations, nous n'aurions pas pu déterminer le planning dans la partie « présentation générale du sujet ». Leur définition étant faite à présent, nous pouvons présenter, alors, le planning du cycle de vie réalisé sous GANTT.

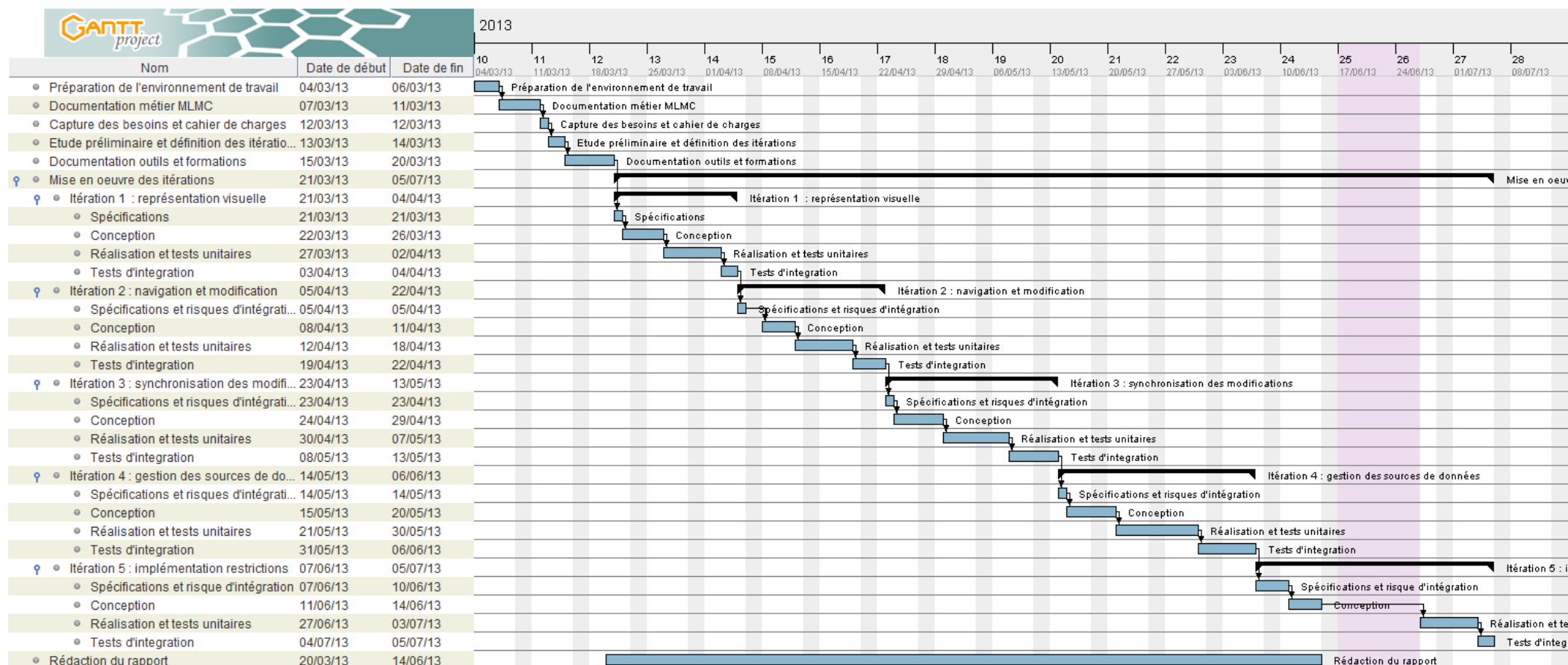


Figure 2: Diagramme de Gantt

2.2. Etude technique

2.2.1. Analyse détaillée des clauses non fonctionnelles

Dans cette partie nous allons nous pencher sur les clauses non fonctionnelles priorisées par l'équipe, afin de les prendre en considération durant la réalisation du projet. Elles sont deux clauses.

S'intégrer dans l'environnement de développement

Le besoin d'intégration exprimé ici revient à deux raisons. Premièrement, l'application doit être intégrée pour ne pas être délaissée par l'équipe. L'intégration dans l'environnement de développement assurera son évolution pour l'implémentation de nouvelles fonctionnalités, et sa maintenance si jamais des correctives seraient prises. Deuxièmement, l'application pourrait faire l'objet d'un nouveau livrable qui n'existe pas chez le client SNCF, les fonctionnalités pourront être réadaptées pour correspondre aux besoins de ce dernier, et la livraison se fera comme habituelle puisque l'application existerait déjà sur l'environnement de développement.

Respecter les règles de développement

Capgemini est une société de service. Les règles de développement lui permettent de tenir ses engagements de durée de livraison et de qualité des livrables. La mise en pratique de ces règles est une recommandation, parfois obligation, pour toutes les activités de développement de Capgemini, dont fait partie, le projet de stage. Et son application se concrétise par le respect des normes et des bonnes pratiques. En plus particulier, il existe des règles de développement propres à chaque client. Dans le cas du projet de fin d'études, les outils de développement en seront grandement influencés [MLMCPre]:

- La présentation est réalisée en FLEX
- Le serveur de déploiement est Tomcat
- Le back Office est réalisé en Java

2.2.2. Architecture du projet

Architecture du projet parent

Les deux précédentes spécifications montrent que l'architecture du grand projet compte beaucoup pour le développement de l'application. Notons que l'intégration d'une application parmi d'autres dans l'environnement de développement n'est pas toujours chose évidente. Alors pour aboutir à une intégration juste sans devoir impliquer des risques inutiles dans l'enjeu, nous devons connaître l'architecture de l'application.

L'application mère est une application web qui s'appuie sur un modèle d'architecture logicielle à 5 couches : Présentation, Coordination, Services, Domaine, et Persistance. Seules nous intéressent les deux couches Présentation et Persistance. La première, parce qu'elle est réalisée en FLEX, implémente des interfaces élaborées et utilise les fonctionnalités les plus utiles de l'outil. La deuxième, parce qu'elle implémente l'accès aux ressources dont l'application aura besoin, et sur lequel, nous pourrions être amenés à apporter quelques changements pour offrir des services plus adaptés à l'application. Toutes les deux feront de bonnes références d'exemples. La figure suivante illustre l'architecture du grand projet :

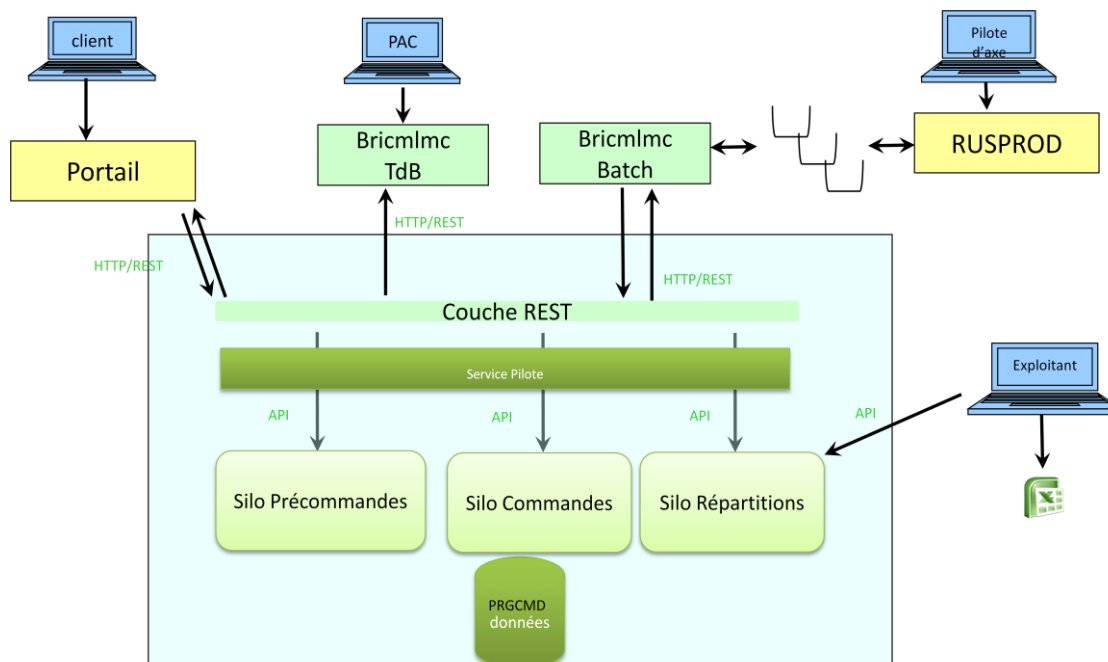


Figure 3: Architecture de bric-MLMC-prgcmd

Nous remarquons les trois divisions (silos) suivant le type de la ressource, et leur interaction avec la couche REST qui propose aux couches supérieures des web services d'accès. Ces derniers seront ré exploités dans l'application [GraPro].

Architecture du projet de stage

L'architecture du projet suivra un modèle à base de couches, car, hormis ses points forts dont nous citons quelques uns :

- maîtriser la complexité des applications. Chaque couche a ses propres responsabilités et propose des services homogènes,
- améliorer le découplage entre composants de l'application. Chaque couche peut être développée et testée de manière individuelle, chose qui rend flexible le développement de l'application et l'application de la méthode spirale.

Il est recommandé pour la raison que l'application est une application web qui sera déployée sur un serveur d'applications, et sera accédée par l'utilisateur sous forme d'une RIA (Rich Internet Application) à base de FLEX. Donc à distinction de ces deux couches différentes, le projet sera réparti en deux sous projets. Un sous projet sera consacré au back office en JAVA et portera le nom « bricmlmc-prgcmd-ihm-service » conformément aux règles de nommage du pôle, et l'autre sera consacré au front office en FLEX/AS et portera le nom « bricmlmc-prgcmd-ihm-flex ».

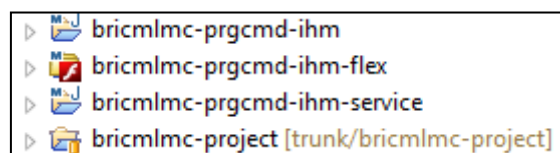


Figure 4: Espace de travail

En haut, le projet de l'application de stage. En bas, le projet de l'application « bric-MLMC », qui contient entre autres, l'application mère « bric-MLMC-prgcmd ». Et les deux restants sont les sous projets en question.

2.2.3. Architecture J2EE ?

Contrairement à ce qu'on pourrait penser, l'application n'implémente l'architecture J2EE. En effet, elle ne se base ni sur les EJB pour l'implémentation métier, ni sur les servlets/JSP pour l'interaction avec l'utilisateur. L'application se base sur le client RIA, qui partage avec le serveur la charge globale de l'application. Les communications passent entre les deux cotés à l'aide d'un petit serveur intégré, en l'occurrence, Blaze DS pour un client FLEX. Une description plus complète sera présentée dans le chapitre « Outils de réalisation ».

2.2.4. Couches métier de l'application

Si le fonctionnel de l'application est défini par l'ensemble des grandes fonctionnalités qu'elle propose à l'utilisateur, pour répondre à son besoin. Le métier de l'application serait défini par l'ensemble des processus internes, implémentés pour correspondre à chacune des fonctionnalités. Dans ce sens, les couches métiers sont des couches indépendantes qui regroupent, sous un même aspect métier, tous les éléments (classes, interfaces, fichiers, etc.) fonctionnant ensemble pour réaliser une étape dans un processus métier.

L'application est répartie sur cinq couches, présentées et décrites ici par ordre hiérarchique:

- Display,
- Services,
- GenericMapping,
- Mapping,
- DAO.

L'illustration montre le positionnement de chaque couche dans le système. Les flèches vides aux deux extrémités représentent l'interaction avec l'environnement extérieur, les sources de données et l'utilisateur. Les flèches remplies montrent les appels entre les couches internes. Ici, la couche Display dépend uniquement de la couche Services, dans la mesure où celle-ci coordonne les appels aux autres couches.

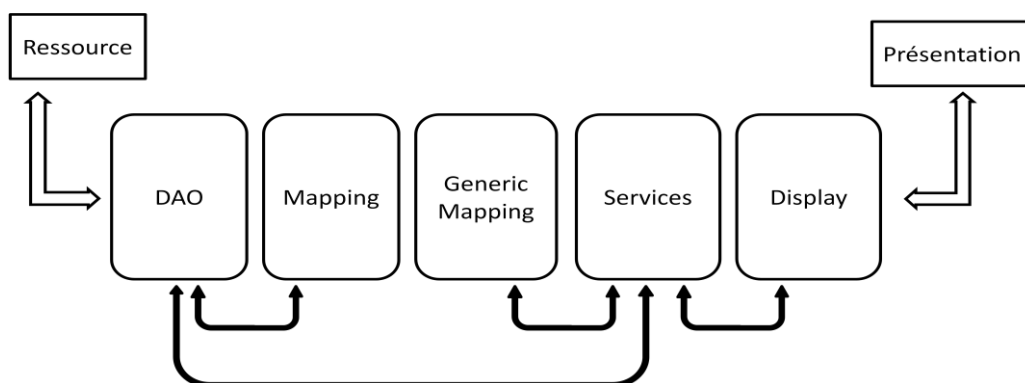


Figure 5: Couches métier de l'application

Display est la couche présentation de l'application. Elle intègre principalement :

- L'affichage,
- l'interaction avec l'utilisateur,
- le contrôle de la cinématique des écrans (page flow),

- les erreurs et les exceptions qui peuvent être levées,
- l'appel aux services dans la couche Services.

Services est une couche particulière du fait de sa position dans le schéma. Elle se tient en tant qu'intermédiaire, non seulement entre la couche Display et les autres au niveau fonctionnel, mais aussi entre les technologies JAVA et FLEX/AS au niveau technologique. Elle intègre principalement :

- la définition des Beans coté JAVA et Action Script
- le mapping entre les Beans
- le transfert à travers Blaze DS
- la coordination des appels aux couches inférieures

GenericMapping est la couche conçue pour résoudre les problèmes qui peuvent survenir lors du transfert des objets de la couche Mapping entre le client RIA et le serveur. En effet, le principe de ce transfert est tel que tout objet transféré devrait avoir un correspondant mappé de l'autre côté. Cependant, les objets échangés peuvent avoir différentes structures, et leur déclaration de l'autre côté peut poser problème. L'idée est de transférer un objet générique capable de représenter les objets de base, et qui n'est constitué que de types prédéfinis. GenericModel intègre principalement :

- La conversion des objets de la couche Mapping en objets Génériques,
- La conversion des objets génériques en objets de la couche Mapping,
- La synchronisation entre un objet de la couche Mapping et son modèle générique,
- La création d'un objet générique vide à partir d'une classe de la couche Mapping.

Mapping est la couche qui accomplit les premiers traitements sur la ressource en format XML. Elle réalise principalement :

- La conversion des objets Java (Bean) en ressources XML,
- La conversion des ressources XML en objets Java (Bean),
- La restitution des restrictions de la ressource sous forme d'objets Java.

DAO est un acronyme pour désigner la couche Data Access Object, La couche la plus basse dans l'hierarchie de la plupart des structures logicielle. Elle se charge d'assurer les services d'accès aux ressources dans différents environnements. Elle intègre principalement :

- La gestion des environnements,
- La recherche des ressources,
- Les actions de manipulation de ressources (Create, Read, Update).

2.3. Conclusion

Dans ce chapitre, nous avons réalisé une étude préalable du projet. Nous avons ainsi entamé réellement la méthode spirale par la définition des itérations. Ensuite nous avons abordé la structure technique de l'application par la définition des couches métiers. Nous pouvons à présent déclencher la première itération.

Chapitre 3

Mise en œuvre

Ce chapitre est consacré aux itérations de l'application. Chaque partie traite une itération à la fois. Et dans chaque partie, nous retrouvons le cycle de développement classique : spécifications, conception, réalisation et tests. Une partie au début mettra en clair les risques de régression de l'itération précédente, et un bilan à la fin pour l'intégration de l'itération en cours.

3. Mise en œuvre

3.1. Itération 1 : Le mapping générique de la ressource

3.1.1. Spécifications

Cette itération représente les premiers développements de l'application. Il ne sera donc pas surprenant de savoir, que les spécifications à ce niveau se restreindront à la seule implémentation d'une représentation visuelle. Cette dernière implique l'implémentation des couches métiers qui vont la supporter, et qui constituent le socle de la gestion des ressources. Ainsi, l'objectif réel de cette première itération est de réaliser une grande partie de la plateforme métier de l'application.

3.1.2. Risques de régression

Un développement qui suit le modèle des itérations est un développement qui encoure des risques de régression. D'un point de vue fonctionnel, le risque de régression est le risque de voir les fonctionnalités d'un système se déstabiliser à cause d'une correction de bogue ou d'ajout d'une nouvelle fonctionnalité. Ce risque ne doit pas être négligé, pour la bonne raison que l'implémentation des nouvelles fonctionnalités ne respecte pas toujours l'architecture déjà existante, et qu'un tout petit changement pourrait mettre en péril certaines fonctions du système.

Afin de se prémunir contre les risques de régression, un tableau d'implémentations nous aidera à cerner l'impact des nouvelles implémentations sur celles-ci. Et constituera un référentiel pour les tests d'intégration et la correction des bogues à la fin de l'itération. Notons qu'à l'itération actuelle, nous ne pouvons pas parler de risque de régression en l'absence du système. Cette partie ne sera réellement utile que lors des prochaines itérations.

3.1.3. Conception de l'itération

Diagramme de classes de la couche GenericMapping

Bien que la logique d'implémentation veuille que notre première conception couvre l'ensemble des couches de bout en bout, depuis la ressource jusqu'à l'affichage chez l'utilisateur. Elle ne concernera que les couches GenericMapping, Services, et Display. En

vérité, nous nous sommes focalisés sur ces trois couches, car elles cachent la plus grande partie du travail. Effectivement, l'implémentation de la couche Mapping est allégée par l'utilisation de l'outil puissant Jaxb. Et la couche DAO fait l'objet d'une implémentation qui sera réalisée dans la quatrième itération. D'autre part, la représentation visuelle de la ressource se base sur son modèle générique, et le transfert des données entre les deux devra passer par la couche Services. Cela montre l'importance de ces trois couches dans l'implémentation de l'affichage. Un premier diagramme de flux de données nous aidera à suivre l'implémentation. Ce diagramme sera enrichi au fur et à mesure que le système se dévoile. À présent, il ne met en clair que les interactions entre les trois couches :

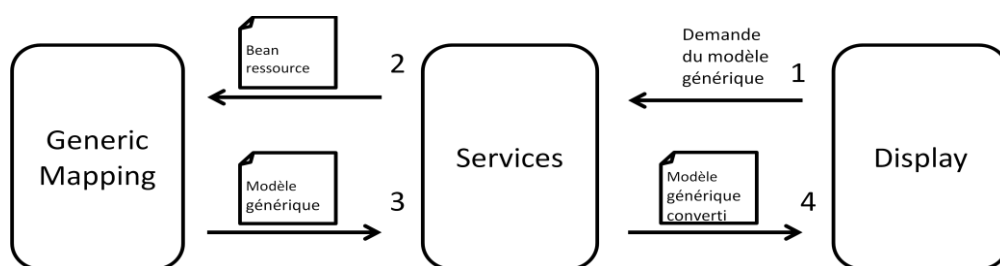


Figure 6: Flux de données entre les trois premières couches

En absence des couches DAO, et Mapping, la couche Services se charge de fournir un Bean de test, créée localement, à la couche GenericMapping. Nous commençons, alors, par la conception de la couche GenericMapping :

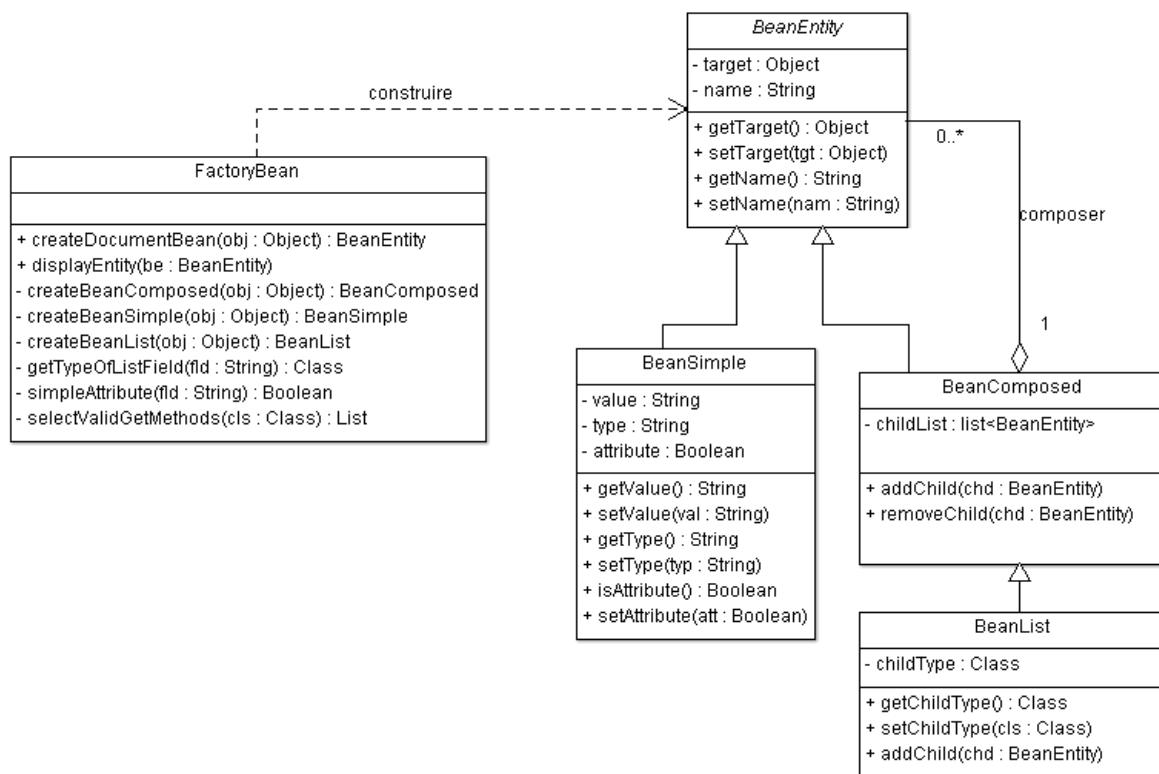


Figure 7: Diagramme de classes de *GenericMapping*

Le *BeanEntity* représente une entité portant un nom et désignant une cible. Le nom est repris sur le nom de l'élément ou de l'attribut (contenu XML), et la cible fait référence à son objet représentant (objet mappé). En héritage, nous retrouvons l'entité *BeanSimple* qui comprend la valeur et son type, Ainsi que l'entité *BeanComposed*, qui se distingue par la composition de plusieurs entités (*ChildList*). Une troisième entité *BeanList* hérite de l'entité *BeanComposed*, et ajoute le type d'entité comme attribut. La différence avec l'entité mère réside, en effet, dans la composition de *ChildList*. La composition peut inclure toutes les entités pour le *BeanComposed*, mais seulement un seul type pour le *BeanList*, exclusivement des *BeanSimples* ou des *BeanComposeds*. La classe *FactoryBean*, est celle qui se charge de générer, à base des entités, le modèle générique sur l'objet mappé de la ressource.

