Réalisation d'un système d'information pour la navigation intérieure

Fadhel Helali

MEMOIRE
Présenté en vue d'obtenir

LE DIPLOME D'INGENIEUR CNAM
SPECIALITE : SYSTEMES D'INFORMATION

REALISATION D'UN SYSTEME D'INFORMATION POUR LA NAVIGATION INTERIEURE

Par
Fadhel **HELALI**

Tuteur CNAM : Claude **GENIER**
Tuteur entreprise : Frédéric **BELLAICHE**

Soutenu le : 27 juin 2011

JURY

PRESIDENT : Christophe **PICOULEAU** (CNAM Paris)

MEMBRES : Claude **GENIER** (CNAM Lyon)
Bertrand **DAVID** (CNAM Lyon)
Frédéric **BELLAICHE** (Osiatis)
Olivier **DEPRIESTER** (Osiatis)

A la mémoire de mon petit frère HAMDI

SOMMAIRE

I. REMERCIEMENTS

Je voudrais remercier toutes les personnes qui, par leur participation et leurs encouragements, m'ont permis de mener à bien ce travail.

Ce mémoire n'aurait vu le jour sans la confiance et la patience de mon tuteur CNAM, M. Claude GENIER et de mon tuteur en entreprise M. Frédéric BELLAICHE, Directeur Technique du Centre de services. Leurs conseils précieux et encouragements m'ont permis de surmonter toutes les difficultés du parcours. Qu'ils trouvent ici l'expression de mes sentiments sincères.

Je tiens à présenter ma vive gratitude à mon Chef de Projet Olivier DEPRIESTER, qui a contribué activement à la réalisation de mes travaux, pour ses conseils efficaces, ses orientations et ses remarques pertinentes.

Merci également à Julien BERNARD, Architecte et Chef de Projet pour son aide, ses précieuses remarques et ses conseils.

Un grand remerciement est adressé à tous mes collègues sur la plateforme du travail, pour l'ambiance très favorable qu'ils ont su créer autour de moi.

Enfin, mes remerciements vont également à tous ceux qui ont participé plus ou moins indirectement au bon déroulement de mon mémoire.

II. INTRODUCTION

1. CONTEXTE

Une étude d'opportunité réalisée en novembre 2009 par la DGITM (Direction Générale des Infrastructures, des Transports et de la Mer) du Ministère du Développement Durable a permis de fixer un périmètre et une stratégie de réalisation pour le Système d'Information. Le système d'information est SINAVI (Système d'Information pour la NAVigation Intérieure). Il se composera à termes de 2 applications différentes : la première ITINAVI (Immatriculation et TItres pour la NAVigation Intérieure) reprendra les fonctionnalités de GROBATO et ETNA, la seconde CERCALINA (CERtificat de CApacité et LIvrets pour la NAvigation) reprendra les fonctionnalités de CERCAFLU.

Compte tenu de la taille du projet et de la volonté de la Maîtrise d'Ouvrage de disposer des anciennes et des nouvelles fonctionnalités, il a été décidé de découper l'application en 2 modules qui seront déployés en 4 livraisons.

Les 4 livraisons du projet ITINAVI suivent le découpage fonctionnel établi comme suit :

- Première livraison « gestion des bateaux » iso-fonctionnalités :
  - Une base bateau
  - Procédures existantes dans GROBATO et ETNA
  - CB, certificat de visite provisoire (CVP)
  - Reprise des données (bateaux, titres et immatriculations)
- Deuxième livraison « gestion des bateaux » :
  - Certificat d'agrément et Titre provisoire
  - Gestion des listes d'équipage
  - Outil de suivi de l'instruction
  - Statistiques
  - Outil de suivi des dossiers à l'intention des usagers sous forme d'un extranet
  - Traçabilité des supports (listes, recherches, etc.)
  - Création de web service pour le dialogue avec les bases européennes (PLATINA) (ou plus tard en attente d'information sur la base européenne)
  - Jaugeage (ou plus tard en attente de la réforme de la procédure)

La troisième et quatrième livraison concernent CERCALINA

- Troisième livraison « gestion des patentes et des certificats de capacité » iso-fonctionnalités :

- Gestion des candidatures, des examinateurs, des examens et des épreuves (théoriques/pratiques)
- Édition des cartes d'épreuves et des attestations radar, gaz

- Quatrième livraison « gestion des patentes et des certificats de capacité » nouvelles fonctionnalités :
  - Éditions diverses liées à l'organisation des examens (convocation, pv, résultats, etc.)
  - Nouvelles procédures (inexistantes dans CERCAFLU)
  - Fonctions de traçabilité des supports
  - Outils à destination des usagers et des utilisateurs

Remarque : Le présent travail ne traite que de la première livraison.

2. PROBLEMATIQUE

La Maîtrise d'Ouvrage a pour objectif le développement d'une application complète et ouverte à toute évolution réglementaire ou technique en centrant les processus métier sur l'objet métier : le bateau.

- Poursuivre et finaliser le développement des modules manquants de l'application ETNA en prenant en compte les éventuelles évolutions réglementaires prévues ou prévisibles
- Résoudre les problèmes de compatibilité fonctionnelle existants entre les diverses applications
- Faciliter l'exploitation des données à des fins statistiques
- Prendre en compte les demandes d'évolution formulées par les utilisateurs

Cependant, d'autres objectifs, plus spécifiquement informatiques, existent

- Résoudre les problèmes signalés lors de l'intégration des dernières versions d'ETNA sur le Centre serveur (liée au développement ACAI)
- Limiter les coûts de maintenance

3. CONTRIBUTION

Notre objectif était la réalisation d'ITINAVI, un système d'information fluvial au niveau européen. Sa réalisation a permis d'approfondir mes connaissances

techniques, fonctionnelles et pratiques. Bien évidemment, ceci n'a pas été des plus faciles et des difficultés ont alors dû être surmontées.

Une des plus grandes difficultés rencontrées fût la complexité de l'architecture ACAI. La prise en main de ce framework puissant fût un véritable challenge, mais les échanges avec le client nous ont permis de surmonter cette difficulté. En effet, au démarrage de réalisation de l'application, je me suis déplacé chez le client où, en collaboration avec les responsables techniques et fonctionnels, nous avons mis en place les premières briques du projet. Nous avons dû, entre autres, apporter des modifications du modèle de données, trouver les solutions techniques pour certains problèmes…

Dans un second temps, une équipe de deux collaborateurs nous a rejoints et, ensemble, nous avons continué la réalisation. Une fois l'autonomie technique voulue atteinte, nous sommes retournés sur la plateforme d'Osiatis où, l'équipe au complet, nous avons pu nous engager sur la réalisation de l'application.

Riche de mon expérience dans le domaine du développement JEE, j'ai pu vite maîtriser l'architecture technique du CETE, laquelle repose sur des frameworks spring, struts, hibernate. J'ai ainsi pu contribuer efficacement à la réalisation de l'application. Le travail que nous avons réalisé au sein d'une équipe motivée est une expérience enrichissante : partage des tâches, partage des connaissances... Nous avons travaillé selon les méthodes agiles et les réunions quotidiennes nous ont permis d'être en constante interactivité avec le client pour mettre en œuvre des solutions techniques qui répondent à ses attentes métier.

La rigueur de l'équipe, couplée aux apports de la plateforme d'intégration continue, garant de la non régression de l'application, nous a permis de répondre efficacement aux attentes du clients.

4. PLAN DU MEMOIRE

Ce document de mémoire se compose de quatre chapitres :

- une introduction,
- le cadre du projet,
- le domaine cible,
- la réalisation.

Le deuxième chapitre présente l'entreprise dans laquelle s'est réalisé ce travail, ainsi que l'organisation de l'équipe de réalisation du projet. Nous travaillons selon le principe agile qui, en plaçant le client au cœur du projet, et en insistant sur l'utilisation de cycles de développement itératifs et incrémentaux, permet à l'équipe de devenir l'élément prépondérant dans la gestion de projet.

Le troisième chapitre est une présentation du domaine cible, ses objectifs et ses exigences. La DGITM est principalement concerné par la mise en place de l'harmonisation des réglementations européennes qui régissent la navigation intérieure (prescriptions techniques, conduite) et la création d'un système d'information fluvial au niveau européen (SIF). Cela consiste pour ce document en la fusion des deux applications GROBATO pour les immatriculations et ETNA pour la délivrance des titres de navigation en une seule application : ITINAVI.

L'écriture du logiciel sera conforme à l'architecture ACAI (paragraphe 5.3) en suivant les guides d'architecture commune des applications informatiques dans le but d'une appropriation facile par l'utilisateur.

Le quatrième chapitre, réalisation, commence par une description fonctionnelle du lot 2 « certificat communautaire » qui permet la gestion de la partie technique du bateau.

Conformément au contexte ACAI, l'application se repose techniquement sur JEE (Java Enterprise Edition), complété par des frameworks puissants : spring, struts et hibernate ; nous montrons l'apport de cette panoplie de technologie et sa contribution à faciliter le travail des développeurs ainsi que la difficulté de sa mise en place.

Nous expliquons le choix des outils techniques, notamment la « mavenisation » du projet et les apports de MAVEN. En effet, l'industrialisation du développement de l'application se base sur 3 concepts : l'intégration continue, le développement piloté par les tests et la mesure continue de la qualité.

Les grandes lignes à respecter pour la réalisation de l'application sont clairement définies dans le framework "commun" du CETE ; ce framework que nous avons intégré, après la « mavenisation », aux dépendances du projet.

Je termine ce document par une conclusion.

III. CADRE DU PROJET

1. PRESENTATION DE L'ENTREPRISE

Pour Osiatis, le système d'information est un levier au service de la performance et de la compétitivité de l'entreprise.

Pour un DSI, toute évolution du système d'information est donc évaluée à la mesure de son impact pour les métiers de l'entreprise et de son retour sur investissement. Base de l'édifice, la disponibilité du système d'information doit, dans tous les cas, être assurée.

Osiatis est un partenaire de performance des DSI, à deux niveaux :

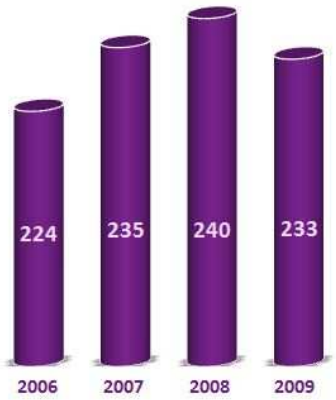
- proposer des solutions efficaces et innovantes de services et de transformation d'infrastructure, et de gestion du patrimoine applicatif
- accompagner les métiers de l'entreprise dans la mise en œuvre de nouvelles applications.

1.1. Chiffres clés

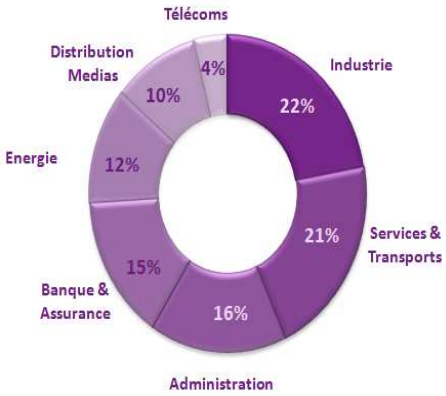
L'année 2009 a été marquée par 2 éléments :

- Une bonne tenue du chiffre d'affaires dans un contexte de crise
- Une nouvelle croissance de la part de l'international.

Evolution du chiffre d'affaires



Répartition du CA 2009 par secteur d'activité



1.2. Structuration de l'offre

Osiatis identifie trois domaines dans le système d'information : utilisateurs, systèmes et réseaux, et applications. Cette segmentation est basée sur des critères technologiques et d'attentes en termes de service :

- Domaine utilisateurs :

Technologies : postes de travail et périphériques, fixes et mobiles.

Attentes : proximité, réactivité et mobilité.

- Domaine systèmes et réseaux :

Technologies : serveurs et réseaux.

Attentes : performance, disponibilité, sécurité.

- Domaine applications :

Technologies : applications standards (messagerie, bases de données, workflow,...) et métiers.

Attentes : réduction des coûts, amélioration des services rendus.

Face aux demandes clients, la cohérence de l'offre Osiatis est assurée en amont par le conseil selon le schéma suivant :

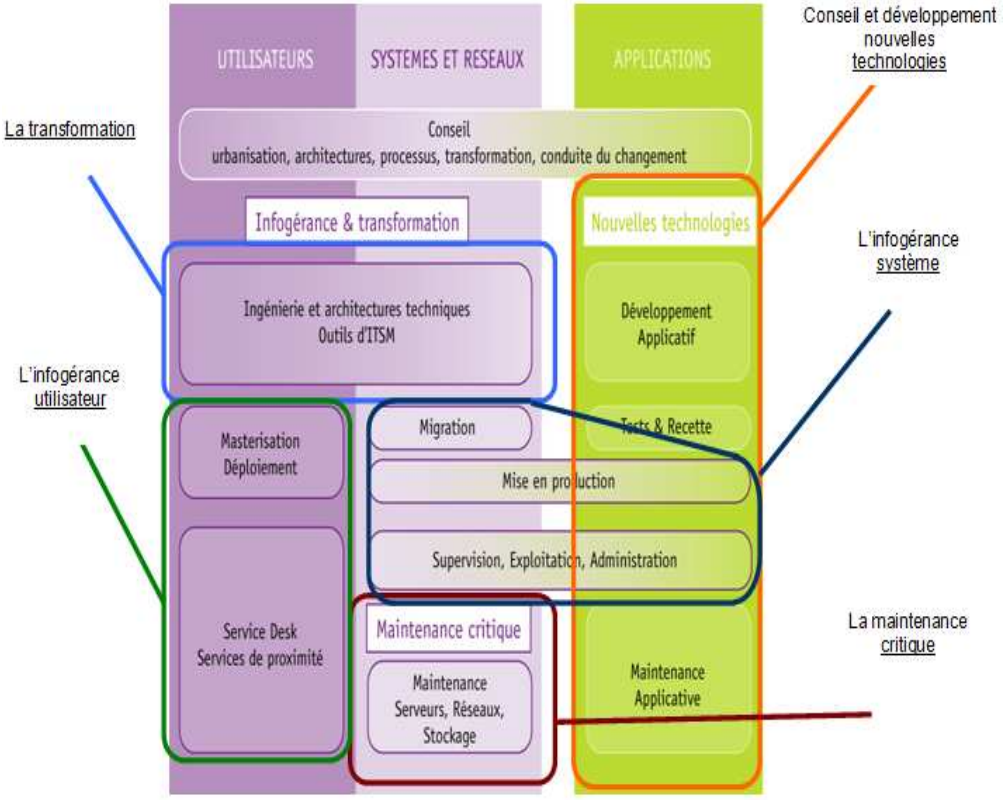


Figure III-1 : Structuration de l'offre

1.3. Métiers

Pour remplir cette mission, Osiatis s'est structuré en deux métiers qui regroupent les ressources et les compétences nécessaires à la production des services :

- L'infogérance et la transformation (services aux infrastructures) : activité principale en France et exclusive à l'international ;
- Le conseil et les développements nouvelles technologies (ingénierie applicative), et plus précisément les projets de développements et la tierce maintenance applicative, commercialisés et produits par sa filiale française, Osiatis Ingénierie.

Ainsi, pour mieux servir ses clients, Osiatis intervient sur l'ensemble du cycle de vie des services, depuis le diagnostic et le design (DIAG), jusqu'aux services récurrents (RUN) en passant par la transformation (TRANSFO).

C'est l'expression de notre signature : « We build the run ».

Ce que nous bâtissons, nous l'exploitons. Ce que nous exploitons, nous le faisons évoluer.

1.4. Implantations

La volonté de respecter les engagements contractuels, en terme de délai d'intervention en particulier, a conduit Osiatis à déployer un réseau d'implantations proche de ses clients.

En France, Osiatis dispose de fortes représentations en Régions (50% des effectifs totaux), ce qui garantit une disponibilité et une qualité de service uniforme sur l'ensemble du territoire.

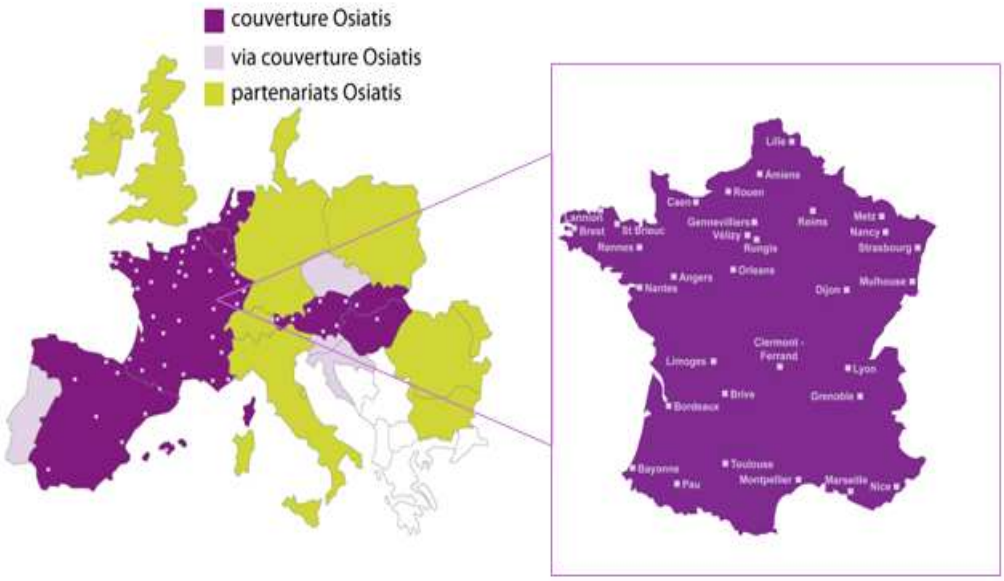


Figure III-2 : implantation d'Osiatis

A l'international, Osiatis assure la couverture de ses services au travers de son organisation dont le mode de fonctionnement est le suivant :

- Dans les pays accueillant une filiale, les services sont assurés en direct par la filiale. C'est le cas en : Espagne, Autriche et Belgique.
- Dans les pays limitrophes de nos filiales, nous savons, à partir de celles-ci, projeter notre capacité d'intervention. Nous agissons ainsi en : Europe centrale (Hongrie, Tchéquie, Slovaquie) à partir de l'Autriche et Luxembourg à partir de la Belgique.
- Autres pays : pour assurer la couverture de nos services sur les autres pays, nous nous appuyons sur notre réseau de partenaires locaux.

1.5. Une entreprise engagée

Osiatis a bâti son succès sur une politique managériale résolument orientée vers l'individu en étant particulièrement attentif à son développement et au maintien de sa motivation.

Les attentes des clients travaillant avec des SSII portent de plus en plus sur un savoir-être des collaborateurs qui puissent s'intégrer rapidement dans les équipes, comprendre et partager l'ambition du client, nous sommes convaincus de cette absolue nécessité et cette recherche de personnalité est au centre de nos campagnes de recrutement.

- Développer les talents des collaborateurs

Osiatis met à disposition de ses clients des savoir-faire, des offres, des processus, des outils. Société de services, sa « matière première », ce sont les Hommes.

Données RH :

- 2518 collaborateurs
- Des investissements de formation parmi les meilleurs du marché
- Plus de 6% du capital détenu par les salariés et management : un outil de motivation à la réussite de l'entreprise, notamment via le canal du PEE. Les 6 % correspondent uniquement aux porteurs déclarés.

- Les valeurs

Construite dès 1998, la culture Osiatis a pour objectif de créer un « esprit de corps » entre les collaborateurs, leur donnant « envie de faire partie de l'équipe », et les inciter à se dépasser par la confiance et la considération qui leur sont accordées.

Cette culture est fondée sur des valeurs partagées et démontrées :

- Transparence : communication directe et franche, reporting précis et ponctuel ;
- Engagement : chacun assure et assume les tâches qui lui sont confiées ;
- Confiance : réciproque et méritée pour réussir une destinée commune ;
- Respect : des individus, de leurs opinions et de leurs différences ;
- Progrès : l'indispensable évolution suppose créativité, prise de risque et innovation.

- Le développement durable : Osiatis a identifié et mène des actions concrètes dans l'esprit d'une démarche sociétale et environnementale selon trois axes :

- Axe économique : continuer à produire la richesse nécessaire à la population
- Axe social : veiller à réduire les inégalités à travers le monde
- Axe environnemental : préserver l'équilibre de l'environnement des futures générations

1.6. Des partenariats au cœur de la stratégie

Pour accompagner son développement et proposer à ses clients les solutions les plus pertinentes, Osiatis a placé sa politique de partenariats au cœur de sa stratégie.

Nous avons sélectionné 6 partenaires stratégiques avec lesquels nous portons, de façon proactive sur le marché, les technologies et les services nécessaires à la transformation ou l'évolution des SI de nos clients.

Figure III-3 : partenaires d'Osiatis

Avec ces partenaires, Osiatis s'engage à :

- obtenir le plus haut niveau de certification

Les principles certifications: Microsoft Certified Gold Partner, Citrix Solutions Advisor Gold Partner, VMware VAC Bronze, BMC Elite Partner, LANDesk Platinum Expert Solution Provider, CA Gold Partner.

- travailler constamment, en étroite collaboration, en amont de leurs innovations technologiques.

Une mention particulière pour l'alliance avec Microsoft : début 2009, Osiatis est l'une des deux SSII françaises retenues par Microsoft pour conduire le programme d'adoption anticipée de Windows 7.

Osiatis a également conclu des alliances avec les principaux acteurs technologiques du marché, sur lesquels elle a développé expertises et expériences pour répondre aux différents besoins de ses clients.

Les technologies sont choisies pour leur capacité à apporter davantage de valeur à ses services et donc à ses clients ; la formation de ses équipes est par conséquent une priorité afin de proposer les meilleurs experts sur ces nouvelles technologies.

1.7. Développements nouvelles technologies

Figure III-4 : développement nouvelles technologies

La stratégie d'Osiatis est de se positionner sur les nouvelles technologies, en privilégiant les activités en centres de services.

Osiatis travaillle en mode projet ou en mode assistance technique, au forfait ou bien en Unités d'Œuvre sur 3 domaines applicatifs majeurs : les applications métiers, les applications décisionnelles, les portails associés à la gestion de contenu.

Dans le cycle de vie des applications, les phases suivantes sont couvertes :

- Conception : cahier des charges, spécifications fonctionnelles et/ou techniques
- Développement : réalisation, tests unitaires, tests d'intégration technique

- Tests et recette : mise en place de cellules de test, validation technique et/ou fonctionnelle

- Mise en production : ensemble des tâches préparatoires et exécution de la mise en production des applications

Ce cycle de vie inclut le Management du patrimoine applicatif : support utilisateurs, maintenance corrective, maintenance évolutive, management des services, transformation du patrimoine, évolution perfective (performances) et adaptative (vis-à-vis du socle technologique).

Toutes les dimensions de ce cycle sont couvertes :

- Le métier (caractérisation du besoin client)

- Le fonctionnel (traduction du besoin client en référentiel fonctionnel)

- L'applicatif et le technique (réalisation et mise en production)

Osiatis couvre les principales familles technologiques du marché et entretient des relations de partenariats avec les acteurs majeurs du monde libre ou propriétaire.

2. METHODES AGILES

Les méthodes agiles selon lesquelles nous avons travaillé, nous permettaient d'être auto-organisés et ainsi d'avoir la capacité à décider de l'organisation de nos propres activités pour atteindre les objectifs fixés, ou pour résoudre les problèmes auxquels nous nous sommes confrontés.

2.1. Organisation de l'équipe

Notre équipe partage une méthode de travail et des objectifs de rendement communs, tous les membres se sentent mutuellement responsables. Cela signifie que chaque personne doit faire preuve du même engagement et du même sens des responsabilités en ce qui a trait aux tâches à exécuter et au but à atteindre, voilà sa composition :

Olivier DEPRIESTER : Chef de Projet

Fadhel HELALI : Ingénieur d'Etudes et de Développement

Elise GALLENSTEIN : Ingénieur d'Etudes et de Développement

Patricia ARTINIAN : Ingénieur d'Etudes et de Développement

Sarah BIGHETTI : Ingénieur d'Etudes et de Développement

Jean marie CROMMEN : Ingénieur d'Etudes et de Développement

2.2. Méthodes agiles

Le terme de « Méthodes Agiles » décrit les différentes manières de gérer un projet informatique dans le but d'augmenter la satisfaction du client tout en rendant le travail de développement plus facile *[Agilité, 1]*. C'est une méthode qui vise à diminuer le cycle de vie d'un logiciel, c'est-à-dire à rendre le développement plus pertinent.

En réponse à des dysfonctionnements et des difficultés de développement, les méthodes agiles s'appuient sur trois valeurs :

- Les individus et les interactions plutôt que sur les processus et les outils : le travail en groupe et la communication sont fortement utilisées avec une interaction permanente entre les différents membres de l'équipe en favorisant parfois le travail à deux;

- Une collaboration avec le client plutôt qu'une négociation du contrat : le client est en contact permanent avec l'équipe de développement permettant un suivi régulier des fonctionnalités développées et d'éventuelles modifications. Un feedback régulier est réalisé permettant d'instaurer une relation de confiance ;

- Acceptation et conduite du changement plutôt que la conformité aux plans : la planification réalisée au début de l'étude est soumise à diverses évolutions en cours de projet, et permet donc une flexibilité de la planification, en s'adaptant alors parfois à des changements de dernières minutes.

2.2.1. PRINCIPES

La plus grande priorité de la méthode agile est la satisfaction du client.

