

Mémoire de stage

Auteur: Guillaume ZAVAN
Version: 1.0
Daté du 25 août 2010
Entreprise : *General Agents*
Tuteur : Damien RUSH

Avant-propos

Ce mémoire présente mon activité au sein de l'entreprise *General Agents* du 19 Avril au 6 Juillet 2010 pour le compte de M. Damien RUSH, de son vrai nom Daniel DE JESUS, qui fût mon tuteur de stage durant cette période. Sans m'étendre d'avantage sur le rôle de cet individu douteux ni sur ses compétences discutables en temps que chef d'entreprise – à qui je réserve toutefois une place dans ma conclusion –, je préciserais que la majeure partie, si ce n'est l'intégralité de ce projet n'aurait pût exister sans le concours de M. Benjamin BRUNET (benjamin.brunet7@gmail.com), qui a travaillé à la réalisation d'un cahier des charges précis et m'a permis d'exploiter au mieux mes compétences dans un environnement de travail professionnel.

En annexe de ce rapport sont joints les cahiers des charges et schéma de conception du projet « *Prospector* », à titre d'exemple mais également pour appuyer la démarche décrite dans le présent mémoire. Celui-ci vise à présenter le projet à proprement parler, son contexte et ses objectifs ; il ne tient nullement lieu de documentation technique, bien que des extraits de code puissent y figurer afin d'illustrer tel ou tel concept.

Titre	Mémoire de stage
Auteur	Guillaume ZAVAN
Courriel	webkawa@gmail.com
Version	1.0
Date	25 août 2010

Table des matières

1.Contexte.....	6
Sujet du stage.....	7
Problématique.....	7
Intervenants.....	8
Planning préliminaire.....	8
Cahier des charges.....	8
Accessibilité.....	8
Services.....	10
Base de données.....	11
Agilité du système.....	11
2.Conception	12
Choix techniques.....	14
Acteurs.....	14
Considérations préliminaires.....	14
Gestion des configurations.....	15
Outils de pilotage.....	16
Plan d'exécution.....	17
Accès aux données.....	18
Génération de l'IHM.....	18
Modules.....	19
Interface BBCode.....	19
Gestionnaire de configuration.....	19
Objets mémorisés.....	19
Gestion des dates.....	19
Encodage SHA256.....	19

Pilote FTP.....	19
Librairie HTML.....	20
Gestionnaire d'installation.....	20
Librairie des langages.....	20
Librairie des pays.....	20
Pages du site.....	21
Fourre-tout.....	21
Pilote SMTP.....	21
Pilote SQL.....	21
Fonctionnement détaillé.....	21
Gestion des configurations.....	21
Pilotes & Utilitaires.....	24
Accès aux données.....	25
Langues & Pays.....	27
Traductions.....	28
Gestion des exceptions.....	29
Génération du code HTML.....	30
3.Base de données.....	31
Présentation.....	33
Liste des tables.....	33
Adresses.....	33
Entreprises.....	33
Disponibilités.....	34
Rencontres.....	34
Erreurs.....	35
Individus.....	35
Opérations.....	36
Codes postaux.....	36

Secteur d'activité.....	37
Statuts professionnels.....	37
Traductions.....	37
Utilisateurs.....	38
Remarques.....	38
4.Développement.....	39
Introduction.....	40
Exécution d'une page.....	42
Exécution de traitements.....	45
Chargement des données.....	47
Chargement d'une configuration.....	49
Remarques d'ordre technique.....	50
5.Conclusion.....	51
Rendu final.....	52
Bilan.....	57
État du projet.....	57
À propos de M. Rush	57
Références.....	59

1. Contexte

Résumé

Sujet du stage.....	7
Problématique.....	7
Intervenants.....	8
Planning préliminaire.....	8
Cahier des charges.....	8
Accessibilité.....	8
Services.....	10
Base de données.....	11
Agilité du système.....	11

Sujet du stage

L'entreprise *General Agents* se présente sous la forme d'une *start-up* Bruxelloise de taille réduite spécialisée dans le domaine de la communication et des médias. Créée au début de l'année 2008, elle propose ses services à des entreprises de taille restreintes désireuses d'assurer à moindre coût une communication directe jusqu'à des clients potentiels, par voie de fax, de téléphone ou de mail. Les tarifs pratiqués sur chaque contrat placent sur un minimum dix fois inférieur à la moyenne du secteur ; le coût final de l'opération évoluant fortement au gré des résultats qu'elle produit. Cette stratégie permet de toucher une clientèle hors d'attente des agences de communication traditionnelles, au prix toutefois d'une contrainte de réussite indispensable à la rentabilité des opérations engagées.



Problématique

Mon stage s'est axé autour d'un travail de rénovation et de complétion du site web utilisé par *General Agents* pour assurer la promotion et le suivi de ses opérations de communication, sous l'appellation *Prospector*.

On notera que General Agents décline son activité en deux services sans rapport l'un de l'autre et gérés par ailleurs de manière indépendante : d'une part, le département Prospector, sur lequel je reviendrai plus loin dans ce rapport ; de l'autre, le département SMB Location Shooting, exerçant une activité dans le domaine de la location à courte durée pour des photographes ou réalisateurs : www.locationshooting.be.



Le site utilisé alors présentait divers défauts : technique dépassée, manque d'interactivité, aspect visuel peu satisfaisant, aucune accessibilité pour les administrateurs et clients finaux. Le projet de

réovation, prévu depuis un peu plus d'un mois par M. Brunet mais ne pouvant se réaliser faute de disponibilité du responsable informatique et principal développeur de l'entreprise, M. Ludovic KUCKART, m'a donc tout naturellement été attribué, essentiellement de par les exigences formulées par l'université quand au caractère d'encadrement de projet devant revêtir le stage.

Le but poursuivit par le projet se présente sous la forme de deux principaux fils conducteurs :

1. Assurer la mise à niveau, sur le plan graphique et technique, du site existant, en incluant diverses mises à jour notamment vis-à-vis des offres proposées.
2. Concevoir une interface simple et intuitive permettant d'une part la simplification des échanges d'information entre l'entreprise, ses clients et ses sous-traitants (nous y reviendrons) ; d'autre part, la constitution d'une base de donnée durable et solide des différents intervenants du système.

La possibilité d'une reprise du site existant afin d'y inclure une simple mise à jour ne me paraissant satisfaisante ni pour mon mandat, ni pour la fiabilité du site, j'ai rapidement suggéré de reprendre à zéro un travail de conception absent, si ce n'est inexistant pour l'ancienne version – dont, par ailleurs, il m'a été impossible de retrouver l'auteur.

Suite à un différent m'ayant opposé à M. Rush et sur lequel je m'épanche d'avantage à la fin de ce mémoire, la nouvelle version du site n'a jamais été déployée. La version qu'il devait à terme remplacer est donc toujours consultable à l'adresse suivante : www.prospector.pro. Par commodité, on la désignera dans la suite de ce rapport comme étant « l'ancienne version », en opposition avec la « nouvelle version », sur laquelle j'ai travaillé.

Intervenants

Le projet s'est réalisé au sein de l'équipe suivante :

- M. Benjamin BRUNET, en qualité de responsable final du projet et maîtrise d'ouvrage ;
- M. Nicolas MARCHAND, en temps que graphiste et concepteur de l'interface web ;
- M. Ludovic KUCKART, pour le suivi technique du projet ;
- M. Maxime GUYON, en temps que responsable des contenus commerciaux.

J'étais pour ma part chargé de la conception et du déploiement technique du projet, ainsi que de l'estimation des coûts et des délais.

Planning préliminaire

Mon temps de travail s'est vu divisé de la manière suivante :

- Une semaine pour assurer l'étude des besoins et la rédaction du cahier des charges, impliquant les choix techniques nécessaires à son déploiement ainsi qu'une première étude des coûts.
- Deux semaines pour la conception préliminaire en UML et la réalisation des diagrammes liés.
- Trois semaines pour le déploiement des éléments d'arrière-plan : base de données, utilitaires, modèle vue-contrôleur.
- Trois semaines pour le déploiement des éléments d'interface : architecture du site, CSS, scripts d'optimisation, pages finales.
- Deux semaines pour les tests de masse en collaboration avec divers intervenants de l'entreprise.

Concrètement, mon planning final a été soumis à divers modifications au fur et à mesure de l'avancement liées à des contraintes dans le planning de l'entreprise et à une certaines sous-estimation des délais liée à mon inexpérience dans des projets d'envergure en JEE.

Cahier des charges

La partie suivante se veut un résumé, classé par thème, des attentes initiales de la MOA pour le projet et à partir desquels j'ai bâti le cahier des charges final de l'application, lié en annexe de ce mémoire.

Accessibilité

Le premier aspect du projet sur lequel il m'a été donné de travailler, mais également le plus flagrant, a été celui lié à l'accessibilité du site.

Mémoire de stage | Contexte

The screenshot shows the homepage of the Prospector website. At the top, there's a navigation bar with links for Accueil, Force de vente, Base de données, Salons et Evenements, Enquêtes téléphoniques, Tarif, and Contact. There are also icons for fax, email, and other services. The main content area features a large orange header with the word "prospector" and a sub-header "Solutions force de vente Starters et Indépendants". Below this, there's a brief description of the service: "Vous souhaitez une solution pour bien démarrer votre activité ? Un levier efficace, alliant économie de temps et d'argent. Un moyen sûr et rapide de conquérir une clientèle". The page is divided into several sections with numbered headings: 1) Le Briefing : Vos attentes et Objectifs, 2) Design & Image : L'attraction visuelle et le reflet de votre image, 3) Envoi de vos emails / Fax : La gestion du Mass Mailing, 4) Appels téléphoniques & prise de rendez-vous : Gestion et Contact Prospect, and 5) Le rapport de résultat :. On the left side, there's a small image of a person in a suit. The overall design is dark with orange highlights.

La capture ci-dessus provient de l'ancien site, et plus précisément de la page présentant les forfaits offerts. Les défauts, certes mineurs mais bien réels, y sont nombreux : en-tête inachevé faute de logo digne de ce nom, absence des mentions légales obligatoires, polices difficilement lisibles (orange clair sur fond bleu), double animation flash (sur les flancs et dans le coin supérieur droit) se répétant en boucle, pas de pied de page à proprement parler, etc.

En terme de portabilité, le bilan n'est guère plus flatteur, le site semblant n'avoir été optimisé qu'à l'adresse des systèmes Apple ; il génère sous Linux des erreurs flagrantes et peine à s'afficher correctement sous Windows.

Le référencement, quand à lui, est médiocre pour ne pas dire mauvais : le mot-clé « Prospector », pourtant associé au nom de domaine du site, ne retourne celui-ci qu'en quatrième place sous Google.

The screenshot shows a Google search results page for the query "prospector". The search bar at the top contains "prospector" and has a "Rechercher" button. Below the search bar, it says "Environ 8 510 000 résultats (0,18 secondes)". The first result is a link to "Prospector Plastics Search Engine & Database for Technical Datasheets" with a snippet about it being a search engine for plastic manufacturers. The second result is "PROSPECTOR CENTRE DE DEMONSTRATION IDES - The Plastics Web®" with a snippet about it being a search engine for plastic manufacturers. The third result is "ProteinProspector" with a snippet about it being a proteomics tool. The fourth result is "Free stuff, freeware, samples - free stuff directory Prospector" with a snippet about it being a directory of free software. The fifth result is "Prospector booster votre entreprise" with a snippet about it being a company specializing in prospecting clients.

Mémoire de stage | Contexte

Le plan du site en temps que tel n'est pas à blâmer, du fait d'un nombre de page très réduit – chaque bouton du menu menant directement à l'une d'entre elle. On reprochera toutefois le manque de transparence quant au statut de l'entreprise, et plus particulièrement l'absence de liens vers le site de *General Agents*.

Prenant en compte ces différents défauts, il m'est rapidement apparu impossible, ou du moins stérile, de reprendre le site existant pour en réaliser la mise à jour. L'absence de sources m'a conforté dans cette décision, et tout au plus n'avons nous gardé de l'ancien site que sa palette de couleur – nuances d'orange et de gris. La conception de l'interface a été assurée par MM. Brunet et Marchand ; mes objectifs concernant cette partie du projets se sont donc limités à :

- La réalisation d'un code HTML, fortement orienté CSS, produisant un résultat équivalent sous IE, Firefox, Safari et Opera, quelque soit la plateforme de lancement ;
- La validation de l'ensemble des contenus finaux – HTML, CSS, JS – au moyen du *W3C Validator* sans retour d'erreur ;
- La possibilité de traduire l'ensemble des contenus dans plusieurs langues et éventuellement en rajouter ;
- L'utilisation d'une interface extensible, permettant d'exploiter au mieux les écrans larges de plus en plus utilisés par le grand public.

Une des priorités avancées par la MOA sera l'absolue nécessité de donner au site un aspect professionnel, sans bavures et donc plus vendeur que la version courante.

Services

Le second souhait formulé par la MOA pour le projet tient dans l'ajout de plusieurs outils aptes à simplifier la logistique de l'entreprise. Pour comprendre les enjeux de telles fonctionnalités, il convient avant tout de se pencher sur le déroulement normal d'une opération réalisée pour le compte d'un client.

Chaque opération se caractérise selon trois facteurs principaux :

- Son médium, soit le courriel, le fax ou le téléphone ;
- Son ampleur, à savoir nombre d'appels ou envois à effectuer ;
- Son objectif, la prise de rendez-vous ou la communication unidirectionnelle.

Concrètement, le contrat passé avec un client comporte, d'une part, un argumentaire de vente éventuellement lié à une ou plusieurs réalisations graphiques, et de l'autre, un descriptif technique décistant de son coût minimal et de ses objectifs.

Dans le cas d'une communication unidirectionnelle, à savoir une promotion classique d'un produit ou des activités du client, la logistique demeure assez simple : les contacts sont comptabilisés au fur et à mesure de leur exécution, et ce au jour le jour.

Dans le cas d'un objectif visant la prise de rendez-vous, les choses sont tout de suite plus compliquées. Il faut alors rédiger avec le client un emploi du temps résument les disponibilités qu'il libère pour rencontrer les tiers intéressés par son offre. Cet emploi du temps est ensuite rempli au fur et à mesure des contacts par les

Mémoire de stage | Contexte

différents sous-traitants de l'entreprise, qui se chargent de confirmer les rendez-vous auprès des intéressés et de tenir chaque agent au courant des évolutions du calendrier. Inutile de le dire, ce travail, réalisé entièrement par voie téléphonique et outil de tableur, peut rapidement prendre une tournure particulièrement délicate, notamment dans le cas où l'opération rencontre un succès inattendu. Il s'agit alors d'arranger avec le client des compromis pour lui permettre d'assister à chaque rencontre, de jongler avec l'emploi du temps ou encore de renoncer purement et simplement à des tiers ayant répondu à l'offre. Étant donné que ces derniers, comme expliqué précédemment, sont précisément ceux qui décident du tarif final de la prestation, on conviendra que la logistique mériterai un meilleur support.

La demande formulée par la MOA peut donc se résumer sur trois axes :

1. Permettre au client de définir son emploi du temps, de surveiller l'avancement de ses opérations et de s'enquérir des rendez-vous pris pour son compte en temps réel ;
2. Donner aux sous-traitants la possibilité de consulter et de remplir directement l'emploi du temps client tel que celui-ci a souhaité le définir ;
3. Laisser libre champ aux administrateurs concernant la gestion finale du système.

Il ne s'agit là que d'un résumé dont une étude plus complète est détaillée dans le cahier des charges.

Base de données

Le troisième objectif formulé par la MOA se base sur la constitution d'une base de données complètes, permettant de garder trace des clients et tiers contactés, des opérations réalisées ou encore de l'état des emplois du temps client.

Nous le verrons, le modèle relationnel permettant de réaliser cet objectif découle purement du cahier des charges ; la principale difficulté de déploiement étant constituée par la nécessité de réaliser les IHM permettant d'administrer simplement la base de données.

Agilité du système

Le dernier point soulevé concerne les évolutions futures et plus que probables du site – l'activité de l'entreprise prenant par moment des tournants inattendus, pour ne pas dire absurdes, suite à l'impulsion de son dirigeant. Il paraît donc nécessaire de pouvoir non seulement récupérer la base de donnée à d'autres fins que celles prévues au départ, mais également de prévoir le travail ultérieurs de développeurs n'ayant pas pris part à la conception du système.

