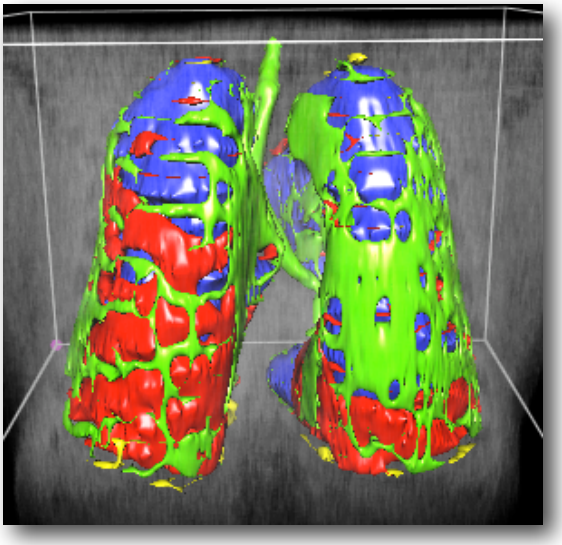
Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

Travail de bachelor 2009

Filière Informatique de gestion

Interface graphique pour analyse et recherche de cas similaires de maladies interstitielles pulmonaires dans le cadre du projet TALISMAN (Texture Analysis of Lung ImageS for Medical diagnostic AssistaNce).



Etudiant : Frédéric Gaillard

Professeur : Henning Müller

www.hevs.ch

Table des matières

Introduction

Informatique médicale et imagerie pour mener la bonne information au bon moment, pour permettre aux soignants de prendre les meilleures décisions possibles et diminuer le risque d'erreur.

Comment un soignant expérimenté fait-t-il son choix pour se diriger vers le bon diagnostic ?

Ses décisions seront déterminées par son expérience professionnelle. En fonction du cas présenté, le soignant se tournera en principe vers un petit nombre cas similaires diagnostiqués par le passé.

L'informatique médicale peut permettre au soignant de rechercher des informations en rapport à un diagnostic, mais elle peut aussi lui fournir des informations en fonction de paramètres cliniques et rassembler les connaissances et l'expérience de plusieurs soignants.

Par simple comparaison de paramètres cliniques comme l'âge ou l'historique de fumeur du patient, certains diagnostics pourront déjà être éliminés et l'orientation se fera vers des cas pertinents.

De plus l'informatique peut intégrer complètement les images en tant que paramètre de recherche et de comparaison à l'aide de traitement d'images et d'intelligence artificielle.

Ce travail se concentre sur la représentation des informations mises à disposition du soignant pour la recherche, l'analyse et l'enseignement de cas similaires de maladies interstitielles pulmonaires.

Présentation de Talisman

Cadre du projet

Les institutions médicales produisent une quantité énorme de données sur les patients, comprenant des données multimédia produites en format numérique.
Dans leur contexte clinique, ces données contiennent beaucoup d'information et représentent une base de connaissances qui ne sont à nos jours pas exploitées de façon optimale.
Les données numériques sont devenues accessibles pour des analyses et des traitements automatiques par de nombreuses applications.

Le nombre et la variété d'images produites peuvent en outre porter à confusion même pour des spécialistes expérimentés, causant un surplus d'information pour de nombreux médecins.

Une base de données contentant des images formatées, munies d'annotations et de données cliniques (dont le diagnostic est confirmé par biopsie) peut être un outil précieux et peut permettre un accès facilité à la bonne information au bon moment.
Les maladies interstitielles pulmonaires (ILDs) qui forment un ensemble d'environ 150 maladies différentes ont la caractéristique d'être rares et d'avoir des symptômes souvent très peu spécifiques.

L'interprétation des images tomographiques à haute résolution (HRCT) des poumons est complexe et peu d'expérience existe dans ce domaine.

Talisman propose une solution informatique capable de catégoriser automatiquement le tissu interstitiel dans les images et de récupérer des cas similaires dans une base de données dédiée aux maladies interstitielles pulmonaires contenant idéalement 150 cas typiques (90 actuellement).