En gardant en tête la structure générale d'un contenu XML, nous avons veillé à ce que sa représentation générique soit bien transcrite dans ce diagramme de classes. Il est impératif à ce que ce modèle puisse modéliser fidèlement les structures de la ressource. Pour en être sûr, le tableau 2 énumère toutes les structures basiques, et leurs modélisations par le diagramme. Soulignons, toute fois, que le design pattern « Composite » utilisé, répond bien à la nature imbriquée du modèle XML.

Dans ce tableau, nous utilisons quelques notions reprises de différents diagrammes UML. Particulièrement, nous avons choisi la syntaxe « [nom_objet] :[nom_classe] » pour désigner les noms des objets et leurs types dans le modèle générique. Nous utilisons aussi des crochets pour renseigner sur d'autres attributs importants dans le cas du *BeanSimple*, et d'une barre verticale pour représenter la liste des fils dans le cas des objets de type *BeanComposed* et *BeanList* :

Tableau 2: Tableau de modélisation

Structure	Exemple de modèle en XML	Représentation dans le modèle Générique
Elément vide	<A/>	<u>A:BeanComposed</u>
Élément contenant seulement du texte	<A> texte	<u>A:BeanSimple</u> [value= "texte";attribute=0]

Élément avec attributs		<u>A:BeanComposed</u> <ul style="list-style-type: none"> a:BeanSimple [value="val1";attribute=1] b:BeanSimple [value="val2";attribute=1]
Élément contenant des éléments avec ou sans répétition	<A> <C> texte </C> <D/> 	<u>A:BeanComposed</u> <ul style="list-style-type: none"> B:BeanList <ul style="list-style-type: none"> O1:BeanComposed <ul style="list-style-type: none"> C:BeanSimple [value="texte";attribute=0] O2:BeanComposed D:BeanComposed
Élément avec attributs contenant des éléments avec ou sans répétition	 <B b="val2" /> <C/> <C> <D> texte </D> </C> <E/> 	<u>A:BeanComposed</u> <ul style="list-style-type: none"> a:BeanSimple [value="val1";attribute=1] B:BeanComposed <ul style="list-style-type: none"> b:BeanSimple [value="val2";attribute=1] C:BeanList <ul style="list-style-type: none"> O1:BeanComposed <ul style="list-style-type: none"> D:BeanSimple [value="texte";attribute=0] O2:BeanComposed E:BeanComposed
Élément mixte contenant du texte avec d'autres éléments ou attributs	<A> Texte 	Le modèle générique n'est pas conçu pour supporter ce cas

Le dernier cas n'a pas été pris en compte lors de la conception du diagramme générique, pour la simple raison que les ressources étudiées ne l'incluent pas dans leur structure. Même à long terme, cette structure n'est pas susceptible d'utiliser le contenu mixte, pour des raisons métier de SNCF.

Diagramme de classes de la couche Display

Passons du côté de la couche Display pour voir le diagramme de classes concernant la représentation visuelle. Nous prévenons, qu'à cette étape, le diagramme est toujours en construction, et que sa présentation ici n'est que pour suivre l'implémentation de l'itération.

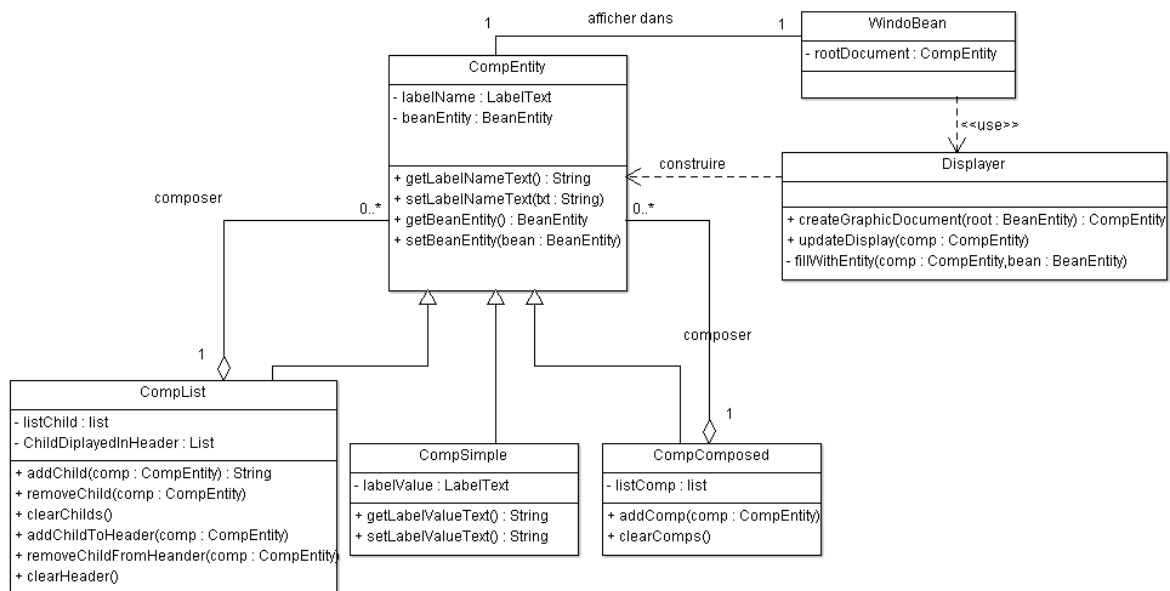


Figure 8: Diagramme de classes en construction de la couche Display

Nous avons défini une classe pour chaque type d'entité. Les classes `CompComposed`, `CompSimple`, et `CompList` correspondent respectivement au `BeanComposed`, `BeanSimple`, et `BeanList`. Ces classes héritent de `CompEntity`, une classe qui a pour but de lier chaque classe à l'entité qu'elle représente. Toute fois, il faudrait savoir que ces classes sont des composants graphiques. En effet, `CompEntity` hérite elle-même du composant graphique `Canvas` (qui ne figure pas dans le diagramme car prédéfini). Et comme tout composant graphique, `Canvas` dispose des attributs spécifiques pour paramétrer son affichage. Remarquons d'ailleurs que, Contrairement au diagramme précédent, `CompList` n'hérite pas de `CompComposed`. Ceci est pour pouvoir personnaliser l'affichage de chaque entité, sans devoir apporter des modifications sur l'autre. Ce point ne semble pas clair ici car nous n'avons pas encore élaboré l'interface, ça le sera dans la prochaine itération. Finalement, La classe `Displayer` est celle qui se charge de générer, à base des classes décrites, le modèle graphique du modèle générique, et l'affiche dans `WindowBean`.

Mise en liaison par la couche Services

La couche Services comporte une seule classe, BeanService. Celle-ci gère tous les transferts de données avec la couche Display. En particulier, elle coordonne les appels aux autres couches pour dérouler les processus. Son rôle de coordination joue un rôle très important dans le métier de l'application.

La figure 9 présente la classe BeanServices, et illustre en même temps les relations entre les classes des trois couches étudiées :

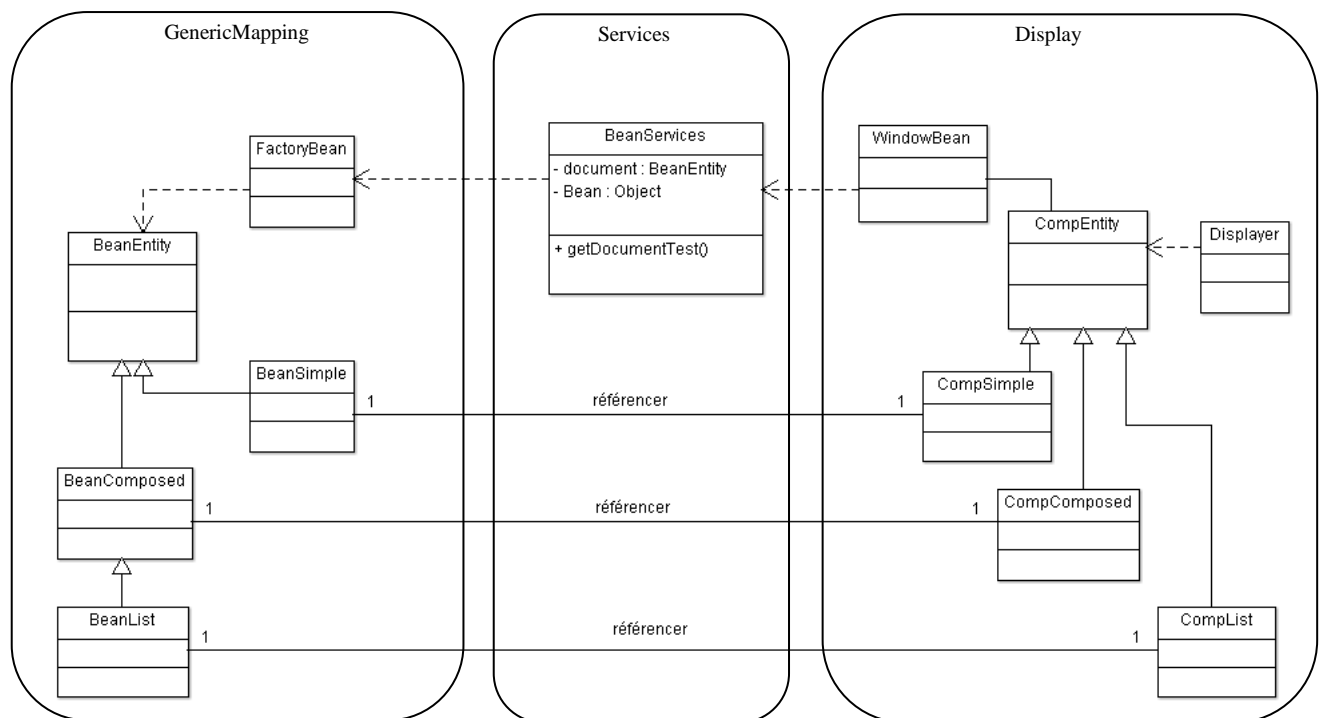


Figure 9: Liaison entre GenericMapping, BeanServices, et Display

La figure reflète une partie de la plateforme métier concernant la gestion des ressources. Toutes les implémentations qui se feront par la suite dans les itérations à venir, viendront enrichir et développer le diagramme.

3.1.4. Réalisation et intégration

Dans cette phase de réalisation, nous avons décidé d'établir un diagramme d'activité pour décrire le comportement de quelques méthodes responsable de construire le modèle générique dans la classe FactoryBean. La plus grande difficulté résidait dans l'utilisation de la réflexion. Les méthodes qui sont décrite dans cette partie sont respectivement, createDocumentBean, et selectValidGetMethods.

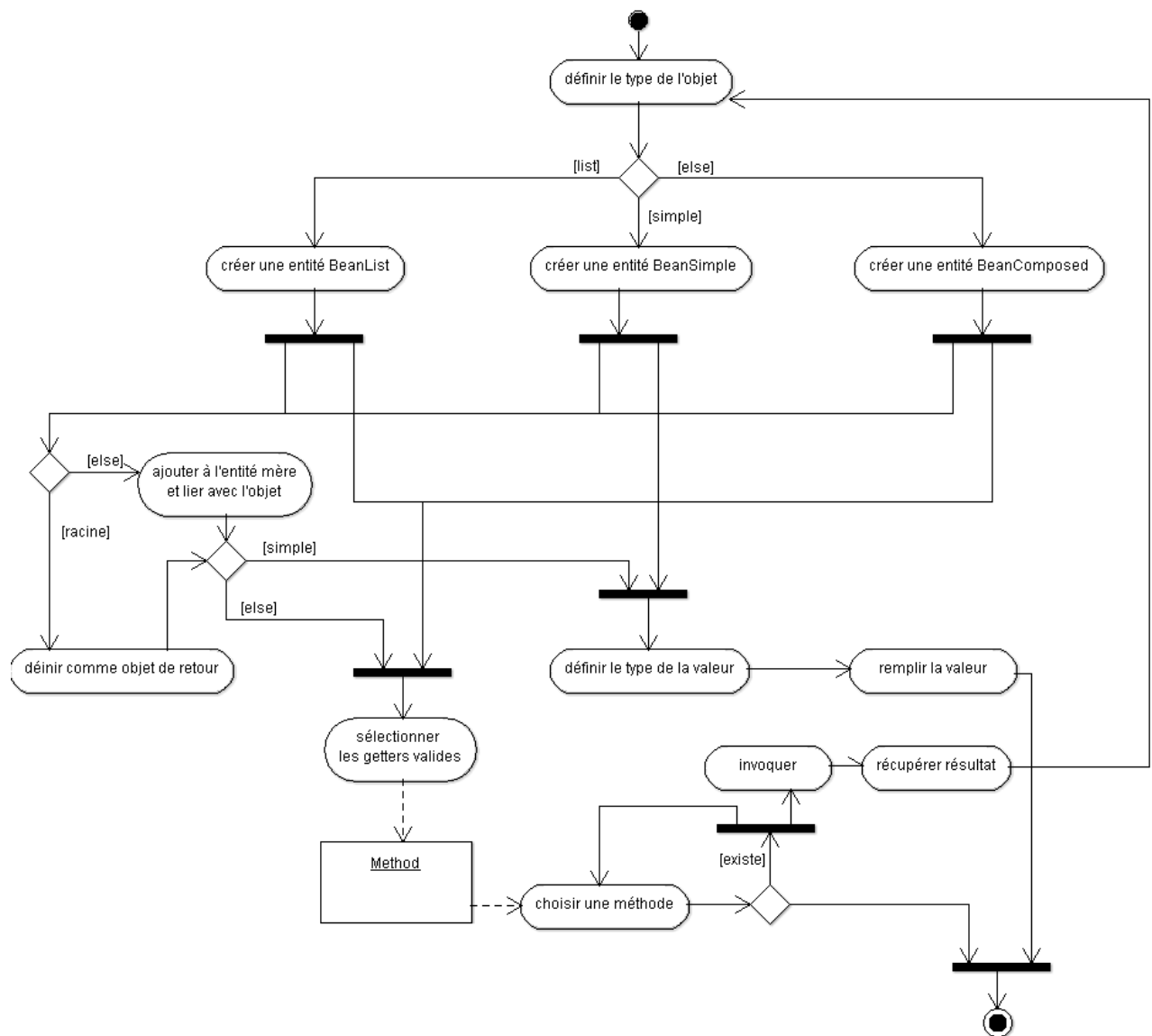


Figure 10: Diagramme d'activité de la méthode *createDocumentBean*

L'implémentation de ce diagramme d'activité s'est fait par récursivité de la méthode *CreateDocumentBean* qui prend en paramètre l'objet mappé de la ressource, un Bean Java. Le principe étant d'utiliser les fonctions de réflexion dans Java, pour accéder aux propriétés de la classe de l'objet. Nous récupérons dès lors le type pour déterminer l'entité qui va le représenter. Ainsi, *BeanSimple* va avec les classes de type primitif, *BeanList* va avec les classes qui implémentent l'interface « List », et toutes les autres classes sont représentées par *BeanComposed*. Ensuite, hormis le premier cas, nous récupérons les méthodes getters de la classe, et nous l'invoquons une à une sur l'objet pour récupérer ses attributs. Et pour chacun, nous relançons la méthode. Le point d'arrêt de la récursivité est soit le *BeanSimple*,

soit que l'attribut récupéré de l'objet est éventuellement vide. CreateDocumentBean fait appel à la méthode selectValidGetMethods dans le nœud d'activité « sélectionner les getters valides ». Son diagramme d'activité présenté dans la figure 11 est assez simple. Néanmoins, la méthode se base essentiellement sur la structure standard d'un Bean en Java. Les getters doivent être de la forme « getAttribute » ou « isAttribut ». Si jamais, l'objet mappé passé en paramètre ne respecte pas la structure, une exception est levée et il ne sera pas mappé dans le modèle générique.

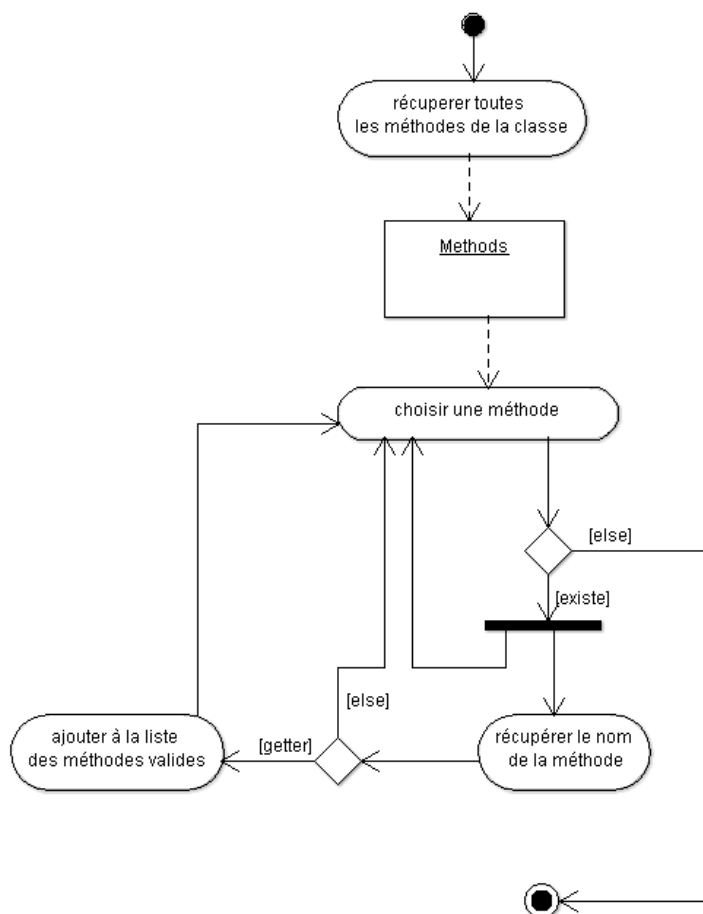


Figure 11: Diagramme d'activité de la méthode selectValidGetMethods

En ce qui concerne l'intégration, nous venons à peine d'ériger les premiers piliers de l'application. Donc tout comme dans la partie des risques, parler d'intégration dans un système inexistant n'as pas de sens. Notons, toute fois, que puisque la couche Mapping n'est pas encore implémentée, ni d'ailleurs la couche DAO, les tests ont été effectués sur l'objet mappée avec Jaxb d'une ressource locale. Ce mapping est effectué au sein de la méthode « getDocumentTest » de la classe BeanServices, en attendant l'implémentation des couches manquantes.

Sur ce, nous donnons dans la figure 12 le résultat obtenu à l'issue de cette itération:

Figure 12: Résultat d'affichage de la première itération

Les nombres entre parenthèses indiquent le nombre des fils contenus dans un BeanList ou BeanComposed. Cette représentation visuelle ne sert pas plus, que de tester l'implémentation de l'itération, d'où l'affichage basique sans fonctionnalité notable.

3.2. Itération 2 : Les fonctionnalités avancées de l'interface

3.2.1. Spécifications

Cette deuxième itération vient élaborer l'implémentation basique de l'interface, faite dans l'itération précédente. Elle a pour rôle de construire une interface offrant des services pratiques de navigation dans le modèle graphique (que nous appellerons désormais, document) et de modification de ses composants. Les ressources pouvant atteindre de très grande taille, les fonctionnalités implémentées doivent permettre à l'utilisateur de mieux se positionner dans le document, et de pouvoir le modifier à sa guise. Nous avons établi deux diagrammes séparés de cas d'utilisation. Le premier décrit la navigation et l'autre la modification.