2. Conception

Résumé

Choix techniques.....	14
Acteurs.....	14
Considérations préliminaires.....	14
Gestion des configurations.....	15
Outils de pilotage.....	16
Plan d'exécution.....	17
Accès aux données.....	18
Génération de l'IHM.....	18
Modules.....	19
Interface BBCODE.....	19
Gestionnaire de configuration.....	19
Objets mémorisés.....	19
Gestion des dates.....	19
Encodage SHA256.....	19
Pilote FTP.....	19
Librairie HTML.....	20
Gestionnaire d'installation.....	20
Librairie des langages.....	20
Librairie des pays.....	20
Pages du site.....	21
Fourre-tout.....	21
Pilote SMTP.....	21
Pilote SQL.....	21

Fonctionnement détaillé.....	21
Gestion des configurations.....	21
Pilotes & Utilitaires.....	24
Accès aux données.....	25
Langues & Pays.....	27
Traductions.....	28
Gestion des exceptions.....	29
Génération du code HTML.....	30

Choix techniques

Au vu des contraintes imposées par le projet et détaillées ci-dessus, les choix fait en terme de technologies utilisées sont les suivants :

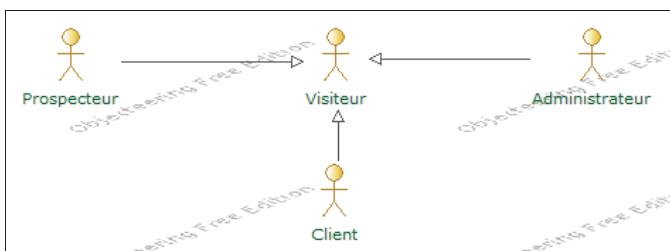
- Pour les traitements, JEE (préféré au langage .NET initialement proposé mais que je maîtrise mal) ;
- Pour la base de données, MySQL ;
- Pour l'IHM, HTML, CSS et Javascript (dans le but de simplifier la navigation mais sans incidence sur les traitements).

L'ancienne version du site étant réalisée en PHP, il a été calculé qu'une dépense avoisinant les 10€ mensuels serait nécessaires pour la location d'un espace de taille restreinte au sein d'un serveur Tomcat mutualisé – le nom de domaine étant déjà enregistré et financé à part.

Acteurs

Après étude préliminaire du projet, on distingue quatre acteurs-types appelés à utiliser le système :

- Les visiteurs, à savoir tout individu accédant au site de façon anonyme dans un but informatif ;
- Les prospecteurs, ou sous-traitants de l'entreprise, appelés à participer aux opérations en cours ;
- Les clients, devant par le biais du système assurer le suivi de leur planning et des opérations dont il bénéficient ;
- Les administrateurs, ou membres de l'entreprise habilités à accéder au système.



A l'exception du premier, chaque acteur du système doit disposer d'identifiants pour accéder aux fonctionnalités qui lui sont dédiées.

Comme présenté sur le diagramme ci-contre, les rapports entre acteurs sont triviaux : chacun d'entre eux possède les mêmes droits qu'un visiteur du système, en plus de diverses fonctionnalités dédiées. Un diagramme de cas d'utilisations plus

ample a été réalisé en début de projet et est disponible en annexe de ce mémoire ; toutefois, suite à des changements ultérieurs de la MOA dans le cahier des charges, il présente certaines imprécisions sur lesquelles je ne m'attarderai pas ici. Une liste plus complète se situe directement dans le cahier des charges

Considérations préliminaires

La majeure partie du travail de conception effectué tient dans le diagramme de classe central servant de base à l'ensemble du travail de développement. A défaut d'avoir suivi une formation sur un MVC compatible avec Java, j'ai abordé le projet en gardant en tête la nécessité de disposer d'outils précis et instinctifs pour simplifier mon développement ; il s'agit là d'une part importante du travail réalisé, aussi bien en terme de volume que de complexité, sur laquelle je m'attarderai plus que sur les classes finales réalisées.

Gestion des configurations

Le premier point auquel j'ai souhaité apporter une attention toute particulière concerne le paramétrage du système une fois ce celui-ci opérationnel et actif. En effet, en l'absence d'un développeur Java apte à me remplacer après mon départ, il m'a paru nécessaire de permettre aux gestionnaires du site la modification, en cas de besoin, du comportement de l'application sans avoir à passer directement par le code source et le remplacement de classes.

Le système de configuration conçu se veut une réponse aux contraintes suivantes :

- L'organisation : il doit être possible de repérer rapidement la configuration d'un objet donné, sans avoir à consulter la documentation, afin d'en modifier le comportement ;
- La simplicité : les fichiers de configurations doivent être facilement éditables, y compris par un individu ne possédant qu'une connaissance limitée de l'informatique, et doivent inclure une documentation embarquée ;
- La sécurité : le système de configuration doit résister aux éventuelles erreurs humaines pouvant se produire (syntaxe) ;
- La portabilité : il doit être possible d'accéder aux fichiers de configuration dans un environnement graphique limité (console) ;
- La souplesse : on souhaite pouvoir restaurer facilement la configuration par défaut d'un objet si un problème sérieux venait à se poser.

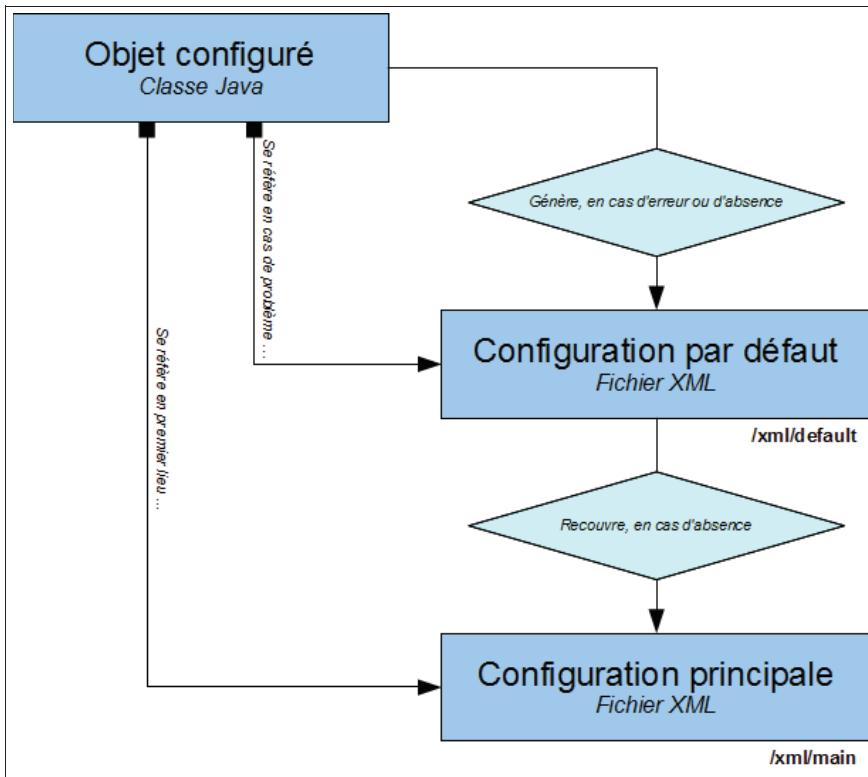
Le format de fichier choisi pour répondre à ses différentes attentes sera le par ailleurs très classique XML, très répandu sur le web et permettant une accessibilité maximale. Un fichier de configuration suit la syntaxe suivante :

```
<configuration>
    <documentation>
        Ce fichier est un exemple de configuration.
        [EXAMPLE_INT : Variable d'exemple entière]
        [EXAMPLE_BOOL : Variable d'exemple booléenne]
        [EXAMPLE_STRING : Variable d'exemple littérale]
    </documentation>
    <variable key="EXAMPLE_INT" type="Integer">123</variable>
    <variable key="EXAMPLE_BOOL" type="Boolean">true</variable>
    <variable key="EXAMPLE_STRING" type="String">Hello world !</variable>
</configuration>
```

Comme on peut le constater, ce format permet l'enregistrement de variables dans trois formats différents (entier, booléen et chaîne de caractère) selon une syntaxe relativement aisée à comprendre.

Chaque objet configuré s'appuie concrètement sur trois configurations distinctes :

- Son réglage machine, inscrit directement dans le code source et utilisé si aucun des autres fichiers ne répond à une requête de l'application ;



- Sa configuration par défaut, à savoir un fichier XML dissimulé servant d'appui en cas d'erreur lors de la lecture du fichier de configuration principal ;
- Sa configuration principale, à savoir un fichier XML destiné à être modifié par les administrateurs du système.

Ce système à trois niveau permet une résistance maximale aux erreurs. L'administrateur désireux de modifier la configuration de l'application peut effectuer des tests directement sur le fichier de configuration principal ; en cas d'erreur de syntaxe ou de valeur, l'application se basera sur le fichier de configuration par défaut. L'administrateur peut également modifier le fichier de configuration

par défaut, mais à la première erreur l'application effectue une restauration de ses réglages machine. Le fichier principal sert donc, dans ce contexte, de « bac à sable » avant toute modification définitive.

Les fichiers de configuration sont organisés dans leur dossier selon leur type et l'objet qu'ils représentent :

/xml/.default/PACKAGE/OBJECT.xml	// Pour les fichiers par défaut
/xml/main/PACKAGE/OBJECT.xml	// Pour les fichiers principaux

Cette organisation permet, d'une part, de maintenir en ordre le dossier des configurations directement à partir du module et du nom des objets concernés et, d'autre part, de simplifier la recherche d'un objet donné. On présente ici l'arborescence locale des fichiers, mais il est également possible de les obtenir sur un serveur distant, par le biais d'une transaction FTP.

La conception et le développement de ce système seront abordés ultérieurement dans ce rapport.

Outils de pilotage

Le deuxième point d'importance mis en avant lors de la conception tient dans les différents outils génériques, ou *drivers*, développés dans le cadre du projet, et visant à simplifier les interactions entre le code Java et les technologies tierces. On trouve dans la version finale du projet cinq de ces utilitaires :

- Pour le SQL, s'appuyant sur la librairie JDBC ;
- Pour le FTP, basé sur les librairies Apache Jakarta ;

- Pour le SMTP, via l'API Javamail ;
- Pour le XML, au moyen de l'API libre JDOM ;
- Pour le SHA256, via un algorithme de cryptage *open-source* ;
- Pour la gestion des dates, basée sur les fonctions fournies directement avec le JDK.

Ces différents outils permettent de simplifier autant que possible la configuration et l'utilisation des librairies qu'ils implémentent. Leur usage, on le verra, vise la simplification à l'extrême via une configuration générique.

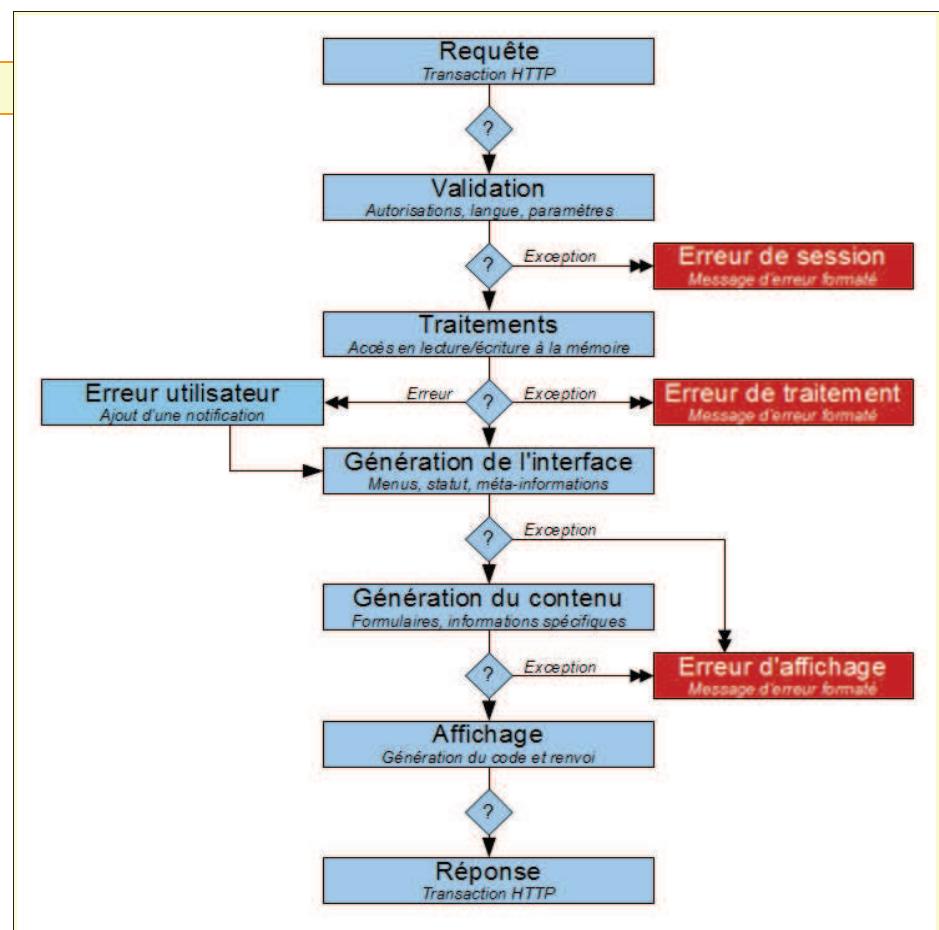
Plan d'exécution

La troisième priorité fixée tient dans l'établissement d'un plan d'exécution clair et précis pour chaque page exécutée sur le site.

Afin de pouvoir développer avec autant de marge que possible des outils à cette fin, le système fait usage de *servlets* classiques à défaut de JSP, plus intuitives mais moins transparentes. Par ailleurs, chaque page du site hérite d'une classe commune en charge de la gestion des erreurs et de la génération de l'interface ; ce afin d'éviter les redondances dans le code.

Le schéma ci-contre présente le processus suivi lors de l'exécution d'une page.

- La requête formulée par le client est dans un premier lieu soumise à une batterie de tests visant à en vérifier la validité : identité de l'utilisateur, langage demandé, paramètres particuliers. Toute erreur lors de cette étape entraîne le renvoi d'un message d'erreur.
- Une fois l'étape de validation terminée, l'application effectue les traitements qui lui sont demandés. Il peut s'agir de simples accès en lecture à la base tout comme de modifications plus profondes. Deux types d'erreurs peuvent découler de la phase de traitement. D'une part, des erreurs inattendues liées au comportement du système ; ces dernières peuvent se traduire par un *bug* dans le code proprement dit et entraînent l'interruption du script. D'autre part, les erreurs liées à un saisie incorrecte de l'utilisateur ; elles entraînent l'interruption de tout ou partie des traitements et l'ajout d'un bandeau de notification clairement visible.



- Une fois les traitements achevés, l'application génère l'interface globale de la page : informations de connexion, menus, mentions légales, etc. Cette étape ne doit normalement pas retourner d'erreur.
- L'application rédige ensuite la partie de code spécifique à la page (vue). Tout comme pour l'interface, ce travail ne doit normalement pas retourner d'erreur. Si toutefois le cas se produit, le script est interrompu.
- Enfin, la page génère le code HTML découlant des différents contenus (voir ci-dessous) et expédie la réponse au client.

A défaut d'être très léger, ce plan d'exécution permet une grande liberté de mouvement lors de la rédaction des traitements finaux : les exceptions retournées sont traitées de manière générique, en amont du traitement proprement dit, et permettent un archivage des erreurs en base de donnée.

Accès aux données

Le quatrième aspect pris en compte dans la conception tient dans la façon dont le système accède au données de la mémoire centrales.

L'API JDBC, utilisée comme intermédiaire entre le code Java et SQL, permet de s'affranchir d'un grand nombre de commandes de lecture ou de mise à jour. L'application met donc à profit ces possibilités par le biais d'une classe commune à tous les objets de la base de donnée, prenant en charge la lecture, l'insertion et la mise à jour de ces derniers en limitant au maximum les commandes SQL contenues dans le code. Elle permet par ailleurs de généraliser certaines erreurs (par exemple, en cas d'objet introuvable). Nous nous pencherons néanmoins d'avantage sur ce point dans la suite de ce rapport.