La conduite du changement est sans cesse mise en pratique avec les méthodes agiles. En effet, les demandes de changements arrivées même tard dans le processus de développement doivent être accueillies « les bras ouverts », ce qui répond au principe de la production flexible du projet et qui procure ainsi un avantage concurrentiel face à des entreprises n'ayant pas mis en place les méthodologies agiles.

Le fonctionnement de l'équipe est basé sur l'auto-organisation et la dispersion des différentes tâches est faite suivant un volontariat. A intervalles réguliers, l'équipe doit faire un point sur son organisation, et s'interroge sur la manière de devenir encore plus efficace pour faire face à cet environnement en perpétuel évolution.

La communication dans les méthodes agiles est l'un des éléments clés dans un projet. Une communication entre le client et l'équipe de développement doit être omniprésente pour s'adapter et faire face aux évolutions constantes.

Une communication à l'intérieur de l'équipe doit être mise en place en privilégiant la communication face à face (réunion).

Une des difficultés de cette méthode est le respect du rythme de développement soutenable pour les développeurs et les utilisateurs clients. Adapter le rythme assure de préserver la qualité du travail sur la durée totale du projet.

Il est important tout au long du projet de garder un œil critique sur la qualité de l'application, du code et du travail fourni par les membres de l'équipe.

2.2.2. SCRUM

La méthode SCRUM est une des méthodes agiles les plus utilisées pour la gestion de projets informatiques. Cette méthodologie peut être utilisée soit pour la réalisation d'applications soit pour la maintenance d'applications. Il s'agit d'une méthode créée pour améliorer la productivité des équipes de développement et d'améliorer la qualité du travail rendu.

L'objectif de cette méthode est de concentrer des équipes de développement sur un ensemble de fonctionnalités métier, au long d'une période appelée un sprint. Chaque itération ou sprint dure 2 à 4 semaines et assure à la fin une livraison avec des fonctionnalités de qualité et opérationnelles. La particularité de cette méthode est la participation active du client qui choisi avec le Chef de Projet l'ensemble des fonctionnalités à livrer pour le prochain sprint. Cependant lorsqu'une itération a déjà débutée, elle ne doit en aucun cas être modifiée sous peine d'un nom respect de la date de livraison du package promis.

La méthode SCRUM utilise une organisation selon trois niveaux : les sprints, les releases et les scrums quotidiens.

Les sprints : Ce sont, comme il a été dit précédemment, des itérations d'une durée de 2 à 4 semaines. Chaque sprint est associé à un but qui représente un ensemble de fonctionnalités à livrer soit une « liste d'items de backlogs » de sprints. Ces items sont alors décomposés par l'équipe en une liste de tâches durant quelques heures appelées « les items de backlogs ». La liste de ces items ne peut pas se voir évoluer durant un sprint. Si les fonctionnalités sont à modifier ou doivent être ajoutées, ces changements seront établis dans la ou les prochaines itérations. La seule exception réside lorsque le développement d'un sprint est terminé à l'avance, dans ce cas il faudra ajouter des items, ou dans le cas contraire reporter certains items dans le prochain sprint ;

Les releases représente le rendu d'un ensemble de livraisons que l'on peut assimiler à un produit intermédiaire représentant une version aboutit pouvant être exploitée en production. Il permet de réaliser des versions de l'application tout au long du développement ;

Les scrums quotidiens représentent dans SCRUM les réunions qui ont lieu quotidiennement avec l'équipe, son directeur de produit et son SCRUM Master afin de définir l'ensemble des tâches à réaliser pour la journée.

➢ *Déroulement d'un sprint*

Tout d'abord un sprint commence par une réunion de planification (Sprint planning) qui consiste à définir le but visé par ce sprint. Elle ne doit pas excéder 4 heures. L'équipe à travers cette réunion s'engage à réaliser un ensemble d'items du backlog du produit c'est-à-dire de l'application. Dans un second temps de cette réunion, l'équipe va découper l'ensemble des items du backlogs du produit défini précédemment en items du backlogs du sprint puis estimer chaque tâche en heure de manière à garder une vision du temps et du nombre de charges nécessaires pour développer ce sprint.

Le déroulement d'un sprint est organisé en mêlées quotidiennes (Daily sprint) qui ne doit pas excéder 15 minutes de manière à ne pas trop empiéter sur le temps de travail de l'équipe. A tour de rôle, chaque membre de l'équipe devra répondre à 3 questions :

- Qu'est ce que j'ai fait hier ?
- Qu'est ce que je compte faire aujourd'hui ?
- Quelles difficultés je rencontre aujourd'hui ?

Cette réunion quotidienne a pour but de synchroniser l'équipe et non pas d'effectuer un reporting d'activité. Elle permet cependant de vérifier que le temps de développement associé à une tâche a été correctement évalué afin de déceler d'éventuels retards ou d'éventuelles erreurs au niveau de l'affectation des tâches, qui permettrait alors de changer l'affectation d'une tâche à un membre de l'équipe si la tâche s'avère être trop compliquée pour lui.

A la fin d'un sprint a lieu ce que l'on appelle la revue du sprint qui ne doit pas excéder 4 heures. Elle a lieu avec le directeur du produit à qui l'équipe présente l'ensemble des items du backlog du sprint réalisé, et en fait alors une démonstration. Le directeur de produit choisit de valider les fonctionnalités développées ou pas. Il est probable que suite à ce bilan de sprint, des aménagements soient fait sur les items backlog du sprint.

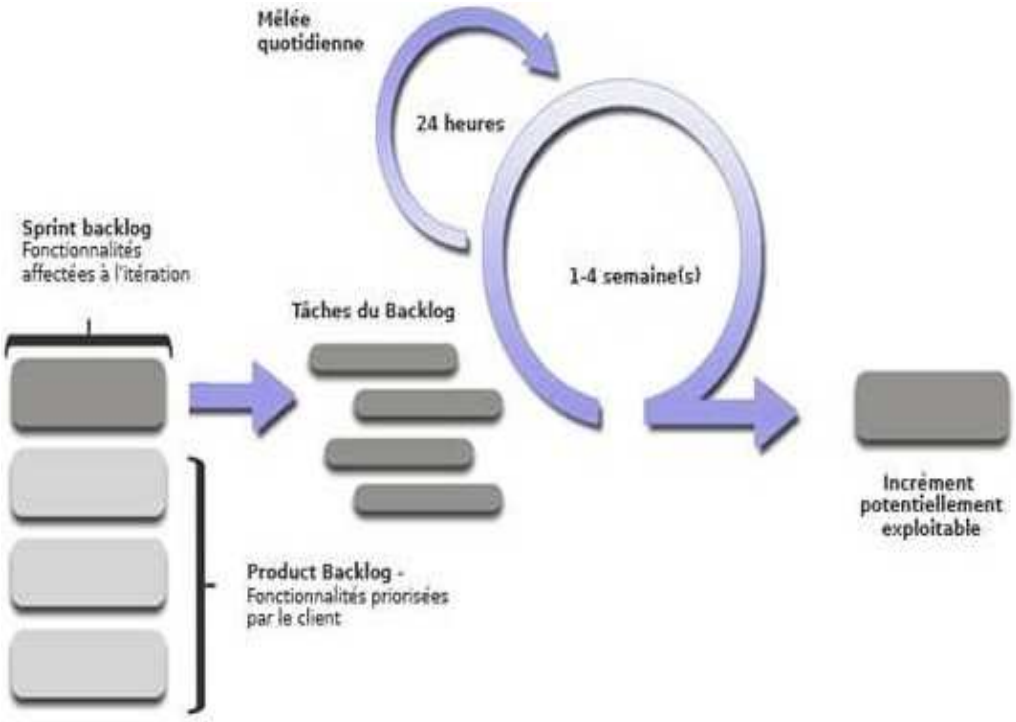


Figure III-5 : Principe de la méthode SCRUM

Enfin a lieu la rétrospective du sprint qui permet à l'ensemble de l'équipe et du SCRUM Master de faire un retour sur le déroulement du sprint : définir ce qui a bien marché de ce qui n'a pas bien fonctionné afin d'éviter de commettre les mêmes erreurs une seconde fois. Ainsi, il est possible d'apporter des aménagements à la méthode SCRUM installée, mais l'erreur à ne pas commettre est de revenir à une gestion de projet rigide avec des méthodologies classiques.

➢ *Les rôles dans la méthode SCRUM*

Lors d'un SCRUM, mêlée qui a lieu quotidiennement, divers personnes tiennent un rôle prépondérant.

- Le directeur de produit est la personne représentant les clients et utilisateurs de la future application. C'est elle qui oriente l'équipe et ses développements vers un ensemble de fonctionnalités dans le prochain sprint. Elle doit rester proche et disponible de l'équipe afin de répondre à un grand nombre d'éventuelles questions et interrogations dans un court délai. Dans notre cas pour le projet ITINAVI, le directeur de produit n'est pas présent dans les locaux d'OSIATIS et la définition du périmètre de livraison a donc lieu dans un comité de suivi hebdomadaire.

- L'équipe est composée des différents intervenants dans le développement de l'application. Il n'y a pas de notion de hiérarchie et

chaque décision est prise ensemble afin de respecter une autogestion du travail à fournir. Ainsi, personne ne donne d'ordre à l'équipe sur sa façon de procéder. Des études ont prouvé que ce sont les équipes qui se gèrent de manière autonome qui rencontrent le plus de succès.

- Le SCRUM Master joue un rôle capital : c'est lui qui est chargé de protéger l'équipe de tous les éléments perturbateurs extérieurs à l'équipe et de résoudre ses problèmes non techniques (administratifs par exemple, pilotage). Il doit aussi veiller à ce que les valeurs de SCRUM soient appliquées, mais il n'est pas un Chef de Projet ni un intermédiaire de communication avec les clients.

IV. DOMAINE CIBLE ET OBJECTIFS

PERIMETRE INFORMATIQUE D'ITINAVI

1. DOMAINE CIBLE

Le transport fluvial est un mode de transport économique, fiable et peu polluant. Aussi, depuis plusieurs années, les instances européennes et nationales souhaitent privilégier le développement du transport fluvial et son inter-modalité avec les transports maritimes, routier et ferré. Pour améliorer la compétitivité du transport par voie d'eau, différentes commissions et différents groupes de travail ont été mis en place.

1.1. Synthèse de l'étude de l'existant

Parmi les nombreux travaux produits (livres blanches, directives) la DGITM est principalement concernés par la mise en place de l'harmonisation des réglementations européennes qui régissent la navigation intérieure (prescriptions techniques, conduite) et la création d'un système d'information fluvial au niveau européen (SIF).

1.1.1. Contexte d'évolution de la réglementation

Sur les 27 pays qui constituent l'Europe, 20 ont des voies navigables dont 12 sont interconnectées.

L'harmonisation de la réglementation applicable aux bateaux fluviaux est déjà réalisée par le biais d'une directive, relative aux prescriptions techniques applicables aux bateaux de navigation intérieure ; cette directive permet aux bateaux de naviguer sur l'ensemble des voies européennes en disposant de titre de navigation communautaire valide quel que soit le pays traversé.

Cette harmonisation a déjà entraîné la modification des titres de navigations quant aux informations recensées et à leur forme. L'évolution des titres va se poursuivre dans les prochaines années, pour renforcer la convergence entre les règles rhénanes et communautaires.

Les certificats de capacité ont également fait l'objet d'une harmonisation qui sera probablement prochainement renforcée pour rapprocher les règles rhénanes et communautaires (aujourd'hui avec un certificat de capacité européen on ne peut pas aller sur le Rhin non canalisé).

Une directive prévoit également la création d'un numéro d'identification unique des bateaux au niveau européen (ce numéro est déjà attribué au bateau lors d'une nouvelle demande ou d'un renouvellement de titres) auquel sont associées des données obligatoires. Pour atteindre cet objectif, une base européenne des bateaux est en cours de création.

Il est probable qu'à moyen terme les titres de conduite soient eux aussi recensés au sein d'une base européenne.

Les règles relatives aux matières dangereuses sont également harmonisées au niveau rhénan et communautaire (ADNR et ADN).

1.1.2. Existant organisationnel

La maîtrise d'ouvrage est assurée par le bureau du transport fluvial (DGITM/DST/PTF/PTF3)

Les services sont chargés actuellement d'organiser les examens des permis de conduire professionnels et de délivrer les certificats correspondants (également pour les bateaux de plaisance mer et rivière), de délivrer les titres de navigation, d'immatriculer les bateaux de plaisance et de commerce, de délivrer les certificats de jaugeage, les attestations matières dangereuses et les attestations radar et passagers ;

La réglementation de la navigation fluviale de commerce est réglementée de façon différente selon que le bateau navigue sur le Rhin ou les eaux intérieures autres que le Rhin. Le Rhin est classé comme zone R. Dans le premier cas, la réglementation à respecter est établie par la Commission Centrale pour la navigation sur le Rhin (CCNR), dans le second, elle est établie par l'État français sur la base de textes européens.

Ces deux réglementations distinguent les bateaux transportant des marchandises (bateaux de marchandises) de ceux transportant des passagers (bateaux à passagers).

Les voies d'eau concernées de la Communauté sont classées en quatre zones navigables. Chaque État membre peut apporter des changements dans la classification de ses voies d'eau, sous réserve de communiquer ces changements à la Commission au moins six mois à l'avance :

- Zone 1 : Zones avec des conditions de navigation particulières, ces zones n'existent pas en France.
- Zone 2 : En général les zones fluvio-maritimes
- Zone 3 : Grands axes fluviaux définis par chaque état membre.
- Zone 4 : Tout le reste

Actuellement, il n'existe en France que les zones 2, 4, et R, mais les grands axes fluviaux comme le Rhône, la Seine ou l'Oise devraient prochainement être classés en zone 3.

➢ *Réorganisation des services*

Suite à leur réorganisation qui s'est achevée en 2009, les services instructeurs de métropole ont maintenant les mêmes compétences administratives. Les six services instructeurs agissent sous l'autorité du préfet compétent (six pour toute la France), ils sont à même de :

- délivrer les titres de navigation
- immatriculer les bateaux,
- organiser les examens et délivrer les certificats de conduite,
- délivrer les certificats de jaugeage,
- délivrer les certificats ADN/ADNR,
- délivrer les attestations radar et passagers.

Il est également possible qu'un service instructeur soit créé en Guyane.

Pour mener à bien ces missions, ils utilisent le système d'information NAVIFLU.

➢ *Services acteurs, rôles*

- La maîtrise d'ouvrage : la maîtrise d'ouvrage est assurée par la Direction Générale des Infrastructures, des Transports et de la Mer (DGITM), pour la réalisation de législation/réglementation et de son évolution (sauf matières dangereuses), l'animation des services instructeurs et de la définition des règles fonctionnelles.
- Les bateliers : les usagers/bateliers effectuent les demandes de tous les documents utiles à la navigation du bateau et à la conduite des bateaux auprès des 6 services instructeurs.

Les usagers doivent fournir aux services instructeurs tous les documents utiles à l'instruction de leur demande

- Les services instructeurs : les 6 services instructeurs sont les services navigation du Nord-Pas-de-Calais, de Rhône Saône, de la Seine, de Strasbourg, de Toulouse et de la DDTM (DDTM Direction Départementale des Territoires et de la Mer) Loire-Atlantique. Ils instruisent les demandes des usagers et délivrent les documents.
- Les commissions de visite : elles interviennent pour le compte du service instructeur, dans le cadre de la délivrance des titres de navigation, leur composition est définie par arrêté préfectoral
- Appui technique : le Centre d'études Techniques Maritimes et Fluviales apporte un appui technique à la maîtrise d'ouvrage et aux services instructeurs
- Assistance aux utilisateurs : l'assistance aux utilisateurs est réalisée par le PND Transports situé au DOM

1.1.3. Existant fonctionnel et Technique

Le système d'information de la navigation fluviale NAVIFLU est composé de trois applications utilisant l'architecture ACAI préconisée au sein du ministère. Les trois applications s'appuient sur une seule base de données.

Elles ont été développées à des périodes différentes et leur technologie ne répond plus complètement aux normes actuelles tant sur le plan ergonomique que technique. Le défaut au niveau technologique peut poser des problèmes pour mettre en œuvre des opérations de maintenance par des équipes différentes de celle qui a assuré le développement, la documentation des applications est dans certains cas incomplète.

Les applications et la base de données sont hébergées au centre serveur de Bordeaux. L'intégration des nouveaux modules est susceptible de poser des problèmes lors de leur intégration.

Les services instructeurs accèdent aux applications par un navigateur internet : Firefox ou Internet Explorer.

Les services instructeurs de Lille et Paris utilisent un fichier tableur pour gérer le suivi des demandes des usagers et le suivi de l'instruction des données.

La plupart des documents utiles à l'échange avec les usagers sont modifiés par traitement de texte alors qu'ils sont normalement édités directement à partir de l'application.

- Eléments et volumétrie

On est passé de 79 (Figure IV-1) à 91 tables dans ITINAVI.

Application	Nombre de tables	Nombre de champs
ETNA	37	314
GROBATO	33	253
ITINAVI	79	694

Figure IV-1 : éléments et volumétrie

1.2. Synthèse des besoins retenus

La synthèse des besoins présentée ci-après est faite à partir des demandes exprimées dans l'étude d'opportunité réalisée en 2009

La première livraison de l'application ITINAVI doit répondre aux besoins suivants :

- Reprendre les données contenues dans la base existante.

- La création d'une base bateau :
  - la création des données des bateaux se fait via les procédures autorisées pour un bateau. La base bateau sera donc alimentée au fur et à mesure par les données des bateaux.
  - affichage de toutes les données administratives et techniques d'un bateau qui auront été saisies dans toutes les procédures (immatriculation, CC, CV, CCS, jaugeage, etc.)
  - affichage de la liste de tous les documents du bateau (immatriculation, titre, etc.)
  - affichage des procédures autorisées pour ce bateau
- Gestion des procédures existantes liées à la navigation du bateau : immatriculation, CB, CVP, certificat communautaire (CC), certificat de visite (CV) et certificat communautaire supplémentaire (CCS)

Les processus métiers étudiés pour ce lot sont : visualiser bateau, gestion des immatriculations et des titres de navigation.

2. LES OBJECTIFS ET EXIGENCES D'ITINAVI

Le principal objectif de la maîtrise d'ouvrage est le développement d'une application complète et ouverte à toute évolution réglementaire ou technique en centrant les processus métier sur l'objet métier : le bateau.

2.1. Les objectifs

Le but est de poursuivre et finaliser le développement des modules manquants de l'application ETNA en prenant en compte les éventuelles évolutions réglementaires prévues ou prévisibles. Les problèmes de compatibilité fonctionnelle existants entre les diverses applications doivent être résolus et permettre ainsi l'exploitation des données à des fins statistiques en prenant en compte les demandes d'évolution formulées par les utilisateurs

Cependant, d'autres objectifs, plus spécifiquement informatiques, existent

- résoudre les problèmes signalés lors de l'intégration des dernières versions d'ETNA sur le centre serveur (liée au développement ACAI)
- limiter les coûts de maintenance

2.2. Les exigences

Le système d'information à réaliser est soumis à deux sortes d'exigences :

2.2.1. Exigences fonctionnelles

Les exigences fonctionnelles listées ci-après sont issues du cahier des charges initial, de l'étude d'opportunité et de différentes informations apportées lors d'échanges avec la maîtrise d'ouvrage. Ces exigences seront traduites par des traitements décrits dans les différents cas d'utilisation de l'application.

L'expression des exigences fonctionnelles est, comme nous le verrons, l'émanation des textes de loi et des règlements.

- Développer une application complète et ouverte à toute évolution réglementaire ou technique. En centrant les processus métier sur l'objet métier : le bateau
- Non-régression par rapport aux fonctionnalités existantes dans ETNA ET GROBATO
- Résoudre les problèmes de compatibilité fonctionnelle existants entre les diverses applications et faciliter l'exploitation des données à des fins statistiques, pour LOT 2
- Poursuivre et finaliser le développement des modules manquants de l'application ETNA en prenant en compte les éventuelles évolutions réglementaires prévues ou prévisibles
- Modernisation du transport fluvial et son développement avec une meilleure sécurité de la navigation
- Prise en compte simplifiée des évolutions de la réglementation
- Sécurité accrue de la délivrance des documents administratifs (certificats, titres...) ;
- Simplifier les relations avec les usagers et faciliter le travail des agents
- Reprendre les données des applications ETNA et GROBATO
- Lors du transfert des données, pas de perte de données, et pas de conflit
- Reprise des données automatique et mention des éventuelles incohérences

➢ ***Profils :***

Prévoir plusieurs profils dont un niveau MOA, un niveau chef de service, un niveau utilisateur et un profil demandeur. Pour chacun des profils un niveau de consultation.

Pour le suivi : le chef de service voit tous ses agents, la MOA voit tous les services, les agents voient leurs suivis.

2.2.2. Exigences non fonctionnelles

L'application devra être conçue de façon à être facilement évolutive et de maintenance aisée. L'écriture du logiciel sera en architecture ACAI en suivant les guides d'architecture commune des applications informatiques dans le but d'une appropriation facile par l'utilisateur. L'accès aux informations doit être sécurisé, des droits d'accès doivent être gérés selon le type d'utilisateur (gestion de profils par CERBERE). L'ergonomie est un facteur important, la simplicité doit primer sur le nombre d'options et de paramètres éventuels, les écrans devront être simples et bien ciblés.

Les temps de réponse doivent être acceptables : ne pas dépasser 2 secondes pour l'affichage d'une page dans le cas de forte affluence. Il est préconisé un temps de réponse inférieur à 0.5s pour un fonctionnement normal ; en mode dégradé suite à incident, il sera toléré un temps de réponse de 10s maximum.

La liste minimum des documents à fournir est :

- document de conception de l'application
- schéma de la base de données
- manuel utilisateur
- manuel administrateur
- manuel d'installation et de para métrage des différents serveurs
- les sources complètes de l'application (le ministère devra être le propriétaire des sources)

3. REGLES ACAI ET PERIMETRE INFORMATIQUE

L'objectif principal des conventions de codage est de permettre une bonne compréhension et une bonne maintenabilité du code.

Tout développeur, qui lit le code d'une méthode pour la première fois, doit facilement en comprendre le contenu. Si ce n'est pas le cas, la maintenance de l'application sera difficile.

Il faut toujours développer en sachant que le code sera revu et maintenu par une autre personne, ce qui nécessite un respect des normes, l'utilisation de commentaires, une clarté du code plutôt que l'utilisation d'expressions confuses...

Il faut veiller à ne pas penser trop tôt à l'optimisation de code. Toute optimisation se fait au détriment de la lisibilité et elle n'a pas sa place durant la phase de développement.

L'application de ces règles par les prestataires est obligatoire ; leur mise en œuvre peut être contrôlée en pratique, par la maitrise d'œuvre, notamment par des revues de livrables. La non-application de ces règles doit pouvoir être justifiée sur demande.

3.1. Règles ACAI

ACAÏ [Acai, 1] définit l'environnement technique pour le développement des applications web du ministère.

Les règles ACAI, un guide à la conception et à la réalisation des applications, est une synthèse des recommandations du ministère à l'adresse des équipes en charge des projets de réalisation technique internet ou intranet. Il donne des recommandations pour leur conception technique, leur construction, leur intégration et leur déploiement.

Les règles ACAI, recensent un ensemble de règles ou d'orientations, techniques ou non, qui ont pour objectif, d'une part de canaliser et d'homogénéiser les efforts des équipes de développement, d'autre part de fournir les bases d'applications robustes, performantes et maintenables.

3.1.1. Conventions de codage

Respecter impérativement les normes de codage qui, de part leur logique, sont très simples à mettre en œuvre et permettent un travail en équipe efficace (partage de code).

Le respect des normes doit être pris en compte dès le début du codage et non pas quelques mois plus tard lors d'une revue de code.

Une règle de codage non obligatoire peut ou non être appliquée à l'écriture des sources de l'application. Cependant, dès lors qu'elle est retenue elle doit être généralisée à l'ensemble des sources, et non pas appliquée sporadiquement selon chaque développeur différent.

Si, pour une raison ou une autre, il est impossible pour le développeur de suivre une norme retenue pour le projet, il faut impérativement notifier (voire justifier) ce non-respect par un commentaire approprié dans le code.

3.1.2. Éléments d'optimisation

- Éviter autant que possible les instanciations d'objets à l'intérieur des boucles.
- Bannir lorsque c'est possible les algorithmes d'une complexité quadratique ou pire.
- Vérifier ou établir dès le début du projet si l'espace mémoire des applications déployées en production est partagé ou si l'espace mémoire de chaque application est isolé des autres applications.
- Penser à l'optimisation des accès aux classes Java.
- Utiliser les collections non synchronisées sauf lorsqu'une synchronisation est nécessaire.
- Utiliser des StringBuilder plutôt que des concaténations de String.

3.1.3. Gestion des exceptions

Prendre en compte le traitement des erreurs dans tous les développements.

Délimiter, avec le bloc try, seulement les expressions qui sont susceptibles de déclencher une exception. Ne pas perdre de précision en englobant un plus grand nombre d'expressions. Il ne doit pas y avoir de bloc catch vide.

Toute exception doit être traitée et journalisée. Le journal doit contenir le message qui a été présenté à l'utilisateur, le contexte ainsi que l'exception avec sa cause originelle et les messages techniques associés. Les exceptions sont transmises et encapsulées au travers des différentes couches de l'application. Les exceptions des couches techniques ne doivent pas être propagées dans les couches applicatives. Elles doivent être transformées en exceptions applicatives. Ne pas traiter globalement des exceptions avec la classe Exception, sauf au niveau le plus haut de l'application où il est nécessaire d'intercepter toutes les exceptions non traitées par ailleurs. Il ne faut absolument pas perdre l'exception originale, pour qu'elle puisse être journalisée correctement.