La base de données contient une sélection de paramètres cliniques selon leur importance, faite en collaboration avec des pneumologues et des bases de connaissances de systèmes informatisés d'aide à la décision.
La base contient aussi des images HRCT annotées et caractérisées par les types de tissus interstitiels pulmonaires qu'elles contiennent.

Motivation

- La recherche d'une solution RIA (Rich Internet Application) qui pourrait s'apparenter à un développement pour client lourd comme se font actuellement les développements en Java, C#, tout en incluant les possibilités d'utilisation d'outils pour faciliter le travail en équipe, les tests et déploiements.
- La recherche pour la représentation des informations par priorité.
- La recherche pour la paramétrisation d'une application web comme un client lourd.
- L'intégration et la communication avec des composants externes.

Contraintes

- La solution devra s'intégrer à l'existant, en rapport avec la base de données et le serveur à disposition.
- Les outils utilisés seront, si possible, libres ou à code ouvert.
- L'ensemble de la solution devra fonctionner uniquement dans un navigateur web.
- La convivialité de l'interface devra faciliter la recherche, la synthèse et la priorisation des informations à consulter.

Structure du rapport

Introduction
Dans ce chapitre la présentation du cadre du projet Talisman est faite avec une brève introduction aux formats spécifiques à l'imagerie médicale utilisée dans ce projet.

Travail de Bachelor
Ce chapitre décrit le travail de Bachelor et son but en incluant les motivations personnelles et la planification, le point central étant le cahier des charges.

Le cahier des charges définit la première analyse de l'application à fournir, il détaille les principales fonctionnalités de l'application et donne les principaux axes à suivre.

Méthodes

Existant
Ce chapitre détaille l'existant en ce qui concerne le matériel et logiciel disponibles en rapport direct avec le projet ainsi que les contraintes qui en découlent.

Analyse, recherche
Plusieurs sujets sont abordés ici, en premier lieu des généralités sur les solutions RIA actuelles.
Ensuite le détail des recherches et choix des solutions faits pour un développement modulaire, des frameworks et une visionneuse DICOM 3D.

Choix des outils et solutions
Ici sont donnés les choix faits par rapport à la solution RIA choisie, à la structuration de l'application, aux outils d'intégration et déploiement, aux connecteurs client serveur choisis, aux frameworks choisis et aux méthodes d'intégration et de communication avec les composants externes.

Résultats

Outils
Les raisons du choix des outils utilisés sont détaillées dans ce chapitre.

Développement
Chaque fonctionnalité et étape du développement est décrite ici, avec introduction et généralité dans les étapes d'initialisation.
Ce chapitre suit en détail toutes les demandes du cahier des charges, décrit les solutions proposées et montre leur application, les changements de direction afin résoudre les problèmes qui sont apparus.

Conclusion
Dans ce chapitre, une conclusion sur les points les plus importants à relever est faite ainsi qu'une conclusion générale sur l'ensemble du travail et mon bilan personnel par rapport à cette expérience.

Remerciements
La description des collaborations apportées se trouve dans ce chapitre