Génération de l'IHM

La cinquième et dernière priorité soulevée lors de la conception vise à limiter au maximum les erreurs lors de la programmation des IHM. Cet objectif découle d'un constat simple : la rédaction « en bloc » du code HTML génère régulièrement des erreurs d'inattention, parfois invisibles à l'écran, complexifiant grandement la compatibilité des pages obtenues avec les spécifications du W3C.

Pour pallier à cet inconvénient, le système génère en premier lieu un modèle objet de la page – chaque *tag* étant considéré comme un objet – à partir de librairies créées à cet effet ; une fois l'ensemble des traitements effectués, il convertit cette image en code HTML affichable.

Image objet	Équivalent HTML
Objet HTML Objet HEAD Objet META Objet TITLE ! Objet BODY Objet P Objet P ! !	<html> <head> <meta name="foo"></meta> <title>foo</title> </head> <body> <p class="title">foo</p> <p class="text">fooo</p> </body> </html>

Modules

Avant toute chose, il convient de se pencher sur la division du projet en différents modules – traduits en Java par des *packages* distincts. La liste ci-dessous les résume par ordre alphabétique.

Interface *BBCODE*

Le module `bb` contient un utilitaire dédié à la conversion entre du texte formaté selon le langage *BBCODE* vers des objets HTML. De taille restreinte, il sert pour permettre une mise en forme basique des contenus commerciaux contenus sur le site.

Pour plus d'informations sur le BBCODE : <http://fr.wikipedia.org/wiki/BBCODE>.

Gestionnaire de configuration

Le module `configuration` contient l'ensemble des classes utiles à la configuration de l'application décrite précédemment. Il possède un sous-module `configuration.exceptions` comprenant l'ensemble des exceptions qu'il projette.

Objets mémorisés

Le module `database` contient l'ensemble des objets dont les instances sont mémorisées en mémoire centrale du système, ainsi que la classe mère évoqué précédemment permettant une généralisation des appels vers la base de donnée. Il possède par ailleurs deux sous-modules, `database.exceptions` – contenant la liste des exceptions qu'il projette – et `database.index` – comprenant un outil d'optimisation dont nous reparlerons ultérieurement.

Gestion des dates

Le module `date` contient un utilitaire permettant de simplifier la gestion des dates et des durées.

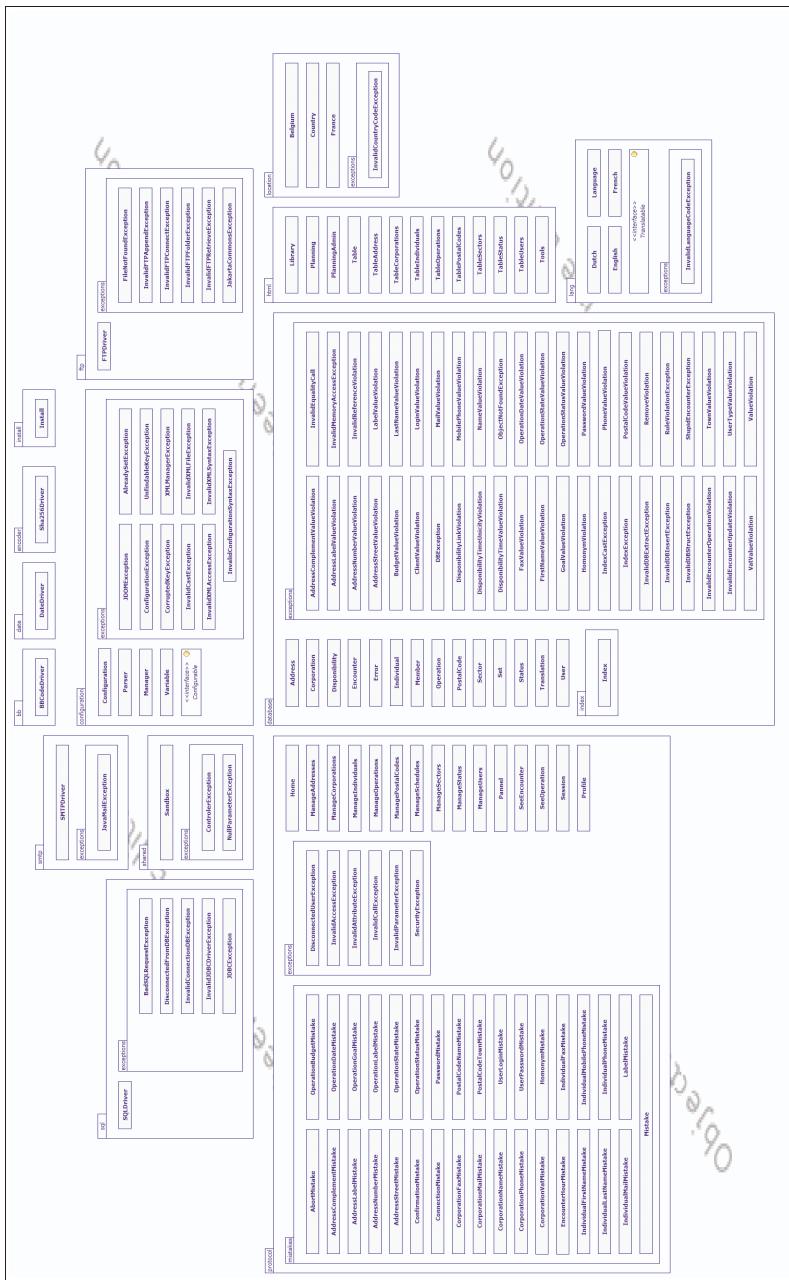
Encodage SHA256

Le module `encoder` contient un utilitaire implémentant un algorithme de codage d'une chaîne de caractères standard vers son *hash* SHA256.

Pilote FTP

Le module `ftp` contient un ensemble d'utilitaires permettant de simplifier le transfert de fichiers en lecture ou en écriture vers des emplacements FTP distants. Il est doté d'un sous-module `ftp.exceptions` comprenant les exceptions qu'il est susceptible de retourner.

Librairie HTML



Dans la version courante, les pays supportés sont la France et la Belgique (General Agents n'exerçant théoriquement pas ailleurs).

Le module `html` contient une liste d'objets et d'utilitaires utilisés dans la phase de construction du code HTML (voir plus haut).

Gestionnaire d'installation

Le module `install` contient une petite application permettant d'installer le programme sur un serveur vide (création des dossiers dédiés à la configuration, création de l'utilisateur `root`, implantation de la base de données, et ainsi de suite).

Librairie des langages

Le module `lang` contient les classes relatives aux langages disponibles sur le site. Il contient un sous-module `lang.exceptions` contenant toutes les exceptions qu'il projette.

Dans la version courante, les langages supportés sont le Français, l'Anglais et le Néerlandais (deuxième langue officielle en Belgique). Une des exigences du cahier des charges tenait dans la possibilité d'ajouter ultérieurement l'Allemand (également parlé dans certaines régions de Flandre).

Librairie des pays

Le module `location` contient toutes les classes relatives aux différents pays mentionnés ou supportés par le site. Il contient un sous-module `location.exceptions` contenant les exceptions qu'il projette.

Pages du site

Le module `protocol` contient les *servlets* disponibles sur le site ainsi que le gestionnaire de session mentionné précédemment. Il inclut par ailleurs les sous-modules `protocol.exceptions` – contenant les erreurs d'affichages – et `protocol.mistakes` – contenant les erreurs utilisateur.

Fourre-tout

Le module `shared` contient l'ensemble des classes ne trouvant pas leur place ailleurs, ainsi qu'un sous-module `shared.exceptions` rassemblant les exceptions « globales » partagées par toutes les entités du système.

Pilote SMTP

Le module `smtp` contient les classes nécessaires à l'envoi de *mail* au format SMTP. Il inclut un sous-module `smtp.exceptions` comprenant les exceptions spécifiques à celui-ci.

Pilote SQL

Le module `sql` contient les classes utilisées dans l'interaction avec la base de données. Il permet de simplifier les opérations de sélection et de mise à jour indispensables aux éléments du module `database`. Il contient par ailleurs un sous-module `sql.exceptions` comprenant les exceptions qu'il est susceptible de projeter.

Fonctionnement détaillé

Cette section aborde dans le détail les classes les plus importantes du système et la façon dont est conçue leur fonctionnement. Il ne s'agit pas d'une documentation exhaustive, loin de là – la Javadoc, jointe en annexe de ce mémoire, pouvant à ce sujet fournir plus de détail –, mais d'un aperçu des réponses données au problèmes formulés dans le cahier des charges et durant la phase de conception. On abordera chacune d'entre elles dans l'ordre de leur réalisation effective.

Gestion des configurations

Le module de configuration a bénéficié d'un soin tout particulier durant les phases de conception et de développement. Il s'appuie sur cinq éléments :

- La classe `configuration.Variable`, représentant une variable de configuration unitaire ;
- La classe `configuration.Configuration`, dont chaque instance représente la configuration d'un objet donné ;

- La classe `configuration.Manager`, fournissant une abstraction entre une configuration et sa représentation physique sur le disque – fichier XML ;
- La classe `configuration.Parser`, faisant office de pont entre un *manager* et la librairie utilisée pour la lecture et l'écriture des fichiers XML, JDOM ;
- L'interface `configuration.Configurable`, implantée par chaque objet bénéficiant d'une configuration propre.

Penchons-nous plus près sur chacun de ces éléments.

Objets configurés

Un objet implémentant l'interface `Configurable` doit répondre à quatre contraintes.

En premier lieu, il doit implémenter une référence vers un objet de type `Configuration` et fournir des accesseurs vers celui-ci. Deuxièmement, il doit pouvoir charger cet objet de manière automatique et via un petit script que nous détaillerons dans le chapitre « Développement ». Troisièmement, il doit pouvoir fournir une configuration par défaut, utilisée si les fichiers principaux et secondaires ne peuvent être lus correctement. Enfin, il doit fournir une documentation, sous forme de chaîne de caractères, des différentes variables qu'il utilise.

Variable de configuration

Une variable de configuration se présente sous la forme d'un objet caractérisé par deux éléments :

- Une clef d'accès, sous forme de chaîne de caractères, permettant de la distinguer des autres variables contenues dans la même configuration – et donc soumise à une contrainte d'unicité ;
- Une valeur sous forme d'objet Java destiné au *cast*.

Son fonctionnement présente un schéma relativement classique sur lequel il semble inutile de s'étendre.

Configuration

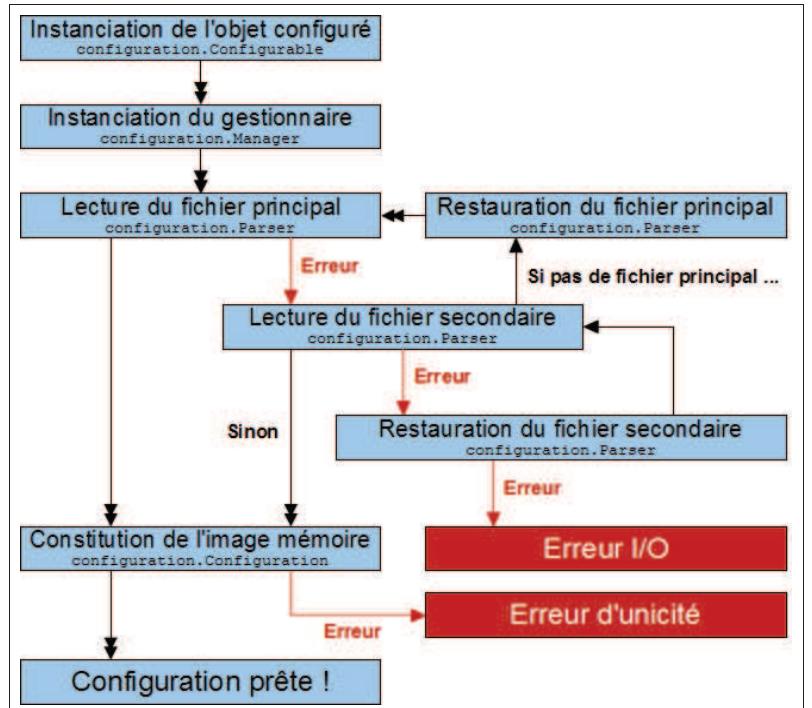
Une configuration à proprement parler implémente une collection de variables dont elle veille à l'unicité et à la validité. Elle ne possède qu'un rôle d'abstraction dont le fonctionnement correspond du reste tout à fait à un objet de type `ArrayList`.

Gestionnaire

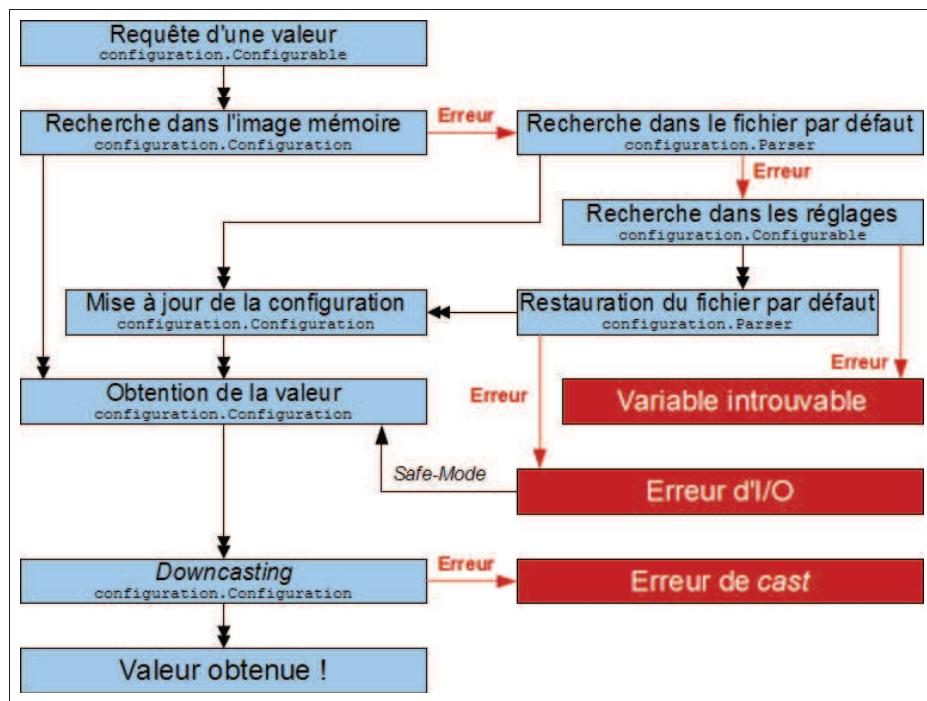
L'objet `Manager` tient le rôle de clef de voûte du système de configuration. Il propose les services nécessaires à la compatibilité entre les fichiers XML contenus sur le disque et la configuration à proprement parler de l'objet qu'il représente.

Le schéma ci-contre présente le cheminement suivi par l'application lors du chargement d'une configuration. Conformément aux prévisions du cahier des charges, la configuration se fait en trois temps : tentative de

lecture du fichier principal, tentative de lecture du fichier secondaire, tentative de restauration des réglages machine. On notera que la restauration du fichier principal se fait uniquement si celui-ci se révèle introuvable en mémoire ; la restauration du fichier secondaire se produit quelque soit l'erreur de lecture – les valeurs attribuées étant alors celles spécifiées par l'objet lui-même. En cas d'échec dans les trois opérations, le gestionnaire retourne une erreur ne pouvant théoriquement être due qu'à un problème dans la configuration *In/Out*. Sinon, le gestionnaire procède à la constitution d'une image « mémoire » de la configuration à partir du fichier déterminé comme fonctionnel – principal ou secondaire. Si une variable s'avère instanciée à plusieurs reprises, une erreur d'unicité est immédiatement retournée. Ce système, en dépit d'une certaine lourdeur, permet de limiter au maximum les erreurs lors de la lecture et de laisser une marge d'essai par le biais du fichier principal.