Toute erreur remontant jusqu'à la couche client doit être affichée à l'utilisateur de manière claire et compréhensible. L'utilisateur sera redirigé sur une page d'erreur spécifique en cas d'exception bloquante pour le reste de l'application.

3.2. Périmètre informatique

Il existe quatre acteurs fonctionnels :

- Profil consultant, il permet la visualisation du bateau et de ses titres de navigation,
- Profil instructeur des immatriculations,
- Profil d'instructeur des titres de navigations,
- Profil administrateur qui permet de gérer les tables de références de l'application.

➢ *Authentification*

L'ensemble des droits d'accès à l'application sera géré par Cerbère. Le déploiement de Cerbère, le serveur d'authentification et de gestion des droits des utilisateurs dans l'application web s'inscrit dans le cadre de l'architecture commune des applications informatiques ACAI.

Intitulé	Code	Habilitation	Peut aussi
Administrateur	Admin	Gestion du référentiel	Consultant
Instructeur titres de navigation	InstNav	Gestion des titres de navigation du centre instructeur d'appartenance	Consultant National
Instructeur Immatriculation	InstImmat	Gestion des titres des immatriculations du centre instructeur d'appartenance	Consultant National
Consultation	Consult	Consulter tous les bateaux et leurs actes administratif associés.	

Figure IV-2 : habilitation Cerbère

Cerbère propose une procédure normalisée qui garantit la sécurité et le contrôle des accès aux données et aux programmes. Ce dispositif générique permet de gérer l'authentification des utilisateurs (internes ou externes) accédant à un site applicatif et les droits d'accès aux différents modules d'une application.

Cette procédure standard permet de réduire le coût des développements de la sécurité dans les applications.

4. SYNTHESE

Les instances européennes et nationales souhaitent privilégier le développement du transport fluvial, un mode de transport économique, fiable et peu polluant. Pour améliorer la compétitivité du transport par voie d'eau, une amélioration du système d'information existant de la navigation fluviale NAVIFLU est nécessaire. NAVIFLU est composé de trois applications utilisant l'architecture ACAI préconisée au sein du ministère. Les trois applications s'appuient sur une seule base de données. Elles ont été développées à des périodes différentes et leur technologie ne répond plus complètement aux normes actuelles tant sur le plan ergonomique que technique.

Le but est de poursuivre et finaliser le développement des modules manquants de l'application ETNA en prenant en compte les éventuelles évolutions réglementaires prévues ou prévisibles. Le développement doit se faire en respectant les règles ACAI.

Les processus métiers étudiés pour ce lot sont : visualiser bateau, gestion des immatriculations et des titres de navigation.

V. REALISATION

CERTIFICAT COMMUNAUTAIRE

Dans ce chapitre nous allons voir tous les aspects techniques utilisés pour mettre en œuvre l'application ITINAVI.

D'abord une description fonctionnelle du module CC et les outils de mise en œuvre de l'application. Des outils dont la plupart sont choisis par le client ; mais nous allons voir aussi les outils que nous avons ajoutés et leurs apports notamment maven et l'intégration continue. Maven permet de faciliter et d'automatiser la gestion et la construction d'un projet java ; l'intégration continue nous a permis d'accélérer les livraisons du logiciel en réduisant le temps d'intégration.

Ensuite, nous abordons l'architecture de l'application : une architecture en couche dont les recommandations pour sa conception technique, sa construction, son intégration et son déploiement sont données par les règles ACAI.

ACAÏ définit l'environnement technique pour le développement des applications web du ministère. Ce guide à la conception et à la réalisation des applications, est une synthèse des recommandations du ministère à l'adresse des équipes en charge des projets de réalisation technique internet.

Enfin nous concluons ce chapitre par une synthèse.

1. DESCRIPTION FONCTIONNELLE

La gestion de la partie administrative du bateau (immatriculation) a été gérée dans le lot 1. Dans le lot 2 (Certificat Communautaire ou CC) on gère la partie technique du bateau.

1.1. Certificat communautaire

L'objectif principal du sous paquetage est de gérer le suivi des Certificats Communautaires Il s'agit pour ce sous-domaine de : saisir, visualiser les informations des certificats communautaires.

1.1.1. Créer Certificat Communautaire

La création du CC est étudiée dans le cas d'utilisation paragraphe 5.4

1.1.2. Gérer Récupération Modification CC

Cette procédure permet de saisir une modification ou un changement de pages effectué à l'étranger d'un CC de bateau initialisé en France.

Cette modification envoyée par le service étranger est à enregistrer par le service instructeur qui a initialisé le premier CC du bateau.

Cette manipulation permet de tenir à jour les données du CC enregistrées dans ITINAVI. Cette procédure est à utiliser pour toutes modifications, prolongation/confirmation, prolongation gaz liquéfiés, suppression gaz liquéfiés effectuées à l'étranger.

- Evénement déclencheur : réception de la photocopie de la page modifiée ou remplacée par une commission étrangère

Le CC du bateau doit être présent dans la base de données et en cours de validité. Si le CC n'est pas dans la base, l'utilisateur devra au préalable faire une reprise (ou récupération) du certificat dont il est le signataire.

Ce n'est qu'ensuite qu'il saisira les modifications des pages de son CC (les données saisies lors de la reprise lui seront réaffichées)

- L'enregistrement des modifications du certificat sont effectuées page par page de façon indépendante.

L'utilisateur renseigne les zones modifiées ainsi que les zones nécessaires à l'identification du signataire.

La première page du CC n'est pas accessible en modification car elle n'est pas modifiable, ni remplaçable.

Toutes les pages peuvent être modifiées, car même les prolongations/confirmations émises à l'étranger devront être saisies :

Les onglets 9 et 10 concernant la procédure de prolongation/confirmation et l'onglet 12 pour la prolongation de l'attestation relative aux gaz liquéfiés ne seront pas accessibles.

1.1.3. Gérer Prolongation/ Confirmation CC

La procédure permet à un centre instructeur de prolonger ou de confirmer un CC déjà existant dans la base de données.

Cette procédure s'applique aussi bien aux bateaux français qu'aux bateaux étrangers.

Le premier onglet du CC n'est pas accessible en prolongation/confirmation car elle n'est pas modifiable L'utilisateur renseigne éventuellement les modifications des autres pages.

- Les onglets 2 à 8 l'onglet 11 pour l'attestation relative aux gaz liquéfiés
- l'onglet 13 pour les annexes
- l'onglet 12 pour la prolongation de l'attestation relative aux gaz liquéfiés

Lors d'une prolongation /confirmation les pages éditées sont les pages 9 et 10 du CC. Ces pages seront remplacées lors de cette procédure en fonction du nombre de prolongations déjà effectuées à partir de la 7ème prolongation/confirmation c'est la première procédure de prolongation qui disparait à l'écran et à l'édition (on ne conserve sur le titre courant que les 6 dernières procédures de prolongations/confirmations.

Après décision, les zones suivantes nécessaires à l'identification du signataire de la prolongation/confirmation sont affichées et seront imprimées. : Lieu, date, de signature, commission, de surveillance, signataire.

1.1.4. Gérer Remplacement CC

La procédure de remplacement est utilisée pour remplacer un CC déjà existant dans la base de données dont le délai de validité du certificat en cours est dépassé.

Un nouveau numéro chronologique sera attribué lors du remplacement du CC.

Cette procédure s'applique aussi bien aux bateaux français qu'aux bateaux étrangers. Elle permet de remplacer le dernier CC connu (et à l'état validé)

Après affichage du CC actuel, l'instructeur vérifie et met à jour s'il y a lieu : l'ensemble des onglets 1 à 8, l'onglet 11 s'il y a une attestation relative aux gaz liquéfiés, l'onglet13 s'il y a des informations à porter en annexe, les zones de bas de page des onglets 1 à 8.

Les pages suivantes ne doivent pas être accessibles à la saisie : L'onglet 12 concernant la procédure de prolongation de l'attestation relative aux gaz liquéfiés et les onglets 9 et 10 concernant la procédure de prolongation, confirmation du CC. En effet les pages 9 et 10 du précédent certificat sont remises à blanc.

2. OUTILS

Le choix des outils, comme la version java et le serveur, est fait par le CETE ; nous avons mavenisé le projet pour pouvoir l'intégrer à notre plateforme d'intégration continue (IT).

2.1. JAVA

Le langage de développement de l'application est JAVA. Il offre une excellente portabilité d'un système à l'autre, et bénéficie du support d'une large communauté de développeurs qui met à disposition de tous sur Internet une vaste étendue de bibliothèques, d'outils et de conseils qui facilitent le développement d'applications.

Java, issu des laboratoires de Sun Microsystems en 1995, a l'avantage d'être un langage de programmation orientée objets dont les environnements de développement et d'exécution (la « machine virtuelle ») sont gratuits. De ce fait, Java est massivement utilisé en milieu universitaire et professionnel, ce qui explique son succès.

La version du JDK (Java Development Kit) utilisée est 1.6

2.2. Serveur et base de données

L'application est déployée sur le serveur Tomcat 6. Apache Tomcat est l'un des composants du projet Jakarta. C'est un conteneur de servlet JEE, qui implémente les spécifications des servlets et des JSP de Sun Microsystems. Il inclut des outils pour la configuration et la gestion.

Le modèle de données ainsi que la base de données ont étaient fournis par le CETE. Toute modification du modèle doit être faite chez eux. On utilise le SGBD (système de gestion de base de données relationnelle) postgresql, version 8.4

➔Nous avons dû revoir le modèle de données, pour ensuite leur faire des propositions de changement, chose qu'ils ont accepté malgré leur réticente au début. Ceci est dû à la confiance qui s'est installé entre notre équipe et la MOA.

2.3. Mantis : gestion et suivi des anomalies

Nous utilisons Mantis comme logiciel spécialisé pour la gestion et le suivi des anomalies.

Mantis est un système de suivi d'anomalies logicielles (bugs) basé sur une interface web. Il est écrit en PHP, requiert une base de données et un serveur web.

Figure V-1 : Gestion des incidents par Mantis

Le principe de cet outil consiste à enregistrer la déclaration d'un bug informatique, puis, pour les techniciens de maintenance informatique concernés, à mettre à jour l'avancement de sa résolution, jusqu'à sa clôture. Le déclarant de l'anomalie peut s'informer à tout moment via le serveur Web de l'avancement du traitement de son problème.

Cet outil nous a permis de gérer les demandes de correction d'anomalies et les demandes d'adaptation/évolution remonté par le client.

2.4. Eclipse Galileo

Pour le développement, on utilise Eclipse Galeleo, un environnement de développement intégré libre extensible, universel et polyvalent. La spécificité d'Eclipse IDE (Integrated Development Environment) vient du fait de son architecture totalement développée autour de la notion de plugin : toutes les fonctionnalités de cet atelier logiciel sont développées en tant que plug-in.

On utilise plusieurs des ses plugins, notamment Subclipse.

2.4.1. Subclipse (SVN) : gestion des versions

Le développement des logiciels implique généralement des développements parallèles, avec des développeurs qui intègrent leurs travaux sur un produit unique. Tous les changements des développeurs ont besoin d'être suivis pour pouvoir identifier et résoudre les problèmes potentiels qui pourraient apparaitre suite à l'intégration de ces changements.

Pour ITINAVI comme pour d'autres projets, nous utilisons SVN *[svn]*, un système de gestion de versions qui propose clairement des avantages qui sont indéniables et qu'il faut souligner.

- les commits, ou publications des modifications sont atomiques. Un serveur Subversion utilise de façon sous-jacente une base de données capable de gérer les transactions atomiques ;
- Subversion permet le renommage et le déplacement de fichiers ou de répertoires sans en perdre l'historique ;
- les métadonnées sont versionnées : on peut attacher des propriétés, comme les permissions, à un fichier, par exemple.

Les opérations à faire :

- On checkout (récupérer en local une version ainsi que ses méta-données depuis le dépot) pour avoir une copie en local
- Avant de coder on fait un update (mettre à jour la copie locale existante depuis la dernière version disponible sur le dépôt)
- On modifie le code
- Avant de commiter on fait un update pour résoudre les conflits éventuels
- On committe nos modifications

SVN permet la consultation et la possibilité de restauration des anciennes versions d'un fichier ; ainsi que de savoir les raisons des modifications apportées et les auteurs de celles-ci. Les propagations de version (commit) sont atomiques. Une propagation réussit uniquement si tous les fichiers de la version sont correctement propagés. Les numéros de versions concernent une propagation et non les fichiers eux-mêmes.

2.5. Pourquoi Maven ?

On a procédé à la mavenisation du projet ITINAVI pour profiter des avantages de cet outil *[Maven, 1]*.

Maven est un outil « open source » d'Apache Jakarta (Jakarta est un ensemble de projets de logiciels libres, écrits en langage Java, développés par la fondation Apache de manière collaborative et consensuelle et tous publiés sous licence Apache). Il permet de faciliter et d'automatiser la gestion et la construction d'un projet java. Il systématise, rationalise et simplifie le développement collaboratif de projets Java, faisant gagner aux entreprises comme aux développeurs du temps et de l'argent ! Il offre des outils de gestion de projet de haut niveau et offre des fonctionnalités réutilisables.

2.5.1. Le modèle objet projet

Le cœur d'un projet Maven 2 est le modèle objet projet (appelé POM pour Project Object Model). Il contient une description détaillée de votre projet, avec en particulier des informations concernant le versionnage et la gestion des configurations, les dépendances, les ressources de l'application, les tests, les membres de l'équipe, la structure et bien plus encore. Le POM prend la forme d'un fichier XML (pom.xml) qui est placé dans le répertoire de base de votre projet.

Une partie du fichier pom.xml du projet **ITINAVI** est présenté ci-dessous :

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
      <modelVersion>4.0.0</modelVersion>
      <groupId>i2.application</groupId>
      <artifactId>itinavi</artifactId>
      <version>0.2.0</version>
      <packaging>war</packaging>
      <name>CETE - Itinavi</name>
      <description>projet du CETE Itinavi</description>
      <url>http://itinavi.developpement-durable.gouv.fr</url>
      <developers>
                  // l'équipe du projet
      </developers>
      <dependencies>
                  <dependency>
                              <groupId>org.springframework</groupId>
                              <artifactId>spring-core</artifactId>
                              <version>2.5.5</version>
                  </dependency>
                  // les autres dependences du projet nécessaires au projet
                  // pour avoir les librairies : struts, hibernate, jasper, tiles,…
      </dependencies>
</project>
```

Sans maven, on devrait ajouter les librairies une par une au projet, alors que dans notre projet mavenisé, la simple configuration du POM permet d'aller chercher tous les artefacts téléchargés depuis le repository distant et les mettre dans le repository local. Le repository local est un répertoire sur le poste du développeur, situé à ce chemin : « %USER_HOME%/.m2/repository », permettant de stocker les jars « librairies » nécessaires au projet.

2.5.2. La structure du répertoire MAVEN

Une partie de la puissance de Maven vient des pratiques standardisées qu'il encourage. Un développeur qui a déjà travaillé sur un projet Maven se sentira tout de suite familier avec la structure et l'organisation d'un autre projet Maven. Il n'y a pas besoin de gaspiller du temps à réinventer des structures de répertoires, des conventions.

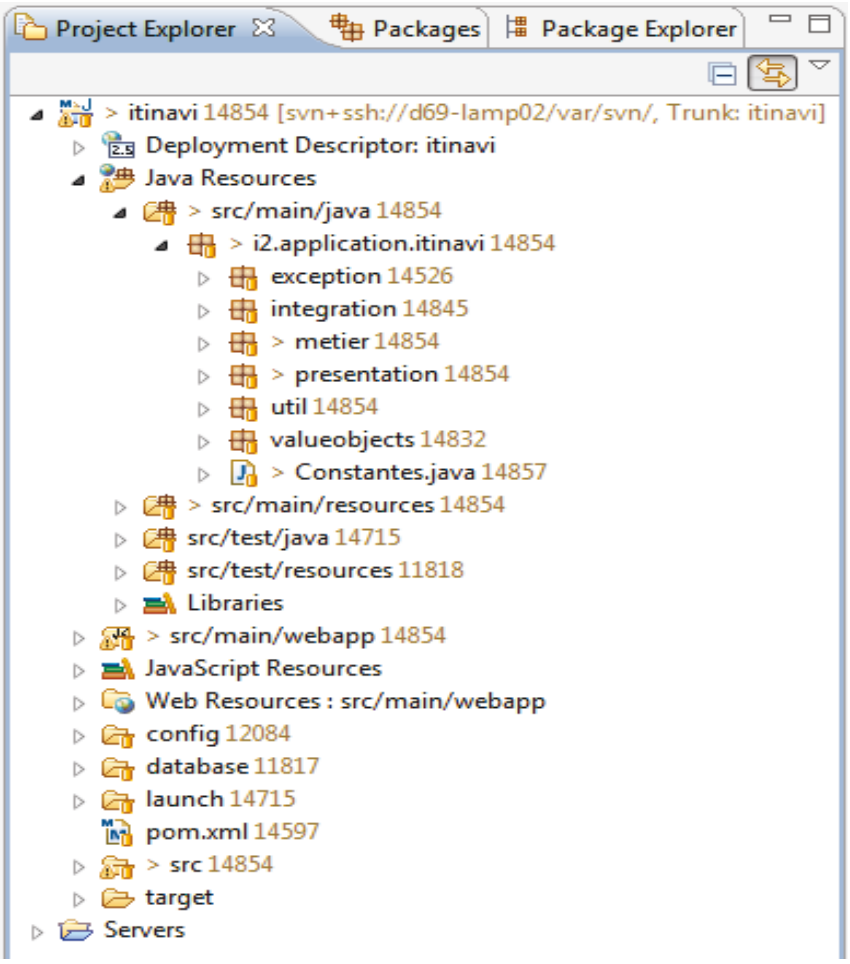


Figure V-2 : le modèle de répertoire du projet ITINAVI mavenisé

La figure ci-dessus présente la liste des répertoires du projet :

- /src : les sources du projet
- /src/main : code source et fichiers source principaux
- /src/main/java : code source, on y trouve des packages pour les différentes couches (présentation, métier et intégration)

- /src/main/resources : fichiers de ressources configuration (strtuts-config.xml, tiles-defs.xml..)
- /src/test/java : code source de test
- /src/test/resources : fichiers de ressources de test
- /src/webapp : webapp du projet contient notamment les fichiers jsp
- /target : fichiers résultat, les binaires (du code), les packages générés et les résultats des tests

On remarque le fichier POM à la racine du projet.

2.5.3. Maven a de multiples apports :

Maven propose de nombreuses fonctionnalités qui répondent plus aux standards et aux besoins des développeurs d'aujourd'hui. Il est aussi plus flexible car il permet de créer plus facilement ses plugins. Maven fournis des fonctionnalités réutilisables.

- Les projets 'maven' ont tous la même structure, ainsi, les développeurs passent plus facilement d'un projet à un autre
- Il permet de réaliser une intégration continue (Hudson)
- Permet la gestion des dépendances (jar), leurs versions, et les conflits associés, cela simplifie les montées de version d'une dépendance

Figure V-3 : automatisation de tâches

Le projet mavenisé nous permet d'automatiser un maximum de tâches. Dans le répertoire « launch » du projet ITINAVI (Figure V-3) on trouve les fichiers de configuration que nous avons créé et qui nous permettent, après un simple click de :

- Générer automatiquement une archive war, fichier « INT-package.launch » dont le contenu est le suivant :

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<launchConfiguration type="org.maven.ide.eclipse.Maven2LaunchConfigurationType">
<booleanAttribute key="M2_DEBUG_OUTPUT" value="false"/>
<stringAttribute key="M2_GOALS" value="tomcat:deploy"/>
<booleanAttribute key="M2_NON_RECURSIVE" value="false"/>
<booleanAttribute key="M2_OFFLINE" value="false"/>
<stringAttribute key="M2_PROFILES" value=""/>
<listAttribute key="M2_PROPERTIES"/>
<stringAttribute key="M2_RUNTIME" value="EMBEDDED"/>
<booleanAttribute key="M2_SKIP_TESTS" value="true"/>
<booleanAttribute key="M2_UPDATE_SNAPSHOTS" value="false"/>
<booleanAttribute key="M2_WORKSPACE_RESOLUTION" value="false"/>
<stringAttribute key="org.eclipse.jdt.launching.VM_ARGUMENTS" value="-Xmx1048M -
Xms512M"/>
<stringAttribute key="org.eclipse.jdt.launching.WORKING_DIRECTORY"
value="${workspace_loc:/itinavi}"/>
</launchConfiguration>
```

- Possibilité de déployer sur un serveur d'intégration en une commande, fichier « INT – Tomcat deploy.launch », avec aussi la possibilité de redéployer, fichier « INT – Tomcat redeploy.launch » ou annuler le déploiment, fichier « INT – Tomcat undeploy.launch »
- Générer un site web complet du projet, fichier «INT – site.launch »

Maven seul est puissant. Maven couplé à notre plateforme d'intégration continue devient une machine à produire du logiciel. Construction automatique du projet, analyse de code, contrôle de qualité et suivie des indicateurs dans le temps, voila ce que permettent ces outils. De quoi améliorer à la fois la qualité des livrables et réduire des temps de développement.

2.6. Plateforme d'intégration continue

Dans le cadre de projets de développement logiciels réalisés par une équipe plus ou moins conséquente, l'évolution du code source de l'application est répartie entre ces plusieurs personnes. Le fait que chaque partie du code soit à la charge d'une personne différente, peut entrainer des régressions, des conflits ou bien tout simplement des bugs fonctionnels lors de la mise en commun des différentes parties du code (En particulier si chacun travaille dans son coin et que la communication dans l'équipe de développement n'est pas forcément le maitre mot).

Le résultat souvent constaté est donc que le code de chaque développeur fonctionne très bien indépendamment du reste mais lors du merge on peut constater des bugs pouvant aller du simple bug/fait amusant à une catastrophe complète. Cet état à la fâcheuse tendance d'entrainer des surcouts de développement importants afin de faire fonctionner l'applicatif final avec toutes les parties de code mergées.

L'intégration continue permet d'éviter cet état de fait. Il permet, aussi la construction automatisé et de délivrer des tests de qualités et des tests fonctionnels.

L'Intégration Continue (IC) est le nom donné initialement par la communauté de l'Extreme Programming (XP) pour désigner la pratique de génie logicielle visant à accélérer la livraison des logiciels en réduisant le temps d'intégration.

Pour que l'Intégration Continue puisse se faire correctement, il faut tout d'abord partager les sources du projet *[IC1]*. Il faut également que les équipes de développements postent (committent) régulièrement les modifications apportées au code en utilisant svn (paragraphe 5.2.4.1), de préférence plusieurs fois par jour.

Des outils vont alors se charger de vérifier régulièrement les modifications sur le SCM (Software Configuration Management) afin d'assurer la non régression de l'application, en compilant l'ensemble du projet.

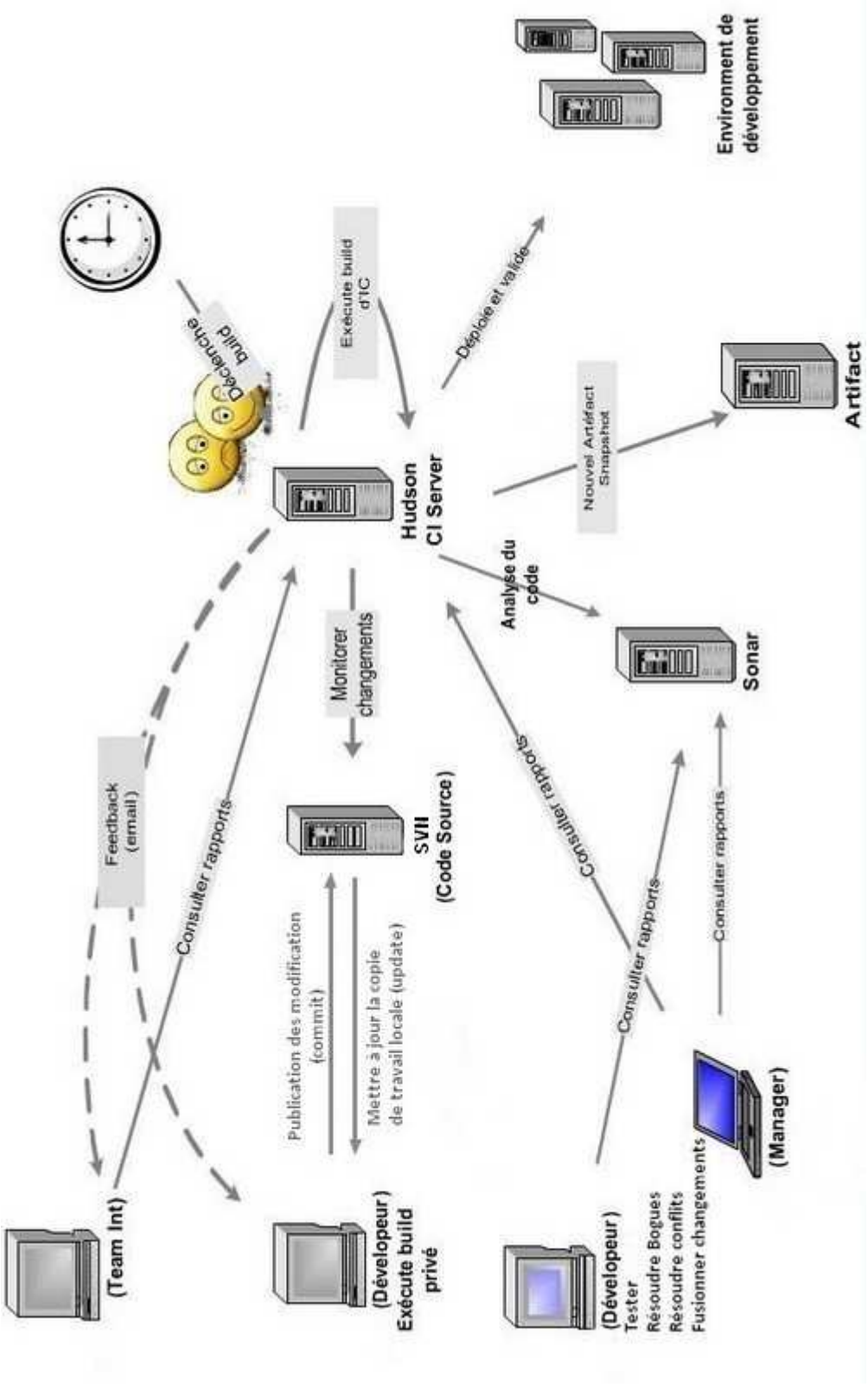


Figure V-4 : Environnement de l'intégration continue

2.6.1. Hudson

Hudson est un serveur d'intégration continue qui permet d'automatiser la reconstruction régulière d'une application afin de détecter le plus régulièrement possible des régressions potentielles. Le serveur d'intégration continue va régulièrement récupérer la dernière version du code source et exécuter des scripts de construction, plus potentiellement des jeux de tests automatisés, voire même déployer l'application, etc. Une interface permet de voir tous les problèmes détectés.