Bibliographie, document administratif et annexes

Travail de Bachelor

Cahier des charges

Présentation

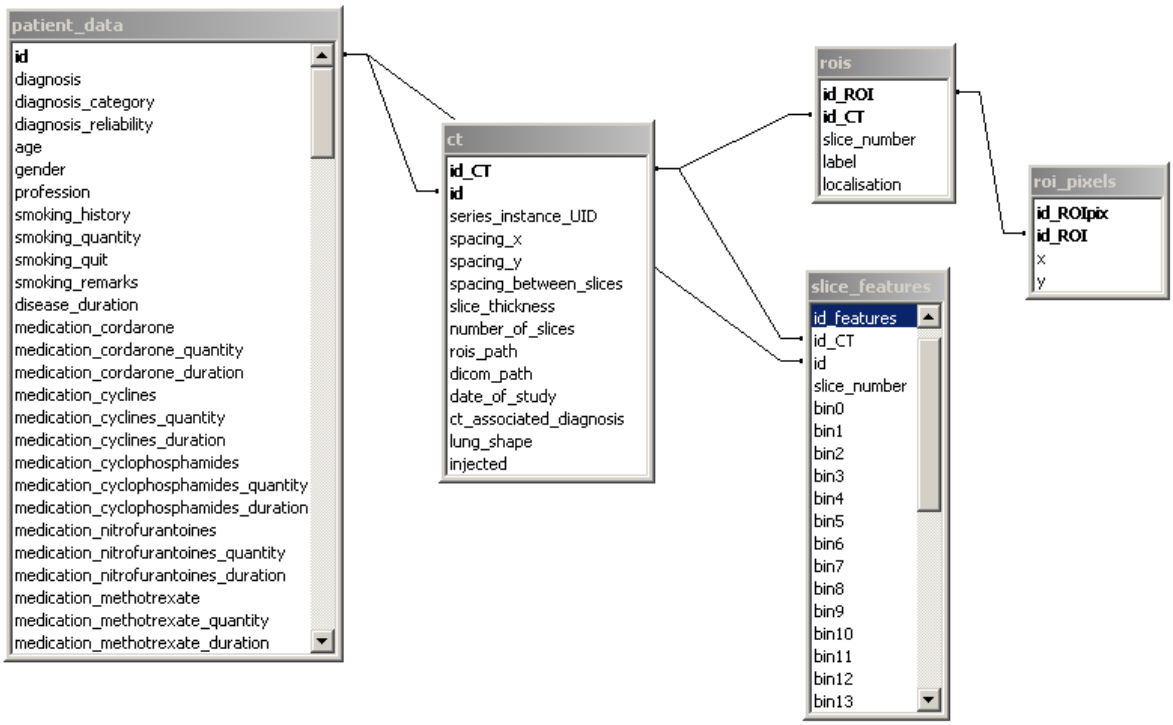
Dans le cadre du projet TALISMAN (Texture Analysis of Lung ImageS for Medical diagnostic AssistaNce)
Le but de ce projet est le développement d'une interface graphique basée web pour l'analyse et la recherche de cas similaires de maladies interstitielles pulmonaires.
Il devra permettre :
- la récupération et recherche de données cliniques des patients sur un serveur MySQL
l'affichage de séries images DICOM (Digital Imaging and COmmunications in Medicine, imagerie et communication numériques en médecine) représentées 3 dimensions avec la possibilité d'interagir sur les couleurs et transparences des images et des zones d'intérêts qu'elles contiennent.
- la liaison à Talisman (Java) pour permettre l'analyse de série d'images CT (Computed tomography) d'un nouveau cas ainsi que la recherche de cas par similarité visuel.

Architecture

Fonctionnel

Données

Modèle des données :



1_patient_data
Contient les 20 paramètres cliniques les plus importants des patients.
2_ct
Référence les séries d'images des patients
3_rois
Contient 2 types de régions d'intérêt :
les régions montrant les différents types de tissus
le poumon segmenté
4_roi_pixels
Contient les points des contours des ROIs Les séries d'images ne sont pas contenues dans la base, mais sont référencées dans la table "2_ct". Les descriptions des tables sont en attachement.
5 slice features
Contient les features pré-calculées pour chaque slice des séries d'images de "2_ct".
Logiciel et outil
Interface client
Adobe Flex builder 3 - visualisation et gestion des données
YaDIV* - visualisation des images
Connecteur client/server
AmfPhp, PHP - accès db
---** - pour accès Java
Serveur de donnée
MySQL
Serveur web
Apache
Serveur d'application
---**
Framework
Cairngorn - MVC (Modèle-Vue-Contrôleur)
---** - O/R mapper (Object-relational mapping)
Maven - Build Deploiment Test

* l'outil ou logiciel n'est pas encore testé ou utilisé, son choix peut encore changer
** le choix et l'utilisation de l'outil n'est pas encore défini.