Figure 13: Diagramme de cas d'utilisation concernant la navigation

Tableau de description du cas d'utilisation « rechercher par élément » :

Tableau 3: Cas d'utilisation "rechercher par élément"

Cas : rechercher par élément
Pré condition : les éléments du document sont affichés
Post condition : les éléments recherchés sont affichés

Scénario principal
<ol style="list-style-type: none"> 1. L'utilisateur remplit le champ de la recherche par élément 2. Le système valide la chaîne de caractère saisie 3. Le système parcourt la racine du document et cache les éléments qui ne correspondent pas à la recherche 4. Le système actualise l'affichage de la fenêtre
Exceptions
<ul style="list-style-type: none"> - La chaîne de caractère saisie comporte des espaces au milieu - Aucun élément ne correspond à la recherche - Le champ de la recherche est vide

Tableau de description du cas d'utilisation « rechercher par contenu » :

Tableau 4: Cas d'utilisation "rechercher par contenu"

Cas : rechercher par contenu
<p>Pré condition : le document contient des BeanSimples</p> <p>Post condition : les éléments recherchés sont affichés</p>
Scénario principal
<ol style="list-style-type: none"> 1. L'utilisateur remplit le champ de la recherche par contenu 2. Le système valide la chaîne de caractère saisie 3. Le système parcourt la racine du document et cache les éléments qui ne correspondent pas à la recherche 4. Le système actualise l'affichage de la fenêtre
Exceptions
<ul style="list-style-type: none"> - La chaîne de caractère ne respecte pas les expressions régulières - Aucun élément ne correspond à la recherche - Le champ de la recherche est vide

Tableau de description du cas d'utilisation « contrôler la profondeur d'affichage » :

Tableau 5: Cas d'utilisation "contrôler la profondeur d'affichage"

Cas : contrôler la profondeur d'affichage
<p>Pré condition : la racine du document est affichée</p> <p>Post condition : les éléments se trouvant au-delà de la profondeur ne sont pas affichés</p>
Scénario principal
<ol style="list-style-type: none"> 1. L'utilisateur clique sur un des boutons de contrôle : incrémenter, décrémenter 2. Le système actualise la valeur de la profondeur dans l'interface 3. Le système parcourt la racine du document et cache les éléments dont la profondeur est supérieure de la profondeur actualisée 4. Le système actualise l'affichage de la fenêtre

Exceptions
<ul style="list-style-type: none"> - La profondeur est négative - La profondeur dépasse la profondeur du document

Tableau de description du cas d'utilisation «afficher certains types d'éléments» :

Tableau 6: *Cas d'utilisation "afficher certains types d'éléments"*

Cas : afficher certains types d'élément
Pré condition : les éléments du document sont affichés Post condition : les éléments de types non sélectionnés ne sont pas affichés
Scénario principal
<ol style="list-style-type: none"> 1. L'utilisateur clique sur les types à afficher parmi : Simple, Attribut, Composé, et Liste 2. Le système parcourt la racine du document et cache les types qui n'ont pas été choisis 3. Le système actualise l'affichage de la fenêtre
Exceptions
aucune

Tableau de description du cas d'utilisation « remonter à la racine » :

Tableau 7: *Cas d'utilisation "remonter à la racine"*

Cas : remonter à la racine
Pré condition : la racine affichée n'est pas la racine du document Post condition : la racine du document est affichée
Scénario principal
<ol style="list-style-type: none"> 1. L'utilisateur clique sur le bouton « racine » 2. Le système change la racine de la fenêtre en la racine du document 3. Le système actualise l'affichage de la fenêtre en respectant la profondeur et les types affichés
Exceptions
aucune

Tableau de description du cas d'utilisation « retourner à l'élément précédent » :

Tableau 8: *Cas d'utilisation "retourner à l'élément précédent"*

Cas : retourner à l'élément précédent
Pré condition : l'élément affiché a été extrait Post condition : l'élément précédent est affiché

Scénario principal
<ol style="list-style-type: none"> 1. L'utilisateur clique sur le bouton « précédent » 2. Le système remplace la racine de la fenêtre par l'élément précédent se trouvant dans l'historique de navigation 3. Le système actualise l'affichage de la fenêtre en respectant la profondeur et les types affichées
Exceptions
<ul style="list-style-type: none"> - L'élément à été éventuellement supprimé

Tableau de description du cas d'utilisation « extraire un élément » :

Tableau 9: Cas d'utilisation "extraire un élément"

Cas : extraire l'élément
<p>Pré condition : l'élément est un élément composé ou une liste</p> <p>Post condition : l'élément est la racine de la fenêtre</p>
Scénario principal
<ol style="list-style-type: none"> 1. L'utilisateur clique sur l'un des boutons suivant : <ul style="list-style-type: none"> - Extraire dans la même fenêtre - Extraire dans une nouvelle fenêtre 2. Le système remplace la racine de la fenêtre par l'élément si 'il est extrait localement, ou crée une nouvelle fenêtre pour l'élément s'il est extrait en dehors. 3. Le système enregistre l'élément extrait dans l'historique de navigation 4. Le système actualise l'affichage de/des fenêtres en respectant la profondeur et les types affichées
Exceptions
<ul style="list-style-type: none"> - L'élément extrait est vide - L'élément extrait est la racine de la fenêtre

Tableau de description du cas d'utilisation « plier déplier » :

Tableau 10: Cas d'utilisation "plier déplier"

Cas : plier/déplier
<p>Pré condition : l'élément affiché comprend des éléments fils</p> <p>Post condition : les fils de l'élément sont cachés/affichés</p>
Scénario principal
<ol style="list-style-type: none"> 1. L'utilisateur clique sur le bouton « plier/déplier » 2. Le système cache les fils s'ils sont affichés et vice versa.
Exceptions
<ul style="list-style-type: none"> - L'élément ne contient aucun fils

S'en suit le diagramme de cas d'utilisation concernant la modification :

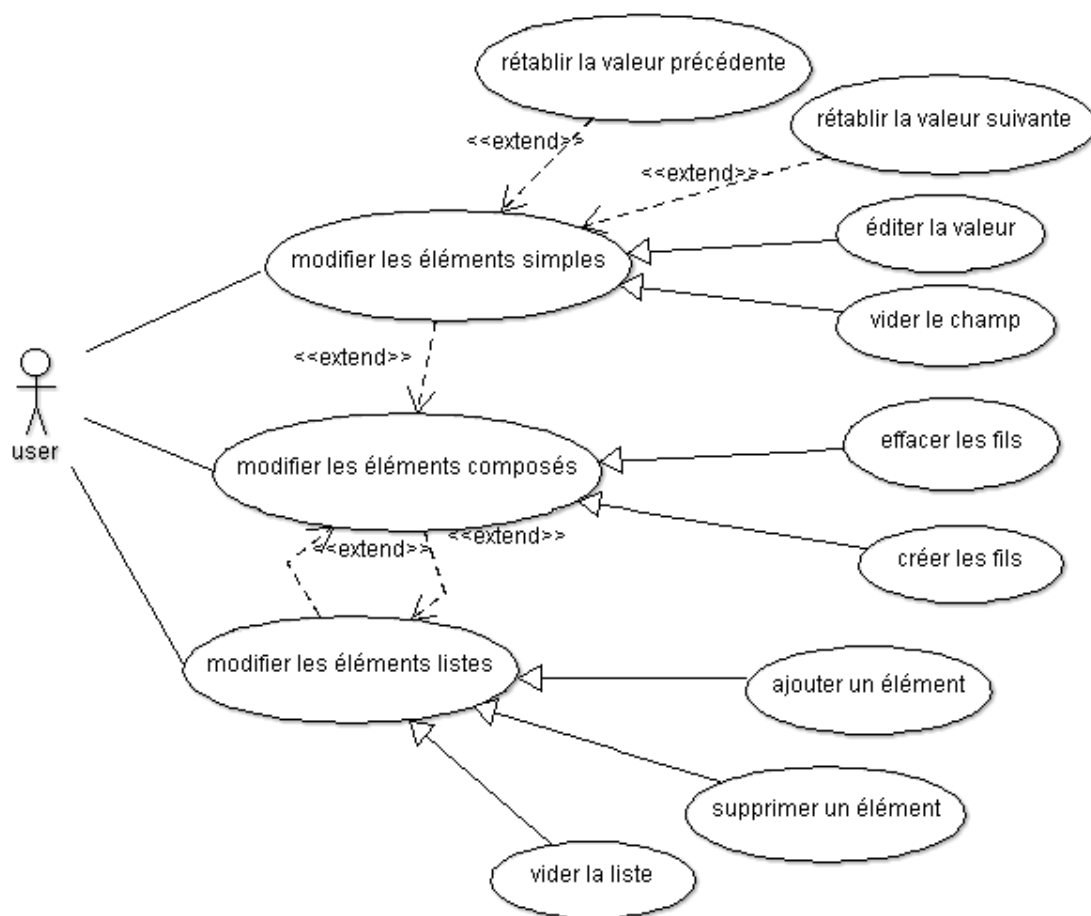


Figure 14: Diagramme de cas d'utilisation concernant la modification

Tableau de description du cas d'utilisation « rétablir la valeur précédente » :

Tableau 11: Cas d'utilisation "rétablir la valeur précédente"

Cas : rétablir la valeur précédente
Pré condition : l'élément affiché est un simple ou un attribut Post condition : la valeur précédente est rétablie
Scénario principal
1. L'utilisateur clique sur le bouton « valeur précédente » 2. Le système remplace la valeur du champ par la valeur précédente de l'historique d'édition 3. Le système actualise l'affichage de l'élément
Exceptions
- L'historique est vide - On ne peut plus revenir en arrière dans l'historique

Tableau de description du cas d'utilisation « rétablir la valeur suivante » :

Tableau 12: Cas d'utilisation "rétablir la valeur suivante"

Cas : rétablir la valeur suivante
Pré condition : l'élément affiché est un simple ou un attribut Post condition : la valeur suivante est rétablie
Scénario principal
1. L'utilisateur clique sur le bouton « valeur suivante » 2. Le système remplace la valeur du champ par la valeur suivante dans l'historique d'édition 3. Le système actualise l'affichage de l'élément
Exceptions
- L'historique est vide - On ne peut plus avancer dans l'historique

Tableau de description du cas d'utilisation « éditer la valeur » :

Tableau 13: Cas d'utilisation "éditer la valeur"

Cas : éditer la valeur
Pré condition : l'élément affiché est un simple ou un attribut Post condition : les fils de l'élément sont cachés/affichés
Scénario principal
1. L'utilisateur clique sur le bouton « éditer » ou double clique sur le champ de valeur 2. L'utilisateur édite la valeur ou saisie une nouvelle 3. Le système valide la valeur saisie selon le type 4. Le système enregistre la valeur dans l'historique 5. Le système actualise l'affichage de l'élément simple
Exceptions
- La valeur saisie est vide - La valeur saisie n'est pas conforme au type qui doit être saisi - La valeur saisie est la même que la valeur précédente

Tableau de description du cas d'utilisation « vider le champ » :

Tableau 14: Cas d'utilisation "vider le champ"

Cas : vider le champ
Pré condition : l'élément affiché est un simple ou un attribut Post condition : le champ de valeur est vide

Scénario principal
<ol style="list-style-type: none"> 1. L'utilisateur clique sur le bouton « vider » 2. Le système vide le champ de valeur et actualise l'affichage de l'élément
Exceptions
<ul style="list-style-type: none"> - L'élément est en édition

Tableau de description du cas d'utilisation « supprimer les fils de l'élément » :

Tableau 15: *Cas d'utilisation "supprimer les fils de l'élément"*

Cas : vider l'élément de ses fils
<p>Pré condition : l'élément affiché est un composite</p> <p>Post condition : l'élément est vide de ses fils</p>
Scénario principal
<ol style="list-style-type: none"> 1. L'utilisateur clique sur le bouton « supprimer les fils de l'élément » 2. Le système demande confirmation 3. L'utilisateur confirme 4. Le système supprime tous le contenu de l'élément 5. Le système actualiser l'affichage de l'élément
Exceptions
<ul style="list-style-type: none"> - L'utilisateur n'a pas confirmé - L'élément est vide

Tableau de description du cas d'utilisation « créer les fils » :

Tableau 16: *Cas d'utilisation "créer les fils"*

Cas : créer les fils
<p>Pré condition : l'élément affiché est de type Composé</p> <p>Post condition : les fils de l'élément sont créés</p>
Scénario principal
<ol style="list-style-type: none"> 1. L'utilisateur clique sur le bouton « créer le fils de l'élément » 2. Le système parcourt la structure de l'élément et crée les éléments fils sans valeur, et les ajoutent à l'élément parent 3. Le système actualise l'affichage des éléments créés
Exceptions
<ul style="list-style-type: none"> - La structure n'est pas trouvée - La création d'un des éléments fils a échoué - L'élément n'est pas vide

Tableau de description du cas d'utilisation « ajouter un élément » :

Tableau 17: Cas d'utilisation "ajouter un élément"

Cas : ajouter un élément
Pré condition : l'élément affiché est de type Liste Post condition : Un nouvel élément est ajouté à la liste
Scénario principal
1. L'utilisateur clique sur le bouton « ajouter un élément » 2. Le système parcourt la structure de l'élément, le crée sans valeur et l'ajoute à la liste 3. Le système actualise l'affichage de la liste
Exceptions
- La structure n'est pas trouvée - La création d'un des éléments fils a échoué dans le cas d'un élément composé

Tableau de description du cas d'utilisation « supprimer un élément » :

Tableau 18: Cas d'utilisation "supprimer un élément"

Cas : supprimer un élément
Pré condition : la liste n'est pas vide Post condition : l'élément est supprimé de la liste
Scénario principal
1. L'utilisateur clique sur le bouton « supprimer l'élément de la liste » 2. Le système demande confirmation 3. L'utilisateur confirme 4. Le système supprime l'élément de la liste 5. Le système actualise l'affichage de la liste
Exceptions
- L'utilisateur n'a pas confirmé - L'élément n'existe pas dans la liste

Tableau de description du cas d'utilisation vider la liste:

Tableau 19: Cas d'utilisation "vider la liste"

Cas : vider la liste
Pré condition : l'élément affiché est de type Liste Post condition : l'élément est vide

Scénario principal	
<ol style="list-style-type: none"> 1. L'utilisateur clique sur le bouton « vider la liste » 2. Le système demande confirmation 3. L'utilisateur confirme 4. Le système supprime tous les éléments de la liste 5. Le système actualiser l'affichage de la liste 	
Exceptions	
<ul style="list-style-type: none"> - L'utilisateur n'a pas confirmé 	

3.2.2. Risques d'intégration

Avec le nombre important de manipulations qui doivent se faire sur le document. Quelques changements doivent être appliqués au niveau des couches existantes pour supporter les nouvelles fonctionnalités. Le tableau 20 montre les risques encourus suite aux changements.

Tableau 20: Tableau des risques de la deuxième itération

Couche	Classe	Changement	Risque
GenericMapping	FactoryBean	Ajout de nouvelles méthodes pour créer des entités vide de valeurs	Pas de risque sur les méthodes existantes
	BeanEntity	Ajout d'un nouvel attribut sur l'état de l'entité	L'attribut n'est pas intégré dans la génération du modèle générique
Services	BeanServices	Ajout de services pour la récupération des entités vides de valeurs	Les services ne sont pas implémentés dans l'interface
Display	Displayer	Ajout de nouveaux paramètres pour les fonctionnalités d'affichage	L'affichage n'est pas correct
	BeanComposed BeanSimple BeanList	Ajout des fonctionnalités des cas d'utilisation	L'affichage n'est pas correct

3.2.3. Conception de l'itération

La couche Display est la couche la plus concernée. Chaque composant d'entité doit, en fait, déclarer des méthodes et des attributs nouveaux pour répondre aux cas d'utilisations. Et compte tenu de leur nombre, cela signifie que les classes du diagramme de la couche Display

vont connaître de grands ajouts. Nous rassemblons tous les changements appliqués à la couche Display dans diagramme de classes de la figure 15. Cependant nous faisons épargne des ajouts concernant les composants graphiques tels que les boutons, les listes, et les images.

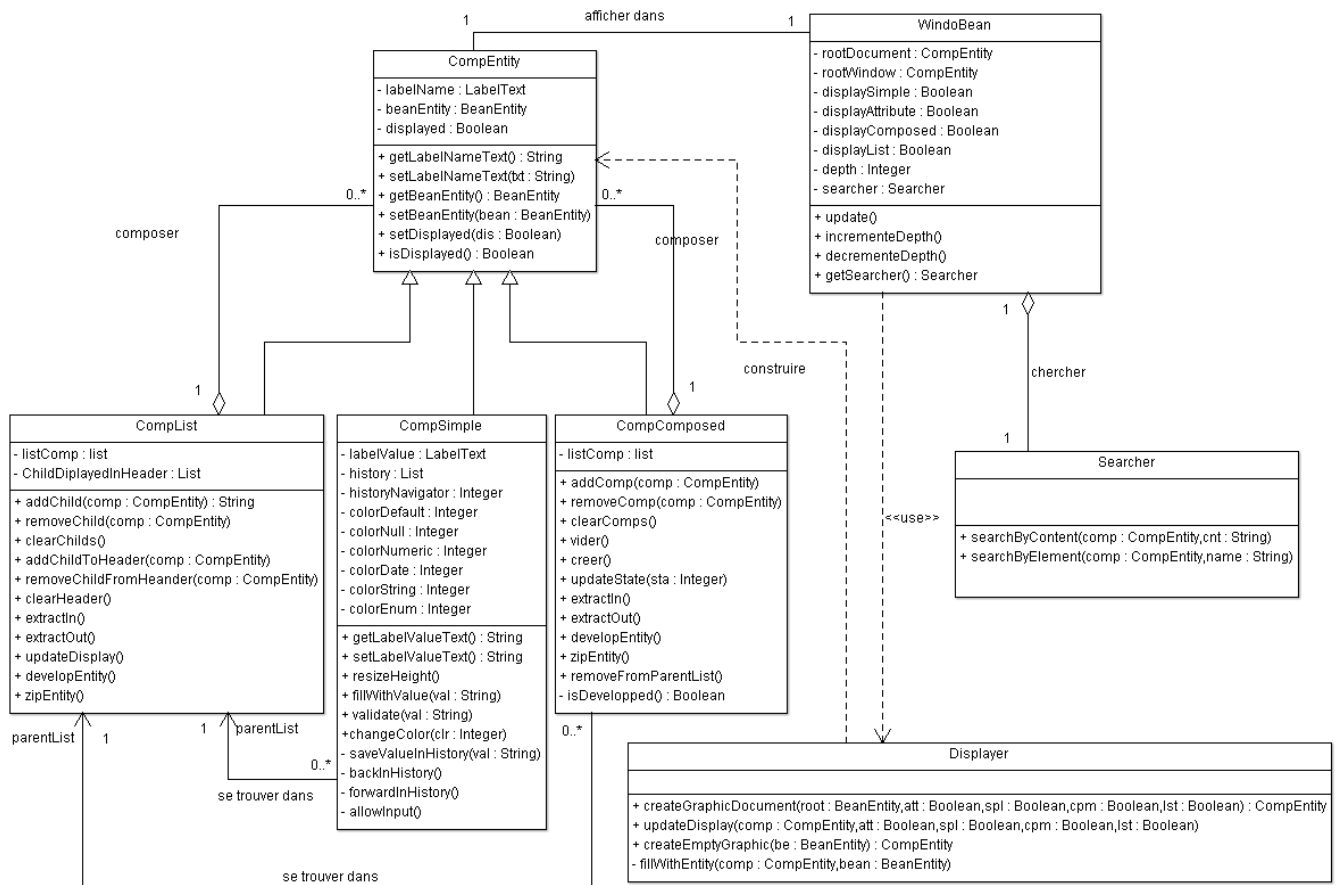


Figure 15: Diagramme actualisé de la couche Display

Remarquons qu'en plus des ajouts d'attributs et de méthodes dans les classes existantes, d'autres relations ont été créées entre elles. L'association uni sens « se trouver dans » liant le CompComposed et CompSimple à CompList, a été ajoutée pour répondre à la fonctionnalité de suppression d'élément dans une liste. La suppression ne pouvant être faite, en effet, qu'au niveau de cette dernière. D'un autre point de vue, les fonctionnalités applicables sur le document (modèle graphique) ont été groupées dans la classe WindowBean aidée par Displayer et Searcher. Notamment la sélection des types, le contrôle de profondeur, l'actualisation et la recherche. Les autres fonctionnalités ont été réparties sur les entités. Les fonctions de modification dans chaque entité actualisent simultanément le modèle graphique et le modèle générique, grâce à la référence de chaque entité générique dans l'entité graphique correspondante (liaison entre BeanEntity et CompEntity).

Plus particulièrement, les fonctions d'ajout dans les entités de type `CompList`, et de création pour les entités de type `CompComposed`, font appel au nouveau service de la classe `BeanServices` « `getEmptyEntity (className : String)` », qui fait lui-même appel à la nouvelle méthode « `createEmptyEntity (className : String)` » de la classe `FactoryBean`. Ces deux dernières méthodes étant les seuls changements dans les couches `GenericMapping` et `BeanServices`, nous n'avons pas repris leurs diagrammes de classe dans cette partie. Par contre, elles sont présentées dans le diagramme de séquences concernant la méthode « `addChild` » (figure 16), de la classe `CompList`, qui ajoute un élément à la liste. La méthode « `create` » de la classe `CompComposed` adopte le même principe, elle n'aura donc pas de diagramme de séquence.