Ce second schéma présente le cheminement suivit par l'application lorsqu'un objet configuré cherche à accéder à l'une de ses variables. Ici encore, le mécanisme se fait en trois temps :



tout d'abord, le *manager* recherche la valeur dans son image mémoire ; en cas d'échec, il lit le fichier secondaire dans le même objectif ; en cas de nouvel échec, il fait appel au réglage machine et procède, le cas échéant, à la restauration du fichier par défaut. Si la variable n'apparaît pas non plus dans le réglage machine, une erreur est retournée. L'idée est ici encore de distinguer les fichiers principaux, qui ne sont pas affectés en cas de variable manquante, des fichiers par défaut, qui se voient restaurés à la première erreur. Deux derniers détails à souligner : d'une part, l'existence d'une *safe-mode*, utilisé dans le cas d'un dysfonctionnement temporaire du système de fichier ; d'autre part, le fait qu'en cas d'erreur la configuration est mise à jour de manière à ne pas avoir à réaliser de nouveau l'opération par la suite.

Pilote XML

Le fonctionnement de l'objet *Parser* ne présente pas d'intérêt particulier sur le plan conceptuel : tout au plus fournit-il une série d'interfaces destinées à abstraire la librairie JDOM, dont il fait usage. Son fonctionnement sera plus explicitement détaillé dans le chapitre « Développement ».

On notera néanmoins que le pilote XML est conçu de manière à fonctionner aussi bien avec des fichiers locaux que distants – en mode FTP.

Pilotes & Utilitaires

Cette section aborde de façon succincte le fonctionnement des différents utilitaires développés dans le cadre du projet. La plupart ne comprennent qu'une classe unique dotée de différents outils ; il peut toutefois être intéressant de connaître les choix de gestion effectués quand à leur fonctionnement.

Pilote des dates

L'objet `date.DateDriver` contient un certain nombre de classes statiques permettant la manipulations des dates par le biais des librairies par défaut de Java, dont entre autres : obtention de dates littérales (sous forme de chaînes de caractères), opérations sur des périodes de temps, obtention des coordonnées d'une semaine ou d'un jour précis, etc.

Il s'appuie sur trois formats distincts :

- L'objet `java.Date`, déprécié dans les JDK récents mais encore utilisé par certains modules ;
- L'objet `java.Calendar`, destiné à remplacer à terme l'objet `Date` ;
- L'objet `java.Long`, correspondant le cas échéant au *UNIX-Time* et préféré dans les fonctions internes pour sa légèreté.

La classe implémente par ailleurs l'ensemble des données de traductions relatives aux dates et heures (nous reviendrons plus tard sur le rôle d'une traduction et son fonctionnement effectif).

Pilote SHA256

Cet utilitaire ne possède qu'une unique méthode, permettant de convertir une chaîne de caractères vers son *hash* selon l'algorithme SHA256. Il ne semble guère utile de s'attarder d'avantage dessus.

Pilote FTP

L'objet `ftp.FTPDriver` permet l'obtention ou le transferts de fichiers par le biais du protocole FTP. Elle se base sur la librairie Jakarta distribuée librairie par la fondation Apache. Son fonctionnement vise une simplification à l'extrême par le biais d'une classe unique et de quatre méthodes :

- Ouverture de transaction ;

Mémoire de stage | Conception

- Lecture d'un fichier sur le serveur distant et réalisation d'une copie locale ;
- Copie d'un fichier ou dossier local sur le serveur distant – fonction récursive implémentée ;
- Clôture de transaction.

Une dernière méthode relative aux *timeout* est également disponibles mais il n'est pas possible d'en détailler l'utilité sans rentrer dans des considérations purement techniques.

Pilote SMTP

L'objet `smtp.SMTPDriver` permet l'expédition de mails via le protocole SMTP. Il s'agit d'une classe unique ne possédant qu'une fonction simplifiée à l'extrême permettant l'envoi unitaire d'un mail. Il s'appuie sur la librairie Javamail fournie avec le JDK.

Pilote SQL

L'objet `sql.SQLDriver` donne une abstraction de la librairie JDBC utilisée dans les interactions avec le serveur de données. Son fonctionnement vise, ici encore, la simplicité ; il possède six méthodes, détaillées ci-dessous :

- Ouverture d'une connexion ;
- Sélection d'une entrée unique ;
- Sélection d'une liste d'entiers (utile dans les sélections de masse d'identifiants) ;
- Sélection d'une liste de chaînes de caractères (utile dans certains cas particuliers) ;
- Exécution d'une requête d'écriture ;
- Fermeture de la connexion.

A l'instar du pilote FTP, cet objet possède également une fonction liée au *timeout*.

Accès aux données

Cette section ne présente pas le modèle relationnel de données à proprement parler (consulter le chapitre « Base de données » pour plus d'informations à ce sujet) mais l'abstraction dont le programme fait usage pour lire, manipuler et enregistrer les objets enregistrés dans la mémoire persistante du système.

Dans une approche orientée purement objet, l'application définit pour chaque table de la base de données une classe permettant d'en représenter les enregistrements et d'effectuer des manipulations, sous contrôle strict, sur les valeurs de ces derniers. Il apparaît donc dès lors possible de généraliser un certain nombre de services : chargement, création, suppression, et de manière plus globale la plupart des opérations nécessitant une requête SQL brute.

En terme de conception, ce principe se traduit par une classe `database.Member`, de laquelle héritent l'ensemble des objets correspondant à des entrées de la base de données. Cette classe fait massivement usage du pilote SQL, `sql.SQLDriver`, mais également à diverses fonctions héritées de JDBC, permettant notamment de manipuler des entrées sans avoir à rédiger manuellement les commandes correspondantes.

Le schéma ci-contre présente le cheminement suivi par l'application lors du chargement d'un objet enregistré sur la base de donnée. La classe `database.Foo` mentionnée hérite de `database.Member`, dont nous parlions au paragraphe précédent. Le processus se fait en plusieurs étapes :

- Dans un premier temps, on instancie la classe `Foo` par le biais d'une identifiant entier correspondant à sa clef d'accès sur la base de donnée.
 - La classe `Member` parente effectue le chargement de la configuration (détaillé précédemment) et des traductions (que nous aborderons dans la section suivante).
 - On effectue une recherche dans l'index pour voir si l'objet a précédemment été chargé ; le cas échéant, on procède à une copie directe (voir ci-dessous pour plus de détails).
 - Si l'objet ne possède pas de représentation dans l'index, la classe `Member` effectue par le biais de JDBC un sélection unique sur la table correspondante.
 - Le résultat de cette sélection est ensuite traduit et attribué par la classe finale et l'instance concernée enregistrée dans l'index pour un usage ultérieur.
 - On déduit ensuite les variables découlant de règles de gestion.
 - L'objet est prêt à l'usage !
-
- ```

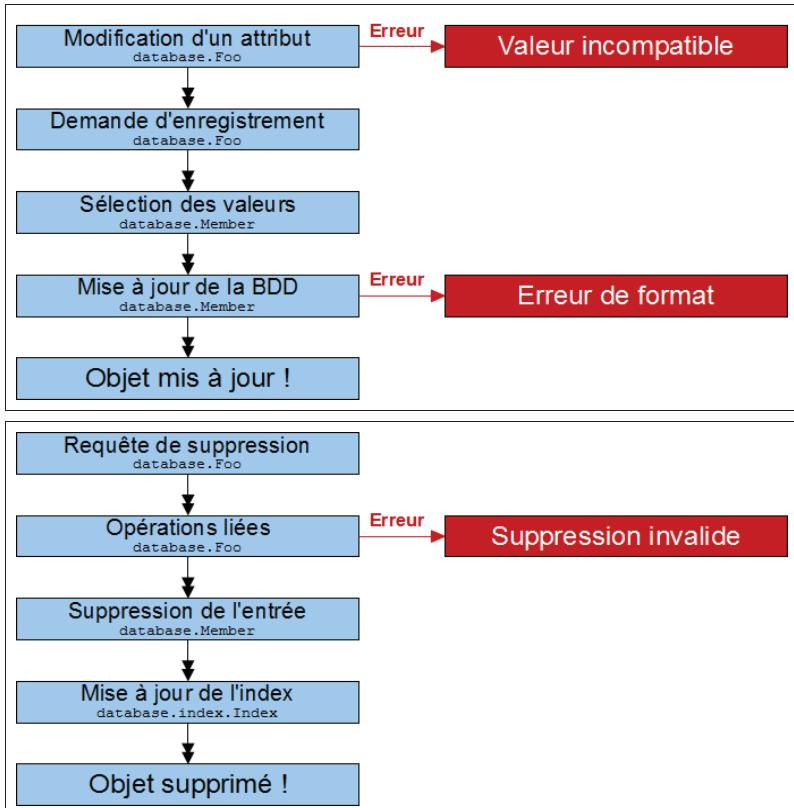
graph TD
 A[Instanciation via ID
database.Foo] --> B[Chargement de la configuration
database.Member]
 B --> C[Chargement des traductions
database.Member]
 C --> D[Recherche dans l'index
database.index.Index]
 D --> E{?}
 E -- Non-chargé --> F[Chargement des données
database.Member]
 F -- Erreur --> G[Objet introuvable]
 E -- Chargé --> H[Copie des données
database.index.Index]
 H -- Erreur --> G
 H --> I[Attribution des données
database.Foo]
 I -- Erreur --> J[Format invalide]
 I --> K[Enregistrement dans l'index
database.Foo]
 K --> L[Attribution des variables tierces
database.Foo]
 L -- Erreur --> M[État de l'objet invalide]
 L --> N[Objet chargé !]

```

Intéressons-nous au rôle de l'index. Ce système, initialement absent de la conception, a été établi suite aux premiers tests unitaires du système, qui révélaient un temps d'accès aux données trop important lié à des redondances multiples dans les accès effectués. Il part du principe suivant : un objet lu une fois sur la base de donnée et non-modifié depuis n'a pas lieu d'être lu une seconde fois – économisant dès lors une requête sélection. Son fonctionnement est du reste assez simple : à chaque nouvelle instanciation, une référence vers l'objet chargé est enregistrée sur la base de données et retournée en cas d'utilisation ultérieure. Cette

méthode présente un autre intérêt, en terme de fiabilité cette fois : un objet de la base de donnée ne peut posséder deux instances le représentant en mémoire, étant donné que le lien est le même ; de la sorte, les valeurs qu'il possède sont systématiquement à jour, et ce quelque soit l'élément qui y accède.

*Vérifié lors des tests unitaire, ce système a permis de supprimer près de 70% des requêtes vers la base SQL.*



Ce second schéma présente le processus suivi lors de la mise à jour d'un objet. Rien de particulier à noter ici, si ce n'est que la mise se fait de manière manuelle, sur demande de la classe finale, et implique donc que le développeur doit appeler la méthode correspondante. La validité des valeurs transmises est testé en premier lieu par l'objet lui-même, puis, en terme de compatibilité avec la base de données par la classe database.Member.

Ce dernier schéma détaille le cheminement suivi lors du processus de suppression. Ici encore, rien de très particulier. On notera seulement que la suppression d'un objet entraîne systématiquement l'exécution d'opérations liées – utiles pour maintenir la cohérence de la base de données – pouvant éventuellement retourner une erreur annulant toute l'opération. L'index est mis à jour une fois la suppression effectuée.

## Langues & Pays

Penchons-nous à présent sur la représentation des langages de traduction et des pays compatibles avec le site. Même si ces deux entités n'ont, en terme de conception, qu'un rapport tout relatif, elles fonctionnent de manières assez identiques.

Dans les deux cas, on distingue un objet parent (`lang.Language` dans un cas et `location.Country` dans l'autre) et des objets héritant représentant tantôt les langages supportés (`lang.French`, `lang.English`, etc.), tantôt les pays compatibles (`location.France`, `location.Belgium`, etc.). Dans les deux cas, il est possible d'obtenir une liste complète des classes héritées par le biais de la classe parente (utile dans de nombreux cas).

Par ailleurs, les objets du module `location` possèdent tous un fichier de configuration comprenant, entre autres :

- La taille et la syntaxe des numéros de téléphone ;
- La taille et la syntaxe des numéros de fax ;

- Le préfixe international appliqué ;
- La taille et la syntaxe des numéros de TVA ou d'inscription au registres de commerces ;
- La taille et la syntaxe des codes postaux.

Ceci permettant de contrôler les adresses ou coordonnées attribuées au pays correspondant – dans le cas des entreprises, par exemple, où individus y résident.

Les objets du module `lang`, quand à eux, ne sont caractérisés que par leur nom et leur code. Leur utilité ne s'applique concrètement qu'aux traductions. On notera, à ce sujet, l'existence d'une interface `lang.Translatable`, dont on fait état dans la section suivante.

## Traductions

La gestion des traductions se veut somme toute assez simple mais mérite néanmoins qu'on s'y attarde ici. On part du principe qu'à tout élément littéral du site, bouton, texte, menu, titre, mot-clef, etc. correspond un ensemble de traductions, enregistré dans la base de données, éventuellement sous format *BBCode*, et pouvant être modifié voir supprimé par les administrateurs du système.

Pour simplifier l'accès à ces données, on fait usage du concept de *set* de traductions, traduit concrètement par l'objet `database.Set`. Un *set* correspond à un objet textuel du site et à ses différentes traductions dans les langues disponibles, par exemple :

```
Set de traductions [Utilisateur]

FR → Utilisateur
EN → User
NL → Gebruiker
```

De cette manière, lors de la génération des pages du site, on fait appels directement aux *sets*, et non aux traductions proprement dites, la valeur finale retournée dépendant du langage de la session.

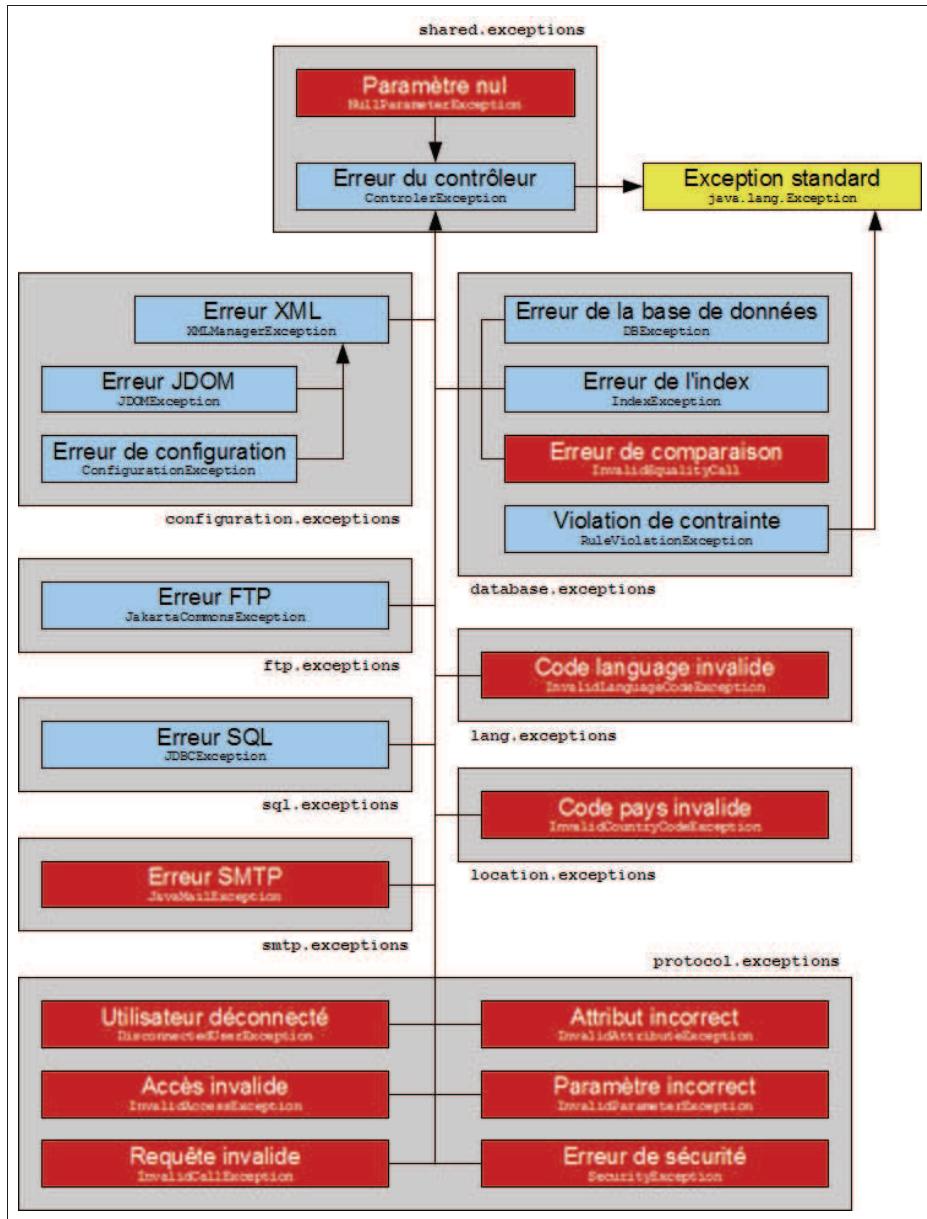
Les différents *sets* de traductions sont initialisés et accessibles par le biais des objets qu'ils concernent. Par exemple, les termes « Login », « Mot de passe » et « Administrateur » seront initialisés dans l'objet `database.User`, alors que « Lundi » et « Janvier » se trouvent dans l'objet `date.DateDriver`. Le fait qu'un objet puisse instancier des traductions se concrétise par l'usage de l'interface `lang.Translatable`, qui exige alors de celui-ci les services suivants :

- Initialisation des traductions liées à l'objet (en bloc, dans la base de données) ;
- Chargement des traductions liées à l'objet (requêtes de sélection auprès de la base de données) ;
- Suppression des traductions liées à l'objet.