Physique

Serveur Linux
Les fonctionnalités
Dans le projet TALISMAN, des outils d'analyse automatique d'images CT(Computed tomography) haute résolution (HRCT) et de récupération de cas similaires sont développés.
L'interface graphique basée web est composée des parties suivantes :
Page d'accueil permettant de choisir entre la navigation dans la base de données et l'analyse d'un nouveau cas.

Navigation dans la base de données :

Récupération de cas existants
Permet le choix des paramètres de récupération des cas existants dans la base de données et envoie une requête sur la base MySQL avec via le connecteur client/serveur
L'interface doit permettre de faciliter la rechercher parmi les nombreux paramètres, proposer et regrouper les critères de recherche selon leur importance et leur hiérarchie.

Affichage des résultats (multi-cas)
Affichage (résumé) de plusieurs cas sur une même page en permettant des facilités de regroupement pour une vision globale

Présentation du cas
Affiche une série d'images DICOM en trois dimensions et les régions d'intérêt (ROI) 3D avec rendus de surface et semi-transparence avec Flex ainsi qu'un choix de paramètres cliniques
Analyse d'un nouveau cas

Saisie d'un nouveau cas
Comme pour la recherche des cas, l'interface doit permettre de faciliter la recherche parmi les nombreux paramètres, proposer et regrouper les critères pour la saisie selon leur importance et leur hiérarchie.
Dans un premier temps les 20 paramètres suivants les plus importants seront mis en avant.

Table 2
List of the first 20 clinical attributes with highest A^{single} when combined with visual features. Abbreviations: HTA: arterial hypertension, subOAP: acute pulmonary edema, LDH: serum lactate dehydrogenase.

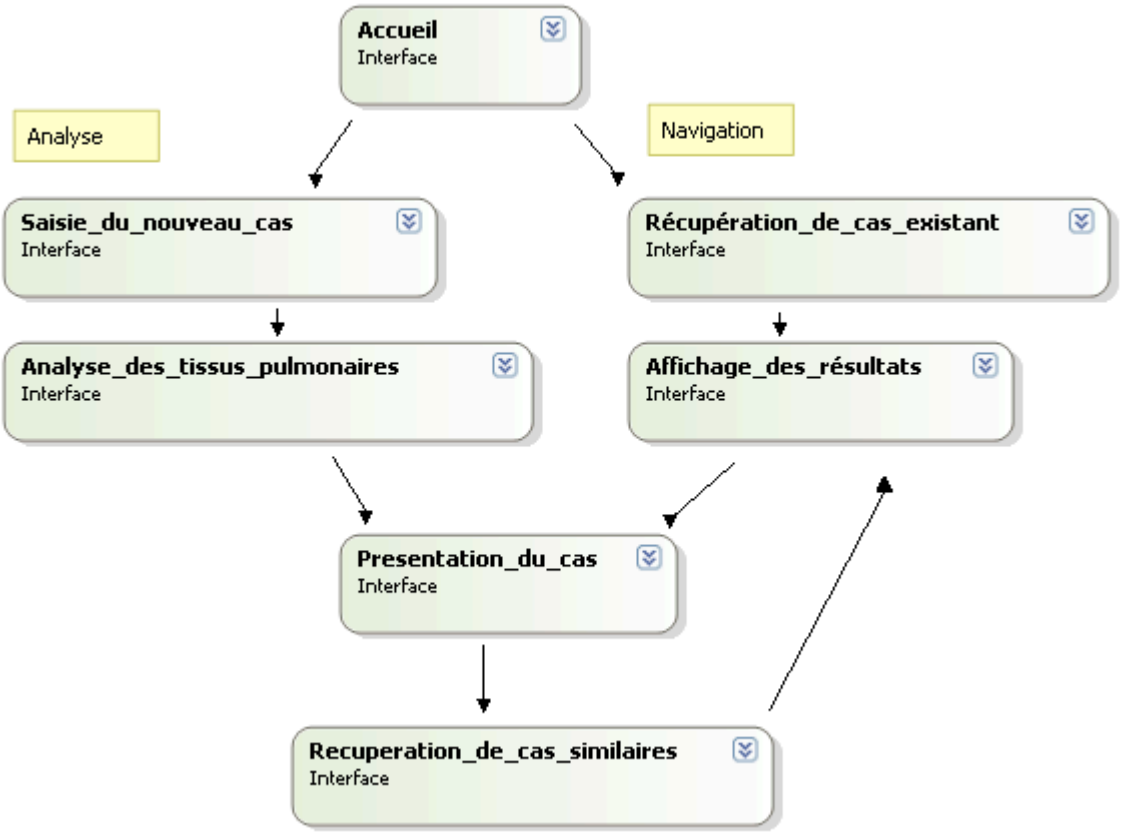
rank	A^{single}	$I_{G_{ratio}}$	name	type
1	0.813	0.403	laboratory_hematocrit	continuous
2	0.809	0.421	age	continuous
3	0.796	0.438	laboratory_hemoglobin	continuous
4	0.794	0.193	past_medical_allergy	binary
5	0.793	0.549	findings_physical_generals_lymph	binary
6	0.784	0.519	past_medical_lymphom	binary
7	0.779	0.469	findings_physical_generals_fever	binary
8	0.779	0.225	medication_cordarone	binary
9	0.778	0.218	host_diabetes	binary
10	0.763	0.227	biopsy_bronchoscopy_transbronchial_eosinophil	binary
11	0.759	0.239	past_medical_HTA	binary
12	0.758	0.261	past_medical_dyspnea_attack	binary
13	0.754	0.237	past_medical_subOAP	binary
14	0.752	0.124	findings_physical_respiratory_tachypnea	binary
15	0.751	0.506	host_chemotherapy	binary
16	0.746	0.182	past_medical_wheezing	binary
17	0.745	0.192	findings_physical_abdominals_liver	binary
18	0.745	0.227	biopsy_bronchoscopy_transbronchial_interstitial_fibrosis	binary
19	0.745	0.361	laboratory_LDH	continuous
20	0.743	0.52	host_hemopathy	binary