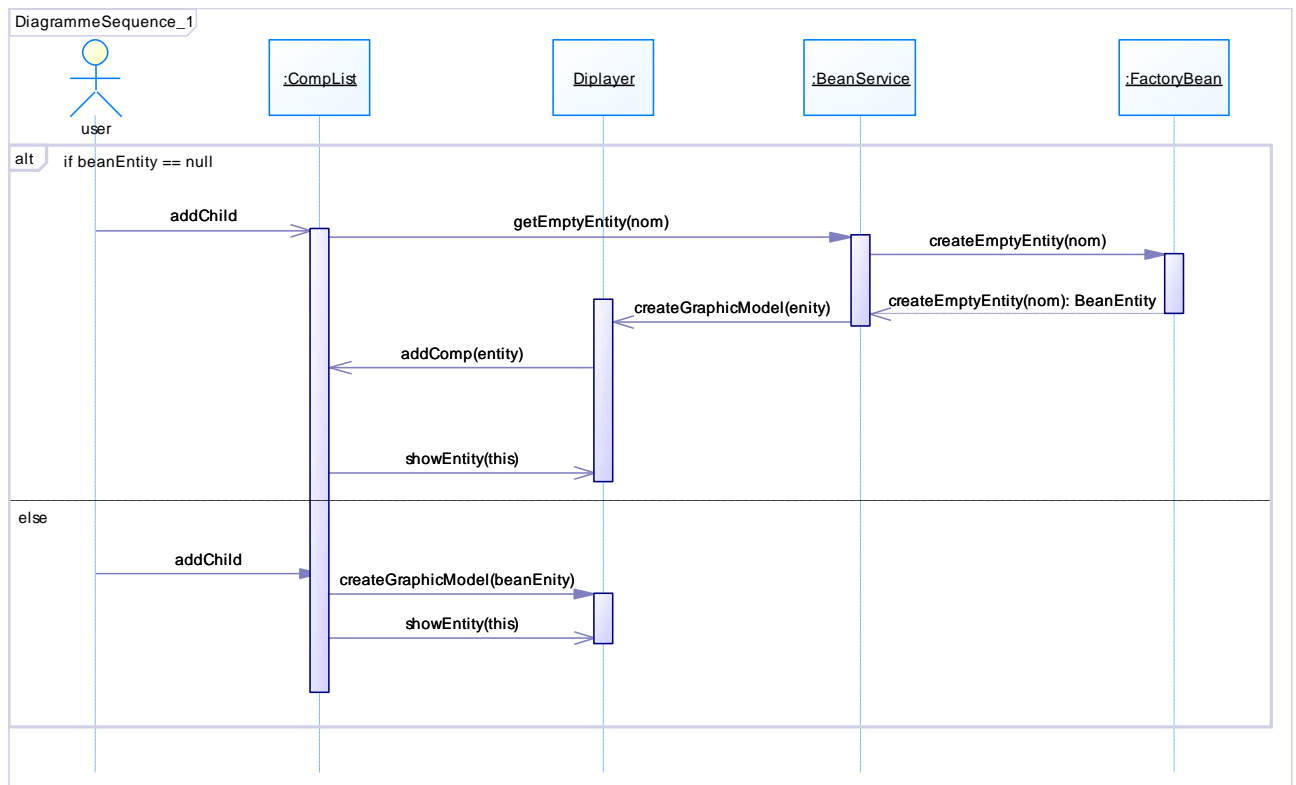


Figure 16: Diagramme de séquence pour la méthode "addChild"

Le but de ce diagramme est de montrer l'aspect d'optimisation de la méthode. En effet, pour ajouter un élément à la liste, l'interface RIA (Rich Internet Application) fait appel au service distant « `getEmptyEntity` » pour récupérer l'entité générique dans l'attribut de classe « `beanEntity` », et générer le modèle graphique correspondant. Cependant, elle ne les lie pas. Elle enregistre l'entité générique, en crée une copie et la lie avec le modèle. La prochaine fois que l'ajout d'un élément doit être fait, elle recopie l'entité déjà enregistrée.

3.2.4. Réalisation et intégration

L'implémentation de cette itération a été réalisée en grande partie, sous Flex/ActionScript. L'utilisation de la notion du « Binding » a grandement facilité le développement des interfaces, puisqu'elle permet de lier les valeurs d'un composant graphique avec des variables d'ActionScript, séparant ainsi le code entre les deux.

D'autre part, le tableau des risques établi en début de l'itération a été repris pour orienter les tests unitaires et les tests d'intégration. Les problèmes et incohérences liés à l'affichage ont été les plus persistants, du fait que chaque fonctionnalité ajoutée modifiait la structure document affichée. Nous avons mis quelques captures d'écrans pour montrer les plus importantes modifications de l'interface.

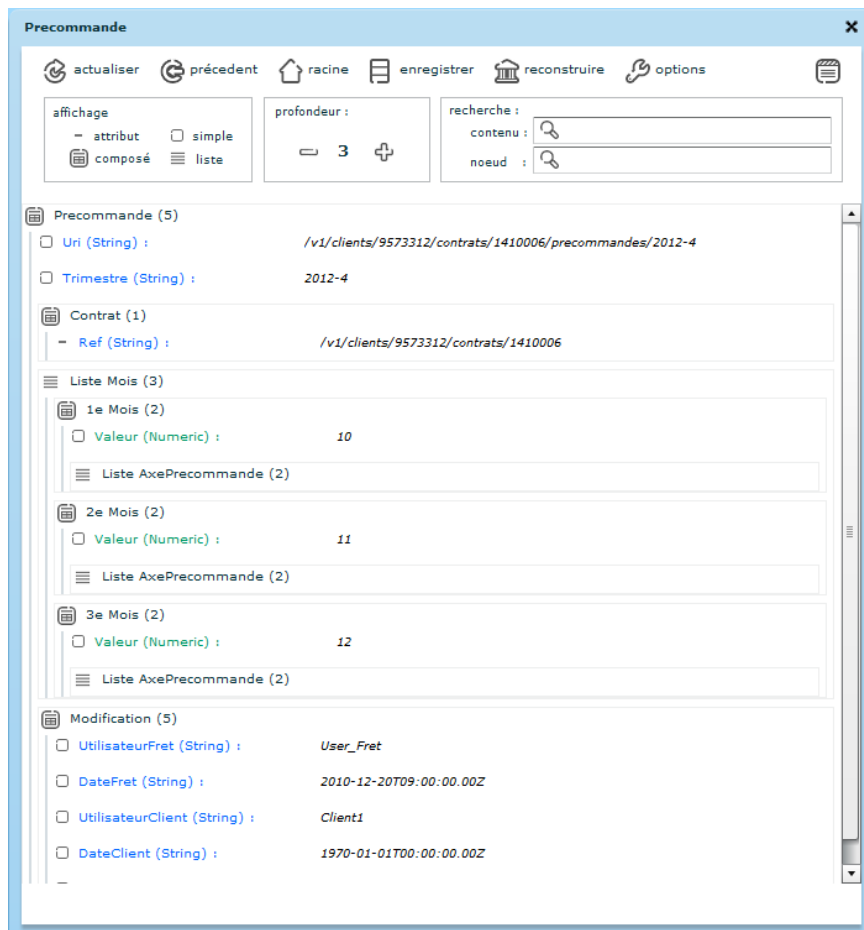


Figure 17: Changements dans l'interface WindowBean

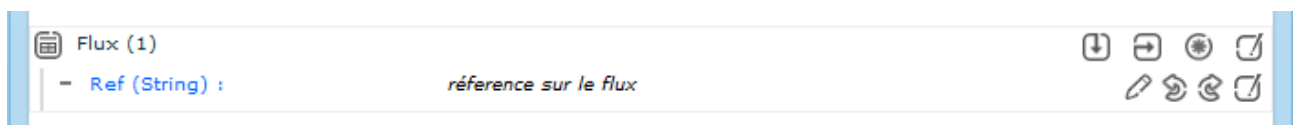


Figure 18: Boutons de modification

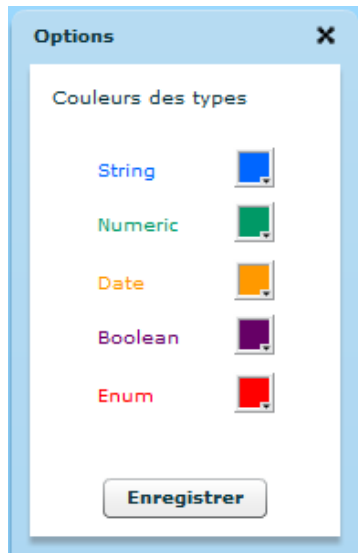


Figure 19: Options de couleurs

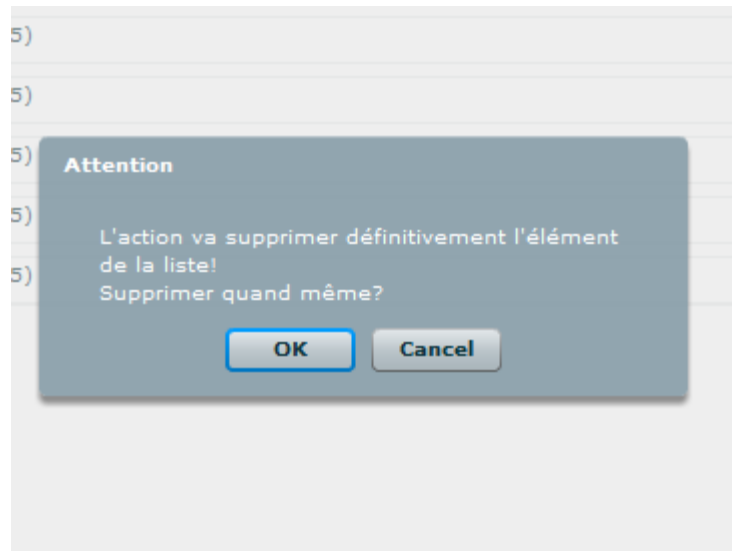


Figure 20: Demande de confirmation lors d'une suppression d'élément

La figure 18 montre la représentation visuelle d'une entité « Flux » de type `CompComposed`, ne contenant qu'un seul élément « Ref » de type `CompSimple`. Lorsque l'utilisateur survole le composant, des boutons d'action propres à chaque type s'affichent à droite.

La représentation visuelle s'appuie sur la couleur, comme critère supplémentaire, pour mieux distinguer entre les types des valeurs, et faciliter la lecture du document. La couleur peut être modifiée au gré de l'utilisateur dans la fenêtre des options (figure 19), et la modification s'applique instantanément sur l'affichage.

La figure 20 expose une alerte affichée, suite à une action de suppression d'un élément d'une liste. L'alerte prévient l'utilisateur de l'irréversibilité de l'action, et demande confirmation avant de supprimer. Cette alerte pourrait devenir obsolète, si un système d'historique, tout comme celui des valeurs, est implémenté. Ce point peut, effectivement, être classé comme perspective intéressante.

3.3. Itération 3 : Les modifications sur le Bean mappé

3.3.1. Spécifications

Ça pourrait sembler déséquilibré, de réserver toute une itération à une seule fonctionnalité ou deux. Et il est vrai que cette itération soit la plus courte parmi toutes les autres, en termes de développement. Mais que ça ne soit pas un facteur de jugement sur son importance dans le système. En effet, l'application des modifications sur le Bean généré (mappé) est l'une des fonctionnalités majeures de l'application. Plus délicate que la génération du modèle générique, et clairement plus difficile à implémenter. Cependant, l'itération n'apporte que deux cas d'utilisations supplémentaires présentés dans le diagramme de la figure 21.

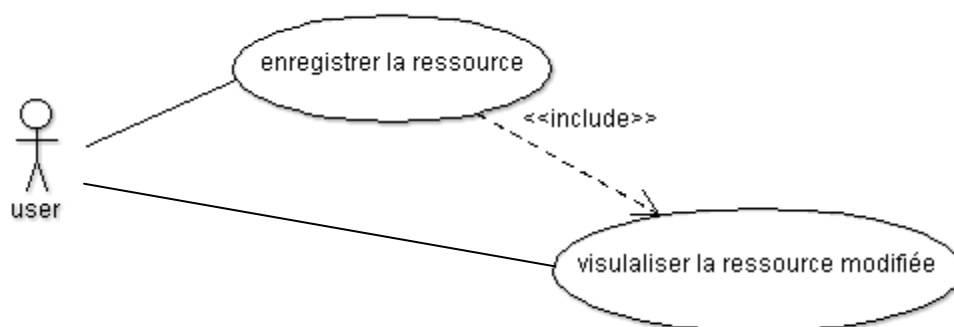


Figure 21: Diagramme de cas d'utilisation de la troisième itération

Tableau de description du cas « enregistrer la ressource » :

Tableau 21: Cas d'utilisation "enregistrer la ressource"

Cas : enregistrer la ressource
Pré condition : les modifications sont visualisées Post condition : les modifications sont enregistrées dans la ressource
Scénario principal
1. L'utilisateur clique sur le bouton « enregistrer » 2. Le système reconvertit l'objet mappé en XML et le passe au web service d'enregistrement
Exceptions
- Les modifications ne sont pas appliquées - Le web service n'est pas accessible

Tableau de description du cas « visualiser la ressource modifiée » :

Tableau 22: Cas d'utilisation "visualiser la ressource modifiée"

Cas : visualiser la ressource modifiée
Pré condition : le document est affiché Post condition : le document modifié est affiché sous format texte
Scénario principal
3. L'utilisateur clique sur le bouton « visualiser la ressource modifiée » 4. Le système applique les modifications sur l'objet mappé de la ressource, et l'affiche sous format texte dans une nouvelle fenêtre
Exceptions
<ul style="list-style-type: none"> - Les modifications ne sont pas appliquées - Le texte affiché est plus long que l'espace d'affichage

Avec ces cas d'utilisation, de nouvelles informations vont être échangées entre les trois couches implémentées. La couche Display devra envoyer le modèle générique modifié localement, afin que la couche GenericMapping puisse modifier l'objet représentant la ressource. Nous mettons à jour le diagramme de flux d'information pour couvrir les nouveaux cas d'utilisation :

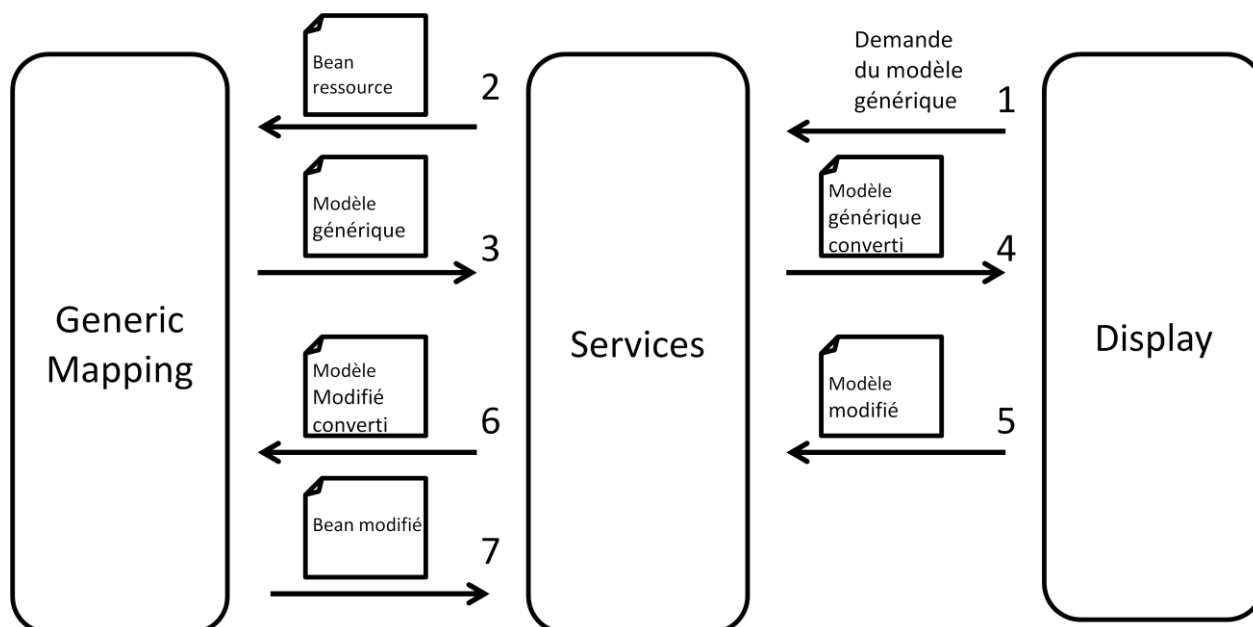


Figure 22: Diagramme de flux d'information de l'itération 3

3.3.2. Risques d'intégration

L'objet mappé qui représente la ressource demeure unique pendant tous le processus de manipulation, depuis la génération de son modèle générique jusqu'à la fermeture de l'application ou l'ouverture d'une nouvelle ressource. C'est une contrainte qui peut être allégée par la sauvegarde d'une copie, ou d'un historique de modification propre au Bean, bien que cela s'avérerait être plus difficile. Mais le problème réside surtout dans la synchronisation entre le Bean et son modèle générique. Une mauvaise synchronisation et l'application perd toute son utilité, et peut devenir contre productive. L'utilisateur pourrait endommager, de manière non intentionnée, une ressource mal saisie au lieu de la corriger. Le tableau 23 présente les risques qui peuvent découler de la modification du Bean mappé :

Tableau 23: *Tableau des risques de la troisième itération*

Couche	Classe	Changement	Risque
GenericMapping	FactoryBean	Ajout de nouvelles méthodes pour la synchronisation entre le Bean et son modèle générique	Le Bean est endommagé Le modèle graphique n'est pas synchronisé avec le Bean et toute l'application est inutile
Services	BeanServices	Ajout de services pour la modification du Bean	Les services ne sont pas implémentés dans l'interface
Display	Displayer	Ajout des fonctionnalités des cas d'utilisation	L'affichage n'est pas correct

3.3.3. Conception de l'itération

Les nouveaux cas d'utilisation ne nécessitent aucune nouvelle classe, par contre, de nouvelles méthodes sont à déclarer dans les anciennes. Plus précisément, la classe FactoryBean ajoutera la méthode qui synchronise le Bean avec l'entité générique « alignObjectWithEntity », et la classe BeanServices ajoutera les méthodes « getModifiedDocument » et « saveDocument », pour la visualisation et l'enregistrement. Nous les mettons en interaction dans le diagramme de séquence de la figure 23.

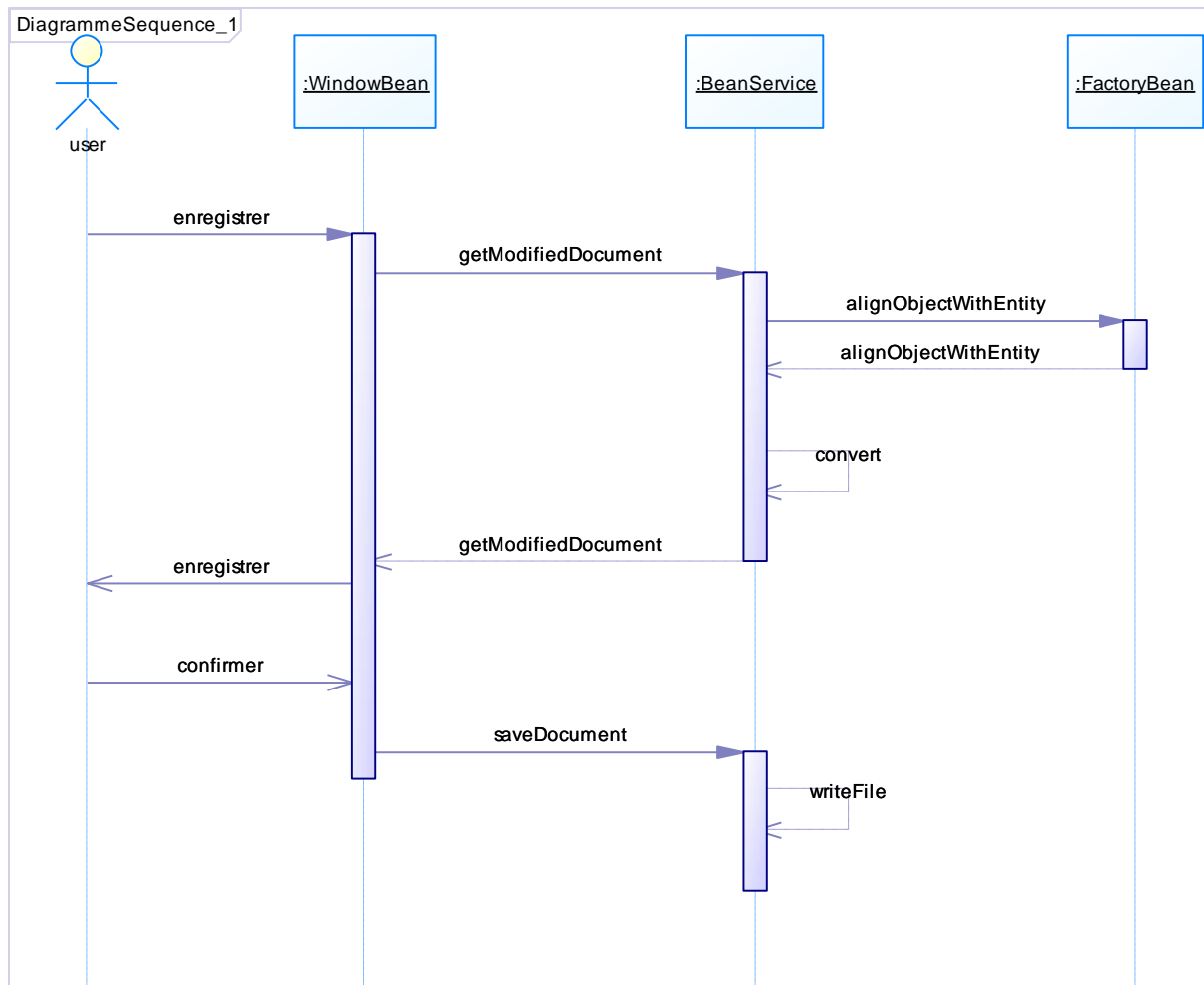


Figure 23: Diagramme de séquence de la fonctionnalité d'enregistrement

Les couches DAO et Mapping n'étant pas encore implémentées, la conversion de l'objet Bean en XML est faite au sein de la classe BeanServices. Quant à l'enregistrement, il se retient à la ressource locale. À vrai dire, le terme « enregistrer » ici fait plutôt allusion à la création d'un nouveau fichier écrasant l'ancien. Notons d'ailleurs, que le plus grand traitement a été fait au niveau de la méthode « alignObjectWithEntity » que nous décrivons par le diagramme d'activité de la figure 24 :