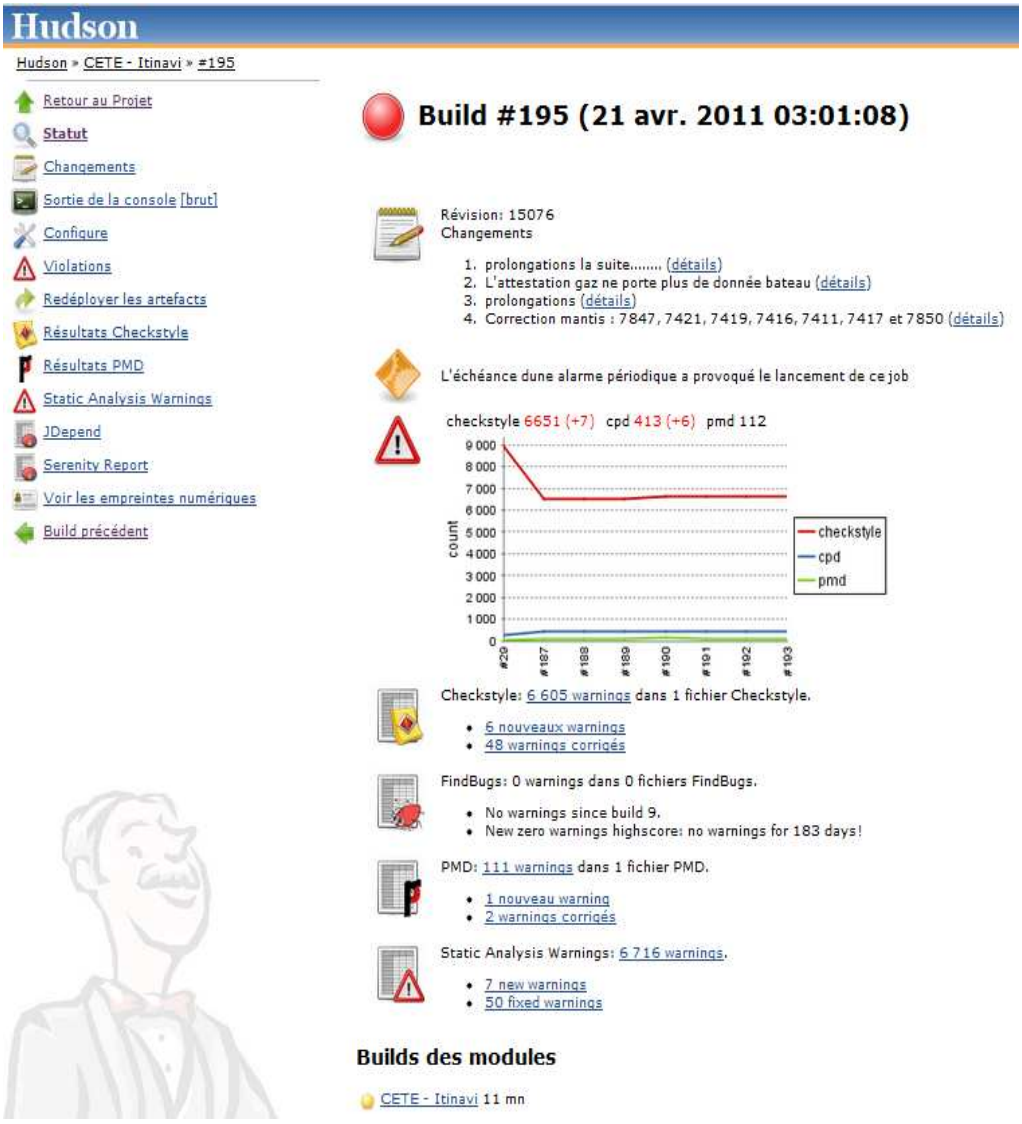


Figure V-5 : Analyse du projet ITINAVI

2.6.2. Sonar

Sonar est un outil qui permet d'évaluer rapidement et clairement la qualité des projets Maven2, mais aussi d'autres types de projets, moyennant une « Mavenisation » assez simple. Il s'appuie sur des outils comme Checkstyle, PMD et Cobertura et offre un reporting web très intuitif pour synthétiser les résultats de ces différents outils et guider l'utilisateur sur la voie de la qualité.

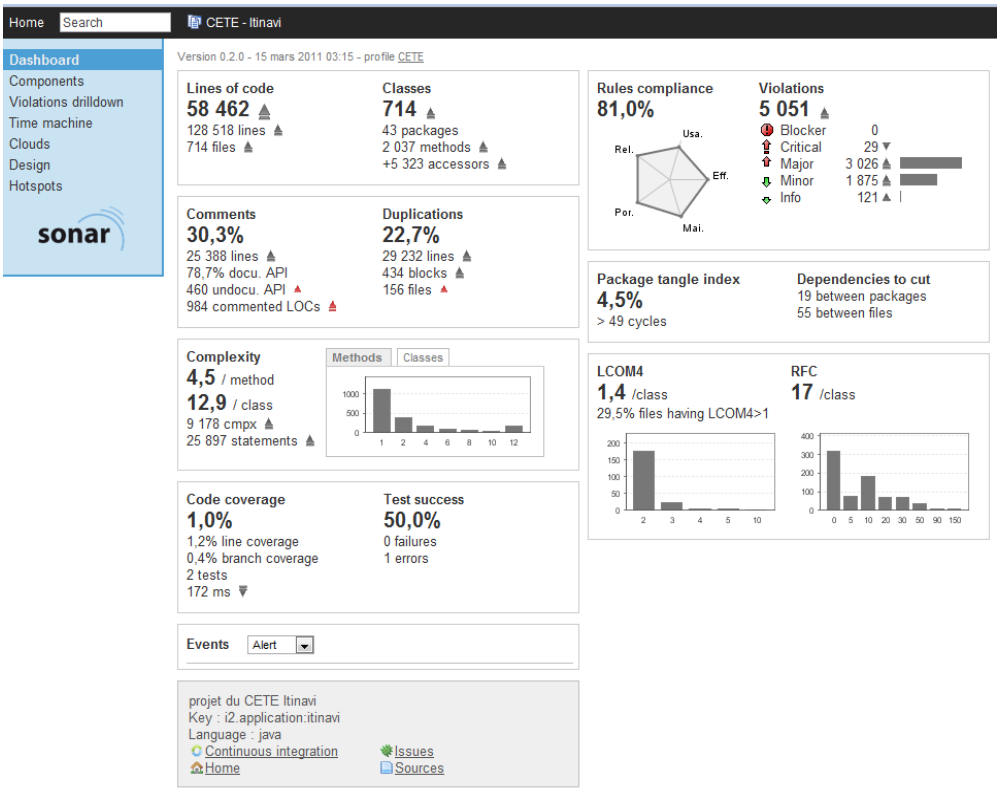


Figure V-6 : Tableau de bord Sonar du projet ITINAVI

Sonar est un serveur de contrôle de qualité logiciel, développé par la société Hortis. Le but principal de Sonar est de proposer une synthèse des résultats de différents outils de contrôle de qualité existant. À cela viennent s'ajouter plusieurs méthodes de calcul permettant d'évaluer la qualité globale du projet.

Sonar nous offre une bonne analyse propre du projet, il permet par exemple de diriger les développements grâce à ces nombreuses vues. La dashboard d'un projet rassemble une série d'analyses comme le montre la figure ci-

dessus (Figure V-6).On retrouve notamment le nombre de lignes de code, de commentaires, de classes, de packages, le pourcentage du code dupliqué (22,7%), etc.

2.7. Apport de l'IC au projet ITINAVI

Globalement, les principaux intérêts de l'IC sont les suivants :

- Vérification fréquente du code, et de sa bonne compilation ; ce qui permet de détecter quasi-immédiatement la présence d'un problème dans le code et donc d'y apporter une solution instantanément !
- Mise à disposition éventuelle d'une version testable comportant les dernières modifications du code.
- Possibilité de créer des rapports périodiques exprimant la qualité du code.

3. ARCHITECTURE GENERALE ADOPTEE

L'architecture en couches proposée pour ce projet est une architecture souple permettant de répondre à tous types de contraintes. La part de risque reste mince, puisque les frameworks utilisés (spring, struts, hibernate) sont des technologies arrivant à phase de maturation. Le CETE, en choisissant de miser sur des technologies émergentes, joue la carte de la stabilité et de l'évolutivité.

3.1. Principes généraux d'ergonomie

Les écrans de l'application doivent respecter des principes généraux d'ergonomie (Figure V-7). La méthode associée aux templates permet de réduire le code redondant dans une application web et dans le même temps de séparer l'ensemble du code de l'application de son apparence.

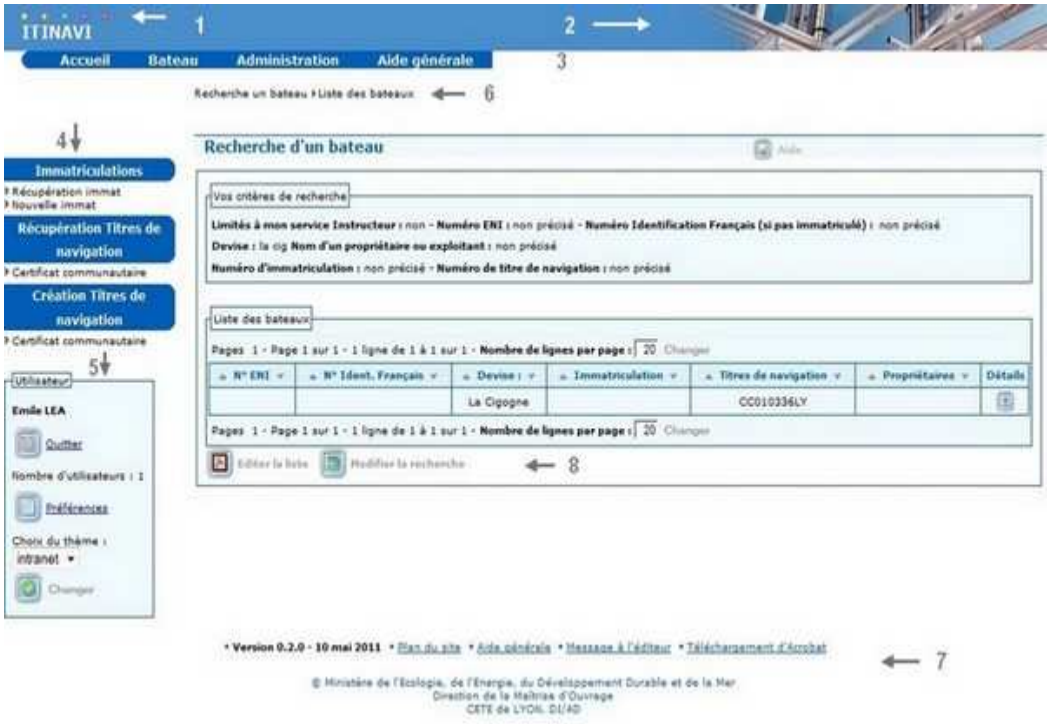


Figure V-7 : structure des pages

- Boutons raccourcis (1)

Le bouton jaune permet d'afficher la politique d'accessibilité, le bouton orange permet d'aller au contenu, le vert envoi à la page d'accueil et enfin le violet pour aller au menu contextuel.

- Bandeau spécifique à l'application (2)

Il est commun à l'ensemble des écrans de l'application et rappelle à l'utilisateur qu'il travaille sur l'application ITINAVI

- Menu principal (3)

Le clic sur une entrée du menu ouvre la page d'accueil correspondante à l'entrée du menu.

La gestion du menu, comme pour le menu contextuel, est gérée dans le fichier menu.xml, ce menu est le même pour toute l'application.

- Menu contextuel (4)

Le menu contextuel est spécifique à l'écran en cours. Il correspond au composant « menu ».

Dans le fichier menu.xml, on définit tous les menus contextuels de l'application, ensuite et selon l'emplacement où on est, on précise les menus à afficher pour la page en cours.

L'exemple suivant est extrait du fichier de configuration menu.xml du projet ITINAVI :

```
<?xml version="1.0" encoding="UTF-8" ?>
<menus>
  <menu nom="menuprincipal">
    <option nom="accueil">
      <texte>MENU_ACCUEIL</texte>
      <titre>Retour à l'accueil de l'application</titre>
      <url>/accueil.do</url>
    </option>
  </menu>
  <menu nom="menucontextuel">
    <menu nom="cc">
      <texte>MENUCONTEXTUEL_TITRES_CERTIFICAT_COMMUNAUTAIRE</texte>
      <option nom="modificationTitreCC">
        <texte>MENUCONTEXTUEL_TITRES_CC_MODIFICATION</texte>
        <url>/creerccinstruction.do?objetTitreNav=Modification</url>
      </option>
    </menu>
  </menu>
</menus>
```

C'est dans l'action qu'on précise la liste des menus à afficher, correspondant aux règles de gestion du CETE. Ensuite dans la classe du commun « MenuManager », où la réelle gestion des menus se déroule, on fait la mise à jour de la liste pour n'en garder que les menus de la page.

On remarque dans ce menu qu'on a des données en dur, comme l'«url» et le «titre» ce qui montre encore une fois les limites du Framework.

- Télécommande (5)

Identification de l'utilisateur connecté, bouton de déconnexion de l'utilisateur (présent sur toutes les pages) et paramétrage de l'affichage.

- Fil d'ariane (6)

Montre où la page se situe dans la hiérarchie du site web. Ils seront générés automatiquement.

C'est le composant « progression ».

- Pied de page (7)

Version de l'application et plan du site, coordonné. Il correspond au composant « pied ».

- Boutons d'action (8)

Les actions principales disponibles pour un utilisateur sont accessibles au travers de boutons composés d'un pictogramme et d'un libellé.

3.2. CETE « Commun » dans le framework

Le projet transmis par le CETE est décomposés comme suit :

- Un projet "commun ", qui contient :
  - Un répertoire "java" pour les classes génériques du Framework
  - Un répertoire "jsp" pour les jsp génériques du Framework
- Un projet java ITINAVI qui contient les éléments spécifiques au projet

Les classes génériques du « commun » regroupent un ensemble de variables et de méthodes communes aux classes d'objets. Ce sont surtout des classes abstraites, les classes héritantes n'ont pas forcément à réimplémenter les méthodes d'où une maintenance du code plus facile.

Classe abstraite : Une classe abstraite est une classe dont au moins une méthode n'est pas implémentée, elle permet de définir les caractéristiques communes à plusieurs classes. On indique qu'une classe est abstraite grâce au modificateur "abstract" au moment de déclarer le nom de la classe. Le nom "abstrait" pour désigner une telle classe est donc bien choisie, car la classe n'est pas complètement écrite.

Pour les méthodes qui ne sont pas implémentées, on met uniquement la signature de la méthode (suivie du point virgule), précédée de "abstract". Une classe abstraite ne peut pas être instanciée, normal car tout son code n'est pas écrit. Une classe qui hérite d'une classe abstraite est instanciable uniquement si elle implémente toutes les méthodes abstraites. Dans le cas contraire, cette nouvelle classe est aussi abstraite.

Une classe abstraite permet à un développeur :

- de fournir à d'autres développeurs une partie de l'implémentation d'une classe
- de laisser aux autres développeurs la manière d'implémenter le reste de la classe
- d'imposer aux autres développeurs d'implémenter certaines méthodes s'ils veulent pouvoir utiliser ses classes

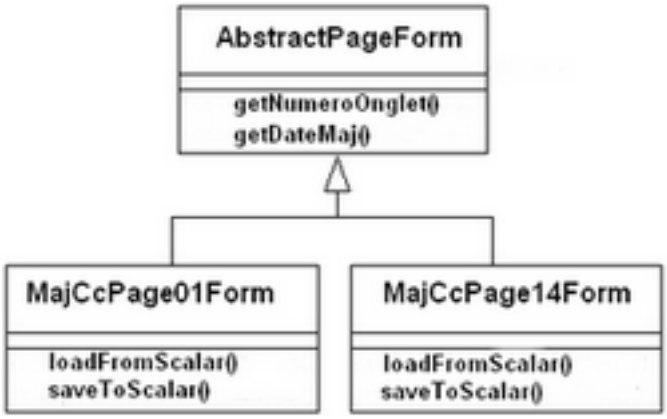


Figure V-8 : Diagramme de classes

Ces classes servent à factoriser du code, ainsi tous les Forms doivent hériter d'AbstractActionForm, laquelle déclare des propriétés communes à tous les formulaires (Figure V-8). Tous les scalar héritent d'AbstractScalarData, elle aussi déclare des propriétés communes à tous les scalars.

L'autre partie du « commun », composée des jsp génériques, constitue l'ensemble des jsp pour les templates de l'ergonomie de l'application (une jsp pour le menu contextuel, une pour le fil d'ariane, une pour le menu contextuel, une pour le pied de page..).

➔Ce commun définit clairement les **grandes lignes à respecter**.

Suite à la mavenisation du projet, nous avons créé deux archives « jar» à partir de ces deux répertoires, qu'on a ajouté aux dépendances du projet.

Pour des besoins de développement, on a dû effectuer des modifications sur des fichiers du projet 'commun'. Après chaque modification, on effectue une régénération des librairies « jar » pour les réinsérer dans les dépendances du projet. Comme le commun est partagé entre plusieurs projets, toute modification doit tenir en compte l'impact sur l'ensemble des projets.

Les modifications effectuées servent en quelque sorte à faire évoluer cette partie commune ; c'est pour cette raison que le CETE est mis au courant de toute modification et doit donner son feu vert pour l'effectuer ou non. Par moment, nos équipes ont dû travailler ensemble pour trouver une solution à un dysfonctionnement.

Ce qui est regrettable est que le CETE n'intègre pas les corrections faites sur le commun à leur framework, chez eux. Ce qui fait qu'on se trouve face aux mêmes erreurs pour les nouveaux projets, chose qu'on a remarqué pour des modifications faites lors d'un ancien projet « sudocu ». Cela ne permet pas de faire évoluer le framework !

3.3. Intégration de struts-spring

Struts est un framework Java utilisé, testé et approuvé depuis 2001, de plus son système de plugin lui permet d'utiliser la puissance du framework Spring pour intégrer les actions Struts au cycle de vie des objets Spring.

3.3.1. Struts

Struts est une implémentation classique du framework MVC (Model View Controler), dont les caractéristiques sont les suivantes *[Struts, 1]*:

- **Modèle**, Constitué de JavaBeans standards. Dans notre cas, ils proviennent des couches inférieures de l'application (couche de service ou couche de domaine).
- **Vue**, Classiquement constituée de pages JSP.
- **Contrôleur**, Constitué de la servlet Struts principale, appelée ActionServlet.

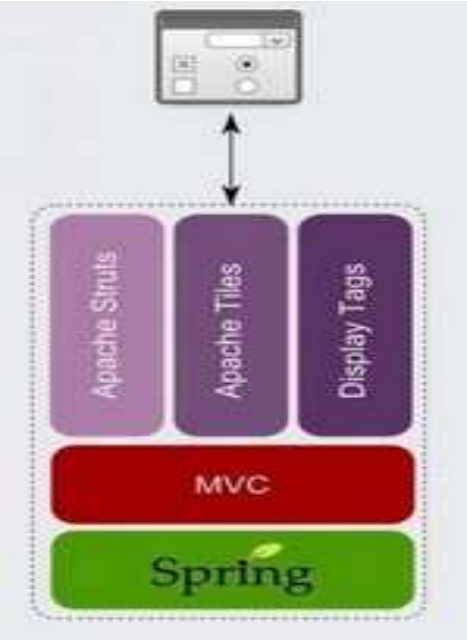


Figure V-9 : Présentation

Dans son implémentation, Struts utilise deux notions principales, qui sont représentées par les classes org.apache.struts.action.**Action**, appelées actions, et org.apache.struts.action.**ActionForm**, appelées FormBeans, ou formulaires :

Action Représente une action d'un utilisateur, telle que valider une sélection, mettre un produit dans un panier électronique, payer en ligne, etc. Cette classe proche du contrôleur sert de lient entre le formulaire envoyé par l'utilisateur, l'exécution de traitements dans les couches métier de l'application et l'affichage d'une page Web résultant de l'opération.

L'ensemble formé par les actions, formulaires et JSP est relié via le fichier de configuration de Struts, struts-config.xml, lequel est stocké dans le répertoire src/main/resources.

Struts se fonde sur une servlet très évoluée pour assurer sa fonction de contrôleur générique. Comme pour toute servlet, il faut la configurer dans le fichier web.xml :

```
<display-name>Itinavi</display-name>
 <context-param>
  <param-name>contextConfigLocation</param-name>
    <param-value>
                /WEB-INF/classes/applicationContext.xml
                /WEB-INF/classes/applicationContext-Hibernate.xml
                /WEB-INF/classes/applicationContext-DAO.xml
                /WEB-INF/classes/applicationContext-Services.xml
                /WEB-INF/classes/applicationContext-Security-ns.xml
    </param-value>
 </context-param>
  ( ... )
 <servlet>
  <servlet-name>action</servlet-name>
  <servlet-class>
                i2.application.commun.presentation.action.GenericActionServlet
  </servlet-class>
  <init-param>
    <param-name>config</param-name>
    <param-value>/WEB-INF/classes/struts-config.xml</param-value>
  </init-param>
 </servlet>
 ( ... )
 <servlet-mapping>
  <servlet-name>action</servlet-name>
  <url-pattern>*.do</url-pattern>
 </servlet-mapping>
```

L'usage veut que l'on fasse correspondre les actions Struts à l'URL *.do, comme dans l'exemple ci-dessus. Cette extension en .do provient du verbe anglais to do (faire). Les actions Struts ont donc toutes une URL se terminant en .do, par exemple l'action suivante nous envoie à la page d'accueil :

```
http:// d69-bdd12:8080/itinavi/accueil.do
```

Ensuite elles seront renvoyées à ActionServlet, qui les traite.