Ce dernier service ne sert toutefois que dans le cadre du développement.

Le principal avantage de ce système tient dans l'idée que l'administrateur final du système est en mesure de modifier, librement et simplement, un élément de traduction, sans avoir à passer par une édition des fichiers source. Il en est de même pour le contenu des pages.

## Gestion des exceptions



L'application utilise une hiérarchie d'exceptions organisée, on l'a vu, par module d'appartenance. Le schéma ci-contre résume l'arborescence de ces dernières ; en bleu, les exceptions sujettes à l'héritage ; en rouge, les exceptions finales.

De façon théorique, la conception prévoit que toutes les erreurs possibles, qu'elles soient liées aux règles de gestion, aux librairies utilisées, ou encore au comportement du serveur soient traitées et éventuellement retournée par le biais d'exceptions personnalisées.

Intéressons-nous à deux particularités de l'arborescence.

Premièrement, l'élément « Paramètre nul » vise à simplifier les appels ou instantiation invalides à cause d'un paramètre prenant la valeur `null`. L'idée est de prévenir, par ce biais, l'apparition des erreurs de type `NullPointerException`, couramment source de problèmes insolubles durant le développement.

Deuxièmement, les exceptions classées comme « Violation de contrainte », correspondant à une erreur d'argument ou de paramètre liée à la violation d'une règle de gestion établie (et non à un dysfonctionnement technique). On remarquera que celles-ci

n'appartiennent pas à la catégorie des « Erreurs de contrôleur », au contraire de toutes les autres, mais constituent une catégorie à part. Le but de cette démarche est de veiller à ce que tous les problèmes pouvant provenir des erreurs de l'utilisateur final soient correctement traités, via des ajustements automatique ou l'usage d'objet du module `database.mistake` (ces derniers ne sont d'ailleurs pas des exceptions à proprement parler et n'apparaissent donc pas dans ce diagramme).

## Génération du code HTML

Comme exprimé dans le cahier des charges, le code HTML retourné sur les pages n'est pas exprimé directement au sein de la source Java, du moins pas sous sa forme textuelle. Il est fait appel une nouvelle fois à la librairie JDOM, permettant de façon simple la manipulation d'un arbre XML ; le projet initial spécifiait la créations d'objets spécifiques à chaque balise HTML, mais cette idée a été abandonnée faute de temps et d'intérêt réel – la librairie fournie dans ce but par le JDK n'a pas été non plus utilisée, car trop imprécise.

L'obtention d'objet HTML pré-configurés se fait par l'intermédiaire de deux classes :

- `html.Library`, pour les éléments de base, dotés d'un minimum de configuration ;
- `html.Tools`, pour les éléments plus évolués : formulaires, boutons, etc.

Le module `html` contient par ailleurs diverses classes représentant les objets les plus complexes, dont, notamment, les tableaux destinés à la manipulation des données.

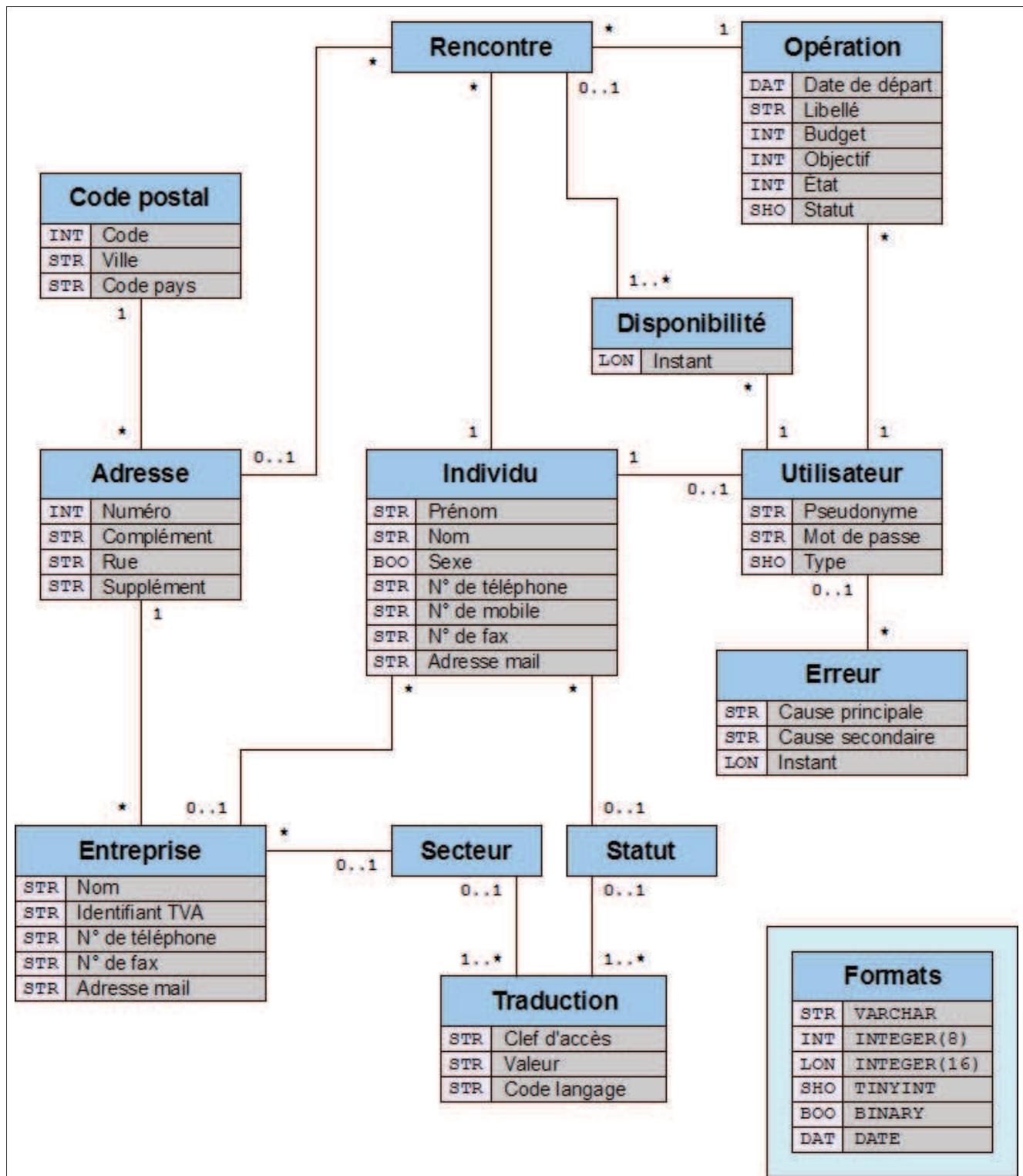
L'objectif suivi par cette méthode, comme expliqué précédemment, est de limiter autant que possible les erreurs de syntaxe et d'attribuer des comportements génériques à chacun des objets HTML.

*Pour l'anecdote, on peut mentionner une autre raison au développement de ce système, mineure et purement pratique : le portable que j'utilise, acheté à l'étranger, ne possède pas la touche « < > » située normalement à droite de « Shift Left ». En conséquence de quoi, pour ouvrir ou fermer une balise manuellement, il m'est nécessaire de changer manuellement la disposition du clavier au moyen de « Alt-Shift » – ce qui, vous en conviendrez, devient rapidement un calvaire lors de la rédaction d'un code XML un tant soit peu complexe ! Or, l'utilisation de JDOM s'affranchit complètement de ce caractère et présente donc un intérêt certain à mes yeux.*

### 3. Base de données

Résumé

|                             |    |
|-----------------------------|----|
| Présentation.....           | 33 |
| Liste des tables.....       | 33 |
| Adresses.....               | 33 |
| Entreprises.....            | 33 |
| Disponibilités.....         | 34 |
| Rencontres.....             | 34 |
| Erreurs.....                | 35 |
| Individus.....              | 35 |
| Opérations.....             | 36 |
| Codes postaux.....          | 36 |
| Secteur d'activité.....     | 37 |
| Statuts professionnels..... | 37 |
| Traductions.....            | 37 |
| Utilisateurs.....           | 38 |
| Remarques.....              | 38 |



## Présentation

Le schéma de la page précédente présente le modèle conceptuel de données réalisées pour le projet. Pour chaque table, il faut également prendre en compte l'existence d'un identifiant, unique, de type `INTEGER(8)`, compatible avec le système d'accès aux données présenté au chapitre précédent. Les liens relationnels donnent également lieu à des entrées de type `FOREIGN KEY`, également absentes du schéma mais découlant directement des multiplicités utilisées. Enfin, on notera que la taille attribuée aux éléments de type `VARCHAR` dépend directement du contenu concerné, dont nous ne préoccupons pas plus ici.

Conformément au système d'accès au donnée, et comme nous en reparlerons dans la partie « Développement », à chaque table correspond une classe du module `database`, dotée d'une méthode `init()`, dans laquelle la commande permettant d'instancier la base de donnée apparaît sous forme littérale. Nous vous invitons donc à consulter directement la source pour tout détail qui n'apparaîtrait pas dans ce mémoire.

## Liste des tables

La section suivante résume pour chaque table chacune des données mémorisées ainsi que les données relationnelles – choix de gestion, décisions d'ordre technique, etc.

### Adresses

La table `Adresses` contient les adresses postales enregistrées sur le site.

- Le champ « Numéro » correspond au numéro de l'adresse dans sa rue ;
- Le champ « Complément » correspond à un éventuel cardinal multiplicatif (BIS, TER, etc.) lié au numéro précédent ;
- Le champ « Rue » contient le nom de la rue, avenue, boulevard ou équivalent ;
- Le champ « Supplément » concerne un éventuel complément d'adresse (numéro d'appartement, de bâtiment, etc.).

Au plan relationnel :

- Chaque adresse est liée à un et un seul code postal ;
- Une adresse peut être associée à des rencontres, même si ce n'est pas forcément le cas ;
- Une adresse peut être liée à une ou plusieurs entreprises (dans le cas d'un bâtiment partagé, par exemple) ;

### Entreprises

La table `Corporations` contient les différentes entreprises référencées sur le site.

## Mémoire de stage | Base de données

- Le champ « Nom » contient le libellé d'activité de l'entreprise ;
- Le champ « Identifiant TVA » contient le numéro de TVA de l'entreprise ou, à défaut, son numéro d'inscription aux registres du commerce (parfois identique) ;
- Le champ « Numéro de téléphone » indique le téléphone principal (secrétariat) de l'entreprise ;
- Le champ « Numéro de fax » indique le numéro de fax principal de l'entreprise ;
- Le champ « Adresse mail » indique l'adresse mail principale (secrétariat) de l'entreprise.

Au plan relationnel :

- Une entreprise est toujours liée à une adresse indiquant son siège social ;
- Une entreprise peut posséder un secteur d'activité, mais ce renseignement est optionnel ;
- Une entreprise peut être liée à des individus, indiquant qu'ils en sont salariés ou associés.

## Disponibilités

La table `Disponibilites` contient la liste des créneaux horaires définis par des clients comme étant disponibles pour la prise de rendez-vous dans le cadre de leurs opérations en cours.

- Le champ « Instant » indique le point de départ de la disponibilité sur l'échelle de temps définie en configuration (dans le projet final, 30 minutes).

Au plan relationnel :

- Une disponibilité est toujours liée à un utilisateur (et plus précisément un client) indiquant à qui elle s'applique ;
- Une disponibilité peut être liée à une rencontre (elle devient dès lors occupée).

## Rencontres

La table `Encounters` possède un rôle de classe-association, permettant de définir une rencontre entre un client de l'entreprise et un tiers dans le cadre d'une opération donnée. Au plan relationnel :

- Une rencontre possède toujours un utilisateur (et plus précisément un client) indiquant à qui elle bénéficie ;
- Une rencontre possède toujours individu tiers ayant pris le rendez-vous en réponse à une des opérations du client lié ;
- Une rencontre possède un ou plusieurs créneaux horaires définis par le client et indiquant le moment où elle doit se tenir ;

- Enfin, une rencontre peut être liée à une adresse, indiquant dès lors en quel lieu elle doit se tenir.

## Erreurs

La table `Errors` recense les erreurs survenues dans le cadre d'une utilisation normale du site (on s'en doute, elle n'a jamais servi :-)).

- Le champ « Cause principale » contient le message d'erreur lié à l'exception projetée ;
- Le champ « Cause secondaire » contient le message d'erreur lié à la cause de l'exception projetée, si elle existe ;
- Le champ « Instant » indique l'heure et la date (en *UNIX-Time*) de l'erreur.

Sur le plan relationnel :

- Une erreur peut éventuellement être liée à un utilisateur, le cas échéant « responsable » de cette dernière.

## Individus

La table `Individuals` contient la liste des personnes physiques enregistrées sur le site : utilisateurs et tiers.

- Le champ « Prénom » contient le ou les prénoms de l'individu ;
- Le champ « Nom » correspond à son nom de famille ;
- Le champ « Sexe » indique s'il s'agit d'un homme ou d'une femme. Pour information, suite à des problèmes techniques avec JDBC, le format de stockage `BINARY` a été abandonné au profil d'un entier de taille courte basé sur la norme ISO-5281.
- Le champ « Numéro de téléphone » indique le numéro de téléphone personnel de l'individu (optionnel) ;
- Le champ « Numéro de mobile » indique le numéro de téléphone portable personnel de l'individu (optionnel) ;
- Le champ « Numéro de fax » indique le numéro de fax personnel de l'individu (optionnel) ;
- Le champ « Adresse mail » contient l'adresse e-mail personnelle de l'individu, utilisée dans le cas où celui-ci est lié à un utilisateur comme adresse du compte.

Au plan relationnel :

- Un individu peut être lié à des rencontres – il y fait alors office de tiers ;

## Mémoire de stage | Base de données

- Un individu peut éventuellement être lié à un compte utilisateur ;
- Un individu peut être lié à une entreprise, indiquant alors qu'il en fait partie par salariat ou association.

## Opérations

La table Operations contient l'ensemble des opérations réalisée par l'entreprise pour le compte de clients.

- Le champ « Date de départ » correspond au jour où doit normalement débuter l'opération ;
- Le champ « Libellé » contient le titre et diverses informations sur l'opération ;
- Le champ « Budget » contient le montant initial, en euros, alloué à l'opération. Ce budget peut, comme on l'a précédemment expliquer, évoluer au fur et à mesure de la prise de rendez-vous (le montant final étant alors défini par calcul) ;
- Le champ « Objectif » indique le nombre de contacts visés par l'opération (optionnel) ;
- Le champ « État » indique le nombre de contacts déjà réalisés dans le cadre de l'opération, sous réserve que celle-ci possède un objectif chiffré ;
- Le champ « Statut » indique si l'opération est en cours, en pause, ou terminée (éventuellement prématièrement).