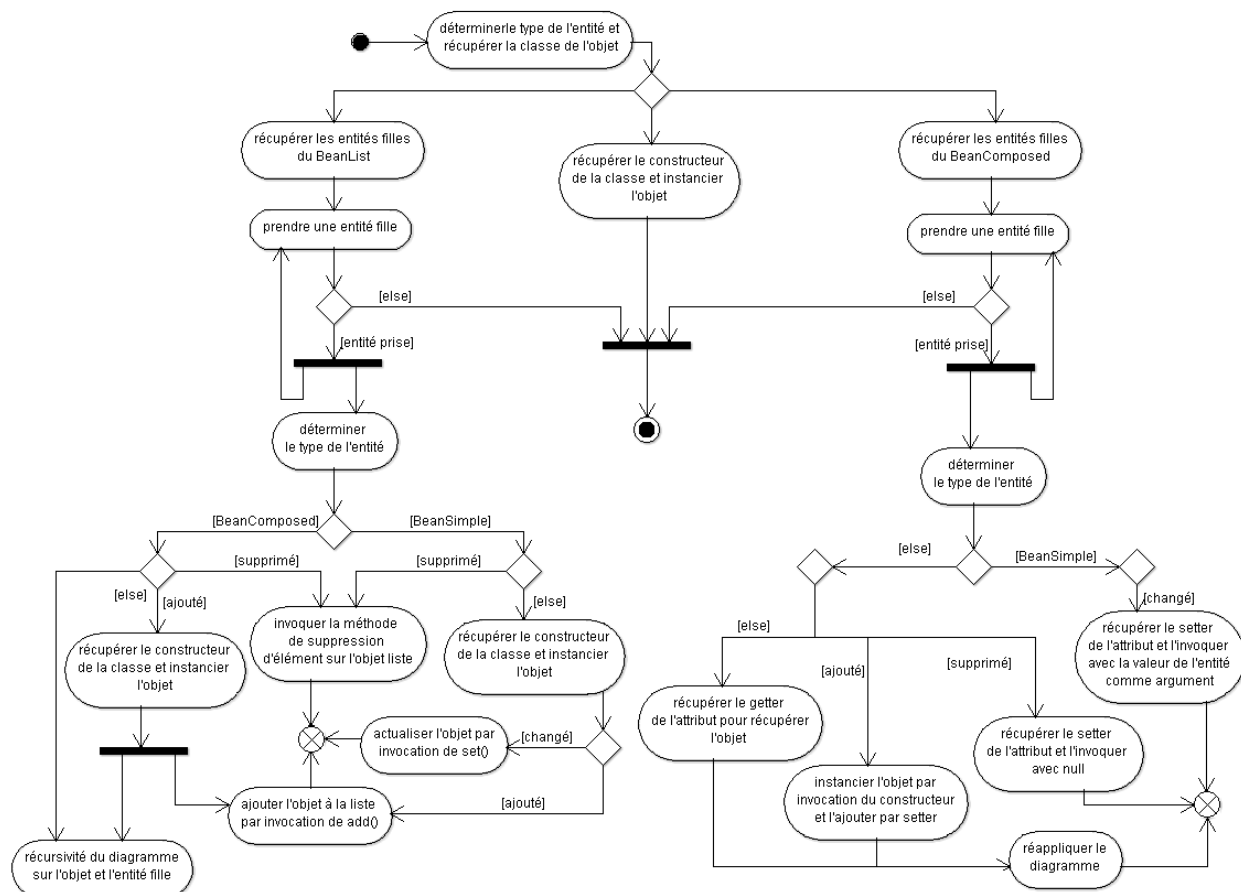


Figure 24: Diagramme d'activité de synchronisation

Le principe de cette synchronisation s'appuie, en fait, sur les états des entités, créées spécialement à cet effet. Entre l'état « supprimé », « ajouté », « changé », et « inchangé », la méthode sait quel objet doit être enlevé du Bean mappé, et quel autre doit être instancié ou changé. Précisons toute fois, que les états « changé » et « inchangé » ne concernent que l'entité BeanSimple, et renseignent en effet sur l'état de sa valeur. Ici, la méthode fait la différence entre les BeanComposed et les BeanList, bien que leurs traitements suivent la même logique. Ceci est du à l'implémentation différente des listes, pour l'ajout et la suppression des éléments.

3.3.4. Réalisation et intégration

L'implémentation de cette itération a été une preuve supplémentaire du constat fait sur la difficulté de travailler avec la réflexion dans Java. Nous avons eu beaucoup de mal à tracer certaines erreurs dues à une mauvaise utilisation d'une méthode, à une mal compréhension du chargement des classes Java, ou mal connaissance des subtilités de Java en général. Cependant, l'implémentation a été réalisée, et nous avons pu visualiser la ressource modifiée.

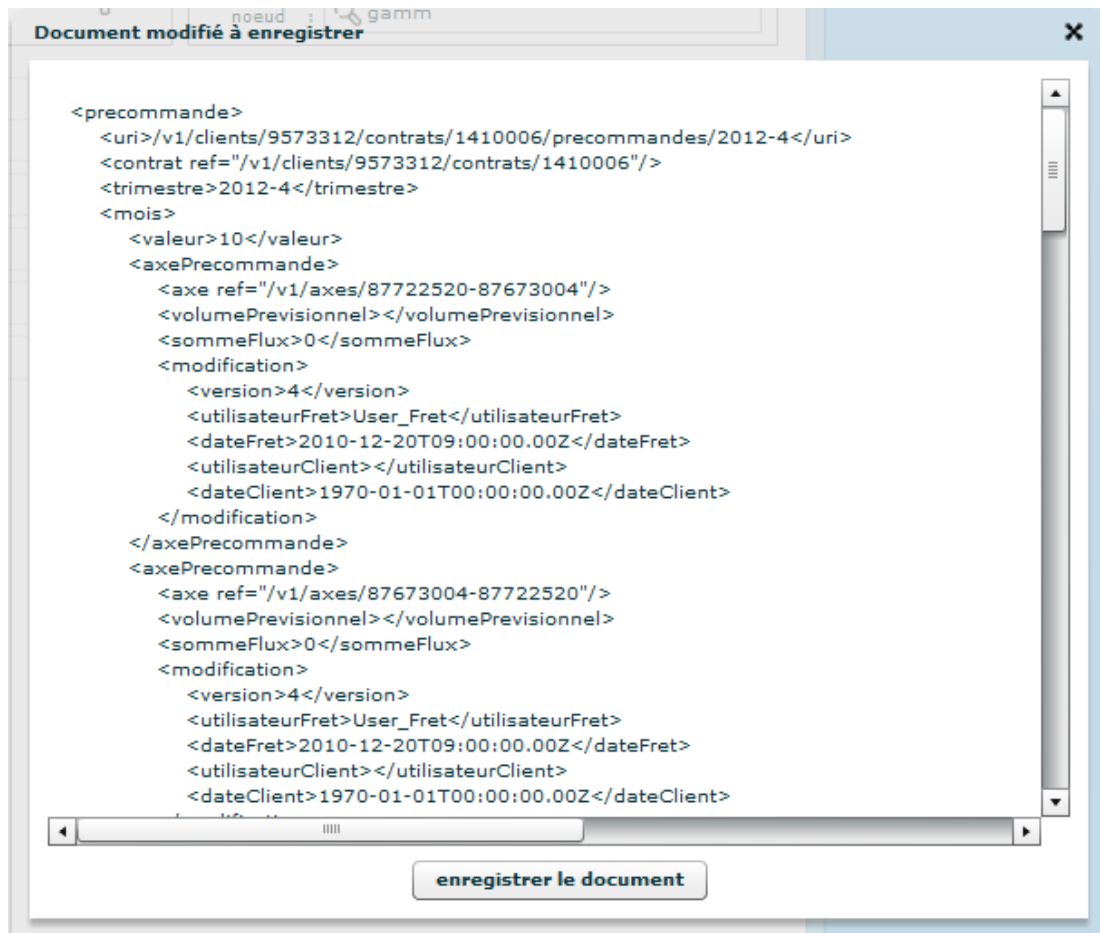


Figure 25: Visualisation de la ressource modifiée

3.4. Itération 4 : La gestion des sources de données

3.4.1. Spécifications

Avant d'entamer l'itération, rappelons d'abord la raison pour laquelle la gestion des sources a été abordée vers la fin de cycle de développement. L'accès aux environnements, en effet, ne pouvait pas être établi, pour des raisons de sécurité. Ceci aurait empêché d'effectuer les tests, et l'itération serait restée suspendue. L'accès étant maintenant octroyé, nous pouvons, désormais, nous lancer dans l'implémentation.

La gestion des sources de données portera sur les différents types d'environnements, où les ressources sont sauvegardées. Par appel au web service, requête dans une base de données, ou recherche dans un système de fichiers, les ressources ne manquent pas chemin d'accès. Et l'application devra supporter la gestion de chaque type. En d'autres termes, l'application devra répondre aux cas d'utilisation de la figure 26:

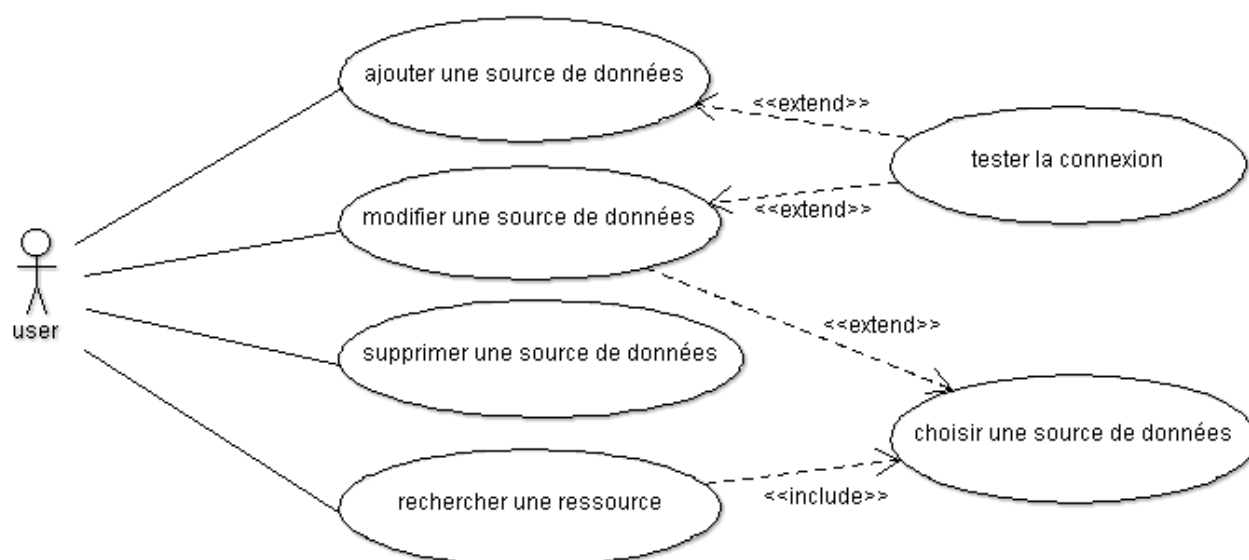


Figure 26: Diagramme de cas d'utilisation de la gestion des sources de données

Tableau de description du cas « ajouter une source de données » :

Tableau 24: Cas d'utilisation "ajouter une source de données"

Cas : ajouter une source de données
Pré condition : La source de données n'existe pas Post condition : La source de données existe
Scénario principal
1. L'utilisateur clique sur le bouton « ajouter » 2. Le système ouvre une nouvelle fenêtre d'ajout 3. L'utilisateur saisit les informations de la source de données 4. Le système valide les informations saisies, ajoute la source à la liste et l'enregistre 5. Le système ferme la fenêtre d'ajout et actualise l'affichage de la liste
Exceptions
- Le nom de la source existe déjà dans la liste - L'enregistrement de la nouvelle liste a échoué

Tableau de description du cas « modifier une source de données » :

Tableau 25: Cas d'utilisation "modifier une source de données"

Cas : modifier une source de données
Pré condition : La source de données existe dans la liste Post condition : La source de données est modifiée

Scénario principal
<ol style="list-style-type: none"> 1. L'utilisateur clique sur le bouton « modifier » 2. Le système ouvre une nouvelle fenêtre de modification avec les informations de la source 3. L'utilisateur modifie les informations 4. Le système valide les informations saisies, change la source dans la liste, et l'enregistre 5. Le système ferme la fenêtre de modification et actualise l'affichage de la liste
Exceptions
<ul style="list-style-type: none"> - Le nom de la source existe déjà dans la liste - L'enregistrement de la nouvelle liste a échoué

Tableau de description du cas « supprimer une source de données » :

Tableau 26: Cas d'utilisation "supprimer une source de données"

Cas : supprimer une source de données
Pré condition : La source de données existe Post condition : La source de données est supprimée
Scénario principal
<ol style="list-style-type: none"> 1. L'utilisateur choisit une source dans la liste et clique sur le bouton « supprimer » 2. Le système demande confirmation de suppression 3. L'utilisateur confirme la suppression 4. Le système supprime la source de la liste, enregistre la nouvelle liste modifiée, et actualise l'affichage
Exceptions
<ul style="list-style-type: none"> - Aucune source n'est choisie dans la liste - L'enregistrement de la nouvelle liste a échoué

Tableau de description du cas « rechercher une ressource » :

Tableau 27: Cas d'utilisation "rechercher une ressource"

Cas : rechercher une ressource
Pré condition : La fenêtre de recherche est ouverte Post condition : Le résultat de la recherche est affiché
Scénario principal
<ol style="list-style-type: none"> 1. L'utilisateur choisit une source de données dans la liste 2. L'utilisateur choisit le type de la ressource à rechercher parmi : Précommande, Commande, et Répartition 3. L'utilisateur remplit les champs de recherche et clique sur le bouton « lancer la recherche » 4. Le système lance la recherche et récupère le résultat 5. Le système affiche le résultat dans la liste de la recherche

Exceptions
<ul style="list-style-type: none"> - Aucune source de données n'est choisie - Aucun type de ressource n'est choisi - Un champ de recherche n'est pas correctement rempli - La source de données n'est pas accessible - Aucun résultat n'est trouvé

3.4.2. Risques d'intégration

Bien que l'axe principal de l'itération soit la gestion des sources de données, son but réel est de récupérer une ressource, où qu'elle se trouve. La couche Services, qui se chargeait auparavant de lire une ressource locale pour effectuer les tests, se verra déléguer cette fonctionnalité d'accès à la couche DAO. Même chose pour la conversion en objet qui sera cédée à la couche Mapping. Ces deux couches, en effet, n'existaient pas dans les itérations précédentes. Et leur implémentation a toutes les chances de compromettre le système. Le tableau 28 liste les risques d'intégration :

Tableau 28: *Tableau des risques d'intégration de la quatrième couche*

Couche	Classe	Changement	Risque
Services	BeanServices	Ajout de services pour la gestion des environnements et la recherche des ressources	Les services ne sont pas implémentés dans l'interface
Display	Displayer	Ajout des fonctionnalités des cas d'utilisation	L'affichage n'est pas correct L'appel aux nouveaux services n'est pas possible
Mapping	#toutes	Création	L'objet mappé ne respecte pas la structure d'un Bean La FactoryBean ne peut pas générer correctement le modèle générique La FactoryBean ne peut synchroniser le Bean avec le modèle générique
DAO	#toutes	Création	La ressource ne peut pas être mappée L'application ne peut pas être utilisée

3.4.3. Conception de l'itération

Dans la conception des couches DAO et Mapping, nous avons veillé à ce que l'architecture puisse être facilement extensible. Cela permettrait d'ajouter d'autres types de source de données, sans devoir toucher à l'implémentation des types déjà existants. Le diagramme de classes de la figure 27 expose l'architecture adoptée :

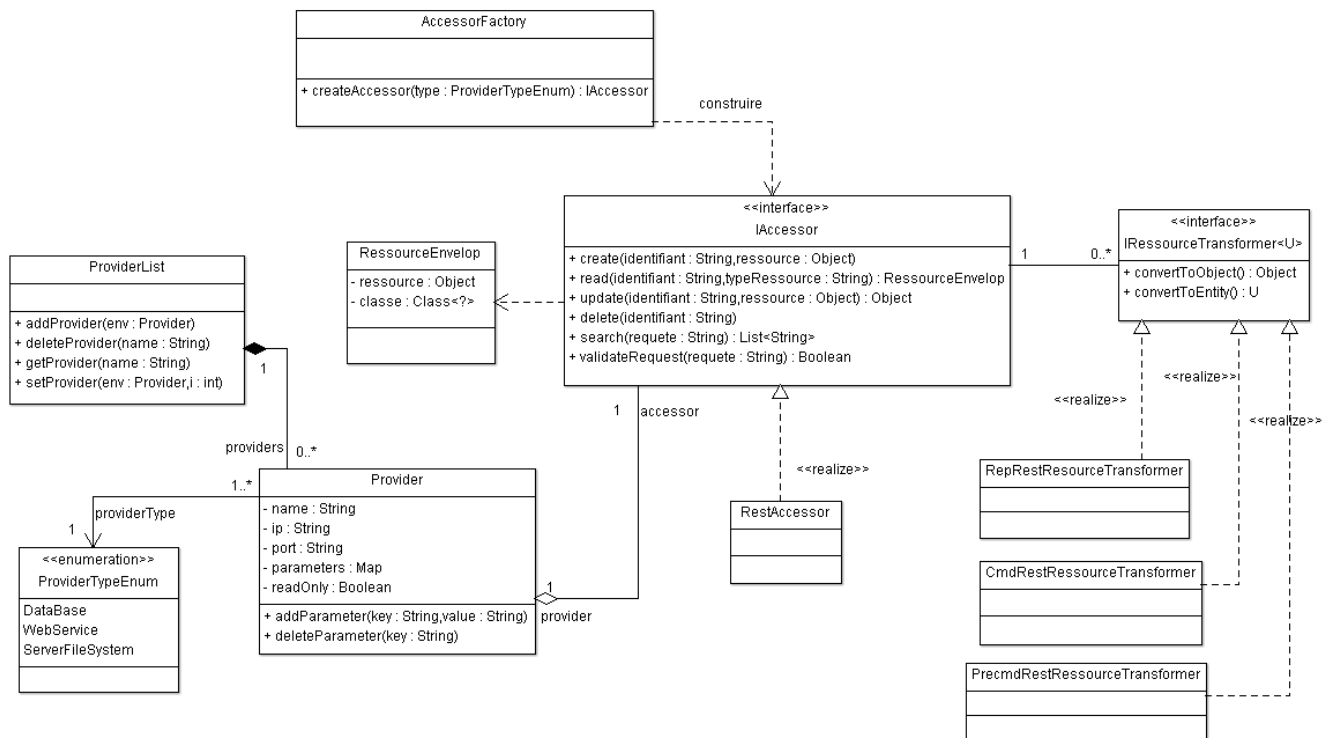


Figure 27: Diagramme de classes de la quatrième itération

Dans le diagramme, la classe `Provider` représente une source de données par un nom, une adresse IP, un port, et des paramètres extensibles sous forme de `key : value`. Les `Providers` doivent porter des noms uniques, c'est leur seul point de distinction. Ils peuvent être de trois types : `DataBase`, `WebService`, et `ServerFileSystem` (énumérés dans `ProviderTypeEnum`), et sont regroupée dans la classe `ProviderList` qui se charge de leur gestion. Pour chaque `Provider`, nous avons besoin d'une classe qui supporte les interactions avec le `Provider`, notamment pour la récupération et l'enregistrement d'une ressource. Ce rôle est joué par les classes qui implémentent l'interface `IAccessor`. Cette dernière propose, en effet, les services de bases qui déterminent le type d'interaction avec le `Provider`, « `create` » pour créer de nouvelles ressources, « `read` » pour la récupération d'une ressource, « `update` » pour la mise à jour d'une ressource suite à des modifications. À noter que chaque type de `Provider` doit avoir son propre `Accessor`, et que les méthodes à définir dans celui-ci doivent prendre en considération les paramètres du `Provider`. La classe `AccessorFactory` se charge de la création des `Accessors`, et est appelée à chaque instantiation d'un `Provider`. En exemple, la classe `RestAccessor` est un `Accessor` pour les provider de type `WebService REST`.

L'interface `IAccessor` est en association avec l'interface `IResourceTransformer`, dans la mesure où celle-ci s'occupe des premiers traitements sur la ressource brute récupérée à partir du `Provider`, car il se peut que la ressource contienne des informations et des ajouts

inutiles pour son traitement. En d'autres termes, le mapping ne doit s'effectuer que sur le contenu utile de la ressource. Le rôle des classes qui implémentent `IRessourceTransformer` est de préparer, d'une part, la ressource à être passée sous forme d'objets à la couche `GenericMapping`, et d'autre part, les objets provenant de ce dernier à être passés à la couche DAO.. Ainsi, les classes `RepRestRessourceTransformer`, `CmdRestRessourceTransformer`, et `PreRessourceTransformer` se chargent respectivement des ressources de type Répartition, Commande, et Précommande. La Figure 28 donne une vue générale du diagramme de classes.

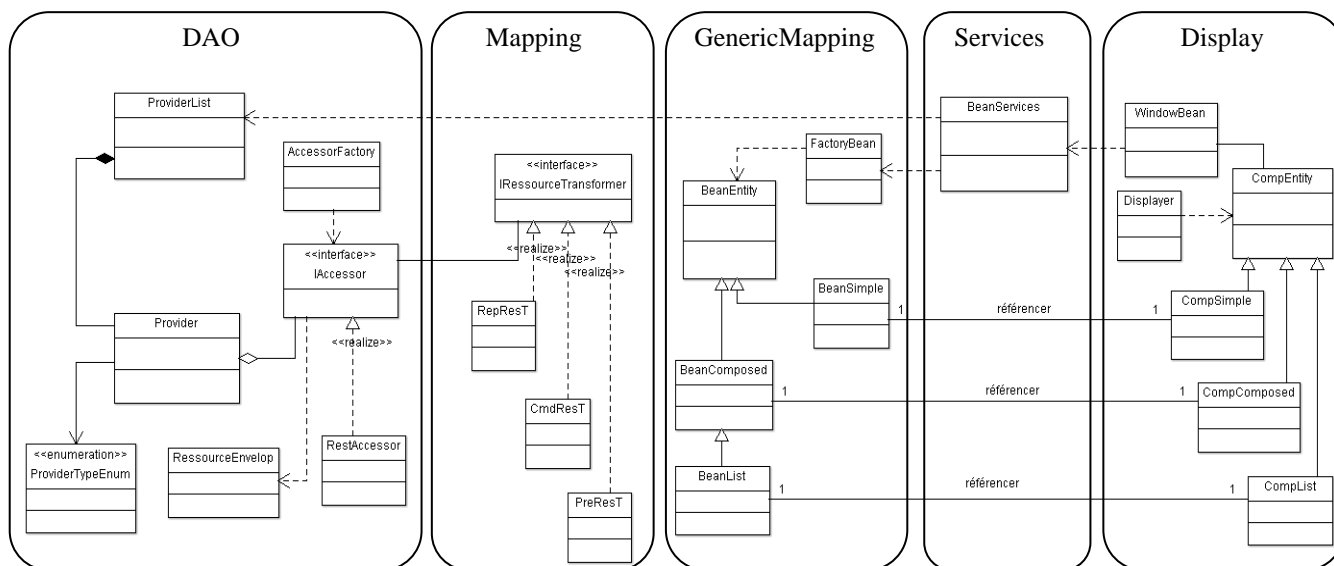


Figure 28: Vue générale sur les classes des couches de l'application

La figure 29 présente le nouveau diagramme de flux d'informations entre couches, enrichi par l'ajout de la couche DAO et Mapping, et complétée par le tableau 29, qui définit les flux numérotés dans le diagramme.