ActionServlet, quant à elle, est configurée via le fichier **struts-config.xml**. Voici une partie du fichier de l'application ITINAVI :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC
        "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
        "http://struts.apache.org/dtds/struts-config_1_2.dtd">

<struts-config>

  <form-beans type="org.apache.struts.action.ActionFormBean">
    <form-bean name="taillePagesForm"
            type="org.apache.struts.action.DynaActionForm"/>
  </form-beans>

  <global-exceptions>
    <exception key="message.erreur.title" path=".erreur500"
             type="javax.servlet.ServletException"
            handler="i2.application.commun.presentation.action.BasicExceptionHandler"/>
  </global-exceptions>

  <global-forwards type="org.apache.struts.action.ActionForward">
    <forward name="accueil" path="/accueil.do" redirect="true"/>
  </global-forwards>

  <action-mappings type="org.apache.struts.action.ActionMapping">
   <action path="/creerccpage10"
     type="i2.application.itinavi.presentation.cc.actions.MajCcAction"
                  name="majCcPage09Form"
                  input=".creerccpage10"
                  validate="false"
                  scope="request">
                  <forward name="success" path=".creerccpage10" redirect="false" />
    </action>
  </action-mappings>
  <message-resources parameter="itinavi"/>
</struts-config>
```

Ce fichier XML comporte les grandes parties suivantes :

<form-beans> Contient la liste des formulaires Struts. Nous donnons un nom à chaque formulaire, comme exampleForm, ainsi que le nom de la classe Java à laquelle il correspond.

<global-exceptions> En fonction de leur type, les exceptions lancées par les actions sont renvoyées par Struts à des classes spécifiques, qui doivent être spécifiquement implémentées. Cette partie est facultative.

<global-forwards> Donne des noms génériques à des pages JSP. Ces noms peuvent ensuite être utilisés depuis n'importe quelle action Struts. Dans notre exemple, nous pouvons renvoyer la page nommée erreur depuis toutes les actions, ce qui peut se révéler très utile. Cette partie est également facultative.

<action-mappings> Définit un fichier de propriétés, dans notre cas itinavi_fr.properties. Situé, comme tous les fichiers de configuration, dans le

package « src/main/resources » de notre projet Eclipse, ce fichier contient les messages utilisés dans les pages JSP et les actions.

➢ *Validation struts*

On utilise les deux principaux mécanismes de validations des données saisies dans les formulaires offertes par Struts :

- Le plug-in Validator.
- La méthode validate() des ActionForms.

Par validation, on entend deux choses :

- une validation qualitative : il s'agit de vérifier que les données saisies sont bien dans la forme attendue (ex : une donnée numérique ne contient que des chiffres) ou une donnée date est de la forme : dd/MM/yyyy

```
<field depends="integer,intRange" property="anneeConstruction_ts">
  <arg0 key="modifiercc.p2.item7.anneeconstruct"/>
  <arg1 key="${var:min}" name="intRange" resource="false"/>
  <arg2 key="${var:max}" name="intRange" resource="false"/>
  <var>
    <var-name>min</var-name>
    <var-value>1800</var-value>
  </var>
  <var>
    <var-name>max</var-name>
    <var-value>2100</var-value>
  </var>
</field>
```

  Dans ce cas, on vérifie que le champ « anneeConstruction_ts » est de type entier, qu'il est inférieur à 2100 et supérieur à 1800.

- une validation sémantique : il s'agit de vérifier que la valeur saisie est bien celle qui est attendue par le système (ex : un numéro de carte bleue valide).

Bien que les deux systèmes sont à même de faire les deux, on utilise le plug-in Validator pour les contrôles qualitatifs, puisqu'il ne nécessite pas d'aller-retour entre le client et le serveur, et la méthode validate() des ActionForms pour la validation sémantique.

Pour utiliser le plug-in Validator, on a ajouté dans le fichier de configuration de l'application ITINAVI « struts-config.xml » le code suivant :

```
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
   <set-property property="pathnames"
       value="/WEB-INF/classes/validator-rules.xml, /WEB-INF/classes/validation.xml"/>
</plug-in>
```

➢ *Les bibliothèques de tags*

Afin d'aider à l'affichage des pages JSP, Struts propose en standard les cinq bibliothèques de tags (tag libraries) suivantes :

- **Bean**. Sert à afficher et manipuler des JavaBeans dans une JSP. Développée avant l'existence des bibliothèques de tags standards, ou JSTL (Java Standard Tag Library), cette bibliothèque présente aujourd'hui peu d'intérêt.
- **HTML**. D'excellente qualité, cette bibliothèque est l'une des raisons du succès de Struts. Très proche dans sa syntaxe des éléments de formulaire HTML, elle est intuitive et permet de réaliser facilement des formulaires Struts. Bien entendu, les formulaires HTML peuvent toujours être réalisés en HTML classique, Struts interprétant uniquement la requête http (HyperText Transfer Protocol) envoyée. Cette bibliothèque n'est donc pas obligatoire mais fournit une aide intéressante.
- **Logic**. Sert à réaliser des boucles et des branchements conditionnels. Comme la bibliothèque Bean, elle n'est plus d'actualité depuis l'apparition de JSTL.
- **Nested**. Reprend l'ensemble des bibliothèques Struts, en leur donnant la capacité de s'imbriquer les unes dans les autres. Cela simplifie considérablement l'utilisation d'arbres d'objets complexes.
- **Tiles**. Permet l'inclusion et le paramétrage de fragments Tiles (voir ci-après).

Pour simplifier le travail dans la couche présentation de l'application, on utilise la librairie JSTL *[JSTL, 1]*. Elle permet de développer des pages JSP en utilisant des balises XML, et ainsi concevoir des pages dynamiques complexes.

L'utilisation des JSTL/Taglibs change radicalement la conception de pages JSP. En effet, les scriptlets Java ne sont pas utilisés et les pages JSP s'apparentes plus à des fichiers XML...

➢ *Utilisation des conversations*

Le scope conversation se situe entre le scope session et le scope request. La portée conversation permet de gérer des instances d'objets pendant une conversation et de les détruire une fois que celle-ci est terminée, ce qui est avantageux par rapport à la portée session, puisque le serveur dans cette

dernière doit garder ses objets pendant toute la durée de la session utilisateur.

Ce mécanisme a l'avantage de nous permettre d'ouvrir plusieurs fenêtres pour un seul utilisateur ; cependant, il est lourd à mettre en œuvre ! En effet, il faut charger tous les formulaires dans la conversation, pour qu'à la fin les récupérer un par un de celle-ci.

Les conversations ont pour objectif de conserver les informations d'un ou plusieurs formulaires tout au long du processus de modification (création ou mise à jour) des données d'un ou plusieurs objets.

Les conversations sont les seuls objets autorisés à être conservé en session (sauf cas exceptionnel). Ceci permet de limiter la taille de la session.

Les conversations sont uniquement utilisées lors des procédures de mise à jour ou de création, et en aucun cas pour les écrans de consultation.

- **Principe d'utilisation**

Lors d'un processus de mise à jour ou de création, la première opération consiste à créer une nouvelle conversation. Lorsqu'elle n'est plus utilisée, la conversation doit être détruite.

- **Propagation des conversations**

Afin de gérer l'aspect transactionnel d'une modification ou d'une création d'un objet dont les attributs sont saisis dans plusieurs écrans, il est nécessaire de propager dans l'URL l'identifiant de la conversation. Sur les actions relatives aux écrans suivant, la conversation est récupérée à partir de cet identifiant et permet ainsi de propager des objets.

➔ Avec le Framework Struts, l'application web est beaucoup plus structurée et mieux organisée afin de permettre une meilleure maintenance au dépit d'un développement plus complexe.

3.3.2. Spring

Le Framework Spring est une boite à outils très riche permettant de structurer, d'améliorer et de simplifier l'écriture d'application JEE. C'est un conteneur dit « léger », c'est-à-dire une infrastructure similaire à un serveur d'application JEE *[Spring, 1]*. Il prend donc en charge la création d'objets et la mise en relation d'objets par l'intermédiaire d'un fichier de configuration qui décrit les objets à fabriquer et les relations de dépendances entre ces objets.

Le gros avantage par rapport aux serveurs d'application est qu'avec SPRING, les classes n'ont pas besoin d'implémenter une quelconque interface pour être prises en charge par le Framework. C'est en ce sens que SPRING est qualifié de conteneur «léger».

Outre cette espèce de super fabrique d'objets, SPRING propose tout un ensemble d'abstractions permettant de gérer entre autres :

- Le mode transactionnel
- La persistance d'objets
- La création d'une interface Web

Pour réaliser tout ceci, SPRING s'appuie sur les principes du design pattern **IoC** (Inversion Of Control) et sur la plateforme **AOP** (Aspect Oriented Programming) *[Spring, 2]*.

- Le design pattern IOC (Inversion Of Control) ou injection de dépendance : il permet d'externaliser la logique applicative d'un projet afin qu'elle puisse être écrite et injectée dans le code client via un fichier XML de configuration. Changer l'implémentation logique est à présent accessible sans trop d'effort, il suffit de modifier un fichier XML et non le code Java directement.

- La programmation orientée aspect ou AOP (Aspect Oriented Programmation) qui a peut-être le plus d'intérêt car il permet de placer des intercepteurs à différents points d'entrée de la logique applicative afin d'y injecter son propre code. Dans ITINAVI, elle est utilisée pour gérer les transactions automatiquement.

Dans le paragraphe 5.3.5 nous gérons les transactions grâce à l'AOP spring.

➢ *Gestion de la sécurité :*

Sécuriser l'application est un besoin fondamental. Restreindre les accès à ceux ayant droit, et permettre des opérations uniquement selon certains critères relatifs à l'utilisateur sont des besoins classiques pour une application Web *[Security]*.

- **Sécurisation des URLs**

Il s'agit de conditionner l'accès à certaines pages à certains rôles. Il s'agit de configurer dans le fichier « applicationContext-Security-ns.xml », les rôles ayant le droit d'accéder à l'URL de la page. Les rôles étant définis dans l'annuaire LDAP (Lightweight Directory Access Protocol). LDAP est un protocole permettant l'interrogation et la modification des services d'annuaire.

L'accès à la page correspondant à l'URL sera intercepté pour vérifier les droits avant d'autoriser ou pas l'accès. Dans l'exemple ci-dessous, il s'agit de donner le droit de consultation d'un CC aux administrateurs et aux consultants :

```
<intercept-url pattern="/visualisercc.do"
            access="ROLE_ADMINISTRATEUR, ROLE_CONSULTANT"
            requires-channel="http"/>
```

- **Sécurisation de la couche Services**

Pour les services qu'on veut restreindre à certains rôles, il faut appliquer un mécanisme de sécurisation particulier. On se base sur les annotations « Secured » à appliquer sur les méthodes à restreindre.

L'exemple ci-desosus présente la manière de restreindre l'appel au service de sauvegarde pour l'ouvrage aux administrateurs, gestionnaire et saisie technique :

```
@Secured({Constantes.ROLE_ADMINISTRATEUR})
public Integer save(ScalarMotorisation data) {
   try {
      //
   } catch (DAOException e) {
      throw new ServiceException("Erreur d'acces a la couche d'integration", e);
   }
}
```

3.3.3. Apport de l'intégration de struts-spring

Struts a été le premier Framework MVC à rencontrer un fort succès auprès de la communauté Java. L'intégration de Spring et de Struts permet de bénéficier des avantages de chacune de ces deux technologies, notamment la vaste communauté d'utilisateurs de Struts et l'injection de dépendances et la AOP proposées par Spring. La combinaison de ces deux technologies permet de réaliser une couche MVC de bonne qualité *[StrutsSpring]*.

La capacité qu'à Spring d'intégrer d'autres Frameworks lui a conféré une grande popularité. En effet, les applications se basant sur Spring bénéficient d'une grande flexibilité permettant le choix des Frameworks spécialisés dans chaque couche de l'application ; par exemple Hibernate pour la gestion de la persistance (Chapitre 5, paragraphe 3.4.4), Struts pour l'implémentation du modèle MVC…

L'intégration de Struts à Spring permet d'ajouter à Struts un accès performant à la couche métier de l'application gérée par Spring. Ce faisant, les actions Struts ne sont plus liées à des implémentations d'objets métier mais à leurs interfaces. Toutes les fonctionnalités de Spring sont disponibles lors de l'accès à cette couche métier, notamment l'utilisation de la AOP pour gérer les transactions.

Spring vient ainsi combler une importante lacune de Struts, contribuant à moderniser ce Framework.

3.4. Le modèle MVC

Le modèle général d'architecture ACAI est le client-serveur de deuxième génération qui propose une répartition des services applicatifs en 3 niveaux (ou "tiers") distincts.

- **1er tiers :** il est pris en charge par le poste client qui gère l'affichage et exécute les traitements locaux (contrôles de saisie, mise en forme de données...).

- **2ème tiers :** il est pris en charge par le serveur applicatif qui réalise les traitements globaux tels que l'exécution des règles métier, la génération du contenu pour le client, les services d'interface avec le tiers "données".

- **3ème tiers :** il est pris en charge par le serveur de persistance qui exécute les traitements réalisant l'ensemble des fonctions de persistance des données (création, modification, suppression).

Ces tiers sont eux-mêmes subdivisés selon un découpage multi-couches précisant de manière plus fine le fonctionnement et les interactions des différentes logiques applicatives initiées dans une application web.

Une couche est une partition logique du système issue du découpage de ce dernier par problématique. Chaque couche a une responsabilité dans le système, est logiquement séparée des autres et est faiblement couplée avec les couches adjacentes. Le système global est une superposition de toutes les couches.

Cette architecture a pour objectif d'organiser une application interactive en séparant : les données, la représentation des données et le comportement de l'application.

Les deux frameworks Spring et Struts sont utilisés conjointement dans l'application, ainsi qu'hibernate.

3.4.1. Couche client

Cette couche représente l'ensemble des systèmes clients de l'application, navigateurs web ou consommateur de service web

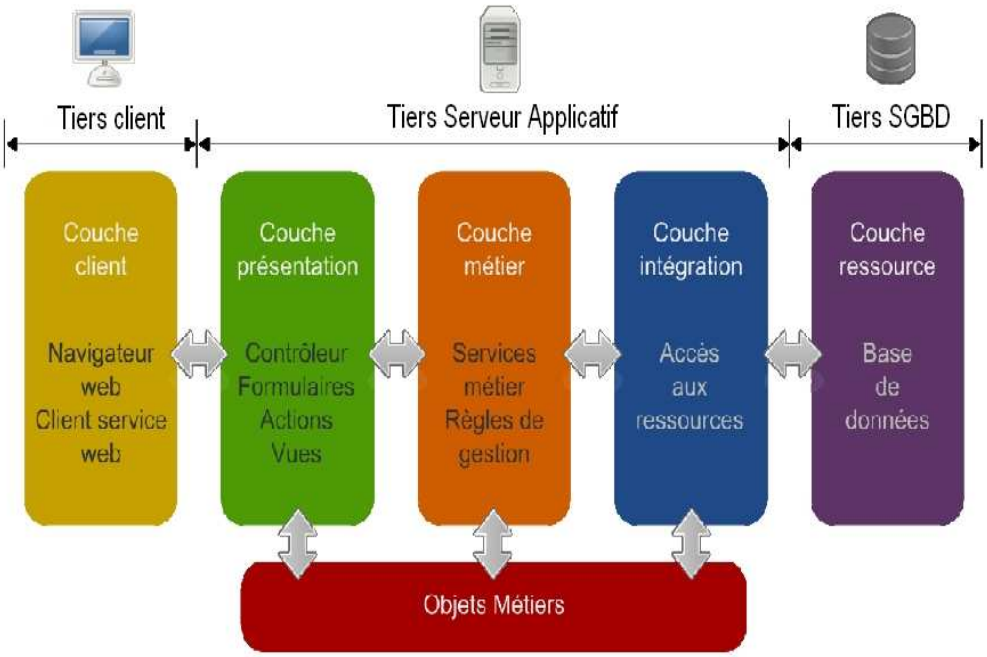


Figure V-10 : Modèle en couches

3.4.2. Couche présentation

Cette couche encapsule la logique de présentation requise pour satisfaire les clients qui accèdent au système. Elle intercepte la requête du client, fournit un service d'authentification unique, gère les sessions, accède aux services métier, construit la réponse et la transmet au client.

Cette couche est composée des pages jsp, des actions struts, des actions Form et des scalars.

- ***Actions et formulaires***

Les actions et les formulaires sont les classes de base utilisées dans Struts. Les actions Struts héritent toutes d'org.apache.struts.action.Action. Cette classe représente une action standard.

L'objectif de l'action est d'effectuer un traitement, généralement réalisé par la couche métier, avec un Bean Spring, puis de renvoyer l'utilisateur vers une nouvelle page Web. Cette page est représentée par l'objet ActionForward, que retourne l'action.

Les formulaires Struts héritent tous de la classe org.apache.struts.action.ActionForm et sont des JavaBeans standards.

Il s'agit de formulaires HTML créés dans une page Web. Lors de la validation d'un formulaire, Struts transforme la requête HTTP envoyée en cet objet Java, que nous pouvons ensuite plus facilement manipuler. Chaque paramètre de la requête HTTP est renseigné dans l'attribut du formulaire qui porte son nom.

Par exemple, pour la requête HTTP suivante :

```
http://localhost/itinavi/rechercherBateauAction.do?id=98526
```

Le formulaire RechercherBateauForm a un attribut id renseigné (méthodes setId), et nous pouvons accéder à cette valeur via la méthode getId.

Il faut cependant être conscient que ce formulaire est l'exacte représentation de ce qu'a envoyé l'utilisateur via son navigateur et que les données qu'il contient ne sont donc pas forcément valides. C'est pourquoi les formulaires Struts possèdent une méthode validate, qui est automatiquement appelée si, dans le fichier struts-config.xml, l'attribut **validate="true"** est mis à une action donnée.

Les 17 forms des onglets du CC héritent tous de la classe AbstractPageForm [Annexe 2]. Cette classe que nous avons créée, permet de factoriser tout ce qui est en commun pour ces Forms comme le pied de page. Cela permet au développeur de se concentrer sur ce qui est propre à l'onglet qu'il est entrain de réaliser, et ainsi éviter la duplication du code d'où un gain important du temps. En cas de modification de tout ce qui est en commun aux onglets, il suffit de faire la modification dans cette classe abstraite et ainsi diminuer le risque d'erreur.

➢ *Les scalars*

L'utilisation des scalars est à respecter selon la norme ACAI. On procède au chargement de chaque form dans un scalar pour obtenir une copie du Form, ainsi, ils auront les mêmes propriétés avec leurs guetteurs et setters.

Chaque form implémente, en plus de la méthode validate() qu'on trouve habituellement dans les form struts, deux méthodes :

- loadFromScalar() : charge les données dans le Form depuis le scalar, voici la méthode de l'onglet 6 du CC. Ceci permet de renseigner les propriétés du Form, qui seront par la suite prêtes à être affichées dans la jsp.
-

```
public void loadFromScalar(AbstractScalarMajPage scalarPage) throws
TechniqueException {

    super.loadFromScalar(scalarPage);
    ScalarMajCcPage06 scalarPageCourant = (ScalarMajCcPage06) scalarPage;
```

```
        this.ancreAvantId = scalarPageCourant.getAncreAvantId();
        this.ancrePoupeId = (scalarPageCourant.getAncrePoupeId());

}
```

- saveToScalar() : Charge les données du Form, récupérées de la page jsp après saisie par l'utilisateur, dans le scalar.

```
public void saveToScalar(AbstractScalarMajPage scalarPage) throws TechniqueException {

        super.saveToScalar(scalarPage);

        scalarPageCourant.setAncreAvantId(this.ancreAvantId);
        scalarPageCourant.setAncrePoupeId(this.ancrePoupeId);
}
```

Le form et le scalar font tous les deux partie de la couche présentation.

Le scalar, à son tour, doit implémenter deux méthodes :

- loadFromValueObject : cette méthode alimente les champs du scalar et par la suite celle du Form, car comme nous l'avons vu, le scalar est un clone du Form.
- saveToValueObject : les données du scalar sont chargées dans les VO (valueObjects).

Comme les Forms et scalars sont liés (un scalar pour chaque form), on a créé une classe abstraite AbstractScalarMajPage, laquelle regroupe ce qui est commun aux scalars des onglets du CC.

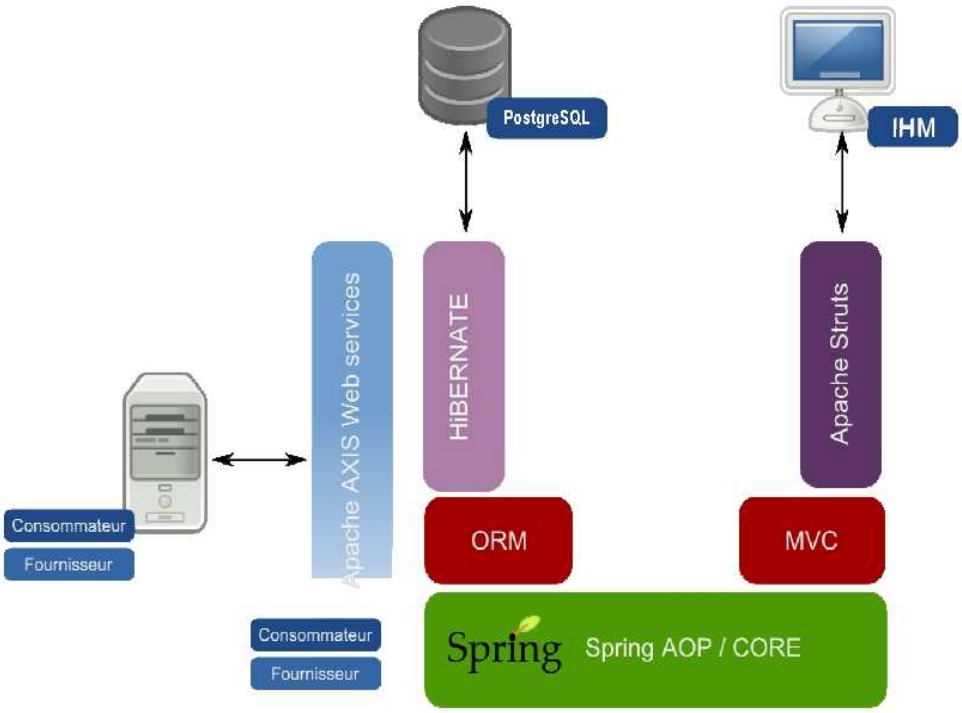


Figure V-11 : Architecture du framwork avec struts, spring et hibernat

3.4.3. Couche métier

Cette couche fournit les services métier nécessaires au traitement des demandes client. Elle contient les données métier et la logique associée. Tous les traitements métier de l'application sont centralisés dans cette couche.

La couche service est la couche où se fait tout le métier, des règles sont à respecter par les développeurs tout le long du développement de l'application :

- Dans cette couche, on trouve des classes dont le nom est de la forme xxxService (exemple : ccService), cette règle de nommage est absolument à respecter pour gérer correctement les transactions (Chapitre 5, paragraphe 3.5). Dans un service, on ne trouve jamais un Form, ceci pour séparer cette couche de la couche présentation, mais on trouve plus tôt des Scalar (clones des Form).

La séparation des deux couches permet, en cas du changement du framework par exemple, de ne pas tout refaire. Il suffit alors de ne modifier que la couche présentation, alors que la couche service est à reprendre tel qu'elle est. Ceci permet un gain de temps énorme et une réutilisabilité du code.

- Ne jamais passer un Bean à la couche service, mais son id (identifiant), cela permet de lever proprement les exceptions étant donnée que tout le traitement métier doit se faire dans cette couche.

Remarque : si la première règle est compréhensible, cette dernière est discutable ; car par moment on est amené à refaire deux fois le même travail notamment lorsqu'un service fait appel à un autre service.

Des revues de code seront faites par le CETE. Le suivi de ces règles de codage est facile à contrôler, il suffit de parcourir les classes de service et vérifier qu'on ne fait pas appel aux Form, qu'on ne passe pas de Bean mais plutôt son id…

3.4.4. Couche intégration

Cette couche a en charge les relations avec les ressources et les systèmes externes, tels que les bases de données et les applications existantes. La couche métier est couplée à la couche intégration dès que les objets métier ont besoin d'accéder à des données ou des services qui se trouvent dans la couche ressources. Les composants de cette couche peuvent utiliser les API des bases de données, les connecteurs, ou tout autre logiciel de liaison (middleware) adapté à la couche ressources.

Figure V-12 : couche intégration

➢ *Hibernate*

Nous utilisons la version 3.2 d'Hibernate, qui est un projet ambitieux dont le but est d'apporter une solution complète aux problèmes de la gestion des données persistantes. Dans une application orientée objet, la persistance permet à un objet de survivre même si le processus l'ayant créé est détruit *[Hibernate, 1]*.

Hibernate est un ORM (Object Relation Mapping), un outil de mapping objet/relationnel pour le monde Java. Le terme ORM décrit la technique consistant à faire le lien entre la représentation objet des données et sa représentation relationnelle basée sur un schéma de base de données.

Non seulement Hibernate s'occupe du transfert des classes Java dans les tables de la base de donnée, mais il permet de faire des requêtes sur des données afin de les récupérer.

Il peut donc réduire de manière significative le temps de développement qui aurait été dépensé autrement dans une manipulation manuelle des données via SQL et JDBC.

Le but principal d'Hibernate est donc de libérer le développeur d'une très grande partie des tâches de programmation liée à la persistance des données communes, ceci étant réalisé en automatisant une tâche à priori complexe : persister les objets Java vers une base de données relationnelle. Effectivement, sans Hibernate, le développeur devrait écrire lui-même le code permettant de mapper le code objet vers une base de données. Ce code est souvent complexe, long et coûteux à écrire. Hibernate possède de plus une architecture flexible et grandement configurable : il a été développé dans le but d'être modulaire, extensible et de permettre aux utilisateurs de pouvoir l'adapter et le configurer selon leurs besoins.

Hibernate est positionné comme une couche entre une application et un système de base de données. La Figure V-13 donne un aperçu de son architecture. Ce type de technologie est appelé Framework de persistance objet des données.

Le lien entre les classes exposées et la source physique des données (pour ITINAVI, c'est une base de données relationnelle Postgre SQL) est définie par un fichier xml, d'où mapping objet-relationnel.

Dans l'application ITINAVI, toute classe **DAO** (Data Access Object) hérite de la classe abstraite AbstractDAO laquelle implémente les méthodes nécessaires à la création, la lecture, la modification, et la suppression d'entités de bases

de données, on dit qu'elle implémente une interface de type CRUD (Create, Read, Update, Delete). Cette classe fait partie du commun (Chapitre 5, paragraphe 3.2). Il nous reste à implémenter les autres méthodes de récupération de données, ceci se fait de plusieurs façons *[Hibernate, 2]*.

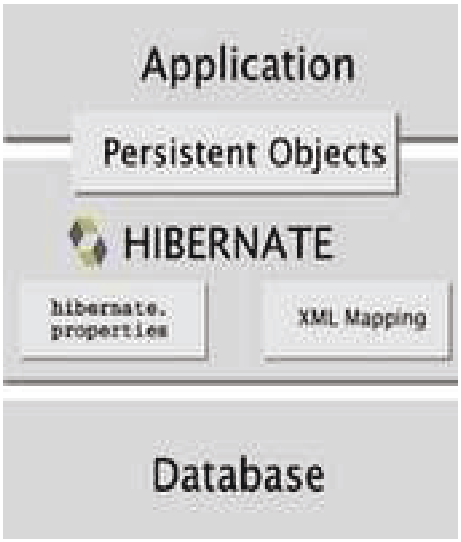


Figure V-13 : Architecture d'hibernate

➢ *Mapping hibernate*

Le mapping du squelette généré contient toutes les relations « simple » (many-to-one) entre les différentes tables de l'application. Pour autant il est nécessaire d'effectuer des adaptations dans le but de :

- Faciliter l'accès aux données,
- Garantir leurs l'intégrité,
- Optimiser l'accès
- Réduire le nombre d'accès a la base.

La stratégie de chargement par défaut utilisée lors de la génération est dite immédiate (« lazy=false »), c'est-à-dire que toute les relations sont chargées avec l'objet.

- **Déclaration et chargement des Collections**

Pour déclarer une collection dans un mapping hibernate il convient de différencier plusieurs éléments : Le type de relation, le type de collection, la relation «père/fils» et Cascade.

Indépendamment du type de relation on se doit de définir le type de la collection.

Hibernate supporte de nombreux type de collection (List, Set, Map, Bag, IdBag…), ayant chacun un impacte sur les performances et la manipulation des objets.

Nous avons choisi, pour l'applicaiton ITINAVI, de mettre en place uniquement des collections de type Set pour plusieurs raisons :

- Garantie l'unicité des objets dans la collection
- On accède directement à l'objet de la collection

Néanmoins, Quelques précautions sont à prendre comme l'initialisation des Set à vide dans les ValueObject et la nécessité d'implémenter correctement la méthode «generateKey», l'équivalent de la clé fonctionnelle de la table, pour ne pas « perdre » des objets.

- **Relation «Père / Fils»**

Hibernate permet de définir l'entité qui porte la relation « père/fils » : Le père déclare ses fils et les fils déclarent leur père.

La propriété «inverse» permet d'ignorer la contrainte de clé étrangère lors d'une association bidirectionnelle. En pratique et pour simplifier cela permet de réduire le nombre de requêtes lors des inserts / update / suppression, en laissant la responsabilité à "l'enfant" de renseigner sa relation.

En accord avec la structures des tables, il faut positionner la propriété «inverse=true» sur les table parent. Ce sera donc au fils de définir sont parent.

- **Cascade**

La propriété «cascade» permet de répliquer des opérations sur les objets associés (save, update, insert…).

Pour une utilisation plus simple, nous avons décidé de répliquer les opérations que du père vers le fils.