Au plan relationnel :

- Une opération peut être liée à des rencontres prévues suite au travail de promotion réalisé ;
- Une opération est toujours liée à un utilisateur (systématiquement un client) indiquant à qui elle bénéficie.

## Codes postaux

La table PostalCodes contient la liste des codes postaux enregistrés sur le système.

- Le champ « Code » indique le numéro national du code postal (sousmis à une syntaxe découlant du pays concerné) ;
- Le champ « Ville » indique le nom de la ville associée au code postal ;
- Le champ « Pays » indique l'identifiant du pays selon la norme ISO-3166-1 Alpha 3.

Au plan relationnel :

- Un code postal peut être lié à une ou plusieurs adresses.

## Secteur d'activité

La table `Sectors` prend le rôle d'une classe-association et contient la liste des secteurs d'activité enregistrés sur le site. Au plan relationnel :

- Un secteur d'activité peut être lié à une ou plusieurs entreprises ;
- Un secteur d'activité est lié à une ou plusieurs traductions permettant d'obtenir son nom dans les différentes langues du site.

## Statuts professionnels

La table `Status` prend le rôle d'une classe-association et contient la liste des statuts professionnels enregistrés sur le site. Au plan relationnel :

- Un statut professionnel peut être lié à un ou plusieurs individus ;
- Un statut professionnel est lié à une ou plusieurs traductions permettant d'obtenir son nom dans les différentes langues du site.

## Traductions

La table `Translations` contient la liste des contenus traduisibles enregistrés sur le site (voir le chapitre précédent).

- Le champ « Clef d'accès » permet de grouper les éléments issus de traductions possédant un sens identique et/ou destinés à un emplacement identique sur le site ;
- Le champ « Valeur » contient le contenu textuel de la traduction ;
- Le champ « Code langage » contient la référence de la langue à laquelle correspond le pays selon la norme ISO-639-1.

Au plan relationnel :

- Une traduction peut être liée à un secteur d'activité (et auquel cas, seulement à celui-ci) ;
- Une traduction peut être liée à un statut professionnel (idem) ;
- Enfin, même s'il ne s'agit pas d'une relation de données à proprement parler, une traduction peut être liée (c'est même le cas général) à une classe implémentant l'interface `Translatable` et y accédant au moyen de sa clef d'accès.

## Utilisateurs

La table `Users` contient l'ensemble des utilisateurs enregistrés au système et capables, donc, de s'y connecter par le biais de leurs identifiants.

- Le champ « Pseudonyme » contient le *login* utilisé pour établir une connexion ;
- Le champ « Mot de passe » contient le *hash* du mot de passe utilisé pour établir la connexion ;
- Le champ « Type » indique si l'utilisateur est considéré comme un client, un prospecteur ou un administrateur.

Au plan relationnel :

- Un utilisateur est toujours lié à un individu, à qui ont été confié les identifiants d'accès au compte ;
- Un utilisateur peut être lié à des erreurs qu'il aurait « provoqué » sur le site ;
- Un utilisateur de type client peut être lié à des opérations dont il bénéficie ;
- Un utilisateur de type client peut être lié des disponibilités qu'il a défini.

## Remarques

La base de donnée et le code SQL permettant son initialisation se veulent simplifiés autant que possible : le but de ce choix étant de réaliser un maximum des traitements de contrôle en amont, dans les sources Java, afin de réserver aux erreurs purement techniques les dysfonctionnements possibles de JDBC.

Pour l'information, l'ensemble des contenus textes de la base sont encodés selon le format UTF-8 ; les tables bénéficient toutes d'un index primaire ordonné sur leur champ `ID`.

## 4. Développement

### Résumé

|                                     |    |
|-------------------------------------|----|
| Introduction.....                   | 40 |
| Exécution d'une page.....           | 42 |
| Exécution de traitements.....       | 45 |
| Chargement des données.....         | 47 |
| Chargement d'une configuration..... | 49 |
| Remarques d'ordre technique.....    | 50 |

## Introduction

Ce chapitre propose plusieurs extraits du code choisis en fonction de leur intérêt, en temps qu'exemple des pratiques mises en œuvre durant le développement, ou de par leur criticité dans l'exécution de l'application. Il n'est évidemment pas possible de commenter l'intégralité des sources réalisées ; des informations supplémentaires sont toutefois disponibles dans la Javadoc jointe en annexe de ce rapport.

Nous nous pencherons ici sur quatre extraits du code :

- Premièrement, le processus global suivi lors de l'appel à une *servlet* du site, les opérations suivies, dont un petit traitement lié au changement de langue ;
- Deuxièmement, un exemple de traitement réalisé par une des pages du site, incluant un accès à la base de donnée et sa modification ;
- Troisièmement, une opération d'instanciation d'un objet (le cas échéant, un utilisateur) possédant une instance sur la base de donnée ;
- Quatrièmement, la création d'un gestionnaire de donnée, longuement évoquée dans le chapitre « Conception ».

Chaque extrait de code est présenté dans son intégralité – avec, au besoin, des sous-sections pour les méthodes les plus intéressantes – et légendé dans les pages qui suivent.

```

public final void doGet(final HttpServletRequest sessionRequest, final HttpServletResponse protocol.Session) {
 try {
 new SQLDriver();
 SQLDriver.connect();
 sessionRequest.setCharacterEncoding(Session.CHARACTER_ENCODING);
 this.request = sessionRequest;
 this.response = sessionResponse;
 this.session = sessionRequest.getSession();
 this.load();
 this.checkSwitch();
 }

 private void checkSwitch() throws ControlerException {
 if (this.isParameter("switch")) {
 if (Language.isCode((String) this.getParameter("switch"))) {
 try {
 this.setLanguage(Language.getLanguage((String) this.get...
 } catch (NullParameterException e) { }
 }
 }
 }

 this.checkLogin();
 this.checkLogout();
 this.checkHelp();
 this.loadTranslations();
 if (!this.checkAccess()) {
 throw new InvalidAccessException();
 }
 this.actualize();
 this.execute();
 this.fill();
 this.response.setCharacterEncoding(Session.CHARACTER_ENCODING);
 XMLOutputter out = new XMLOutputter(format);
 out.output(this.encapsulate(), this.response.getWriter());
 SQLDriver.disconnect();
} catch (ControlerException e) {
 try {
 if (this.isUser()) {
 new Error(e, DateDriver.getDate(), this.getUser());
 } else {
 new Error(e, DateDriver.getDate());
 }
 } catch (Exception f) {
 throw new ServletException("Critical error, please check help", f);
 }
}
}

```

1  
2  
3  
4  
5  
6  
6.1  
6.2  
6.3  
6.4  
6.5  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20

## Exécution d'une page

Cet exemple présente le processus standard suivi par l'application lors de l'exécution d'une page du site par le biais de la méthode héritée `doGet()` de l'objet `javax.HttpServlet()`, l'objet `protocol.Session` en héritant.

- 1 La fin de cette ligne correspond aux paramètres standard de la méthode `HttpServlet.doGet()`.
- 2 Ces deux commandent instantient le pilote SQL et établissent une connexion avec le serveur de données.
- 3 On règle l'encodage des caractères reçus sur la valeur standard du site (UTF-8). Cette commande se retrouve à l'alinéa 13, pour les caractères envoyés cette fois.
- 4 On récupère et on stocke dans une variable indépendante les informations de session.
- 5 La méthode `protocol.Session.load()` permet d'initialiser plusieurs variables utiles dans l'exécution de la page sur des valeurs par défaut, au besoin.
- 6 La méthode `protocol.Session.checkSwitch()` permet de vérifier une requête de changement dans le langage du site. Observons-la plus en détail.
  - 6.1 La méthode peut projeter des erreurs de type `ControlerException`, car elle manipule des objets de la base de données. Toutefois, étant donné que ces erreurs n'ont pas lieu d'exister dans le cadre d'une exécution normale, elles sont transmises normalement.
  - 6.2 Cette fonction teste si un paramètre de type `_GET` ou `_POST` de nom `switch` existe dans la requête courante. Si c'est le cas, c'est que l'utilisateur souhaite procéder à un changement de langue.
  - 6.3 On détermine si la valeur du paramètre `switch` correspond à un code langage compatible ; si ce n'est pas le cas, on interrompt l'opération.
  - 6.4 On procède au changement via la méthode `setLanguage()`.
  - 6.5 L'exception `NullParameterException`, présentée précédemment, survient lorsqu'un paramètre nul est passé en argument d'une fonction du site. Ici, les arguments qu'on utilise sont sûrs et éprouvés, donc on intercepte l'exception.
- 7 Cette méthode vérifie, sur un modèle proche de la précédente, si l'utilisateur fait une demande de connexion auprès du système.
- 8 Cette fonction effectue le même travail, en cas de déconnexion cette fois.
- 9 Enfin, cette méthode détermine si l'utilitaire souhaite activer ou désactiver l'aide contextuelle (voir plus loin).
- 10 On appelle la méthode `lang.Translatable.loadTranslations()`, décrite précédemment, qui indique à la page en cours de charger les éléments traduisibles lui étant liés (contenus, formulaires, aides, etc.).
- 11 Cette instruction teste si la page courante est exécutable à partir du statut courant de l'utilisateur (visiteur, client, prospecteur ou administrateur). Si ce n'est pas le cas, une exception de type `InvalidAccessException` est projetée.
- 12 Ici, on fait appel à la méthode abstraite `protocol.Session.actualize()`. Elle permet d'effectuer un rafraîchissement, si nécessaire, des informations de la page (données, vues, formulaires, etc.).
- 13 Cette ligne entraîne les traitements (éventuels) à effectuer avant affichage définis dans la méthode

abstraite protocol.Session.execute().

14 Cette ligne entraîne la création des objets HTML par la classe finale au moyen de la méthode abstraite protocol.Session.fill().

15 Voir l'alinéa 3.

16 Ces deux lignes entraînent la rédaction du code HTML final et son renvoi à l'écran via la librairie JDOM et les fonctions par défaut du JDK.

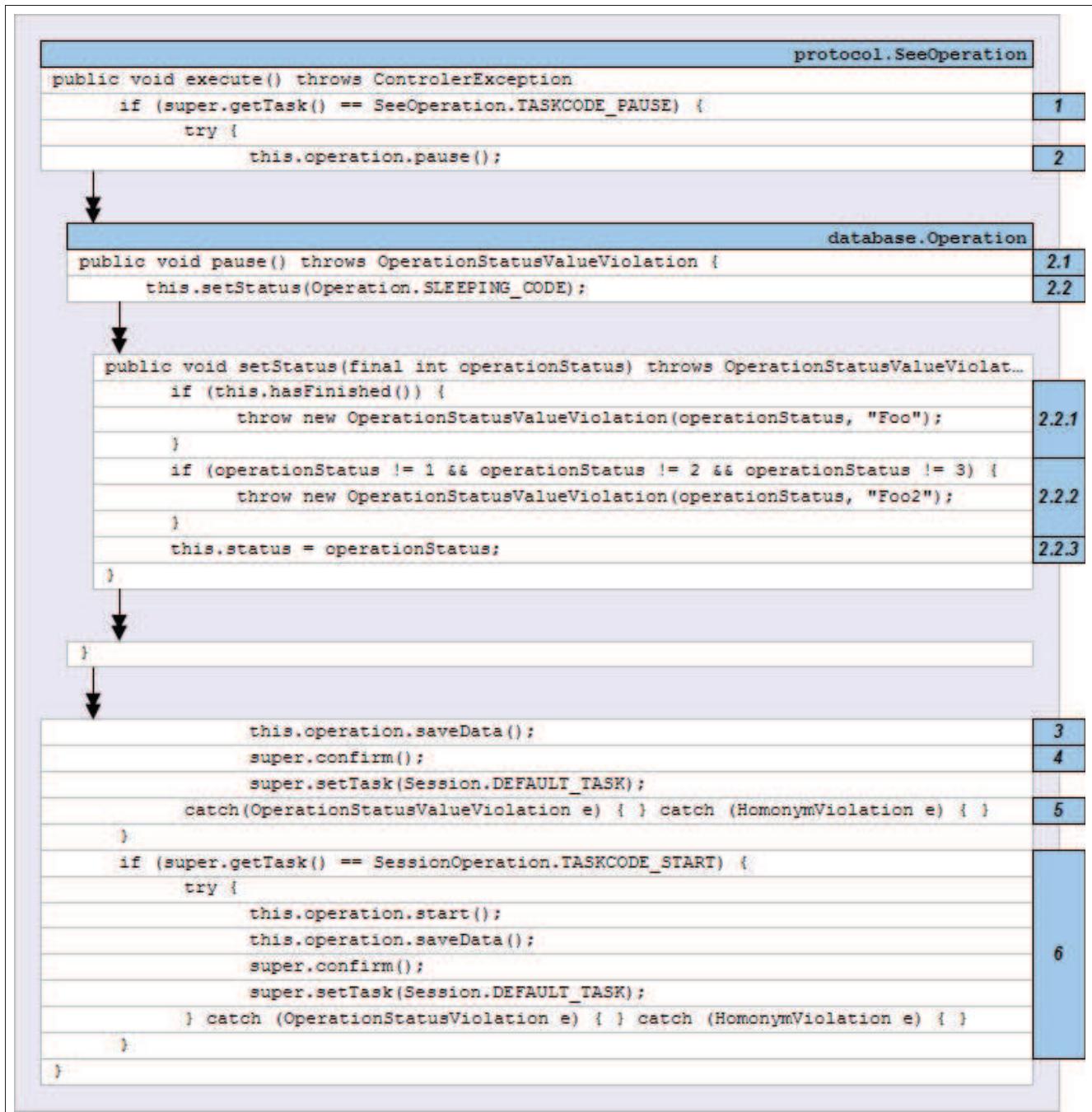
17 On effectue la déconnexion du serveur SQL.

18 Le but de ce bloc est d'attraper et d'enregistrer d'éventuelles erreurs du contrôleur s'étant produites durant la lecture de la page. On notera que dans l'état actuel du site, les erreurs ne sont pas soumises à un traitement particulier autre que leur indexation.

19 Ici, on procède à l'enregistrement avec ou sans signature. On notera que la simple instanciation de l'objet database.Error suffit à créer une entrée sur la base correspondante, sans besoin d'appeler une autre fonction.

20 En cas d'échec lors de l'indexation, on transmet l'erreur telle quelle à l'utilisateur (affichage de la *stack trace* à l'écran).

Dans l'exemple suivant, on se penchera sur le fonctionnement de protocol.Session.execute(), rencontrées ici à l'alinéa 13.