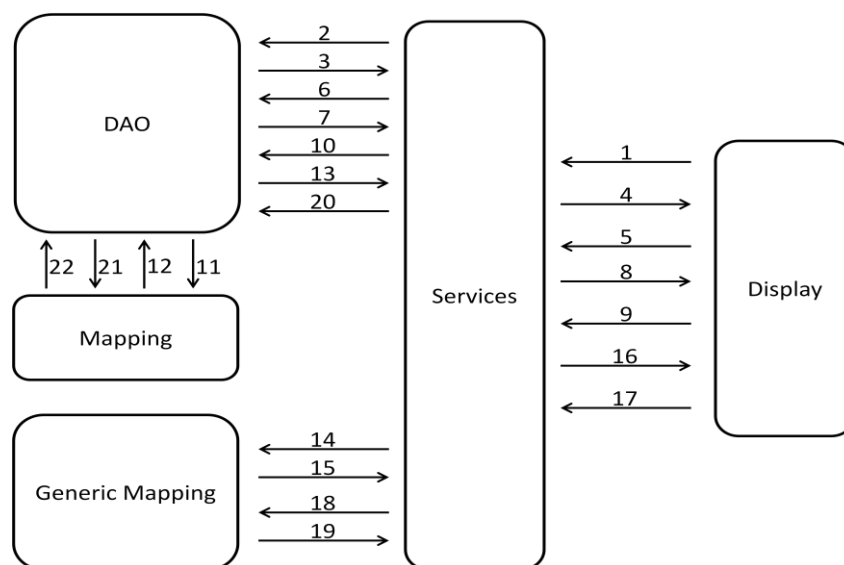


Figure 29: Diagramme de flux d'information entre les couches de l'application

Tableau 29: Légende du diagramme de flux d'information des couches de l'application

Numéro	Flux d'information	Numéro	Flux d'information
1	Demande des sources de données	12	Objet mappé de la ressource
2	Demande de la liste des sources	13	La ressource transformée en objets Java
3	Liste des sources de données	14	Demande de génération du modèle générique
4	Liste des sources convertie	15	Entité générique de l'objet mappé
5	Critères de recherche de ressources	16	Entité générique convertie
6	Demande de recherche	17	Entité générique modifiée
7	Liste des résultats de la recherche	18	Demande de synchronisation
8	Liste des résultats convertie	19	Objet synchronisé avec l'entité
9	URI de la ressource	20	Demande de mapping en XML
10	Demande de récupération de la ressource	21	Demande de génération du contenu XML
11	La ressource	22	La nouvelle ressource modifiée

Remarquons toute fois que l'ordre de positionnement des couches GenericMapping, DAO, et Mapping, est à l'inverse de l'ordre d'appel. L'ordre de positionnement, en effet, fait référence aux passages de transformation d'une ressource depuis sa source, jusqu'à l'affichage dans l'interface RIA.

3.4.4. Réalisation et intégration

Contrairement aux itérations précédentes, qui n'avaient nécessité que la création d'une seule interface. Dans cette quatrième itération, nous avons créé une interface pour l'affichage des sources de données, une interface pour leur gestion, et une pour la recherche des ressources. Nous avons, ensuite, adapté l'interface d'affichage de ressource (WindowBean) pour intégrer l'enregistrement, cette fois dans le Provider. Nous avons aussi implémenté, à la demande du client, la lecture sans possibilité de modification, pour les Providers en mode lecture seule. Aussi, nous avons ajouté les services des nouvelles fonctionnalités dans la classe BeanServices, et leurs appels au niveau des interfaces.

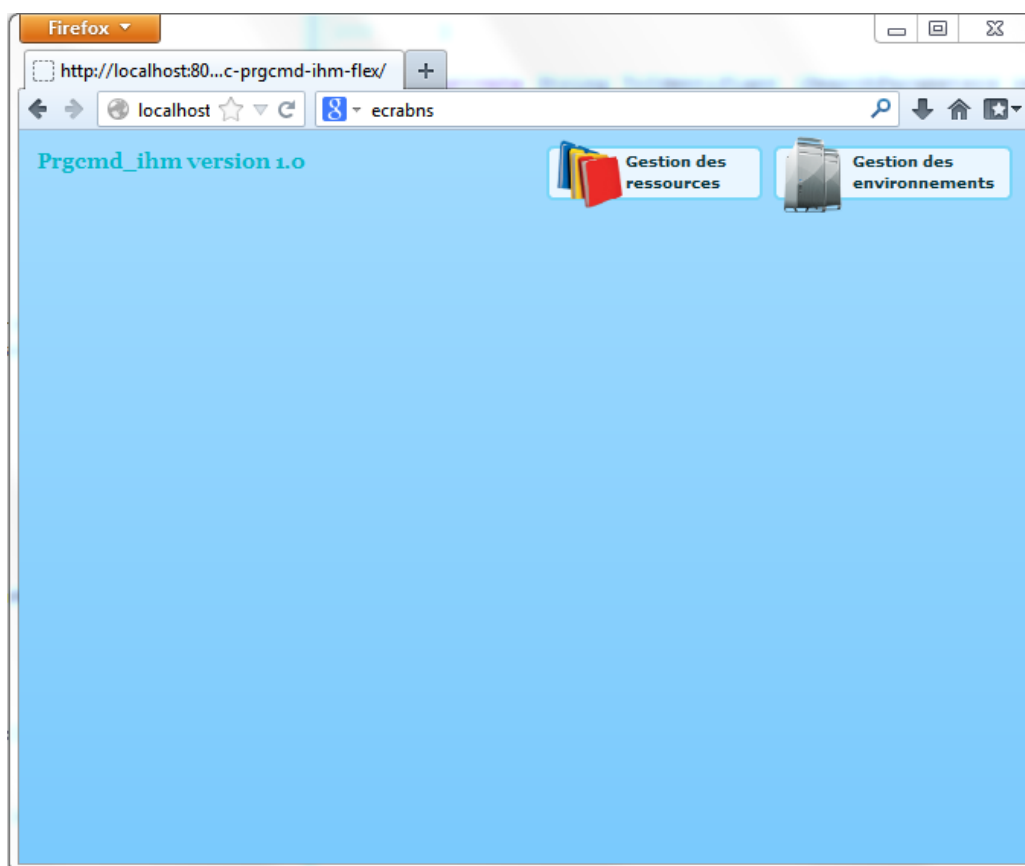


Figure 30: Interface d'accueil de l'application

Le design de l'interface d'accueil de l'application a été repris sur le design de l'interface d'accueil de l'application « BricMLMC-prgcmd ». Elle propose les deux fonctionnalités générales de l'application.

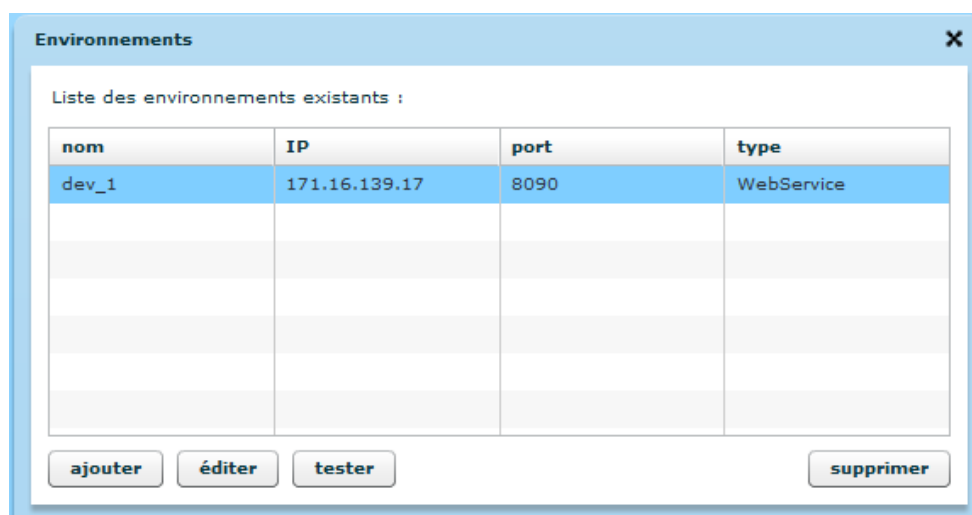


Figure 31: Interface d'affichage des sources de données

L'interface affiche la liste des sources de données avec les informations communes, le nom, l'IP, et le port. L'utilisateur peut choisir d'ajouter une source, d'éditer, ou d'en supprimer une,

Figure 32: Interface d'ajout d'une source de données

Quand l'utilisateur choisit d'ajouter une source de données, l'interface de la figure 32 apparaît et propose à l'utilisateur trois onglets, pour remplir les champs d'informations selon le type.. Dans cette itération, nous n'avons implémenté que l'ajout d'une source de type Webservice. Les autres types ne sont, en effet, pas utilisés par l'équipe de travail à l'heure où nous avons réalisé l'implémentation.

Figure 33: Interface de recherche des ressources

L'interface de recherche dans la figure 33, contient à gauche, une liste des sources rangée par type, dont l'utilisateur choisit où effectuer la recherche. L'utilisateur, ensuite, sélectionne le type de ressource, et remplit les champs dynamiquement affichés.

Dans la figure 34, l'interface affiche les résultats sous formes d'URI de ressources en bas. Remarquons, au passage, les champs d'information affichés lors du choix de la précommande comme type de ressource. Un double click sur une ligne dans la liste des résultats ouvrirait l'interface d'affichage de la ressource (WindowBean).

Concernant l'intégration, nous avons constaté une petite incohérence au niveau de l'affichage des interfaces dans l'application. Le WindowBean devait être affiché en mode modal, afin d'empêcher l'utilisateur d'ouvrir d'autres ressources en parallèle, mais ça a fini par créer des problèmes entre les fenêtres ouvertes par extraction des éléments. Chose que nous avons résolu par un retour à l'affichage normal (non modal), et l'apparition d'une alerte à l'ouverture d'une ressource en parallèle.

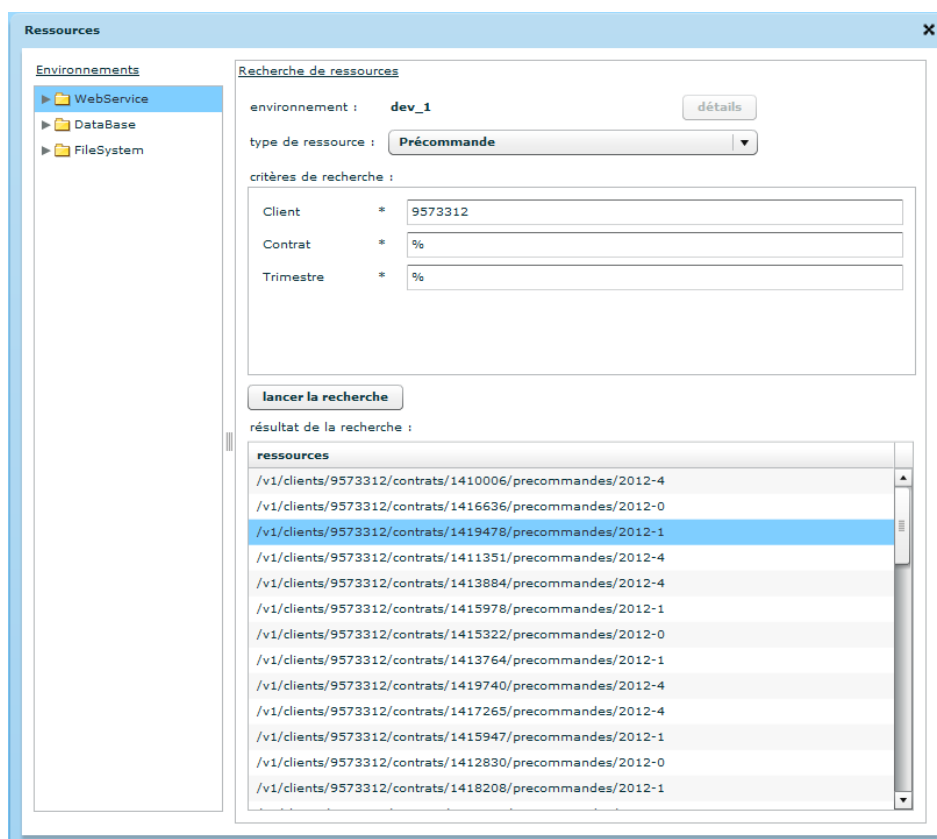


Figure 34: Résultat d'une recherche

Le caractère « % » utilisé dans le champ « Contrat » et « Trimestre », est un Wildcard pour désigner « peut importe ». En l'occurrence, l'application affiche les précommandes du client identifié par 9573312, peut importe le contrat, et peut importe le trimestre.

3.5. Itération 5 : l'implémentation des restrictions

3.5.1. Spécifications

Lorsque l'utilisateur modifie une ressource en vue de l'enregistrer, il lui incombe de respecter la conformité avec les restrictions de son schéma XSD. Ceci l'éviterait de créer des incohérences pouvant compromettre les processus métiers de SNCF, une fois la ressource est remise pour exploitation. Cependant, pour implémenter les restrictions, nous savons que le schéma se trouve à l'entrée de l'application, et que la modification s'effectue à l'autre bout. Le seul moyen de lier les deux, est de transformer les restrictions en objets, qui vont s'ajouter au modèle générique, et accompagner ainsi la ressource jusqu'à l'affichage. Dès lors, l'utilisateur pourra choisir un des cas d'utilisation de la figure 35.

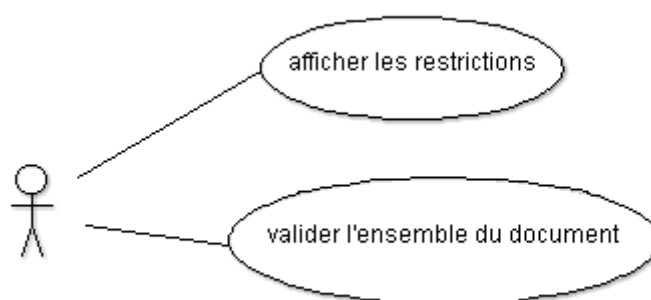


Figure 35: Diagramme de cas d'utilisation de la quatrième itération

Les tableaux 30, et 31 décrivent les cas d'utilisation de la figure 35.

Tableau 30: Cas d'utilisation "afficher les restrictions"

Cas : afficher les restrictions
Pré condition : l'élément est affiché Post condition : les restrictions sont affichées dans une nouvelle fenêtre
Scénario principal
1. L'utilisateur clique sur le bouton « afficher les restrictions liées à l'élément » 2. Le système affiche les restrictions de l'élément dans une nouvelle fenêtre
Exceptions
- L'élément ne dispose d'aucune restriction

Tableau 31: Cas d'utilisation "valider l'ensemble du document"

Cas : valider l'ensemble du document
Pré condition : Le document est affiché Post condition : Le document est validé
Scénario principal
1. L'utilisateur clique sur le bouton « valider le document » 2. Le système vérifie la conformité avec les restrictions 3. Le système remonte des alertes sur les éléments non validés
Exceptions
aucune

3.5.2. Risques d'intégration

Passer les restrictions à travers les couches de l'application ne sera pas sans risque. En effet, chaque couche jouera un rôle dans l'implémentation des restrictions. La couche DAO se chargera de récupérer le schéma XSD, la couche Mapping proposera des méthodes pour parcourir le schéma et extraire les restrictions. Ces méthodes seront utilisées dans GenericMapping pour greffer les restrictions au modèle générique, puis transféré par Services. Et finalement, la couche Display implémentera les interactions nécessaires. Le tableau 32 liste les risques d'intégration qui pourraient avoir lieu.

Tableau 32: Tableau des risques d'intégration de la quatrième itération

Couche	Classe	Changement	Risque
DAO	IAccessor	Ajout du type « schéma » dans les types de ressources	Le schéma n'est pas récupérable de la même manière que les autres ressources
Mapping	—	Ajout d'une classe de parcours du schéma et d'extraction de contenu	Temps de parcours long
GenericMapping	FactoryBean	Ajout des restrictions aux entités	Le modèle générique n'est pas correctement construit La synchronisation ne peut pas être effectuée

Services	BeanServices	Déclaration des nouvelles classes en ActionScript et liaison avec leurs homologues en Java	Le transfert des données ne marche plus L'application ne peut pas être utilisée
Display	CompComposed CompSimple CompList	Ajouter la validation à l'élément	La validation ne s'applique après chaque modification
	WindowBean	Ajouter la validation de tout le document	La document est erronément validé

3.5.3. Conception de l'itération

Dans cette partie de conception, l'idée est de représenter les restrictions du schéma XML par une hiérarchie de classes, qui seront liées aux entités génériques par une relation de composition (figure 36).

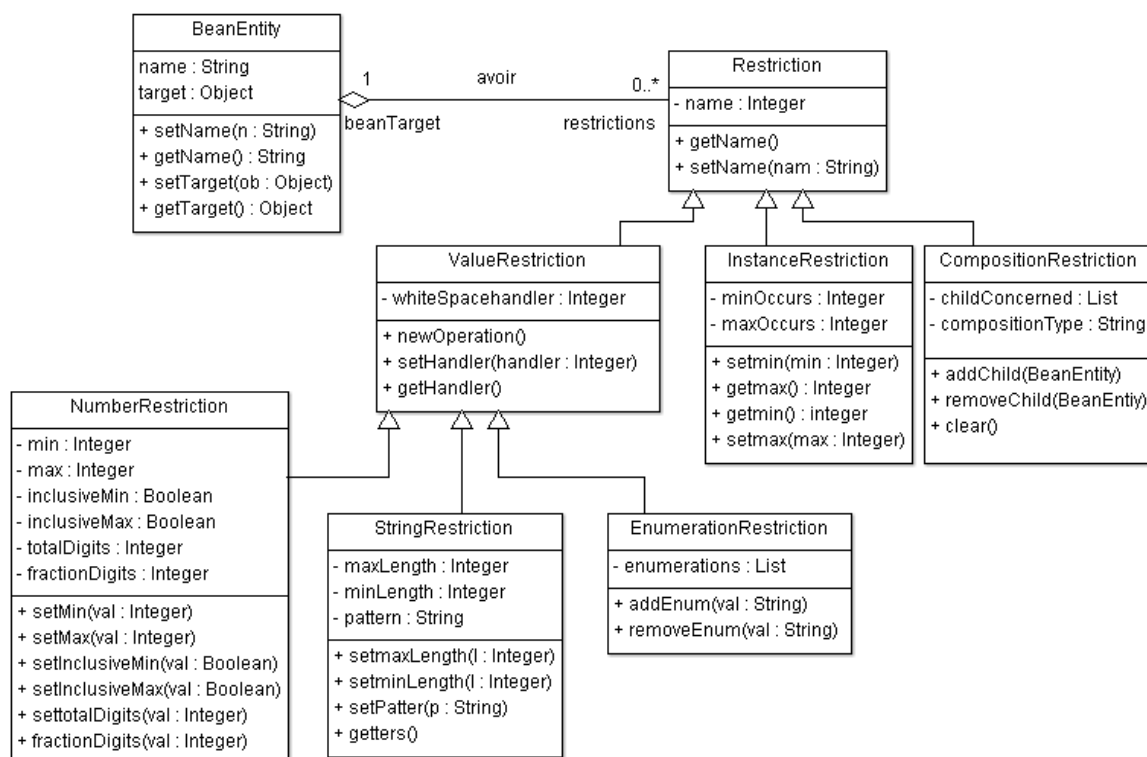


Figure 36: Diagramme de classes de la cinquième itération

La classe Restriction contient un nom, et un attribut « beanTarget » qui fait référence à l'entité cible. En dessous, nous distinguons trois classes filles :

- InstanceRestriction pour l'occurrence de l'élément cible dans une BeanList,
- CompositionRestriction donne des restrictions sur la composition de l'élément cible, dans le cas des BeanComposed,

- ValueRestriction concerne les valeurs des BeanSimples, elle définit le comportement sur les espaces blancs et spécifie la restriction suivant trois classes filles :
 - NumberRestriction, qui définit les contraintes sur la valeur numérique
 - StringRestriction, qui définit les contraintes sur une chaîne de caractères
 - EnumerationRestriction qui définit les choix possibles d'une valeur

3.5.4. Réalisation et intégration

La réalisation de cette itération, à l'heure de rédaction de ce présent rapport, et sous la contrainte de la date de soutenance, est toujours en cours. Elle n'est donc pas abordée.

3.6. Conclusion du chapitre

Dans ce chapitre, nous avons présenté l'implémentation de quatre itérations sur cinq, la dernière étant toujours en cours. Chaque itération s'est caractérisée par ses propres spécifications, analyse de risques, conception et mise en œuvre. La première itération a servi de plateforme pour les autres, durant laquelle nous avons implémenté trois couches métier, Display, Services, et GenericContainer. La deuxième itération s'est concentrée sur les interactions de l'interface. Elle a implémenté la majorité des actions de navigation et de modification locale. Ensuite, dans la troisième itération, nous avons implémenté la synchronisation entre l'objet mappé distant et les modifications faites sur l'objet générique. Finalement, nous avons complété les processus métier, par la gestion des sources de données, afin de pouvoir rechercher les ressources.