```
<set name="listeMoteurs" cascade="all,delete-orphan" inverse="true" lazy="true" >
  <key column="noseriedonneebateau" not-null="true" />
  <one-to-many class="i2.application.itinavi.valueobjects.donnee.Moteur" />
</set>
```

Dans cet exemple du fichier de mapping de « motorisation » d'ITINAVI, on a déclaré un SET de moteur, pour une motorisation. La clause cascade="all-delete-orphan" permet d'effacer les moteurs correspondant à une motorisation lors de la suppression du moteur.

➢ *Requêtage*

Hibernate propose plusieurs solutions pour réaliser des requêtes en base de données. Lorsque que le développeur doit écrire une requête avec Hibernate, il a trois possibilités :

- **SQL natif**

L'un des objectifs d'Hibernate est de s'éloigner de la base de données. Or, l'utilisation du SQL natif va à l'encontre de cet objectif. De ce point de vue, le

SQL natif est à éviter. Néanmoins, l'équipe du projet le conseille seulement de façon exceptionnelle pour des points qui ont besoin d'utiliser une optimisation liée spécifique à la base de données.

➔Des problèmes de performances ont été remarqués dans ITINAVI lors de la recherche d'un bateau. En effet, l'affichage de la page du résultat de la recherche mettait plus d'une minute ce qui n'est pas agréable pour un utilisateur et surtout non conforme aux normes ACAI qui ne tolère pas plus de 5 seconde pour afficher une page.

Pour pallier au problème, on a dû revoir la recherche faite initialement en Criteria, pour utiliser du SQL natif ***[Annexe 3]***. Le résultat est remarquable, juste quelques seconde pour avoir le même résultat.

- **HQL : Langage de requêtage d'Hibernate**

Hibernate fourni un langage d'interrogation extrêmement puissant qui ressemble au SQL. Il se distingue par sa syntaxe qui se veut totalement orienté object, comprenant des notions d'héritage, de polymorphisme et d'association.

Une panoplie d'expressions est autorisée dans la clause WHERE, (and, or, not …).

Voici un exemple de requête HQL (Hibernate Query Language) :

```
public List<Acteur> rechercherListeActeursPrincipauxPourDossier(List<Integer>
listeIdDossiers) {
  // Requête hql à passer
  StringBuffer hqlQueryActeurs = new StringBuffer("select acteur from Acteur as acteur");
  hqlQueryActeurs.append(" inner join acteur.donneeBateau as donneeBateau");
  hqlQueryActeurs.append(" inner join donneeBateau.dossier as dossier");
  // Acteur principal du dossier
  hqlQueryActeurs.append(" where acteur.proprietairePrincipal = true");
  // Critère : Liste des dossiers à considérer
  hqlQueryActeurs.append(" and dossier.id in (:listeIdDossiers)");
  // Définition de la requete
  Query queryActeurs = getSession().createQuery(hqlQueryActeurs.toString());
  // Valeur : liste des identifiants de dossiers
  queryActeurs.setParameterList("listeIdDossiers", listeIdDossiers);
  // Exécution de la requête
  List<Acteur> listeActeurs = queryActeurs.list();
  return listeActeurs;
}
```

Cette requête nous retourne la liste des acteurs principaux pour un dossier.

- **Criteria API : Requêtes par critères**

Hibernate offre aussi une API (Application Programming Interface) d'interrogation par critères intuitive, extensible, puissante et élégante qui permet de réaliser autant que l'utilisation du langage HQL. Elle est surtout adaptée pour l'écriture dynamique de recherche multi-critères où les requêtes doivent être construites à la volée.

Nous utilisons souvent l'API Criteria et on la préfère au HQL car il y a généralement beaucoup moins de lignes de code à saisir, le code en ressort plus propre et facile à maintenir. Bien sur, à part dans le cas d'optimiser les performances nous n'utilisons pas du SQL natif.

➢ *Une cache du second niveau pour ITINAVI*

Pour améliorer les performances, on a mis en place un cache de second niveau. Cette solution permet d'optimiser les accès aux données, mais ce n'est pas le remède miracle pour pallier à des problèmes de performances. La mise en place d'un tel cache doit être étudiée avec soin, car employé à mauvais escient, cela peut dégrader les performances ou engendrer des problèmes d'accès concurrent.

Hibernate fonctionne avec deux niveaux de cache. Le cache de premier niveau est lié à la session Hibernate, et les objets qui sont cachés ne sont visibles que pour une seule transaction. Quand la session est fermée, le cache n'a plus d'existence. Ce cache ne peut pas être désactivé. Le cache de second niveau est, quant à lui, lié à la session factory d'Hibernate. La portée de ce cache est la JVM (Java Virtual Machine), où l'application est déployée. Les objets cachés sont donc visibles depuis l'ensemble des transactions. Par défaut, ce cache n'est pas activé. Nous avons fait une configuration pour l'activer.

Le cache de second niveau permet de cacher des entités, ainsi que des associations de type collection. Il convient donc de sélectionner les classes et les collections à cacher, puis de configurer les fichiers de mapping correspondant.

Dans le fichier de configuration « applicationContext-Hibernate.xml » nous avons activé le fournisseur de cache :

```
<prop key="hibernate.dialect"> org.hibernatespatial.postgis.PostgisDialect </prop>
<prop key="hibernate.cache.use_second_level_cache">true</prop>
<prop key="hibernate.cache.use_query_cache">true</prop>
<prop
    key="hibernate.cache.provider_configuration_file_resource_path">/ehcache.xml</prop>
<prop key="hibernate.cache.provider_class">
                        net.sf.ehcache.hibernate.SingletonEhCacheProvider </prop>
<prop key="hibernate.cache.use_structured_entries">true</prop>
```

Le second niveau de cache est partagé et permanent contrairement au premier. Le premier niveau étant isolé et relatif à un contexte transactionnel spécifique et son cycle de vie dépend du cycle de la session Hibernate.

En plus du cache des objets persistants, le second niveau opère sur les requêtes fréquemment exécutées ainsi que leurs résultats.

Hibernate propose des stratégies pour la mise en place du cache de second niveau :

- **Read-only :** adaptée aux données qui sont lues fréquemment et qui ne sont jamais mises à jour. Cette stratégie est la plus performante.
- **Read-write :** adaptée aux données mises à jour.
- **Nonstrict read-write :** fonctionne de la même façon que la stratégie précédente mais reste adaptée aux données qui sont mis à jour très occasionnellement voire rarement.

➢ *Avantages d'Hibernate*

Une application Hibernate est très facilement et rapidement configurable pour fonctionner avec un tout autre type de base de données que celui utilisé actuellement par cette dernière. Il suffit de modifier quelques lignes dans le fichier de configuration. Hibernate permet donc d'éviter une dépendance d'un logiciel avec un certain type de base de données.

Les gains de performances sont grandement améliorés grâce à l'emploi d'une mémoire cache.

Hibernate dispose également d'une fonction 'lazy initialisation'. Elle permet de charger les états de divers objets depuis la base de données seulement lorsque l'application les utilise. Des gains de performance sont également réalisés car l'application initialise simplement les objets qu'elle utilise et non tous les objets.

➢ *Inconvénients d'Hibernate*

Absence de maîtrise du chargement : chaque appel peut potentiellement provoquer une requête en base de données, et ce, de manière complètement transparente et incontrôlable. On peut ainsi assister à des effondrements spectaculaires des performances de l'application, dus à un excès de requêtes SQL.

Seule solution, non pas pour maîtriser mais pour diagnostiquer ce genre de problématique : passer le paramètre de configuration " show_sql " à 'true' afin de visualiser en développement les requêtes générées automatiquement par les proxies Hibernate.

3.5. Le mode transactionnel

Une série d'opération est valide uniquement si l'exécution s'est effectuée convenablement. Ce mode est une transaction ACID (atomicity, consistency, isolation, durability).

Elle se définit par :

- **Atomique :** Toutes les opérations sont exécutées ou aucune ne l'est.
- **Cohérente :** Une base de données passe d'un état cohérent à un autre.
- **Isolée :** Aucune modification n'est visible sur la base tant que toutes les opérations n'ont pas été réalisées et validées.
- **Durable :** Les transactions validées sont conservées même si le système tombe en panne.

Les trois couches de l'architecture de l'application ITINAVI sont strictement disjointes. La gestion des transactions est une des clés de ce type d'architecture : elle est implémentée au niveau services.

Les composants de DAO sont intégrés à Spring, ce qui peut faciliter la gestion de la session et des transactions.

Le démarrage et la manipulation des transactions se fait par l'intermédiaire de l'objet de session Hibernate. Plusieurs requêtes peuvent donc participer à une même transaction si elles utilisent la même session.

3.5.1. Gestion des transactions spring-hibernate

Pour que Spring puisse gérer des transactions Hibernate, nous avons ajouté un bean spécialisé dans le fichier de configuration « applicationContext-Services.xml » :

```
<bean id="txManager"
            class="org.springframework.orm.hibernate3.HibernateTransactionManager">
      <property name="sessionFactory" ref="sessionFactory" />
</bean>
```

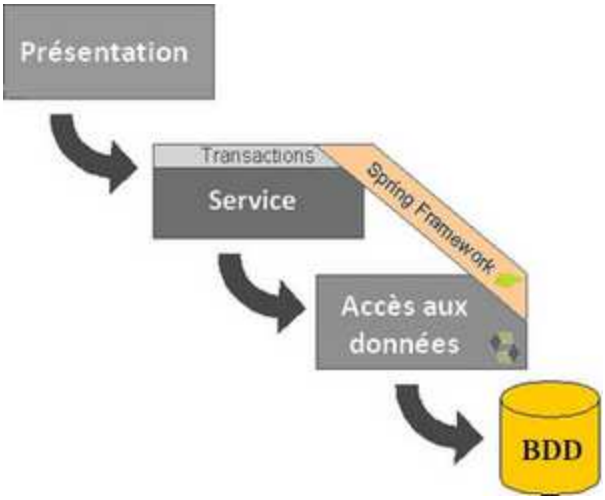


Figure V-14 : Gestion des transactions spring-hibernate

Nous utilisons les transactions par AOP, elle ne nécessite aucune modification dans le code source. Son principal défaut est qu'elle se base sur des conventions de développement relativement fortes, sur le nommage des classes, packages et méthodes.

La partie <**aop**:config> paramètre les classes qui seront prises en compte et la partie <tx:advice> précise la statégie de transaction en fonction des méthodes.

```
<tx:advice id="serviceTxAdvice" transaction-manager="txManager">
 <tx:attributes>
   <tx:method name="compter*" propagation="REQUIRED" read-only="true"
      rollback-for="i2.application.commun.exception.TechniqueException,
      i2.application.commun.exception.RegleGestionException,
      i2.application.commun.exception.AccessForbiddenException,
      i2.application.commun.exception.ServiceException,
      i2.application.commun.exception.DAOException" />
   <tx:method name="rechercher*" propagation="REQUIRED" read-only="true" .. />
     load, valider, save, persist, enregistrer, change, supprimer, creer, editer ..
 </tx:attributes>
</tx:advice>

<aop:config>
  <aop:pointcut id="serviceMethods"
   expression="execution(* i2.application.itinavi.metier.**.*Service.*(..))" />
</aop:config>
```

La partie <**tx**:method name= … /> restreint le préfixe du nom donné aux méthodes pour qu'elles soientt prises en compte dans la transaction. On peut avoir par exemple : *enregistrer*Immatriculation, *editer*FicheBateau,…

Le package **i2.application.itinavi.metier** est celui de tout traitement métier, le nom d'un service doit être de la forme « xxxService » exemple : Bateau**Service**.java pour celui du bateau. Ainsi à chaque action, s'il y a une exception dans l'un des services appelés, la transaction ne se termine pas et il y a un rollback.

➔Donc après configuration du fichier et pour que le mode transactionnel soit effectif, des règles doivent être respectés dans toute l'application ITINAVI :

- Placer la classe dans le package de services
- Préfixer le nom du service par « Service »
- Préfixer le nom de la méthode par l'un des éléments de la liste

3.6. Journalisation

Les traces sont toutes les informations fournies par une application qui permettent de suivre son comportement en exécution. Ces traces servant au débogage doivent être globalement débrayables pour la mise en production de l'application. Le composant Log4J pour java doit être utilisé pour les traces et journaux.

3.6.1. Log4J : journalisation (log) des applications Java

Les bonnes pratiques de développement actuelles recommandent l'utilisation d'un logger tel Log4J *[Log4j, 1]* apportant plus de souplesse.

Log4J simplifie les gestions des logs et le débogage des applications Java en fournissant des classes et des méthodes pour l'enregistrement de ces informations. Les fichiers journaux d'une application représentent la mémoire d'une application, un historique permanent de la vie de celle-ci, il est donc important de correctement enregistrer ces messages.

Les messages sont envoyés au logger pour affichage ou enregistrement en lui assignant un certain niveau de criticité (DEBUG, INFO, WARNING, ERROR, CRITICAL) et indiquant la classe/la méthode à l'origine de ce message, la ligne dans le code source, ou toute autre information utile. Grâce à cette couche, on peut facilement demander par exemple à une application d'afficher tous les messages de niveau DEBUG et supérieur à l'écran lors de la phase de développement puis lui demander de n'afficher que les messages de niveau WARNING et supérieur dans un fichier de log en phase de production.

Ces différents types d'affichage des messages sont configurés facilement au runtime de log4j par le fichier XML log4j.xml de façon totalement externe au code.

Log4J est constitué de 3 composants principaux qui permettent de configurer le dispositif de journalisation : les Loggers pour les écrire les messages, les

Appenders pour sélectionner la destination des messages et les Layouts pour la mise en forme des messages.

Pour le développement et pour voir les logs dans la console, on a effectué la configuration suivante au fichier Log4J :

```
log4j.logger.i2.application.itinavi=debug
log4j.rootLogger=error, stdout
log4j.logger.org.hibernate=debug
log4j.logger.org.hibernate.SQL=debug
log4j.logger.org.hibernate.hql=debug
```

Le fait de mettre « debug », le niveau le plus bas, nous permet de voir tous les traces des niveaux supérieurs (info, warning, error et critical).

Log4J permet donc non seulement de gagner en flexibilité sur la gestion des messages d'une application mais également de faciliter la recherche et la détection d'erreur.

3.7. Gestion des exceptions dans ITINAVI

Les exceptions constituent une solution élégante pour dissocier le code lié au fonctionnement normal du programme et celui lié à la gestion des erreurs. Elles représentent le mécanisme de gestion des erreurs intégré au langage Java. Il se compose d'objets représentant les erreurs et d'un ensemble de trois mots clés qui permettent de détecter et de traiter ces erreurs (try et catch) et de les lever ou les propager (throw et throws).

Lorsqu'une exception se produit, un objet qui hérite de la classe Exception est créé (on dit qu'une exception est levée) et propagé à travers la pile d'exécution jusqu'à ce qu'il soit traité.

Dans toute l'application ITINAVI, on doit systématiquement catcher les exceptions suivantes (et dans cet ordre) :

- **En lecture**

```
try {
	// Bloc d'instructions...
} catch (AccessForbiddenException afe) {
	// Instructions de traitement d'exception
} catch (ServiceException se) {
	// Instructions de traitement d'exception
}
```

- **En écriture**

```
try {
	// Bloc d'instructions...
} catch (AccessForbiddenException afe) {
	// Instructions de traitement d'exception
} catch (RegleGestionException rge) {
	// Instructions de traitement d'exception
} catch (ServiceException se) {
	// Instructions de traitement d'exception
}
```

L'instruction try contient un bloc d'instructions à l'intérieur duquel une exception pourrait se produire.

L'instruction catch doit être associée à try et contient un bloc d'instructions susceptible de gérer un type d'exception particulier.

Bien sûr, sans oublier que le bloc catch doit être fonctionnel, c'est à dire qu'il ne soit pas vide et qu'il faut traiter l'exception ! Il est possible de laisser ce bloc vide, le programme ne s'arrête pas, mais on ne saura pas qu'il y a eu erreur.

Le CETE a défini ses propres exceptions, par exemple, une exception **RegleGestionItinaviException** doit être levée dans le service lors du non respect d'une règle de gestion.

```
private void validerRgCcEditions(Cc certificat) throws RegleGestionItinaviException {
	/* Date de validité du certificat */
	if (certificat.getDateLimiteValidite() == null) {
	    throw new RegleGestionItinaviException ("rg.cc.dateLimiteValidite");
	}
}
```

Dans cet exemple, on vérifie que la date limite de validité est renseignée ; si elle est nulle, une exception sera levée. Son traitement sera fait dans l'action et un message sera affiché ; le texte du message d'erreur sera récupéré du fichier de ressources avec la clé « rg.cc.dateLimiteValidite ».

➢ *Les exceptions liées aux conversations*

Il faut impérativement "catcher" les exceptions liées aux conversations dans les actions lorsque l'on appel les fonctions du « ConversationManager » (surtout à l'appel de la méthode getConversation(session)) et rediriger vers la page approprié en cas d'erreur (cf : "global-forward" de l'application).

Il existe 4 types d'exceptions, qui héritent toutes de TechniqueException :

- ConversationNotFoundException
- ConversationFormOptimisticLockingFailureException
- ConversationFormNotFoundException
- IllegalConversationFormException

Les deux premières sont lancées respectivement : lorsque la conversation est expirée, lorsque l'on post deux fois un formulaire avant que sa version ne soit incrémentée (cad lors de l'affichage). Les deux dernières relèvent plus d'erreurs de programmation.

Les mécanismes d'exceptions permettent de renforcer la sécurité du code Java.

3.8. Tiles

Tiles, « Tuiles » en français, apporte beaucoup d'avantages dans un projet web. C'est une méthode qui fait appel au « Template » *[Tiles, 1]*.

Le Template (ou modèle), est le squelette d'une application web. Les différentes parties présentes dans le Template, seront générées au fur et à mesure de leur exécution. Donc le contenu d'un Template est constitué de « Tiles », ces « Tuiles » que l'on peut pluguer aux différents emplacements. Ainsi nous obtenons une uniformité de l'apparence pour toutes les pages appelées et nous profitons de la même occasion, des avantages du Template.

Les templates réutilisent beaucoup de codes qui normalement auraient été dupliqués sur toutes les pages. Ce qui permet de réduire considérablement la taille des pages.

L'autre intérêt que nous fourni les « Tiles », est la flexibilité de l'apparence web. C'est-à-dire que nous pouvons personnalisés facilement l'affichage si l'envie de changer le look du site nous prend. Entre autre, Le Framework Tiles fournit les fonctionnalités suivantes :

- Système de Template
- Construction et chargement de pages JSP dynamiques
- Supporte l'internationalisation (I18N)

Dans le fichier tiles-config.xml, on a une Template ou précisément la page «modelescc.jsp » qui permet d'assembler tous les Tiles dans une même page afin de l'afficher au client.

Pour pouvoir utiliser Tiles, il suffit d'ajouter les lignes suivantes à la fin du fichier struts-config.xml :

```
<plug-in className="org.apache.struts.tiles.TilesPlugin">
  <set-property property="definitions-config" value="/WEB-INF/classes/tiles-config.xml"/>
  <set-property property="moduleAware" value="true"/>
  <set-property property="definitions-parser-validate" value="true"/>
</plug-in>
```

Nous utilisons un fichier de définition. Une définition est la déclaration d'un composant tiles qui pourra être utilisé dans tous les contextes (pages, template, ...). C'est une déclaration d'une balise d'insertion de manière générale que l'on va pouvoir insérer dans n'importe quel contexte. Voici une partie du fichier de définitions « tiles-config.xml » utilisé dans itinavi :

```
<?xml version="1.0" encoding="UTF-8"?>
 <!DOCTYPE tiles-definitions PUBLIC
       "-//Apache Software Foundation//DTD Tiles Configuration 1.1//EN"
       "http://jakarta.apache.org/struts/dtds/tiles-config_1_1.dtd">
<tiles-definitions>

    <definition name="modele2colonnes" path="/WEB-
INF/jsp/itinavi/commun/modelecc.jsp">
      <put name="titrepage"/>
      <put name="accessibilite" value="/WEB-INF/jsp/commun/accessibilite.jsp"/>
      <put name="menuprincipal" value="/WEB-INF/jsp/commun/menuprincipal.jsp"/>
      <put name="messagedeservice" value="/WEB-
INF/jsp/commun/messagedeservice.jsp"/>
      <put name="progression" value="/WEB-INF/jsp/commun/progression.jsp"/>
      <put name="messages" value="/WEB-INF/jsp/commun/messages.jsp"/>
      <put name="telecommande" value="/WEB-INF/jsp/itinavi/commun/telecommande.jsp"/>
      <put name="menu" value="/WEB-INF/jsp/itinavi/commun/menu.jsp"/>
      <put name="utilisateur" value="/WEB-INF/jsp/itinavi/commun/utilisateur.jsp"/>
      <put name="recherche" value="/WEB-INF/jsp/itinavi/commun/recherche.jsp"/>
      <put name="pied" value="/WEB-INF/jsp/itinavi/commun/pied.jsp"/>
   </definition>

   <definition name=".creerccpage10" extends=" modele2colonnes ">
       <put name="contenu"
       value="/WEB-INF/jsp/itinavi/modification/titrenavigation/cc/majccpage10.jsp"/>
   </definition>
</tiles-definitions>
```

La page « modelecc.jsp » fait appel aux différents composants (accessibilite, progression, telecommande, menu... pied).

Le mot "extends" permet d'effectuer de l'héritage entre les définitions. Une définition fille va hériter de tous les paramètres de la définition mère et

pouvoir redéfinir certain de ces paramètres ou en ajouter des nouveaux. La notion d'héritage est très pratique dans le cas de spécialisation de certaines pages.

On remarque que, pour l'onglet de création de la page 10, **« .creerccpage10** » dérive de « modele2colonnes ». L'intérêt est, que nous avons juste à redéfinir les « tuiles » spécifiques à notre page tout en gardant les composants communs, c'est de la surcharge.

On peut donc constater la facilité et la vitesse de conception d'un site avec le principe des Tiles.

Pour ce module (Certificat communautaire), on a factorisé et mis dans une Template tout ce qui est, dans le corps de la page, commun à toutes les pages, a savoir, les entêtes et pieds de pages. Et ainsi tirer profit des avantages des Tiles.

Les différents composants du Template principal « modele2colonnes», représentent la structure générale des pages de l'application, qu'il faut respecter (Figure V-7).

4. ETUDE DE CAS : CREER UN NOUVEAU CC

Un bateau français ou étranger peut avoir un ou plusieurs CC. On accède à la création d'un CC depuis le menu contextuel.

L'instructeur doit remplir :

- l'ensemble des onglets 2 à 8 (pour les zones minimales obligatoires)
- l'onglet 11 s'il y a une attestation relative aux gaz liquéfiés
- l'onglet 13 s'il y a des informations à porter en annexe

Les onglets suivants ne doivent pas être accessibles à la saisie :

- Les onglets 9 ,10 concernant la procédure de prolongation, confirmation du CC l'onglet 12 concernant la procédure de prolongation de l'attestation relative aux gaz liquéfiés

4.1. Couche présentation

Le traitement de l'action créée suite au click sur le lien de création du nouveau CC dans le menu contextuel « majccinstruction.do » se fait dans le

contrôleur MajCcAction. Cette action est déclarée dans le fichier de configuration de struts « struts-config.xml » :

```
<action path="/majccinstruction"
      type="i2.application.itinavi.presentation.cc.actions.MajCcAction"
      name="majCcInstructionForm"
      input=".majccinstruction"
      validate="false"
      scope="request">
      <forward name="success"        path=".majccinstruction"        redirect="false" />
      <forward name="forbidden"      path="/erreur403.do"            redirect="true" />
</action>
```

C'est dans cette classe « MajCcAtion » que se fait le traitement pour préparer le premier affichage de la page de saisie du CC :

```
package i2.application.itinavi.presentation.cc.actions;
public class MajCcAction extends AbstractItinaviAction {
public ActionForward doExecute(ActionMapping mapping, AbstractActionForm form,
HttpServletRequest request, HttpSession session, HttpServletResponse response, String
sousAction) throws TechniqueException {

/* Déclaration */
IDossierService dossierService = ServiceFactory.getInstance().getDossierService();
AbstractPageForm formCourante = (AbstractPageForm) form;
MenuComposite menuContextuel = null;

try {

      // Création de la conversation
      Conversation conversation = ConversationManager.createNewConversation(session);
      // Création d'un nouveau certificat communautaire
      IObjetTitreNavService objetTitreNavService =
ServiceFactory.getInstance().getObjetTitreNavService();
      // Création d'un CC pour un bateau existant déjà en base
      Cc certificat                certificat = (Cc)
dossierService.loadDossierAvecBateau(idBateau,
EnumTypeDossier.TYPE_CERTIFICAT_COMMUNAUTAIRE);
      certificat.setObjetNav(objetTitreNavService.rechercherObjetTitreNavParCode(sObjetTitr
eNav));

      AbstractScalarMajPage scalarMajCcPage = null;

      // Parcourt les onglets
      for (EnumNoOngletCC enumNumeroOnglet : EnumNoOngletCC.values()) {
                  int numeroOnglet = enumNumeroOnglet.getId().intValue();
                  // exp. : formCourante = new MajCcPage01Form();
                  formCourante = this.initialiserFormPourPage(enumNumeroOnglet);
                  // initialiser le formulaire
                  formCourante.init(request);
                  // exp. : scalarMajCcPage = new ScalarMajCcPage01();
                  scalarMajCcPage = this.initialiserScalarPourPage(enumNumeroOnglet);
```

```
                    // charget le scalar depuis le valueObject
                    scalarMajCcPage.loadFromValueObject(certificat, basPage);
                    // charger le formulaire depuis le scalar
                    formCourante.loadFromScalar(scalarMajCcPage);
                    // pour notre cas c'est Nouveau
                    formCourante.setCodeObjetTitreNav_ts(sObjetTitreNav);
                    //Ajout du formulaire à la conversation
                    conversation.addForm(formCourante);
        }

        // Préparation du menu contextuel
        menuContextuel = dossierService.creationMenuContextuelMaj(request, idBateau,
EnumTypeDossier.TYPE_CERTIFICAT_COMMUNAUTAIRE.getId());
        request.setAttribute(ConstantesCommunes.MENU_CONTEXTUEL, menuContextuel);
        // Aller à la JSP
        return mapping.findForward(Constantes.FORWARD_SUCCESS);

        }  catch (AccessForbiddenException afe) {
                    return mapping.findForward("forbidden");
        }catch (ServiceException se) {
                    // traitement
        }
}
}
```

On crée la conversation puis dans la boucle, on fait dans l'ordre :

- Initialisation des scalars pour tous les onglets
- Initialisation des formulaires pour chaque onglet
- On ajoute les formulaires à la conversation comme le préconise le framework du CETE.

Ensuite, et si tout se passe bien et qu'il n'y a pas d'exceptions, struts nous redirige à la page « majccinstruction.jsp » qu'on récupère du fichier tiles par le nom « .majccinstruction ».