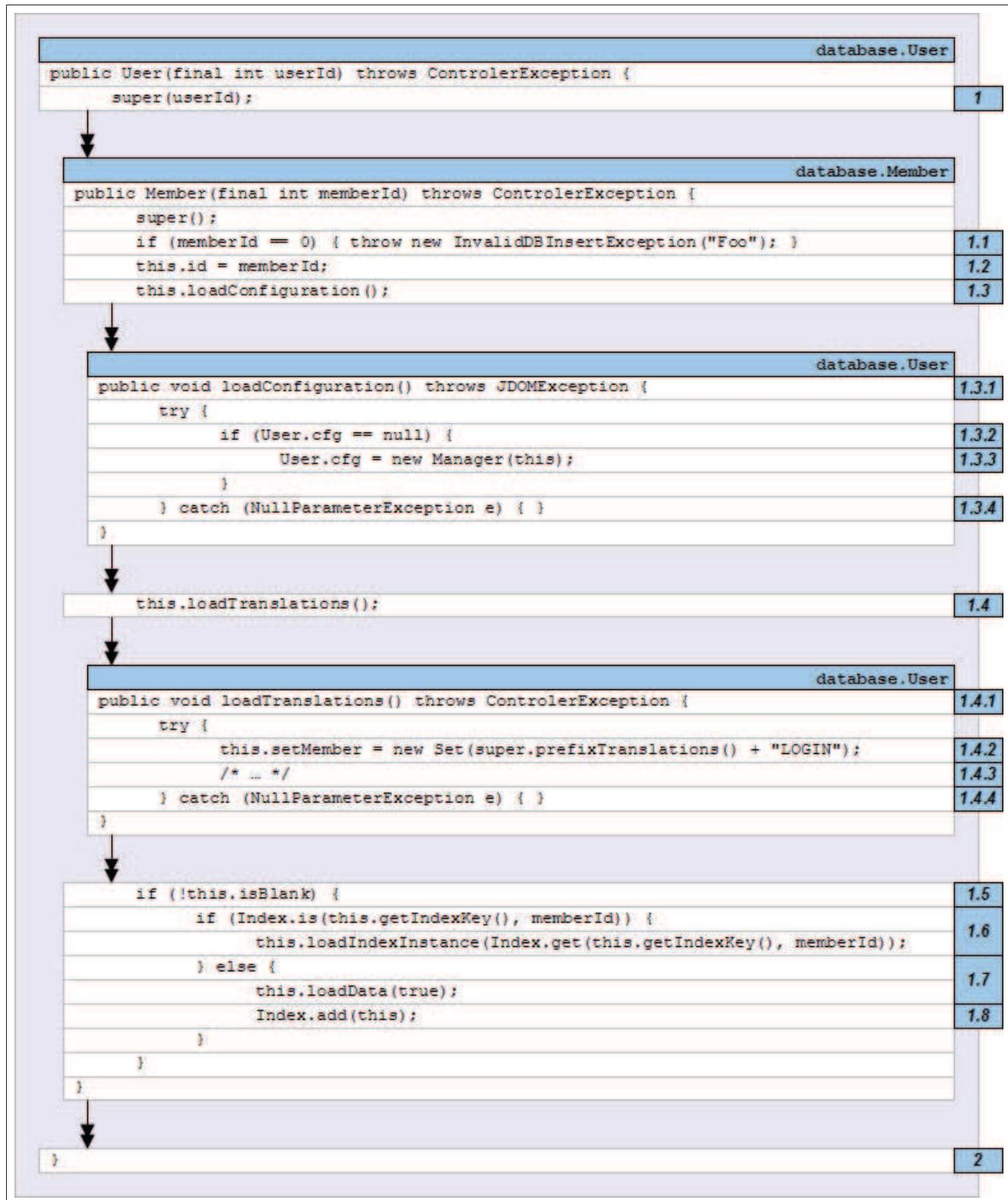


## Exécution de traitements

Dans cet exemple, on étudie le fonctionnement de la méthode abstraite `protocol.Session.execute()` appliquée à l'objet `protocol.SeeOperation()`.

La classe `protocol.SeeOperation()` hérite de `protocol.Session` (c'est donc une page, ou *servlet finale* du site). Elle permet à un client de visualiser les opérations dont il bénéficie, et les gérer dans une certaine mesure. Par ses liens d'héritage, elle doit implémenter la fonction `execute()`, qui contient l'ensemble des traitements produits par la page.

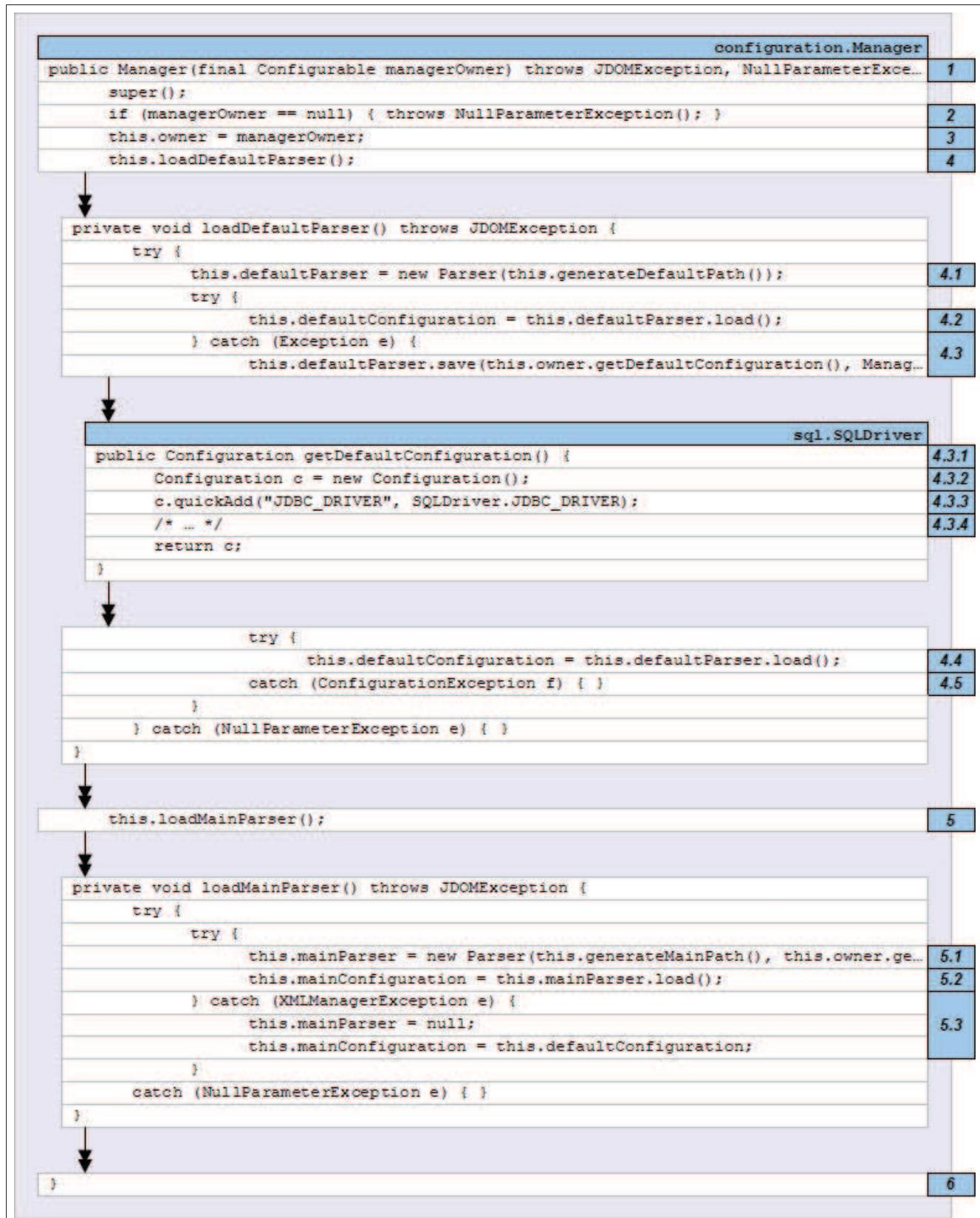
- 1 La méthode `protocol.Session.getTask()` permettant d'obtenir un identifiant chiffré, désigné sous le nom de tâche, correspondant à l'action requise par l'utilisateur lors de sa dernière requête. Ici, on regarde si elle correspond à « Mettre l'opération en pause ».
- 2 Si c'est effectivement le cas, on appelle la méthode `database.Operation.pause()`, qui permet de mettre en pause l'opération concernée (ici `this.operation`).
  - 2.1 La méthode `pause()` projette l'exception `OperationStatusValueViolation` dans le cas où il soit impossible de mettre l'opération en suspens.
  - 2.2 On appelle la méthode `Operation.setStatus()`, qui permet de régler le statut de l'opération (à savoir, en cours, en pause, ou terminée). Ici, on lui attribue la valeur `pause`.
    - 2.2.1 Lors du changement de statut, il est nécessaire de valider l'opération par plusieurs tests. Le premier d'entre eux vérifie que l'opération n'est pas déjà terminée par le biais de la méthode `Operation.hasFinished()`.
    - 2.2.2 Second test, visant à déterminer si le nouveau statut possède une valeur correcte.
    - 2.2.3 Enfin, une fois les tests passés sans encombre, on attribue la nouvelle valeur.
- 3 De retour dans la méthode principale, on effectue l'enregistrement des changements effectués par le biais de la méthode `Member.saveData()` - dont `Operation` dispose par héritage. Comme expliqué précédemment, toute mise à jour de la base de donnée doit se faire par un appel explicite dans le code.
- 4 Cette petite méthode ajoute une notification de réussite en haut de l'écran.
- 5 Enfin, on attrape deux erreurs à la volée.  
`HomonymException` se produit lors d'un cas d'égalité parfaite entre deux membres de la base de données ; ici, on est sûr que le cas ne se produira pas, on ne transmet donc rien.  
`OperationStatusValueViolation`, mentionné à l'alinéa 2.1, se produit lorsque le changement de statut échoue à cause d'une valeur invalide. Ici, étant donné qu'on fait appel à la méthode `pause()` – d'où une valeur demandée systématiquement correcte – et qu'un client ne peut pas modifier une opération déjà achevée, on ne donne pas suite.
- 6 Ce cas, exactement similaire au précédent, vise à démarrer ou redémarrer l'opération.



## Chargement des données

Cet exemple illustre la façon dont il est possible d'instancier un objet enregistré sur la base de donnée par le biais de la classe `database.User` (héritant de `database.Member`).

- 1 Le chargement se fait par le biais d'un paramètre unique : l'identifiant, ou clef primaire, de l'objet sur la base de données (ici `userId`).
  - 1.1 L'essentiel du travail s'effectue en amont. On commence par vérifier que l'identifiant demandé n'est pas nul (en théorie, il ne doit pas être non plus négatif, mais on verra plus tard que c'est parfois le cas).
  - 1.2 On mémorise l'identifiant de l'objet.
  - 1.3 On procède au chargement de la configuration. La méthode `Member.loadConfiguration()` est abstraite, et donc définie dans la classe finale.
    - 1.3.1 Le chargement de la configuration peut éventuellement projeter des erreurs JDOM, en cas d'erreur de chargement (voir le chapitre « Conception »).
    - 1.3.2 On commence par vérifier que la configuration n'a pas déjà été chargée. Pour des motifs logiques, la configuration d'un objet est systématiquement instanciée sous forme d'attribut statique. Il n'est donc pas nécessaire de le charger plus d'une fois, à la première instantiation d'un objet de la classe.
    - 1.3.3 Le chargement de la configuration ne prend qu'un argument : l'objet lui-même. A partir de ce dernier, le gestionnaire (`configuration.Manager`) pourra déterminer à quel fichiers se référer.
    - 1.3.4 Comme la plupart des objets de l'application, le gestionnaire de configuration vérifie que l'instanciation ne se fait pas à partir d'un paramètre nul. Ici, on est sûr que ce n'est pas le cas ; l'exception est donc interceptée sans plus de poursuite.
  - 1.4 On procède au chargement des traductions. La méthode `Member.loadTranslations()` est, tout comme pour les configurations, abstraite et donc définie dans la classe finale.
    - 1.4.1 Le chargement des traductions nécessite un ou plusieurs accès à la base de donnée ; l'erreur `ControlerException` est donc possible.
    - 1.4.2 Cette ligne contient l'instanciation « standard » d'un *set* de traductions. Celle-ci se fait par le biais d'un argument unique, à savoir la clef d'accès. Le cas échéant, on cherche les traductions associées au mot « Login ».
    - 1.4.3 Suivent un certain nombre de lignes similaires à la précédente.
    - 1.4.4 Voir 1.3.4.
  - 1.5 Ce test vérifie que l'instanciation de l'objet ne se fait pas « à blanc ». Cette fonctionnalité permet d'instancier un objet sans données (pour accéder à sa configuration ou à ses traductions).
  - 1.6 On recherche si l'objet est déjà chargé dans l'index, et, le cas échéant, on procède à la copie des données.
  - 1.7 Sinon, on effectue le chargement manuel des données ...
  - 1.8 ... et on enregistre le résultat dans l'index.
- 2 L'instanciation de l'objet est terminée. Comme on peut le constater, la méthode finale est très pratique d'utilisation et concentre un maximum de code dans la classe `database.Member`. On notera tout de même que chaque objet héritant doit implémenter plusieurs méthodes pour fonctionner.



## Chargement d'une configuration

Cet exemple illustre le chargement d'un gestionnaire de configuration `configuration.Manager` par un pilote SQL `sql.SQLDriver` (on a choisi celui-là mais le fonctionnement est le même pour tous les objets de type `Configurable`).

- 1 On constate ici que l'instanciation d'un gestionnaire de configuration nécessite que son propriétaire, implémentant l'interface `Configurable`, se déclare. Pour plus de détails, voir l'alinéa 1.3 de l'exemple précédent.
- 2 On commence par vérifier la validité du propriétaire ; `NullParameterException`, rencontré précédemment à plusieurs reprises, rentre ici en jeu.
- 3 On mémorise ensuite une référence vers le propriétaire.
- 4 On procède ensuite au chargement du fichier de configuration par défaut. Penchons-nous sur le mécanisme de chargement.
  - 4.1 On commence par tenter un chargement du fichier par défaut. Concrètement assuré par l'instanciation de l'objet `configuration.Parser`, cette opération consiste à vérifier l'existence du fichier correspondant (indiqué par la méthode `generateDefaultPath()` et dépendant du propriétaire de la configuration) et créer les liens logiques nécessaires à sa lecture.
  - 4.2 On tente ensuite, si le chargement s'est bien passé, de charger la configuration par défaut au moyen de la méthode `configuration.Parser.load()`.
  - 4.3 Si la moindre erreur se produit durant les deux étapes précédentes (signifiant, *a priori*, un problème de syntaxe ou de variable), on écrase immédiatement le fichier de configuration à partir des réglages « machine » de l'objet. Ceux-ci s'obtiennent pas le biais de la méthode `configuration.Configurable.getDefaultConfiguration()` ; détaillons-la plus amplement.
    - 4.3.1 Comme on peut le constater, la génération des réglages machine ne doit générer aucune erreur (en soit, la seule erreur imaginable pourrait être lié à une homonymie de variables, cas peu plausible étant donné que le programmeur les déclare lui-même).
    - 4.3.2 On commence par créer une configuration vide ; comme vu précédemment, cette classe fonctionne de façon similaire à une liste, malgré quelques contraintes supplémentaires.
    - 4.3.3 L'ajout de variable se fait via `configuration.Configuration.quickAdd()`. On notera que cette méthode peut être vu comme la « voie rapide » ; en effet, en cas d'homonymie entre deux variables, la précédente valeur est écrasée. Il existe d'autres méthodes d'ajout soumises cette fois à contrôle ; ici, on s'en dispense, vu le faible risque d'erreur.
    - 4.3.4 S'en suit un certain nombre de lignes similaires à la précédente, une par variable de configuration à enregistrer.
  - 4.4 Une fois les valeurs restaurées, on effectue un nouveau chargement des valeurs par défaut ...
  - 4.5 Étant donné que le fichier est, pour ainsi dire, « neuf », on ignore les erreurs pouvant provenir de la configuration en elle-même (`ConfigurationException`). Toutefois, les problèmes d'I/O sont transmis normalement.
- 5 On procède ensuite au chargement du fichier de configuration principal. Celui-ci se fait d'une manière légèrement différente du précédent.
  - 5.1 Tout d'abord, on instancie le *parser* en incluant une configuration et une documentation par défaut,

au cas où le fichier aurait été supprimé – le cas échéant, il s'agit de la configuration par défaut précédemment chargée et la documentation fournie par l'objet. Le cas échéant, le fichier est recréé et rempli avant lecture.

- 5.2 On procède ensuite à la lecture du fichier et son attribution à la configuration principale.
- 5.3 En cas d'erreur liée à l'utilisation de JDOM ou de la configuration en elle-même, on abandonne le *parser* principal et on copie la configuration par défaut comme configuration principale de l'objet. De cette manière, le fichier principal défini par l'utilisateur n'est pas affecté ; il ne fonctionne juste pas.

6 Le chargement de la configuration est terminé.

Comme on peut le voir, la distinction entre les fichiers de configuration principaux et secondaires est bien respectée : dans le premier cas, la restauration est immédiate dès la première erreur ; dans le second, elle ne se produit que si le fichier n'existe pas ou est totalement vide.