Chapitre 4

Outils de réalisation

Dans ce chapitre, nous allons voir un descriptif des outils mis à notre disposition pour la mise en œuvre des itérations. Certains outils ont été fixés par le cahier de charge pour des raisons d'intégration. D'autres ont été choisis pour des critères spécifiques au besoin.

4. Outils de réalisation

4.1. Configuration et gestion de projet

4.1.1. Maven

Maven est un outil open-source de build pour les projets Java très populaire, conçu pour supprimer les tâches difficiles du processus de build. Maven utilise une approche déclarative, où le contenu et la structure du projet sont décrits, plutôt qu'une approche par tâches. Cela aide à mettre en place des standards de développements au niveau d'une société et réduit le temps nécessaire pour écrire et maintenir les scripts de build. Raison pour laquelle Capgemini privilégie Maven dans tous les projets java [Maven].

4.1.2. Spring

Spring est un Framework qui s'appuie sur les principes du design pattern IoC, et sur la programmation par aspects (AOP) pour la création d'objets, et la mise en relation d'objets par l'intermédiaire d'un fichier de configuration qui décrit les objets à fabriquer et les relations de dépendances entre ces objets. En ce sens SPRING est une infrastructure similaire à un serveur d'application J2EE. L'avantage étant que les classes n'ont pas besoin d'implémenter une quelconque interface pour être prises en charge par le Framework (au contraire des serveurs d'applications J2EE et des EJB) d'où son appellation de conteneur « léger ».

Spring est très utilisé dans les environnements de développement de Capgemini pour la flexibilité qu'il ajoute à la configuration du projet. Un facteur très recherché dans les processus de test d'intégration. En effet, Spring peut être couplé efficacement avec des Frameworks de test d'intégration s'appuyant sur la notion du Mocking [Spring].

4.1.3. SVN

SVN est l'acronyme du nom Subversion, un logiciel de gestion de versions, distribué sous licence Apache et BSD. Il a été conçu pour remplacer CVS (Concurrent Version System). Ses auteurs s'appuient volontairement sur les mêmes concepts (notamment sur le principe du dépôt centralisé et unique) et considèrent que le modèle de CVS est le bon, et que seule son implémentation est en cause. Subversion fonctionne donc sur le mode client-serveur, dont le serveur, en principe centralisé et unique, héberge les fichiers constituant la référence. Et dont

le client peut modifier localement les fichiers et les synchroniser manuellement ou automatiquement avec le serveur [SVN].

4.2. Développement et tests

4.2.1. Flex

Flex est une solution de développement créée par Macromedia en 2004 puis reprise par Adobe en 2006, permettant de créer et de déployer des applications Internet riches (RIA) multiplateformes grâce à la technologie Flash et particulièrement son lecteur. Son modèle de programmation fait appel à MXML (basé sur XML) et ActionScript 3.0, reposant sur ECMAScript.

La technologie Flex produit un fichier .swf intégré dans une page html. La richesse de l'interface graphique ainsi créée présente l'inconvénient, comme toute applet, de créer un fichier .swf sur le serveur un peu long à télécharger. Le client doit attendre une certaine durée avant de pouvoir réellement interagir avec l'interface. Mais cet inconvénient se voit compensé par la rapidité de traitement local, du fait que l'application riche peut proposer des fonctionnalités avancées n'obligeant pas le recours au serveur [Flex].

4.2.2. Blase DS

Blase DS est un mini serveur dédiée aux applications réalisées en Flex ou en Adobe Air. Il s'intègre à un serveur d'application J2EE et prend le relais sur toutes les communications qui peuvent avoir lieu entre l'interface riche et le serveur. Il est aussi le pont de communication entre la technologie Action Script et la technologie JAVA, et il assure l'appel aux services entre les deux par le transfert des objets et le mapping des classes.

Au vu de l'architecture de mon application, Blase DS est utilisé uniquement dans la couche Services. Ainsi toutes les interactions entre l'interface riche et les couches métiers passent par ce pont [BlaseDS].

4.2.3. REST

REST est l'acronyme de Representational State Transfer. C'est un style d'architecture. Un style d'architecture est un ensemble de contraintes qui permettent, lorsqu'elles sont appliquées aux composants d'une architecture, d'optimiser certains critères propres au cahier des charges du système à concevoir. Cela signifie que REST définit un ensemble de contraintes sur les systèmes hypermedia distribués comme le *World Wide Web* afin d'optimiser les qualités désirées comme la séparation des tâches, la simplicité, la généricité ou les performances réseaux. REST fournit un véritable modèle pour décrire le fonctionnement que devrait avoir le web aujourd'hui. Bien que moins formel, ce modèle peut être comparé aux paradigmes objet ou entité-association. En pratique, ce style est appliqué dans le développement des web services par l'attribution d'une Uri à chaque service. La couche DAO fait appel à ce genre de service pour l'accès aux ressources [REST].

4.2.4. Jaxb

JAXB est l'acronyme de Java Architecture for XML Binding. Le but de l'API et des spécifications JAXB est de faciliter la manipulation d'un document XML en générant un ensemble de classes qui fournissent un niveau d'abstraction plus élevé que l'utilisation de JAXP (SAX ou DOM). Avec ces deux API, toute la logique de traitements des données contenues dans le document est à écrire. JAXB au contraire fournit un outil qui analyse un schéma XML et génère à partir de ce dernier un ensemble de classes qui vont encapsuler les traitements de manipulation du document.

Le grand avantage est de fournir au développeur un moyen de manipuler un document XML sans connaître XML ou les technologies d'analyse. Toutes les manipulations se font au travers d'objets java. Ces classes sont utilisées pour faire correspondre le document XML dans des instances de ces classes et vice versa : ces opérations se nomment respectivement unmarshalling et marshalling [Jaxb].

4.2.5. JUnit

JUnit désigne un Framework de rédaction et d'exécutions de tests unitaires. Le but principal de JUnit est d'offrir au développeur un environnement de développement simple, le plus familier possible, et ne nécessitant qu'un travail minimal pour rédiger de nouveaux tests. Leur écriture ne représente pas le seul intérêt d'un tel Framework. Une fois ces tests mis en place, il est très important de pouvoir les exécuter et d'en vérifier les résultats très rapidement. L'idée principale de JUnit est de représenter chaque test par un objet à part. Un test correspond souvent à une classe du programme. En d'autres termes, un test pourra être composé de plusieurs tests unitaires dont le rôle sera de valider les différentes méthodes des classes [JUnit].

4.3. Conclusion

Dans ce chapitre, nous avons fait un tour sur les outils utilisés dans la configuration générale du projet, et la mise en œuvre des itérations. Remarquons toute fois qu'avec Jaxb, toute la logique des algorithmes s'est faite sur les objets java.

Conclusion Générale

Le projet de stage consistait à mettre à la disposition de l'équipe de travail, une application pour la gestion délocalisée de commandes Fret SNCF, qui en automatiserait la procédure faite manuellement, et permettrait à l'équipe d'effectuer une gestion plus facile et rapide.

L'application, dont une grande partie a été réalisé en quatre itérations, a implémenté les principales fonctionnalités, répondant ainsi à tous les besoins exprimés, au début, par l'équipe. La première itération a donné naissance à l'application en supportant sa première fonctionnalité, la représentation visuelle. Certes, c'est une fonctionnalité qui était basique car n'incorporait pas de manipulation d'affichage, mais son implémentation a impliqué la construction des couches nécessaires pour transporter la ressource depuis sa représentation en objet java jusqu'à l'interface RIA (Rich Internet Application). Chose qui a permis, alors, de nous concentrer sur l'affichage dans la deuxième itération. Dans cette dernière, nous avons travaillé sur l'interface RIA afin de l'enrichir par des fonctionnalités de navigation et de modification de contenu. Ces fonctionnalités ont été implémentées en ActionScript, tout en bénéficiant de la complémentarité avec le Framework Flex et des atouts de ses composants graphiques. Ensuite, à l'issue de la troisième itération, l'application s'est dotée d'une fonctionnalité lui permettant de reporter sur l'objet mappé distant, les modifications faites localement, afin de le synchroniser et de maintenir la cohérence de l'application. La quatrième itération a été consacrée à la gestion des sources de données, et plus particulièrement, à l'implémentation des couches DAO et Mapping, les deux dernières couches restantes de l'application. Nous y avons tâché d'établir une architecture extensible, qui permettrait, une fois l'implémentation réalisée, d'ajouter d'autres types de sources de données, sans être amené à refaire la conception. Enfin, dans la dernière itération, nous avons envisagé d'implémenter les restrictions, afin que l'utilisateur puisse valider ses modifications conformément au schéma, bien que qu'elle ne fût pas exprimée en besoin par l'équipe.

Le projet de fin d'études a été une réelle opportunité pour mettre en pratique un ensemble de connaissances acquises durant mes études à l'ENSIAS. Par ailleurs, ce stage était l'occasion de développer plusieurs qualités, aussi bien au niveau métier et professionnel qu'au niveau relations humaines.

Perspectives

Le projet n'en est qu'à ses débuts, et d'autres fonctionnalités viendront s'ajouter pour offrir une marge de manœuvre plus large pour la gestion des commandes. Notamment des fonctionnalités de comparaison entre éléments d'une liste, de recherche avancée, ou d'affichage personnalisé. Cependant, avant de se projeter à moyen ou à long terme, il y a trois points prioritaires :

- Terminer l'implémentation de la cinquième itération,
- Passer le projet en phase de validation, avec rédaction du cahier de recettes,
- Rédiger un manuel d'utilisateur décrivant les fonctionnalités de l'application

Ces points seront la clôture du projet de stage d'étude, et une ouverture d'un projet en sa version 1.0, au sein des projets de développement de SNCF, qui évoluera avec l'émergence de nouveaux besoins.











Bibliographie












- [CapPre] Documentation interne. PCM. Présentation Capgemini Maroc
- [MLMCPre] Documentation interne. PG1. Présentation Générale Capgemini
- [UML] Laurent AUDIBERT. De l'apprentissage à la pratique. Ellipse.2009
- [Spirale] Laila KJIRI. Présentation de processus de développement logiciel. Module1
- [GraPro] Documentation interne. FT03. Fiche technique de MLMC Programme de commandes
- [Maven] Page web. maven.apache.org/plugins/Documentation
- [Spring] Gary Mak. Spring par l'exemple. Pearson Education. 2009
- [SVN] Page web. subversion.apache.org/docs/
- [Flex] Etienne NASSE. Formation Framework Flex. Capgemini. 2010
- [BlaseDS] Documentation Officielle. livedocs.adobe.com/blazeds/1/blazeds_devguide/
- [REST] Tutoriel Interne. Guide d'utilisation de REST.
- [Jaxb] Documentation Oracle. www.oracle.com/technetwork/articles/javase/index-140168.html
- [JUnit] Benoit GANTAUME. Junit Mise en œuvre pour automatiser les tests en Java.

Annexe

Tableau des icônes et leur description

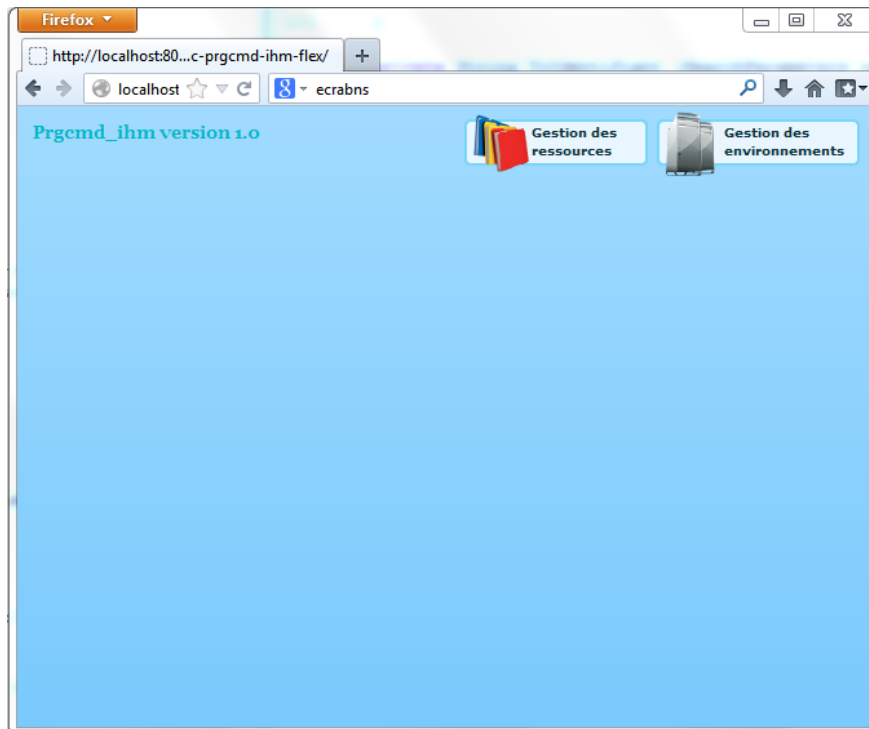
Nous avons privilégié les icônes aux boutons avec label, pour ne pas encombrer la fenêtre d'affichage, et nous leur avons ajouté une petite description dans des info-bulles.

	Actualiser l'affichage, pour prendre en considération la profondeur et les types sélectionnée
	Retourner à la racine de la fenêtre précédente
	Afficher la racine première de la fenêtre
	Ouvrir la fenêtre des options pour changer la couleur des types de valeur
	Montre que le champ de saisie est un champ de recherche
	Appliquer les modifications sur le document, faites en dehors de la fenêtre
	Elargir la fenêtre pour afficher les valeurs longues qui ne tiennent pas sur la taille par défaut
	Petite icône qui s'affiche à gauche des CompSimples de type attribut
	Petite icône qui s'affiche à gauche des CompSimples
	Petite icône qui s'affiche à gauche des CompLists, permet aussi de plier/déplier l'élément

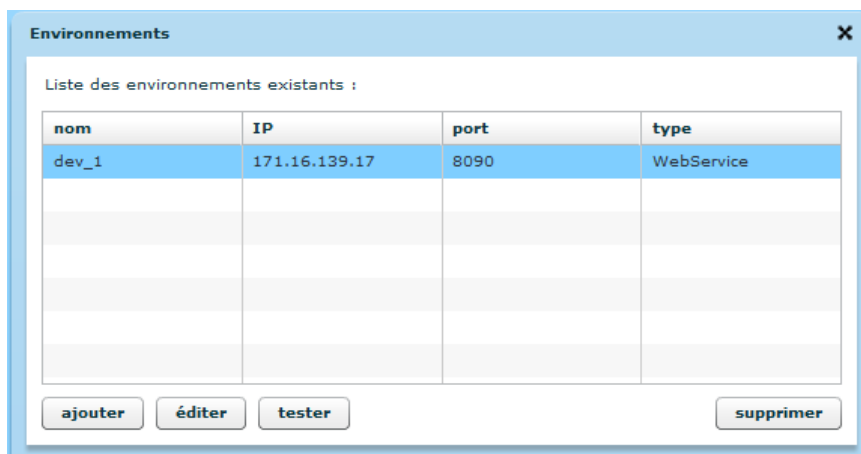
	Petite icône qui s'affiche à gauche des CompComposeds, permet aussi de plier/déplier l'élément
	Extraire l'élément dans une nouvelle fenêtre, ça permet de naviguer partiellement dans le document
	Même utilité que sa précédente, sauf que l'extraction s'effectue dans la même fenêtre. L'icône « précédent » permet, dans ce cas, de revenir à l'élément père affiché précédemment.
	Ajouter un élément à la liste
	Créer les fils de l'élément
	Vider la valeur pour un CompSimple, les éléments pour un CompList, et effacer les fils pour un BeanComposed
	Supprimer l'élément de la liste
	Editer la valeur du BeanSimple
	Rétablir la valeur précédente, dans l'historique de valeur, du CompSimple
	Rétablir la valeur suivante dans l'historique de valeur, du CompSimple
	Confirmer la saisie dans le champ de valeur du CompSimple

Exemple d'utilisation de l'application

Dans cette partie, nous allons illustrer l'utilisation de l'application, par des captures d'écran. Nous allons supposer, que l'utilisateur veuille définir une nouvelle source de données, il ouvre l'application dans un navigateur web :



Puis clique sur le bouton « Gestion des environnements », une nouvelle fenêtre s'ouvre, affichant la liste des environnements déjà sauvegardés :



Nous avons déjà l'environnement « dev_1 » enregistré, le bouton « tester » indique si une connexion avec le serveur a été établie ou non. Pour ajouter un nouveau, l'utilisateur clique sur le bouton « ajouter » :

Détails de l'environnement

Type de l'environnement :

Service Web | Système de fichiers | Base de données

nom :

IP :

port :

racine :

url :

L'utilisateur choisit quel type d'environnement veut il créer, en navigant entre les onglets. L'application ne supporte que les environnements de type « Service Web », les deux autres onglets sont, par conséquent, grisés. L'utilisateur remplit les champs, teste la connexion, et si tout marche bien, clique sur ajouter. L'application procède à une petite vérification du nom avant d'ajouter l'environnement. Le nom doit, en effet, être unique. Une fois ajouté, l'application affiche le nouvel environnement dans la liste.

L'utilisateur revient sur la première page d'accueil, et clique sur le bouton « Gestion des ressources ». Une nouvelle fenêtre s'affiche.

Ressources

Environnements

- WebService
- DataBase
- FileSystem

Recherche de ressources

environnement : aucun environnement

type de ressource : Choisissez le type de ressource

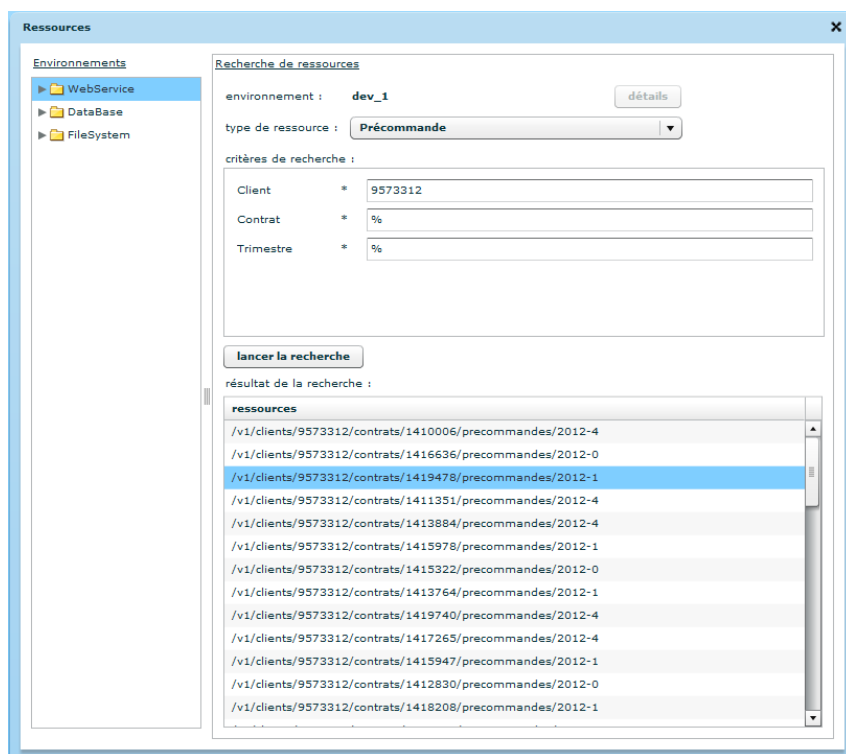
critères de recherche :

résultat de la recherche :

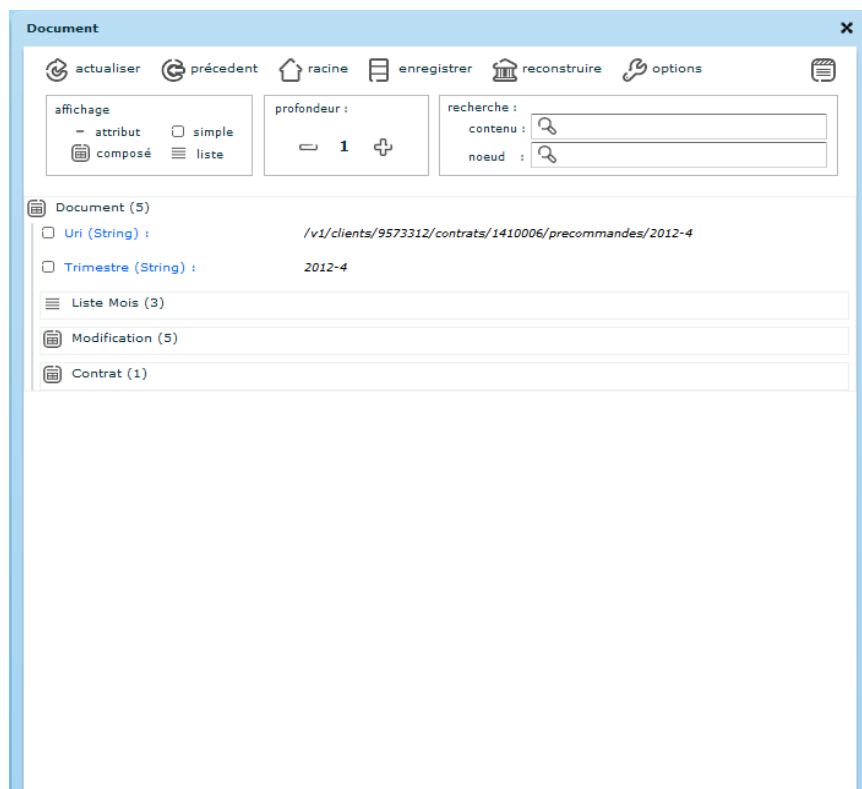
ressources

La liste, à gauche, classe les environnements par catégorie, dont l'utilisateur choisit un pour effectuer sa recherche, puis choisit un type de ressource, et remplit les champs affichés, avant

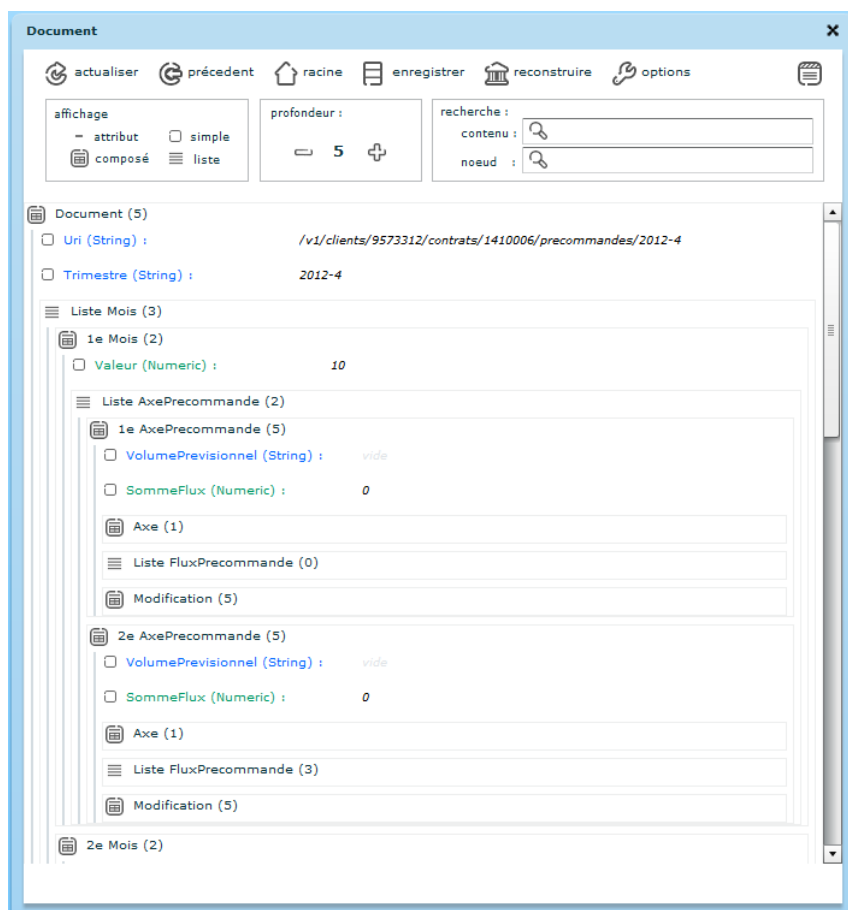
cliquer sur le bouton « lancer la recherche ». Les résultats sont affichés dans la liste en bas de la fenêtre.