```
<definition name=".majccinstruction" extends="modele2colonnes ">
      <put name="titrepage" value="vide.title"/>
      <put name="contenu" value="/WEB-
INF/jsp/itinavi/modification/titrenavigation/cc/majccinstruction.jsp"/>
</definition>
```

La page hérite du template « medele2colonnes » ce qui permet de ne pas tout réécrire (Chapitre 5, paragraphe 3.8), on se concentre ainsi que sur la page actuelle de la figure suivante :

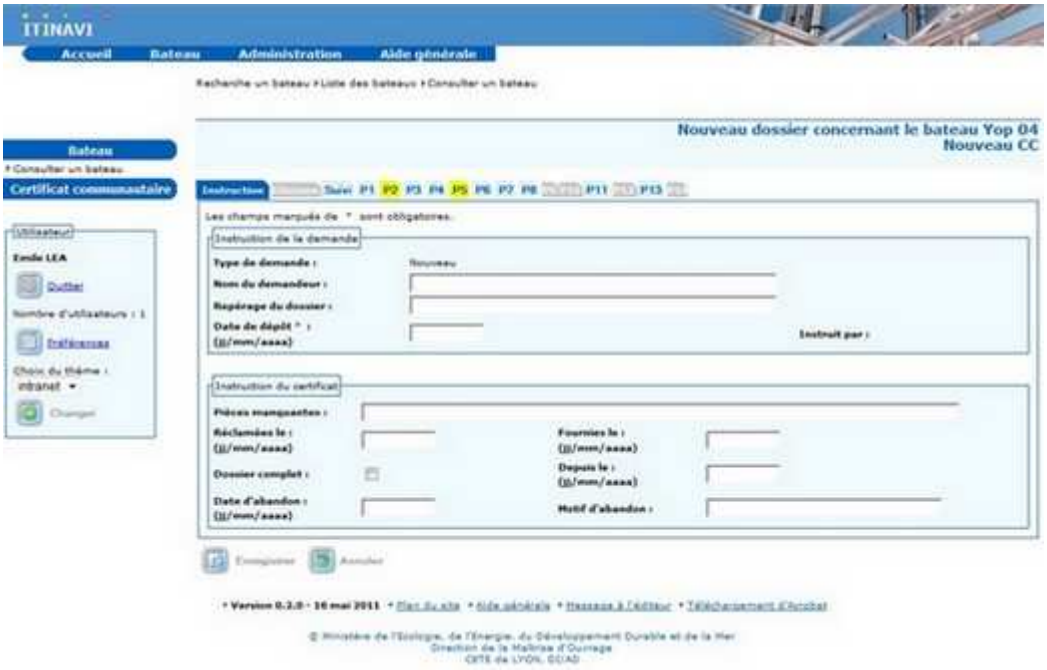


Figure V-15 : Nouveau CC

Pour afficher la page JSP, les bibliothèques de tags de struts sont là pour nous aider, en voici un petit bout de code qui nous permet d'afficher la zone de saisie de date de dépôt :

```
<tr>
  <td>
    <label>
      <bean:message key="modifierDossier.instruction.dateDepot"/>
      <span class="obligatoire"> * </span> :<br/>
      <bean:message key="format.date.avecParentheses"/>
    </label>
  </td>
  <td>
    <html:text name="majPageForm" property="dateDepot_ts"/>
  </td>
</tr>
```

- **Validation struts**

A chaque changement d'onglet, il y a une validation struts, il faut que tous les champs obligatoires soient saisis et dans le bon format. La saisie d'une date de dépôt est obligatoire comme le montre l'astérisque (*) à côté du champ de saisie (Figure V-15).

Si on essai de changer d'onglet sans saisir de date de dépôt, un message d'alerte s'affiche nous informant que cette date est obligatoire, le contrôle est fait dans le fichier « validation.xml » :

```
<field depends="required,date" property="dateDepot_ts">
  <arg0 key="modifierDossier.instruction.dateDepot"/>
  <var>
    <var-name>datePatternStrict</var-name>
    <var-value>dd/MM/yyyy</var-value>
  </var>
</field>
```

Comme pour tous les champs de saisi du framework, la date de dépôt se termine par « _ts ». Ici deux contrôles :

- La date est obligatoire : required
- Elle est de la forme « dd/MM/yy »

Un changement d'onglet ou l'enregistrement du CC ne peut pas se faire si les deux conditions ne sont pas remplies.

La saisie de données correctes et le click sur le bouton d'enregistrement nous envoie à l'action « MajCcProcessAction » *[Annexe, 4]*. Dans cette classe on charge dans une boucle les données des Forms, qui ne sont que les données saisies dans les différents onglets, dans les scalars. Les Forms et les scalars sont les mêmes initialisés dans la classe MajCcAction.

L'enregistrement se fait dans le la méthode « save » du service « CcService ».

4.2. Couche métier

Comme tous les services d'ITINAVI et selon les règles ACAI, le service « CcService », étend d'« AbstractCRUDService » du commun du CETE.

```
package i2.application.itinavi.metier.dossier;
public class CcService extends AbstractCRUDService<Cc> implements ICcService {

public Integer save(List<AbstractScalarMajPage> listeScalarMajCcPage) throws
ServiceException, TechniqueException {

    IBateauService bateauService = ServiceFactory.getInstance().getBateauService();
    ITitreNavService titreNavService = ServiceFactory.getInstance().getTitreNavService();
    IDossierService dossierService = ServiceFactory.getInstance().getDossierService();
```

```
      Integer bateauId = null;
      Integer dossierId = null;
      Cc certificat = null;
      Cc certificatHistorique = null;
      Cc certificatPrecedent = null;
      Map<String, Dossier> listeDossiers;
      Bateau bateau = null;

      /* création nouveau bateau*/
      if (!ItinaviUtils.isIdValide(bateauId) && !ItinaviUtils.isIdValide(dossierId)) {
        /* créér un nouveau bateau */
       bateau = bateauService.creerMonNouveauBateau();
      }

      /*Création certificat communautaire */
      certificat = new Cc();
      certificat.setBatiment(bateau);

      /* modification du dossier et gestion de l'historique*/
      /* récupère la liste des dossiers à mettre à jour (C, P, H) */
      listeDossiers = titreNavService.configurerDossier(certificat, bateauId,
EnumTypeDossier.TYPE_CERTIFICAT_COMMUNAUTAIRE.getId(),
EnumObjetTitreNav.NOUVEAU.getId());
      /* certificat créé ou mis à jour*/
      certificat = (Cc) listeDossiers.get(Constantes.DOSSIER);
      /* certificat courant passé en précédent */
      certificatPrecedent = (Cc) listeDossiers.get(Constantes.DOSSIER_COURANT);
      save(certificatPrecedent);
      /* certificat précédent passé en historique */
      certificatHistorique = (Cc) listeDossiers.get(Constantes.DOSSIER_PRECEDENT);
      save(certificatHistorique);

      /* Copier les scalars dans les objets */
      if (!CollectionUtils.isEmpty(listeScalarMajCcPage)) {
        for (AbstractScalarMajPage scalarMajCcPage : listeScalarMajCcPage) {
          scalarMajCcPage.saveToValueObject(certificat);
        }
      }

      /*vérifie les regles de gestion communes aux dossiers */
      dossierService.validerRgCommunesDossiers(certificatPrecedent, certificat);

      /* Enregistrement du dossier courant passé en précédent * */
      if (certificatPrecedent != null && !ItinaviUtils.isIdValide(dossierId)) {
                  save((Cc) certificatPrecedent);
      }
      /*enregistrement du dossier précédent passé en historique*/
      if (certificatHistorique != null) {
                  save((Cc) certificatHistorique);
      }
      /* enregistrement du certificat en cours */
      save(certificat);
      // Renvoie l'id du certificat sauvegardé
      return certificat.getId();
}
}
```

On remarque la seule présence des scalars dans le service et aucun Form, le Form étant chargé dans le scalar dans MajCcAction. C'est le principe de séparation des couches. Les données dont on a besoin sont maintenant dans les scalars. On charge le valueObject CC depuis les scalars, c'est la boucle qu'on a dans le service. Une fois toutes les données sont chargées dans le valueObject, on fait la sauvegarde.

Pour un bateau on peut avoir plusieurs dossiers. Lors de la création du nouveau CC, il y a des règles de gestions à respecter. Une de ces règles est la vérification de la date du dépôt du nouveau dossier : elle doit être supérieure à celle du précédent.

Voici, la méthode qui permet de gérer cette règle de gestion, elle est appelé depuis la méthode précédente « save » du service « CcService » :

```
public void validerRgCommunesDossiers (Dossier dossierPrecedent, Dossier
dossierCourant) throws RegleGestionException {

      // la date du dépôt du dossier courant doit être supérieure à celle du précédent
      if (dossierPrecedent != null) {
          if (dossierPrecedent != null && dossierPrecedent.getDateDepot() != null &&
dossierCourant.getDateDepot().before(dossierPrecedent.getDateDepot())) {
              throw new RegleGestionException("rg.DossierService.dateDepotInvalide");
          }
      }
}
```

➢ *Gestion des exceptions*

La saisie d'une date antérieure à la date de dépôt du dossier précédent lève une exception : RegleGestionException. Elle est interceptée et gérée dans l'action MajCcProcessAction, de la couche au dessus. C'est dans le bloc catch() qu'elle sera traité.

Avec la clé « rg.DossierService.dateDepotInvalide », on récupère du fichier de ressources « reglesgestion.properties» un message pour alerter l'utilisateur. C'est dans ce fichier qu'on trouve tous les messages des règles de gestion.

V-16 : Message d'alerte pour l'utilisateur

Avant de vérifier cette règle de gestion, on a chargé et configuré la liste des dossiers (courant, précédent et historisé), c'est la méthode *titreNavService.configurerDossier().* Dans cette méthode nous avons enregistré les dossiers (précédent et historisé).

➢ *Gestion des transactions*

Nous avons vu dans le paragraphe (Chapitre 5, paragraphe 3.5) de gestion des transactions, que les transactions sont automatisées en utilisant la Programmation Orientée Aspect (AOP) de Spring. Pour que la transaction soit gérée par spring et hibernate, il y a des règles à respecter, rappelons-les :

- Placer la classe dans le package de services
- Préfixer le nom du service par « Service »
- Préfixer le nom de la méthode par l'un des éléments de la liste

Notre service est bien placé dans le package « i2.application.itinavi.metier », son nom « Cc**Service** » rempli la deuxième condition. Le nom de la méthode « save » répond à la dernière exigence. Par conséquence, le mode transactionnel doit être bien géré. Seule la règle de gestion de précédence des dates des dépôts n'est pas respectée, par conséquent elle empêche le « commit » de la transaction.

➢ *Journalisation*

Suite au rollback, une exception est levée, le développeur voit apparaître la trace suivante dans la console d'eclipse :

```
org.hibernate.transaction.JDBCTransaction DEBUG - rollback
(BCA8475F4333658A8D1637850E6EC4FE)
```

```
org.hibernate.transaction.JDBCTransaction DEBUG - rolled back JDBC Connection
(BCA8475F4333658A8D1637850E6EC4FE)
org.hibernate.jdbc.ConnectionManager DEBUG - transaction completed on session with
on_close connection release mode; be sure to close the session to release JDBC resources!
(BCA8475F4333658A8D1637850E6EC4FE)
```

C'est la configuration du fichier log4j qui nous permet de voir la trace (Chapitre V, paragraphe 3.6.1). Cette trace confirme qu'il y a eu rollback et que le dossier n'est pas enregistré.

4.3. Couche integration

La couche métier est couplée à la couche intégration dès que les objets métier ont besoin d'accéder à des données ou des services qui se trouvent dans la couche ressources.

Dans l'onglet 6, on saisie un ensemble d'ancres pour le bateau.

Un bateau a plusieurs ancres, donc pour hibernate on a une relation père/fils entre le père, ici c'est le bateau et l'ancre, son fils.

Dans le fichier de mapping de donneeBateau on a un set d'ancres, le choix du set comme collection nous garantie l'unicité des éléments et l'accès direct à la collection :

```
<set name="listeAncres" cascade="save-update,evict,delete-orphan" inverse="true"
lazy="true">
  <key column="noseriedonneebateau" not-null="true"/>
  <one-to-many class="i2.application.itinavi.valueobjects.donnee.Ancre"/>
</set>
```

Le chargement de la liste des ancres se fait au fur et à mesure des demandes car lazy="true". Le père « donneeBateau » déclare ses fils et les fils déclarent leur père. En effet dans la méthode saveToValueObject() du ScalarMajCcPage06 on a :

```
ancre.setDonnee(donneeBateau);
donneeBateau.getListeAncres().add(ancre);
```

On a vu dans MajCcAction qu'on fait appel au service pour récupérer objetTitreNav par son code. Le service, à son tour, fait appel à la couche DAO, on a la requête criteria d'hibernate :

```
public ObjetTitreNav rechercherObjetTitreNavParCode(final String codeObjetDemandeTitre)
{
  ObjetTitreNav objetTitreNav = null;
  if (!StringUtils.isEmpty(codeObjetDemandeTitre)) {
    Criteria criteria = getSession().createCriteria(ObjetTitreNav.class);
    criteria.add(Restrictions.eq("code", codeObjetDemandeTitre));
```

```
    objetTitreNav = (ObjetTitreNav) DataAccessUtils.singleResult(criteria.list());
  }
  return objetTitreNav;
}
```

Tout accès à la base de données se fait dans cette couche.

Après l'enregistrement, l'instructeur édite le CC. Il valide ensuite les données après confirmation de vérification de son édition. Les informations concernant ce certificat ne seront plus modifiables sauf par de nouvelles procédures.

5. SYNTHESE

Nous avons vu dans ce chapitre que nous réalisons le développement de l'application ITINAVI dans le respect des règles ACAI qui définissent l'environnement technique pour le développement.

Pour ce qui est de la réalisation, le développeur doit se baser sur le framework du CETE lequel offre une panoplie de frameworks : string, spring, hibernate… des frameworks puissants qui ont fait leurs preuves. La prise en main du framework CETE est difficile, mais une fois maîtrisés, le développement devient facile.

Nous avons vu l'avantage de la mavenisation du projet, surtout ses conventions et en particulier sur l'organisation des répertoires. Cet outil de build, maven, permet entre autres de générer un site ou une archive et la déployer.

Nous avons vu ensemble un certain nombre de concepts inhérents à spring, ainsi que leurs implémentations. A ce conteneur léger nous avons intégré Struts pour la couche présentation et hibernate pour la couche intégration. Si la prise en main de struts est relativement facile, l'ORM hibernate, lui, nous a posé beaucoup de difficultés. Notamment avec les problèmes de performances que nous avons résolu en activant la mémoire cache du second niveau et en optant pour du SQL natif à la place de l'API criteria.

VI. CONCLUSION ET PERSPECTIVES

Au cours de ce travail de mémoire, nous nous sommes focalisés sur la réalisation de l'application ITINAVI.

Rappel des objectifs

L'objectif à atteindre était la réalisation d'ITINAVI, une application de gestion des bateaux. C'est une application ouverte à toute évolution réglementaire ou technique en centrant les processus métier sur l'objet métier : le bateau.

Notre équipe travaillait selon les méthodes agiles scrum. Elles nous permettaient d'être auto-organisés et ainsi d'avoir la capacité à décider de l'organisation de nos propres activités pour atteindre les objectifs fixés, ou pour résoudre les problèmes auxquels nous nous sommes confrontés.

Travail réalisé

La réalisation du travail s'est faite dans le cadre du respect des règles ACAI, le guide du ministère à la conception et à la réalisation des applications. Les grandes lignes à respecter sont clairement définies dans le framework "commun" du CETE, que nous avons intégré aux dépendances du projet après mavenisation.

L'architecture en couche proposée utilise des framwork arrivant à phase de maturation (struts, spring, hibernate). Nous avons montré l'apport des ces technologies comme pour l'AOP spring qui permet la gestion des transactions. Ces frameworks libèrent le développeur d'une très grande partie des tâches de programmation. Néanmoins, ce sont des technologies complexes, les problèmes de performances en sont un bon exemple. Nous avons su faire face et résoudre ces problèmes de performances par la mise en place d'une mémoire cache de second niveau.

La plateforme d'intégration continue assure une vérification fréquente du code, et de sa bonne compilation. Cette vérification permet de détecter quasi-immédiatement la présence d'un problème dans le code et donc d'y apporter une solution instantanément. Elle permet aussi de créer des rapports périodiques exprimant la qualité du code.

Perspectives

Le système d'information à réaliser se composera à termes de 2 applications différentes : ITINAVI et CERCALINA. La première étant en fin de réalisation, nous allons poursuivre notre travail avec la réalisation de CERCALINA, la seconde application, dont l'étude est déjà faite.

VII. BIBLIOGRAPHIE

[Acai, 1] http://regles.application.equipement.gouv.fr/

[Log4j, 1] http://beuss.developpez.com/tutoriels/java/jakarta/log4j/

[Hibernate, 1] http://docs.jboss.org/hibernate/core/3.3/reference/fr/html/

[Security] http://baptiste-meurant.developpez.com/tutoriaux/acegi-dwr-tapestry5-spring-hibernate/

[Hibernate, 2] http://docs.jboss.org/hibernate/core/3.2/reference/fr/html_single/

[Struts, 1] http://javaweb.developpez.com/faq/struts/

[Spring, 1] http://spring.developpez.com/faq/

[Spring, 2] http://zekey.developpez.com/articles/spring/

[StrutsSpring] http://www.javabeat.net/articles/70-integrating-struts-with-spring-1.html

[Tiles, 1] http://igm.univ-mlv.fr/~dr/XPOSE2003/tiles/presentation.html

[JSTL, 1] http://adiguba.developpez.com/tutoriels/j2ee/jsp/jstl/

[Maven, 1] http://java.developpez.com/faq/maven/

[Agilité, 1] http://www.synergeek.fr/2011/02/methode-agile-scrum-cest-quoi/

[IC1] http://carmaworld.free.fr/blog/wp-content/posts/00004/Dossier_CI.pdf

[svn] http://dev.nozav.org/intro_svn.html

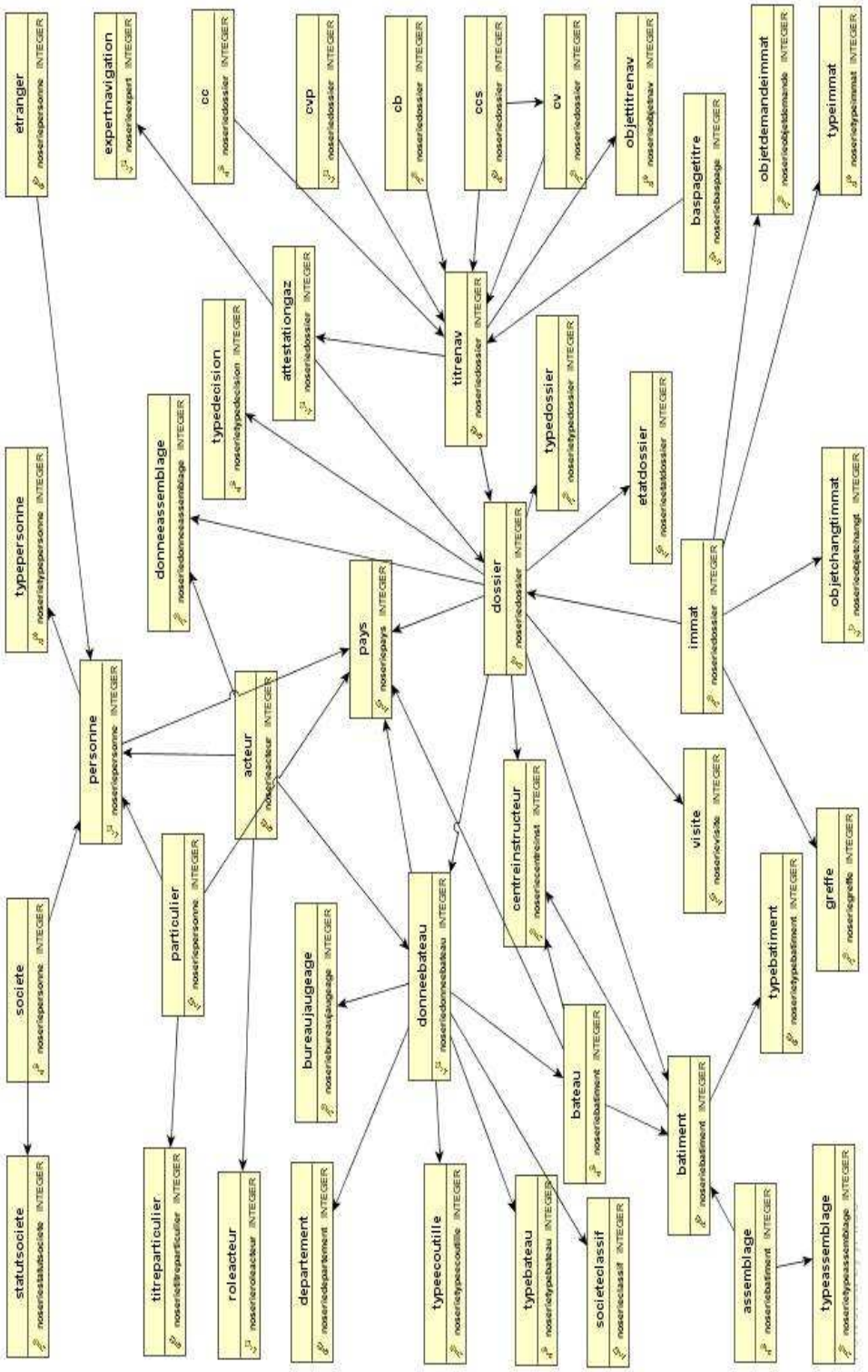
VIII. TABLE DES ILLUSTRATIONS

IX. GLOSSAIRE

AOP	Aspect Oriented Programmation
API	Application Programming Interface
CB	certificat de bateau
CC	certificat communautaire
CCS	certificat communautaire supplémentaire
CRUD	Create, Read, Update, Delete
CV	certificat de visite
CVP	certificat de visite provisoire
CERCALINA	CERtificat de CApacité et LIvrets pour la NAvigation
DAO	Data Access Object
DDTM	DDTM Direction Départementale des Territoires et de la Mer
DGITM	Direction Générale des Infrastructures, des Transports et de la Mer
HQL	Hibernate Query Language
HTTP	HyperText Transfer Protocol
IC	Intégration Continue
IDE	Integrated Development Environment
ITINAVI	Immatriculation et TItres pour la NAVigation Intérieure
IoC	Inversion Of Control
JDK	Java Développement Kit
JEE	Java Enterprise Edition
JSTL	Java Standard Tag Library
JVM	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol
MVC	Model View Controler
ORM	Object Relation Mapping
POJO	Plain Old Java Object
POM	Project Object Model
XP	Extreme Programming
SCM	Software Configuration Management
SIF	Système d'Information Fluvial

X. ANNEXES

[Aannexe 1] : Diagramme des principales tables (91 au total)



[Annexe 2] : La classe obstraite AbstractPageForm