*On ne s'étendra pas plus ici sur le système permettant d'utiliser des fichiers de configuration distants via une transaction FTP. Inabouti, il ne ferait qu'embrouiller l'exemple, mais est bel et bien fonctionnel, quoi que beaucoup trop lent pour une utilisation autre que locale.*

## Remarques d'ordre technique

L'ensemble du code réalisé respecte la syntaxe SUN telle que proposée par le *plugin Checkstyle* (à l'exception de la taille des lignes, dépassant parfois largement les 80 caractères). Le HTML retournée se plie sans erreur à la norme W3C Strict, tout comme les fichiers CSS et Javascript. Enfin, l'ensemble des sources possède une documentation complète, rédigée en Français par le biais de l'outil Javadoc.

## 5. Conclusion

### Résumé

|                           |    |
|---------------------------|----|
| Rendu final.....          | 52 |
| Bilan.....                | 57 |
| État du projet.....       | 57 |
| À propos de M. Rush ..... | 57 |
| Références.....           | 59 |

## Rendu final

Cette section présente quelques captures d'écran du travail fini. Je préfère ne pas m'attarder sur le CSS, bien qu'il y ait quelques petites choses à dire, ni sur la structuration du code HTML ; il ne s'agit là que d'une partie mineure du travail et le rendu final, me semble-t-il, parle suffisamment de lui-même.

The screenshot shows the Prospector software interface. At the top, there is a navigation bar with links for Accueil, Espace administrateur, Mon profil, and Contact. On the right, it shows a user is connected as Guillaume Zavan [Administrateur] with a 'Déconnecter' button and a link to disconnect. The main window has two tabs: 'Liens' (links) and 'Profil'. The 'Profil' tab is active and displays a sidebar with 'Mes informations'. The main content area shows various profile details:

|                    | Pseudonyme                              | Type           | Adresse mail      |
|--------------------|-----------------------------------------|----------------|-------------------|
| Nom                | Zavan                                   | Administrateur | webkawa@gmail.com |
| Prénom             | Guillaume                               |                |                   |
| Sexe               | Homme                                   |                |                   |
| N° Téléphone       | +(0032) 012345321                       |                |                   |
| N° Mobile          | +(0032) 0741258962                      |                |                   |
| N° Fax             | Non-renseigné                           |                |                   |
| Entreprise         |                                         |                |                   |
| Nom                | Prospector                              |                |                   |
| Secteur d'activité | Informatique                            |                |                   |
| Adresse            | 2588 avenue des Caribous 1051 Bruxelles |                |                   |
| Pays               | BELGIQUE                                |                |                   |
| Statut             | Stagiaire                               |                |                   |

At the bottom of the interface, there is a footer with the text: 'Prospector' - TVA : 0123456789  
2588 avenue des Caribous Bureau 4445 1051 Bruxelles BELGIQUE  
Téléphone : 0123456789 - Fax : 0123455777

Page profil (aide conceptuelle visible en haut à gauche)

## Mémoire de stage | Conclusion

Connected : Guillaume Zavan [Administrator] [Disconnect](#)

[Home](#) [Administrator board](#) [My profile](#) [Contact](#)

**Individuals list**

| Records 1 - 10 / 11 |                         |                          |                                          |                           |                                                                                                         |
|---------------------|-------------------------|--------------------------|------------------------------------------|---------------------------|---------------------------------------------------------------------------------------------------------|
|                     | Last name<br>First name | Contact                  | Mail address<br>Fax number               | Corporation Status        |                                                                                                         |
| M                   | BRUNET<br>Benjamin      | 0125632563<br>0588888888 | benjamin.brunet@free.fr<br>0288885558    | Prospector<br>Executive   | <a href="#">Edit</a> <a href="#">User account</a><br><a href="#">Delete</a> <a href="#">Corporation</a> |
| M                   | BUSH<br>George          | Undefined<br>Undefined   | gbush@us-gov.org<br>Undefined            | Undefined                 | <a href="#">Edit</a> <a href="#">Delete</a>                                                             |
| F                   | CLAUDE<br>Amandine      | Undefined<br>Undefined   | claude.amandine@hotmail.com<br>Undefined | Prospector<br>Designer    | <a href="#">Edit</a> <a href="#">User account</a><br><a href="#">Delete</a> <a href="#">Corporation</a> |
| M                   | JOHNSON<br>Bryan        | 0124567893<br>Undefined  | webkawa@gmail.com<br>0125874369          | Exxon Débile<br>Slave     | <a href="#">Edit</a> <a href="#">User account</a><br><a href="#">Delete</a> <a href="#">Corporation</a> |
| M                   | KUCKART<br>Ludovic      | Undefined<br>Undefined   | webmasteragents@gmail.com<br>Undefined   | Prospector<br>Developper  | <a href="#">Edit</a> <a href="#">User account</a><br><a href="#">Delete</a> <a href="#">Corporation</a> |
| M                   | MAXWELL<br>Maxime       | Undefined<br>Undefined   | max@max.com<br>Undefined                 | Prospector<br>Executive   | <a href="#">Edit</a> <a href="#">Corporation</a><br><a href="#">Delete</a>                              |
| M                   | MUADDIB<br>Paul         | 0741258963<br>Undefined  | pmd@dune.com<br>Undefined                | Undefined                 | <a href="#">Edit</a> <a href="#">Delete</a>                                                             |
| M                   | RUSH<br>Damien          | Undefined<br>Undefined   | damienrush@d-rush.be<br>Undefined        | Prospector<br>Director    | <a href="#">Edit</a> <a href="#">User account</a><br><a href="#">Delete</a> <a href="#">Corporation</a> |
| M                   | SARCASTIK<br>Nicolas    | Undefined<br>Undefined   | ns@french-gov.free.fr<br>Undefined       | Exxon Débile<br>Undefined | <a href="#">Edit</a> <a href="#">User account</a><br><a href="#">Delete</a> <a href="#">Corporation</a> |
| M                   | WALKER<br>Johny         | Undefined<br>Undefined   | kikoulol@lol.com<br>Undefined            | Grosoft<br>Undefined      | <a href="#">Edit</a> <a href="#">Corporation</a><br><a href="#">Delete</a>                              |
| M                   | ZAVAN<br>Guillaume      | 0123485321<br>0741258962 | webkawa@gmail.com<br>Undefined           | Prospector<br>Slave       | <a href="#">Edit</a> <a href="#">User account</a><br><a href="#">Delete</a> <a href="#">Corporation</a> |

[Filter](#)

**Tools**

**Create an individual**

Individual

Gender:

First name:

Last name:

Mail address:

[Create a corporation](#)

[Create a status](#)

'Prospector' - VAT : 0123456789  
2588 avenue des Caribus Bureau 4445 1051 Bruxelles BELGIUM  
Phone number : 0123456789 - Fax number : 0123455777

Écran administrateur, gestion des individus (version anglophone)

## Mémoire de stage | Conclusion

Connexion : Guillaume Zavan [Administrateur] [Déconnecter](#)

**prospector** | Accueil Espace administrateur Mon profil Contact

### Opération réussie

**Modifier**

**Mardi 7 Septembre 2010 / 10h > 15h**

Ce ou ces créneaux horaires sont actuellement assignés à un rendez-vous.

[Modifier un rendez-vous](#)

| Disponibilité |     |
|---------------|-----|
| Départ        | 10h |
| Fin           | 15h |

**Rencontre**

Client: SARCASTIK Nicolas  
 Individu: BRUNET Benjamin  
 Adresse: Non-renseigné  
 Opération: Non-renseigné

[Modifier](#)

[Annuler le rendez-vous](#)

**Planning**

| SARCASTIK Nicolas          | 8h | 9h | 10h    | 11h    | 12h    | 13h    | 14h    | 15h    | 16h    | 17h    | 18h    |
|----------------------------|----|----|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| Lundi 6 Septembre 2010     |    |    |        |        |        |        |        |        |        |        |        |
| Mardi 7 Septembre 2010     |    |    | Occupé |
| Mercredi 8 Septembre 2010  |    |    | Occupé |
| Jeudi 9 Septembre 2010     |    |    |        |        |        |        |        |        |        |        |        |
| Vendredi 10 Septembre 2010 |    |    |        |        |        |        |        |        |        |        |        |
| Samedi 11 Septembre 2010   |    |    |        |        |        |        |        |        |        |        |        |
| Dimanche 12 Septembre 2010 |    |    |        |        |        |        |        |        |        |        |        |

Mode sélection: << Aujourd'hui : Mercredi 25 Août 2010 / 18h57 >>

**Légende**

- Indisponible.
- Disponible
- Occupé.

**Outils**

- Définir les disponibilités
- Créer une opération client
- Créer un individu
- Créer une adresse postale

'Prospector' - TVA : 0123456789  
 2588 avenue des Caribous Bureau 4445 1051 Bruxelles BELGIQUE  
 Téléphone : 0123456789 - Fax : 0123455777

Écran administrateur, gestion des plannings

## Mémoire de stage | Conclusion

The screenshot displays the Prospector software interface, specifically the 'Gestion des statuts professionnels' (Management of professional statuses) module.

**Top Bar:** Includes links for 'Accueil' (Home), 'Espace administrateur' (Administrator space), 'Mon profil' (My profile), and 'Contact'. It also shows the user is connected as 'Guillaume Zavan [Administrateur]' and includes a 'Déconnecter' (Logout) button.

**Modifying an Employee Status:** A modal window titled 'Modifier un statut d'employé' (Modify an employee status) is open. It contains a table with three rows for 'Libellé' (Label) and three columns for 'FR', 'NL', and 'EN'. The 'FR' row contains 'PDG'. The 'NL' row contains 'Dutchpédégé'. The 'EN' row contains 'Director'. A 'Modifier' (Modify) button is at the bottom.

**List of Employee Statuses:** A table titled 'Liste des statuts d'employé' (List of employee statuses) shows five registered entries. Each entry has columns for 'Libellé' (Label), 'Actions' (Modifier, Supprimer), and a detailed description in French, Dutch, and English. The entries are:

| Libellé                                             | Action                | Description                                        |
|-----------------------------------------------------|-----------------------|----------------------------------------------------|
| FR : PDG<br>NL : Dutchpédégé<br>EN : Director       | Modifier<br>Supprimer | FR : Stagiaire<br>NL : Ducthslave<br>EN : Slave    |
| FR : Développeur<br>NL : Dutchev<br>EN : Developper | Modifier<br>Supprimer | FR : Cadre<br>NL : Dutchwindow<br>EN : Executive   |
| FR : Graphiste<br>NL : Dutchgraph<br>EN : Designer  | Modifier<br>Supprimer | FR : Graphiste<br>NL : Dutchgraph<br>EN : Designer |

**Outils (Tools):** A panel titled 'Créer un statut d'employé' (Create an employee status) provides a button to create new statuses.

**Footer:** Displays the company address: 'Prospector' - TVA : 0123456789, 2588 avenue des Caribous Bureau 4445 1051 Bruxelles BELGIQUE, Téléphone : 0123456789 - Fax : 0123455777.

Écran administrateur, gestion des statuts professionnels

## Mémoire de stage | Conclusion

The screenshot shows the Prospector software interface. At the top, there is a navigation bar with links for 'Accueil', 'Espace client', 'Mon profil', and 'Contact'. On the left, there is a sidebar titled 'Afficher' containing sections for 'Opération' (with a dropdown menu showing 'Contrôle du monde'), 'Informations' (showing 'Date de départ: Jeudi 24 Juin 2010', 'Etat: En cours', 'Budget: 12000€', 'Objectif: 1000 appel(s)', and 'Avancement: 25.6%'), 'Rendez-vous' (showing 'Rendez-vous lié(s) à cette opération: 3'), and 'Outils' (with a button 'Endormir l'opération'). The main window is titled 'Mes opérations' and displays an operation named 'Opération 'Contrôle du monde''. It includes a summary section and two tabs: 'Rencontres à venir' and 'Rencontres passées'. The 'Rencontres passées' tab lists three meetings:

- Vendredi 25 Juin 2010 / 9h**  
Pair: ZAVAN Guillaume  
Adresse mail: webkawa@gmail.com  
Entreprise: Prospector  
Adresse: Non-renseigné
- Vendredi 25 Juin 2010 / 12h**  
Pair: BRUNET Benjamin  
Adresse mail: benjamin.brunet@free.fr  
Entreprise: Prospector  
Adresse: Non-renseigné
- Vendredi 2 Juillet 2010 / 11h30**  
Pair: ZAVAN Guillaume  
Adresse mail: webkawa@gmail.com  
Entreprise: Prospector  
Adresse: 2588 avenue des Caribous 1051 Bruxelles

At the bottom of the main window, there is a footer with the text: 'Prospector' - TVA : 0123456789  
2588 avenue des Caribous Bureau 4445 1051 Bruxelles BELGIQUE  
Téléphone : 0123456789 - Fax : 0123455777

Écran client, gestion des opérations

## Bilan

Comme je le disait en introduction, ce mémoire se veut une présentation des points les plus importants de mon travail à Bruxelles, et non une documentation exhaustive du travail réalisé. Celui-ci est d'ailleurs trop massif pour pouvoir se résumer en soixante pages (plus de 70.000 lignes de code rédigées, une dizaine de fichiers dédiés à la mise en page et aux scripts d'interface, sans parler du travail effectué pour le cahier des charges) ; j'invite donc encore une fois le correcteur à se référer aux annexes jointes pour plus de détails.

La question la plus importante que je me pose, à la suite de ce travail, peut se résumer ainsi : étant donné que ma démarche conceptuelle et technique revient, plus ou moins, à la création d'un MVC applicable au JEE, quelle nécessité avais-je, au moment des fait, de « réinventer la poudre », plutôt que de me baser sur un produit existant – Spring, Struts, ou un autre – qui, sans aucun doute, se serait révélé plus performant et m'aurait épargné de nombreux efforts ?

Tout d'abord, je pense que ce travail m'a permis d'acquérir un réel recul sur les principaux points indispensables à tout projet applicable au web : en terme d'agilité et de portabilité, en premier lieu, puis vis-à-vis du code à proprement parler. Par ailleurs, je pense qu'il intéressant de se poser la question, ne serait-ce qu'une fois, des bonnes pratiques nécessaires pour assurer une qualité optimale dans la réalisation d'une application de taille conséquente. Enfin, je crois avoir pu perfectionner, au fil du développement, mes habitudes de rédaction, tout particulièrement en terme de documentation.

L'utilisation d'un MVC simplifie certes beaucoup de choses, mais elle les rend également plus opaques ; et c'est la une des raisons pour lesquelles j'ai préféré partir de zéro. C'est dans le même esprit que j'ai privilégié le développement de *servlets* authentiques, plutôt que de JSP plus intuitives.

Je tire néanmoins une certaine satisfaction du travail réalisé ; n'ayant jamais travaillé en JEE par le passé, j'ai compris l'intérêt certain présenté par ce format par rapport à ses concurrents, et notamment le PHP. J'ai par ailleurs découvert certains inconvénients du langage Java que je ne soupçonnais pas forcément dont, en tête de liste, la lourdeur du processus de développement. Je mise néanmoins dessus pour la suite de mes études et, ayant été amené depuis à chercher du travail, je me rend bien compte qu'il s'agit là d'une valeur sûre sur le marché du travail.

## État du projet

Au moment de mon départ (voir ci-dessous), le travail de développement était quasiment achevé : tout au plus restait-il une dizaine de *servlets* à déployer (dont le *front office*), le gestionnaire de *BBCode*, ainsi qu'un outil pour gérer l'intégration d'éléments au format Flash.

A mon grand regret, il n'a pas été possible de réaliser les tests unitaire initialement prévus, pas plus que la location de l'espace dédié au site. Même s'il ne s'agit là que d'une partie restreinte du travail, n'ayant jamais mené de projet JEE de bout en bout, j'aurais apprécié pouvoir en faire l'expérience. Enfin, la version en Néerlandais n'a put être réalisée faute d'un traducteur compétent au moment de mon passage.

Un set de données compatible se trouve hébergé sur mon espace de développement personnel.

### Serveur SQL de développement

-----  
Adresse : mysql6.celeonet.fr  
Login : kawa  
Mot de passe : lolacb