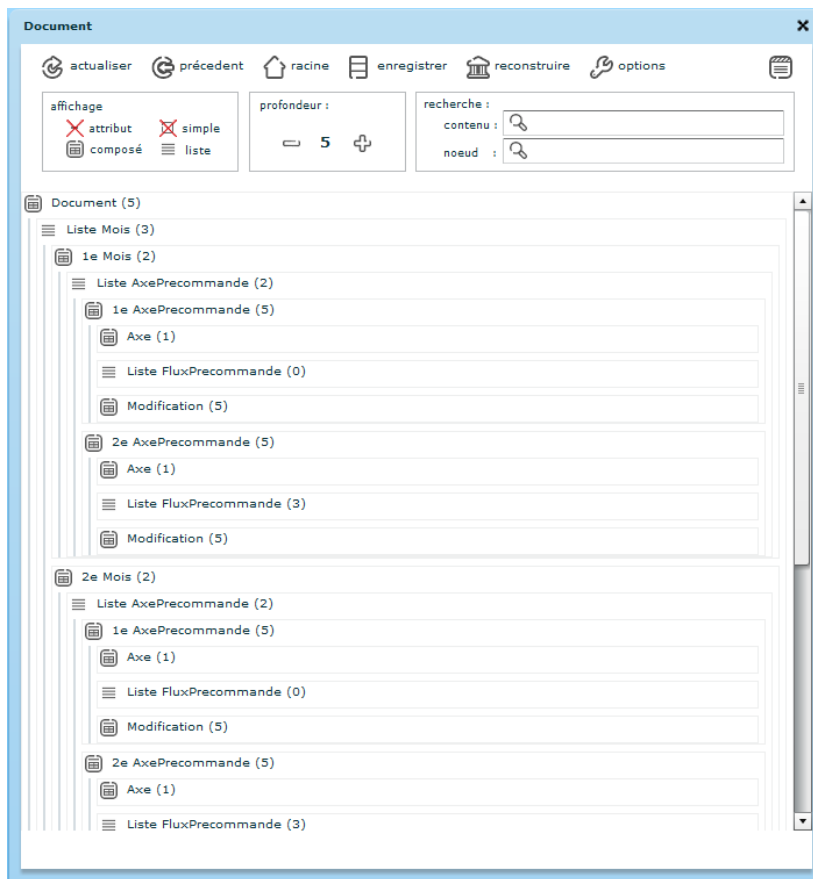
L'utilisateur choisit, dès lors, une ressource parmi celles dont l'URI est affiché dans la liste. Un double click sur la ressource ouvre une fenêtre d'affichage de la ressource.



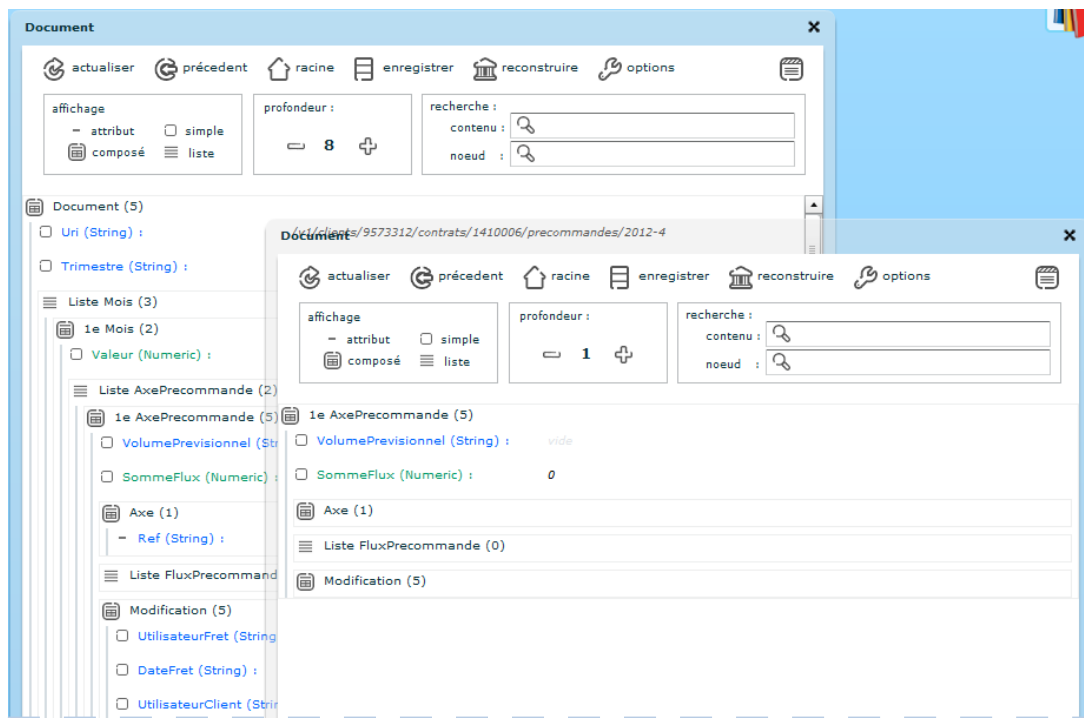
La fenêtre ouverte propose toutes les fonctionnalités implémentées, de navigation et d'édition du document. La fenêtre n'affiche qu'un seul niveau de profondeur. L'utilisateur pourrait choisir d'afficher davantage d'éléments par incrémentation de profondeur, ou par dépliage des éléments complexes. Pour un affichage rapide, l'utilisateur choisit d'afficher les cinq premiers niveaux du document.



Les couleurs aident l'utilisateur à mieux lire le document. Toutefois, des fonctionnalités de sélection de types, peuvent l'aider à mieux contrôler l'affichage, n'afficher que les éléments complexes, par exemple. Dans ce cas, l'utilisateur barre par click, dans le panel « affichage », sur les types à ne pas afficher. Ainsi, il barre les attributs, et les simples.

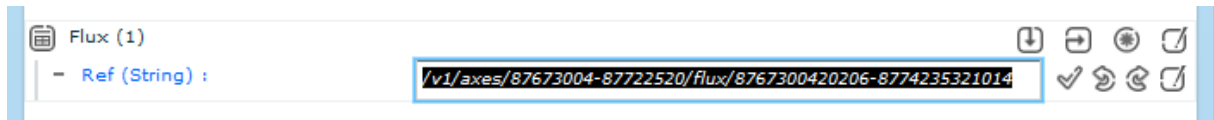


En n'affichant que les éléments complexes, l'utilisateur obtient, en effet, l'arbre du document. Couplée à la fonctionnalité d'extraction, ça permet de répartir l'affichage du document sur plusieurs fenêtres. Du fait, que chaque élément complexe est doté d'un bouton pour l'extraction locale, et d'un bouton pour l'extraction dans une nouvelle fenêtre.

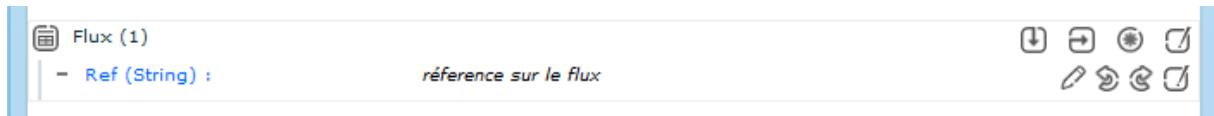


Un avantage d'une telle extraction est de restreindre les fonctionnalités globales, comme la recherche, à l'élément extrait.

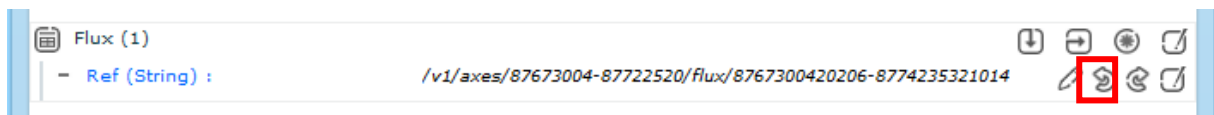
Pour modifier la valeur d'un élément simple, l'utilisateur double-clique sur le champ de valeur, ou clique sur le bouton « éditer ».



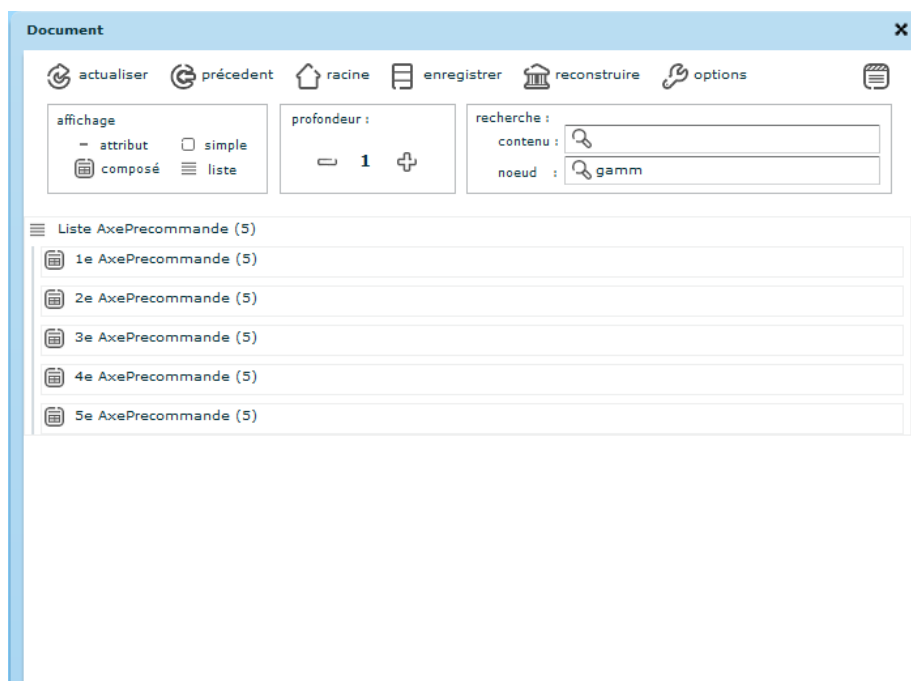
L'utilisateur saisie la nouvelle valeur, et tape sur la touche entrée. L'application valide, ensuite, la valeur avant de l'appliquer.



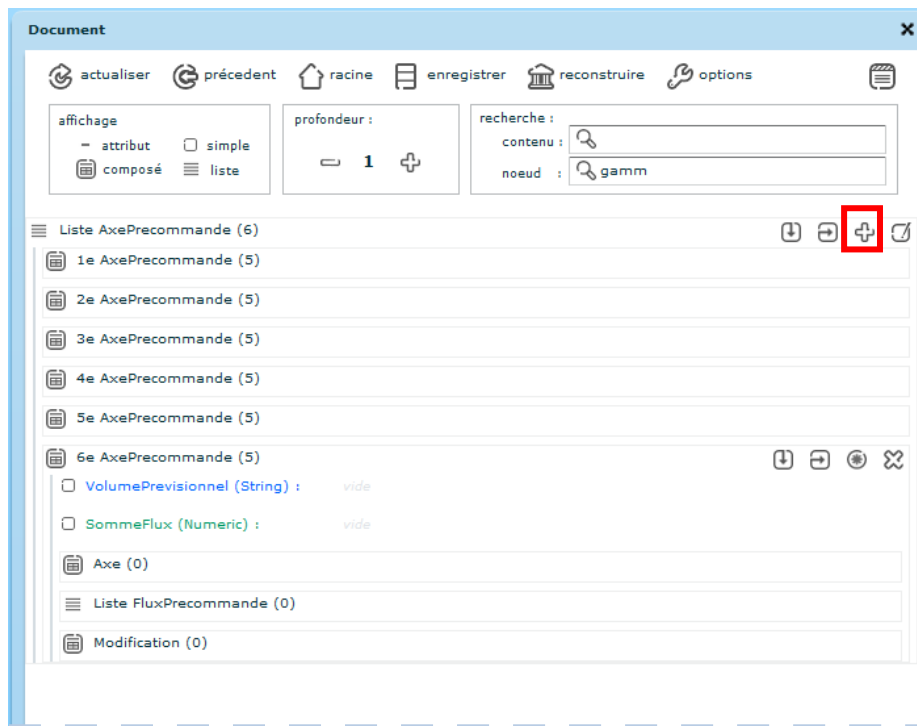
Toutes les modifications liées à chaque élément simple, sont enregistrées dans un historique de valeurs. L'utilisateur pourrait y naviguer, à l'aide des boutons « undo » et « redo », pour rétablir les valeurs qui on été saisies avant.



Concernant la modification des éléments complexes, de type BeanComposed et BeanList, l'utilisateur dispose des actions nécessaire pour l'ajout et la suppression. Dans l'exemple, l'utilisateur extrait localement une liste « AxePrecommande ».

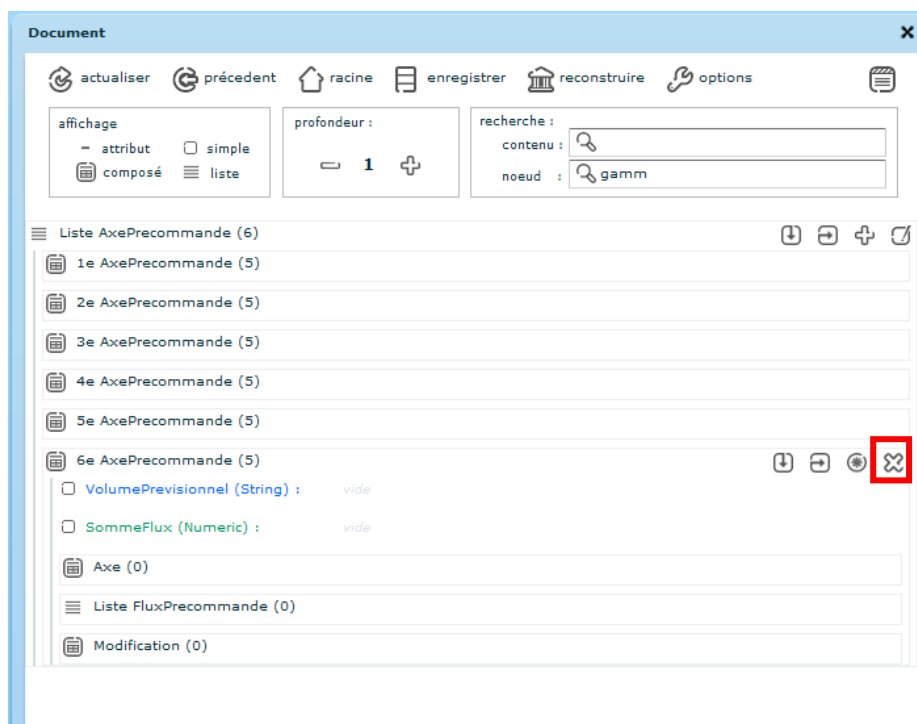


Puis, choisit d'y ajouter un nouvel élément.

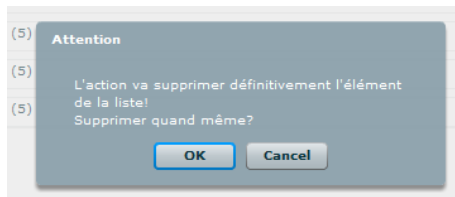


L'application crée la structure de premier niveau de l'élément, et l'ajoute à la liste. L'utilisateur pourrait choisir de compléter la structure, par la création des éléments fils, et ainsi de suite.

La suppression d'un élément d'une liste, se fait par click sur le bouton « supprimer », de l'élément en question.

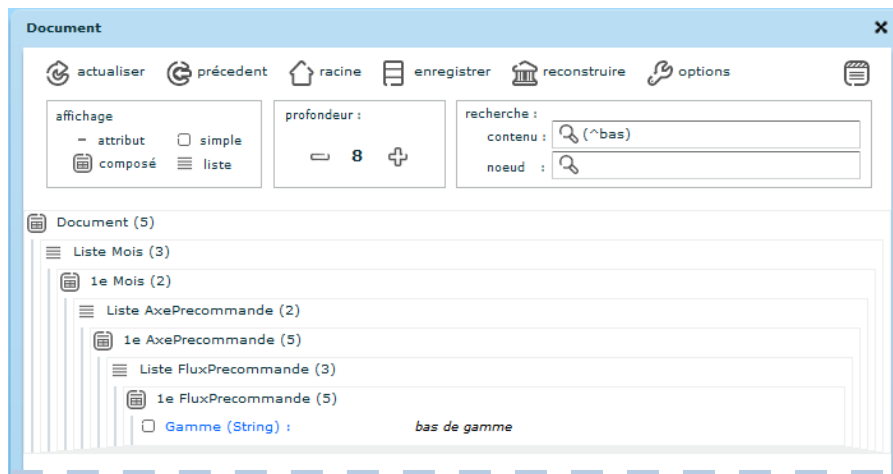


L'application demande, aussitôt, la confirmation.

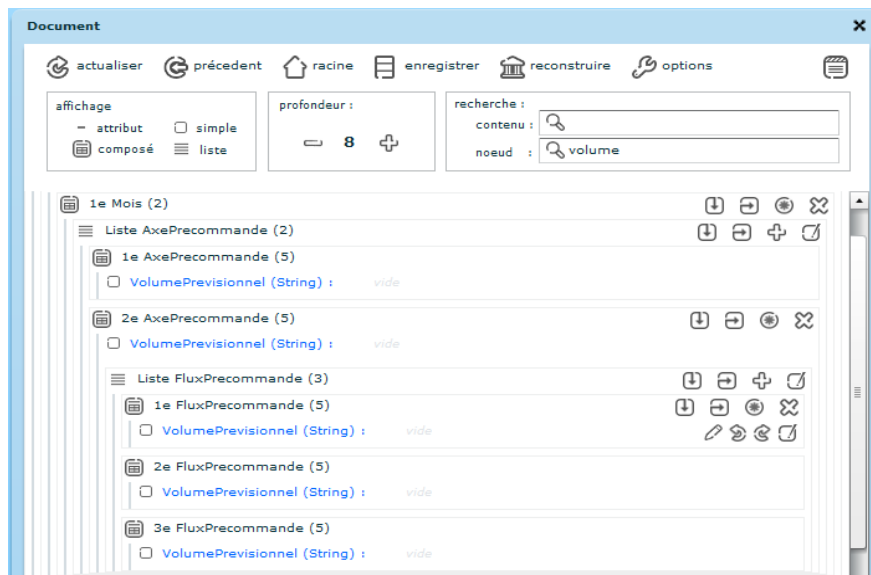


Et supprime l'élément de la liste.

La recherche dans le document s'applique la racine d'un WindowBean, et est implémentée en deux fonctionnalités, recherche par contenu, qui permet de rechercher dans les valeurs des BeanSimples, à bases d'expressions régulières.



Et recherche par élément, qui permet de rechercher les éléments dans le nom commence par la chaîne de caractères saisie.



À la fin des modifications, l'utilisateur clique sur le bouton « enregistrer ». L'application affiche, dès lors, la ressource qui va être enregistrée, et attend l'utilisateur de cliquer sur enregistrer, en guise de confirmation.