```
/**
 * Form abstraite représentant une page d'un certificat communautaire ou de visite.
 */
public abstract class AbstractPageForm extends
ItinaviOngletForm<AbstractScalarMajPage> {

	/** Numéro de la page tel qu'apparaissant à l'affichage. */
	private String numeroPage_ts;

	/** The numero dossier. */
	private String numeroDossier_ts;

	/** The bateau id_ts. */
	private String bateauId_ts = "0";

	/** Premier signataire de la page. */
	private String signataire1ModifPage_ts;

	/** Second signataire de la page. */
	private String signataire2ModifPage_ts;

	/** Commission ayant fait la modification du certificat. */
	private String commissionModifPage_ts;

	/** Date de dernière modification de la page. */
	private String dateModifPage_ts;

	/** Lieu de dernière modification de la page. */
	private String lieuModifPage_ts;

	/** The mention modif page_ts. */
	private String mentionModifPage_ts;

	/** l'objet de changement d'un titre(Nouveau, Mutation, Radiation .... */
	private String codeObjetTitreNav_ts;

	/** Titre de la page. */
	private String titre_ts;

	/** The version bas page_ts. */
	private String versionBasPage_ts;

	/** Numéro de l'onglet. */
	private int numeroOnglet;

	/** The nom onglet. */
	private String nomOnglet;

	/** Masque bit à bit des onglets à cacher. */
	private String afficheOnglet_ts;

	/** The est onglet affiche. */
	private boolean[] estOngletAffiche;

	/** The date maj. */
	private String dateMAJ;
```

```
	/** Le bateau sur lequel on travaille a-t-il une immatriculation ? */
	private boolean possedeImmat;

	/**
	 * Instantiates a new abstract page form.
	 */
	protected AbstractPageForm() {
		super();
		this.estOngletAffiche = new boolean[17];
		for (int i = 0; i <= 16; i++) {
			this.estOngletAffiche[i] = true;
		}
	}


	@Override
	public void loadFromScalar(AbstractScalarMajPage scalar) throws TechniqueException
{
		super.loadFromScalar(scalar);
		this.setNumeroPage_ts(scalar.getNumeroPage_ts());
		this.setNumeroDossier_ts(scalar.getNumeroDossier_ts());
		this.setBateauId_ts(scalar.getBateauId_ts());
		this.setSignataire1ModifPage_ts(scalar.getSignataire1ModifPage_ts());
		this.setSignataire2ModifPage_ts(scalar.getSignataire2ModifPage_ts());
		this.setCommissionModifPage_ts(scalar.getCommissionModifPage_ts());
		this.setDateModifPage_ts(scalar.getDateModifPage_ts());
		this.setLieuModifPage_ts(scalar.getLieuModifPage_ts());
		this.setMentionModifPage_ts(scalar.getMentionModifPage_ts());
		this.setCodeObjetTitreNav_ts(scalar.getCodeObjetTitreNav_ts());
		this.setTitre_ts(scalar.getTitre_ts());
		this.setVersionBasPage_ts(scalar.getVersionBasPage_ts());
		this.dateMAJ = scalar.getDateMAJ();
	}


	public boolean[] getEstOngletAffiche() {
		return estOngletAffiche;
	}

	public String getNomOnglet() {
		return nomOnglet;
	}


	public void setNomOnglet(String nomOnglet) {
		this.nomOnglet = nomOnglet;
	}


	private int ongletAfficheFromArray() {
		int afficheOnglet = 0;
		for (int i = EnumNoOngletCC.INSTRUCTION.getId().intValue(); i <=
EnumNoOngletCC.PAGE14.getId().intValue(); i++) {
			if (this.estOngletAffiche[i - 1]) {
				int x = (int) Math.pow(2, i - 1);
				afficheOnglet = afficheOnglet | x;
			}
		}
		return afficheOnglet;
	}
```

```
	@Override
	public void saveToScalar(AbstractScalarMajPage scalar) throws TechniqueException {
			// Save parent
			super.saveToScalar(scalar);
			// Numéro de la page
			scalar.setNumeroPage_ts(this.getNumeroPage_ts());
			scalar.setNumeroDossier_ts(this.getNumeroDossier_ts());
			scalar.setBateauId_ts(this.getBateauId_ts());
			scalar.setSignataire1ModifPage_ts(this.getSignataire1ModifPage_ts());
			scalar.setSignataire2ModifPage_ts(this.getSignataire2ModifPage_ts());
			scalar.setCommissionModifPage_ts(this.getCommissionModifPage_ts());
			scalar.setDateModifPage_ts(this.getDateModifPage_ts());
			scalar.setLieuModifPage_ts(this.getLieuModifPage_ts());
			scalar.setMentionModifPage_ts(this.getMentionModifPage_ts());
			scalar.setCodeObjetTitreNav_ts(this.getCodeObjetTitreNav_ts());
			scalar.setTitre_ts(this.getTitre_ts());
			scalar.setNumeroOnglet(this.getNumeroOnglet());
			scalar.setVersionBasPage_ts(this.getVersionBasPage_ts());
	}


	@Override
	public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {
			return super.validate(mapping, request);
	}


	public void init(HttpServletRequest request) {

			/* Remettre les paramètres de navigation dans la requete */
			request.setAttribute(Constantes.PARAMURL_IDHISTO,
request.getParameter(Constantes.PARAMURL_IDHISTO));
			request.setAttribute(Constantes.PARAMURL_IDBATEAU,
request.getParameter(Constantes.PARAMURL_IDBATEAU));
			request.setAttribute(Constantes.PARAMURL_OBJET_TITRE_NAV,
request.getParameter(Constantes.PARAMURL_OBJET_TITRE_NAV));
			if (request.getAttribute(Constantes.PARAMURL_EXISTE_GAZ) == null) {
				request.setAttribute(Constantes.PARAMURL_EXISTE_GAZ,
request.getParameter(Constantes.PARAMURL_EXISTE_GAZ));
			}
			if (request.getAttribute(Constantes.PARAMURL_EXISTE_P14) == null) {
				request.setAttribute(Constantes.PARAMURL_EXISTE_P14,
request.getParameter(Constantes.PARAMURL_EXISTE_P14));
			}

			String idBateau = null;
			if
(ItinaviUtils.isIdValide(request.getParameter(Constantes.PARAMURL_IDBATEAU))) {
				idBateau =
request.getParameter(Constantes.PARAMURL_IDBATEAU);
				this.bateauId_ts = idBateau;
			}
			String id = null;
			if (ItinaviUtils.isIdValide(request.getParameter(Constantes.ID))) {
				id = request.getParameter(Constantes.ID);
				this.id = ItinaviUtils.integerValueOf(id);
			}
```

```
		String objetTitreNav =
request.getParameter(Constantes.PARAMURL_OBJET_TITRE_NAV);
		EnumNoOngletCC enumNumeroOnglet =
EnumNoOngletCC.getEnumFromId(this.numeroOnglet);

		if (enumNumeroOnglet != null) {
			this.nomOnglet = enumNumeroOnglet.toString().toLowerCase();
			// Initialise le bas de page sur les pages concernées en cas de
récup ou de récup de modif
			if (enumNumeroOnglet.getNoPage().intValue() > 0) {
				if
(EnumObjetTitreNav.RECUPERATION.getId().equals(objetTitreNav) ||
EnumObjetTitreNav.RECUPMODIF.getId().equals(objetTitreNav)) {
					if (!ItinaviUtils.isIdValide(this.getId())) {
						this.mentionModifPage_ts =
EnumOuiNon.OUI.getStringValue();
					}
				}
			}
		}

		if (!EnumObjetTitreNav.PROLONGATION.getId().equals(objetTitreNav) &&
!EnumObjetTitreNav.RECUPERATION.getId().equals(objetTitreNav)
			&&
!EnumObjetTitreNav.RECUPMODIF.getId().equals(objetTitreNav)) {
			this.estOngletAffiche[EnumNoOngletCC.PAGE09.getId() - 1] =
false;
			this.estOngletAffiche[EnumNoOngletCC.PAGE10.getId() - 1] =
false;
			this.estOngletAffiche[EnumNoOngletCC.PAGE12.getId() - 1] =
false;
		}

		this.afficheOnglet_ts = String.valueOf(this.ongletAfficheFromArray());

		// Définit l'adresse de retour
		this.setRetour(this.gererRetour(request));

		/* renseigne l'action en cours */
		if (!ItinaviUtils.isIdValide(this.id)) {
			this.setActionEnCours(true);
		} else {
			this.setActionEnCours(false);
		}

		this.initListes();
	}

	protected String gererRetour(HttpServletRequest request) {
		StringBuffer actionRetour = new StringBuffer();
		if (ItinaviUtils.isIdValide(this.id)) {
			// Le dossier existe -> Retour vers la visualisation du dossier :
			actionRetour.append("visualisercc.do?");
			actionRetour.append(Constantes.ID);
			actionRetour.append("=");
			actionRetour.append(this.id);
		} else {
			if (ItinaviUtils.isIdValide(this.bateauId_ts)) {
				// Le bateau existe -> Retour ver la visualisation du
bateau
				actionRetour.append("visualiserbateau.do?");
```

```
                                        actionRetour.append(Constantes.ID);
                                        actionRetour.append("=");
                                        actionRetour.append(this.bateauId_ts);
                                } else {
                                        // Le bateau n'existe pas : on retourne sur le résultat
de la recherche des bateaux
                                        actionRetour.append("rechercherbateauprocess.do");
                                }
                        }
                        return actionRetour.toString();
        }

        @Override
        public boolean estNew() {
                        if (!ItinaviUtils.isIdValide(this.id))
                                return true;
                        return false;
        }
}
```

[Annexe 3] : recherche de la liste de bateaux, du SQL natif

```
public List<BateauOverview> rechercherListeBateaux(final BateauCriteresBean bean)
throws DAOException {
        StringBuffer sqlQuery = new StringBuffer();
        // Id batiment, Id dossier, N° Titre/Immat, NIF, EN I
        sqlQuery.append("select distinct b.noseriebatiment, d.noseriedossier, d.nodossier,
b.noidentificationfrance, cast(bateau.nouniqueeni as varchar),");
        // Devise, Type de dossier, N°Immat, Flag donneeba teau courant/précédent/historique
        sqlQuery.append(" db.devise, td.codetypedossier, db.noimmatriculation,
cast(db.histodonneebateau as varchar),");
        // Id propriétaire principal, Nom [+ prénom] propriétaire principal
        sqlQuery.append(" coalesce(pa.noseriepersonne, ps.noseriepersonne) as
noseriepersonne, p.nompersonne || ' ' || coalesce(pa.prenomparticulier,'') as nom");
        sqlQuery.append(" from dossier d");
        sqlQuery.append(" inner join bateau on bateau.noseriebatiment = d.noseriebatiment");
        sqlQuery.append(" inner join batiment b on b.noseriebatiment = d.noseriebatiment");
        // TypeDossier et DonneeBateau d'un des dossiers courants pour la récupération
        sqlQuery.append(" inner join typedossier td on td.noserietypedossier =
d.noserietypedossier");
        sqlQuery.append(" inner join donneebateau db on db.noseriedonneebateau =
d.noseriedonneebateau");
        // Dossier, TypeDossier, DonneeBateau liés à la DonneeBateauCourante pour la
recherche (sauf si recherche sur n°Immat ou titrena v)
        sqlQuery.append(" inner join donneebateau dbc on b.noseriebatiment =
dbc.noseriebatiment");
        sqlQuery.append(" inner join dossier dc on dc.noseriedonneebateau =
dbc.noseriedonneebateau");
        sqlQuery.append(" inner join typedossier tdc on tdc.noserietypedossier =
dc.noserietypedossier");
        if (!StringUtils.isEmpty(bean.getNomProprietaire())) {
                // Recherche un nom parmi les acteurs (attention : pas de restriction sur
propriétaire ou exploitant)
```

```
                    sqlQuery.append(" left join acteur ac on ac.noseriedonneebateau =
dbc.noseriedonneebateau");
                    sqlQuery.append(" left join personne pc on ac.noseriepersonne =
pc.noseriepersonne");
      }
      // Récupére les propriétaires principaux
      sqlQuery.append(" left join acteur a on db.noseriedonneebateau =
a.noseriedonneebateau and a.proprietaireprincipal = true");
      sqlQuery.append(" left join personne p on p.noseriepersonne = a.noseriepersonne");
      sqlQuery.append(" left join particulier pa on pa.noseriepersonne = p.noseriepersonne");
      sqlQuery.append(" left join societe ps on ps.noseriepersonne = p.noseriepersonne");
      // Récupère uniquement les dossiers courants des différents bateaux
      sqlQuery.append(" where            d.histodossier = '" +
EnumTypeHisto.TYPE_COURANT.getId() + "'");
      // Recherche parmi les dossiers courants
      sqlQuery.append(" and dc.histodossier = '" + EnumTypeHisto.TYPE_COURANT.getId()
+ "'");
      // Ne pas récupérer les dossiers de type Attestation gaz
      sqlQuery.append(" and td.codetypedossier != '" + EnumTypeDossier.TYPE_GAZ.getId()
+ "'");

      // Si le numéro d'immat ou le code du titre de navigation sont renseignés, on va
chercher
      // dans tous les dossiers courants et pas seulement sur la DB courante
      if (StringUtils.isEmpty(bean.getNumimmat()) &&
StringUtils.isEmpty(bean.getNumtitnav())) {
                    sqlQuery.append(" and dbc.histodonneebateau = 'C'");
      } else {
                    // N°Immat ou N°titre de navigation
                    sqlQuery.append(" and dc.nodossier = :noDossier");
                    if (StringUtils.isEmpty(bean.getNumimmat())) {
                                  // Cherche parmi les titres de navigation
                                  sqlQuery.append(" and tdc.codetypedossier !=
:codeTypeImmat");
                    } else {
                                  // Cherche parmi les immatriculations
                                  sqlQuery.append(" and tdc.codetypedossier =
:codeTypeImmat");
                    }
      }

      /* Nom du propriétaire */
      if (!StringUtils.isEmpty(bean.getNomProprietaire())) {
                    sqlQuery.append(" and pc.nompersonne ilike :nomPersonne");
      }

      /* Devise */
      if (!StringUtils.isEmpty(bean.getDevise())) {
                    sqlQuery.append(" and translate(dbc.devise, '" +
Constantes.SQL_RESTRICTION_CARACTERES_AVEC_ACCENT + "', '" +
Constantes.SQL_RESTRICTION_CARACTERES_SANS_ACCENT
                                  + "') ilike translate(:devise,'" +
Constantes.SQL_RESTRICTION_CARACTERES_AVEC_ACCENT + "', '" +
Constantes.SQL_RESTRICTION_CARACTERES_SANS_ACCENT + "')");
      }

      /* Centre instructeur */
      if (bean.getCentreInstructeurId() != null && bean.getCentreInstructeurId() != 0) {
                    sqlQuery.append(" and dc.noseriecentreinst = :centreInstructeurId");
      }
```

```
	/* Numéro unique ENI */
	if (!StringUtils.isEmpty(bean.getNoUniqueENI())) {
			sqlQuery.append(" and bateau.nouniqueeni = :noUniqueEni");
	}

	/* Numéro identification france (NIF) */
	if (!StringUtils.isEmpty(bean.getNoIdentificationFrance())) {
			sqlQuery.append(" and b.noidentificationfrance = :noIdentificationFrance");
	}
	// Trie par bateau et codetypedossier pour que les numéros de titre de navigation soient
tjs affichés dans le même ordre
	sqlQuery.append(" order by b.noseriebatiment, td.codetypedossier");

	SQLQuery query = getSession().createSQLQuery(sqlQuery.toString());

	// Si le numéro d'immat ou le code du titre de navigation sont renseignés, on va
chercher
	// dans tous les dossiers courants et pas seulement sur la DB courante
	if (!StringUtils.isEmpty(bean.getNumimmat()) ||
!StringUtils.isEmpty(bean.getNumtitnav())) {
	query.setString("codeTypeImmat",
EnumTypeDossier.TYPE_IMMATRICULATION.getId());
	// N°Immat ou N°titre de navigation
	if (StringUtils.isEmpty(bean.getNumimmat())) {
			// Cherche parmi les titres de navigation
			query.setString("noDossier", bean.getNumtitnav());
	} else {
			// Cherche une immat
			query.setString("noDossier", bean.getNumimmat());
	}
	}

	/* Nom du propriétaire */
	if (!StringUtils.isEmpty(bean.getNomProprietaire())) {
			query.setString("nomPersonne", bean.getNomProprietaire() + "%");
	}

	/* Devise */
	if (!StringUtils.isEmpty(bean.getDevise())) {
			query.setString("devise", bean.getDevise() + "%");
	}

	/* Centre instructeur */
	if (bean.getCentreInstructeurId() != null && bean.getCentreInstructeurId() != 0) {
			query.setInteger("centreInstructeurId", bean.getCentreInstructeurId());
	}

	/* Numéro unique ENI */
	if (!StringUtils.isEmpty(bean.getNoUniqueENI())) {
			query.setString("noUniqueEni", bean.getNoUniqueENI());
	}

	/* Numéro identification france (NIF) */
	if (!StringUtils.isEmpty(bean.getNoIdentificationFrance())) {
			query.setString("noIdentificationFrance", bean.getNoIdentificationFrance());
	}
	// Exécution de la requête
	List<Object[]> listeResultats = query.list();

	// Vérifie s'il y a des données dans la requête
	if (!CollectionUtils.isEmpty(listeResultats)) {
```

```
		// Initialise la liste des bateaux
		List<BateauOverview> listeBateaux = new ArrayList<BateauOverview>();
		// Overveiw du bateau en cours de traitement
		BateauOverview bateau = null;
		// Id du bateau précédent pour vérifier s'il y a chgt de bateau
		Integer batimentPrecedentId = null;
		for (Object[] item : listeResultats) {
			Integer batimentCourantId =
			(Integer)item[RECHERCHE_BATEAU_IDBATIMENT];
			// Vérifie s'il y a un changement de bateaux
			if(batimentPrecedentId == null ||
			!batimentCourantId.equals(batimentPrecedentId)) {
				bateau = new BateauOverview();
				// Id de l'overview = Id bateau
				bateau.setId(batimentCourantId);
				// Id bateau
				bateau.setNoSerieBatiment_(batimentCourantId);
				// NIF

	bateau.setNoIdentificationFrance_((String)item[RECHERCHE_BATEAU_NIF]);
				// ENI

	bateau.setNoUniqueENI_((String)item[RECHERCHE_BATEAU_ENI]);
				// Ajoute le bateau à la liste des bateaux créés
				listeBateaux.add(bateau);
				batimentPrecedentId = batimentCourantId;
			}
			// Vérifie qu'on soit bien sur la donnée bateau courante pour
renseigner les éléments à afficher

	if(EnumTypeHisto.TYPE_COURANT.getId().equals((String)item[RECHERCHE_BATEA
U_HISTODB])) {
	// Devise

	bateau.setDevise_((String)item[RECHERCHE_BATEAU_DEVISE]);
	// Nom du propriétaire principal

	bateau.setNomProprietaire_((String)item[RECHERCHE_BATEAU_NOMPERSONNE]);

	bateau.setProprietaires_((String)item[RECHERCHE_BATEAU_NOMPERSONNE]);
			}

	// Renseigne les numéros d'immat et de titres de navigation

	if(EnumTypeDossier.TYPE_IMMATRICULATION.getId().equals((String)item[RECHERC
HE_BATEAU_CODETYPEDOSSIER])) {
	// Numéro d'immatriculation du bateau

	bateau.setNumimmat_((String)item[RECHERCHE_BATEAU_NODOSSIER]);
			} else {
				if(bateau.getNumtitnav_() == null) {
					// Initialise la liste des titres de navigation

	bateau.setNumtitnav_((String)item[RECHERCHE_BATEAU_NODOSSIER]);
				} else {
					// Concatène le titre de navigation
					String numTitreNav =
bateau.getNumtitnav_();
					bateau.setNumtitnav_(numTitreNav + " - "
+ (String)item[RECHERCHE_BATEAU_NODOSSIER]);
				}
```

```
                        }
                    }
                    return listeBateaux;
        }
        return null;
}
```

[ANNEXE, 4] : Enregistrement d'un CC

```
public class MajCcProcessAction extends AbstractConversationItinaviAction {
      public ActionForward enregistrer(ActionMapping mapping, ActionForm form,
HttpServletRequest request, HttpServletResponse response) throws TechniqueException,
DAOException {

      ActionMessages messages = new ActionMessages();
      Conversation conversation = null;
      HttpSession session = request.getSession();
      ICcService ccService = ServiceFactory.getInstance().getCcService();
      List<AbstractScalarMajPage> listeScalarMajCcPage = new
ArrayList<AbstractScalarMajPage>();

      /*
       * 1 - Récuperation des formulaires
       * 2 - Validation de tous les formualaires
       * 3 - Récupération de la conversation
            */
      AbstractPageForm majCcPageFormCourante = (AbstractPageForm) form;
      /* Validation de tous les onglets */
      try {
            validateAllOnglet(mapping, request, majCcPageFormCourante);
      } catch (ValidationException ve) {
                  // traitement
      }

      // Récupération de la conversation à partir de l'onglet 1
      conversation = ConversationManager.getConversation(session,
majCcPageFormCourante.getConversationId());
      String sObjetTitreNav =
request.getParameter(Constantes.PARAMURL_OBJET_TITRE_NAV);

  for (int numeroOnglet = EnumNoOngletCC.INSTRUCTION.getId().intValue();
       numeroOnglet <= EnumNoOngletCC.PAGE14.getId().intValue(); numeroOnglet++) {
      EnumNoOngletCC enumNumeroOnglet =
                  EnumNoOngletCC.getEnumFromId(numeroOnglet);
      String nomForm = this.recupererNomFormPourPage(enumNumeroOnglet);
      AbstractPageForm majCcPageForm =
                   (AbstractPageForm) conversation.getListeForms().get(nomForm);
      // Initialisation des scalar, exp : scalar = new ScalarMajCcPage01()
      AbstractScalarMajPage scalarMajCc =
                  this.initialiserScalarPourPage(enumNumeroOnglet);
      // Remplir les scalars a partir des formulaires
      majCcPageForm.saveToScalar(scalarMajCc);
```

```
        listeScalarMajCcPage.add(scalarMajCc);
    }

    try {
        Integer id = ccService.save(listeScalarMajCcPage);
        // Fil d'ariane
        ItinaviUtils.progressionCouperChemin(session,
I18nHelper.getLocalizedMessage(request, "progression.nouvelle.immat"));
        return mapping.findForward("enregistrement_ok");
    } catch (AccessForbiddenException afe) {
        // Traitement
    } catch (RegleGestionException rge) {
        // Traitement
    } catch (ServiceException se) {
        // traitement
    }
  }
}
```

Mémoire Ingénieur CNAM : REALISATION D'UN SYSTEME D'INFORMATION POUR LA NAVIGATION INTERIEURE

RESUME

L'objectif de ce travail est le développement d'une application complète « ITINAVI » en centrant les processus métier sur l'objet métier : le bateau.

L'écriture du logiciel sera en architecture ACAI, un modèle client-serveur qui propose une répartition des services applicatifs en 3 niveaux. La réalisation suit les guides d'architecture commune des applications informatiques dans le but d'une appropriation facile par l'utilisateur.

C'est une architecture en couche en environnement JEE (Java Enterprise Edition) utilisant des frameworks puissants : Spring, Struts et Hibernate. Nous montrons l'apport de cette panoplie de technologies et sa contribution à faciliter le travail des développeurs ainsi que la difficulté de sa mise en place.

Nous montrons l'apport de l'industrialisation du développement, notamment l'intégration continue (IC). L'IC consiste à déployer des pratiques et outils visant à rendre le logiciel développé plus robuste, tout en restant dans des délais et coûts maîtrisés.

Mots-clées : JAVA, JEE, Spring, Struts, Hibernate, Tiles, Maven, Hudson, Intégration Continue, ACAI

SUMMARY

The objective of this work is the development of a complete new software named "ITINAVI", centered on the business processe carried by the business object.

This software relies on the "ACAI" client-server architecture, which provides a breakdown of application services in 3 levels, following the guidelines of common best partices.

It is a layered JEE (Java Enterprise Edition) architecture environment based on powerful frameworks: Spring, Struts and Hibernate. We show the contribution of these technologies and their contributions to facilitating the work of developers but also the difficulty of their implementation.

We show the contribution of industrial developments, including continuous integration, which facilitates deployments and the use of best practices to build more robust sotfware, while remaining in a timely and cost control.

Keywords: JAVA, JEE, Spring, Struts, Hibernate, Tiles, Maven, Hudson, Intégration Continue, ACAI