Permet la saisie des paramètres cliniques et de l'upload des images

Fait l'appel de l'algorithme d'analyse d'images en local dans une applet Java (calcul des features de l'image et classification des régions pulmonaires par blocks).

Récupération de cas similaires

Stockage des features pré-calculées de chaque cas dans la base (table 5 slice_features), puis et récupération une applet en local pour les comparer à celles du cas en cours et calcul des distances de similarité.
Les images des cas similaires identifiés peuvent être téléchargées par l'utilisateur.



Intérêts et axes

Les principaux intérêts et axes mis à par l'intégration d'un lecteur DICOM sont :

- le développement avec une solution RIA (Rich Internet Application) pour un développement souple et modulaire avec bonne séparation des couches données, métier et graphique.

- l'utilisation de nouveaux framework MVC, d'O/R mapper, d'object remoting et d'outil de gestion de projet pour les tests/intégrations/déploiements pour les RIA.

- la recherche pour faciliter l'utilisation et la représentation des informations selon leur importance

- la recherche pour la paramétrisation de l'interface par l'utilisateur pour son utilisation et son mode de fonctionnement (saisie, consultation, raccourcis…).

-l'intégration et l'utilisation des composants externes (viewer, lien avec application) comme des composants internes.

Motivations personnelles

Le choix de ce sujet s'est fait tout d'abord par rapport à l'imagerie médicale qui est un domaine complètement inconnu pour moi et qui peut m'apporter une nouvelle expérience.
Ensuite l'utilisation de mon expérience en client lourd pour la comparer à un développement d'une RIA.
Et pour finir mon expérience en développement web est pauvre et se résume à quelques très petites applications créées avec de nouvelles technologies mais sans jamais se confronter à de vrais problèmes pour pourvoir acquérir une vraie expérience.

Planification

Détail des tâches

La planification s'est faite en deux étapes, tout d'abord en détaillant les différentes tâches et en essayant de faire une répartition par priorité et importance sur le nombres d'heures disponibles, tout en prévoyant pendant la recherche et le développement du temps pour documenter les différents choix et développement effectués

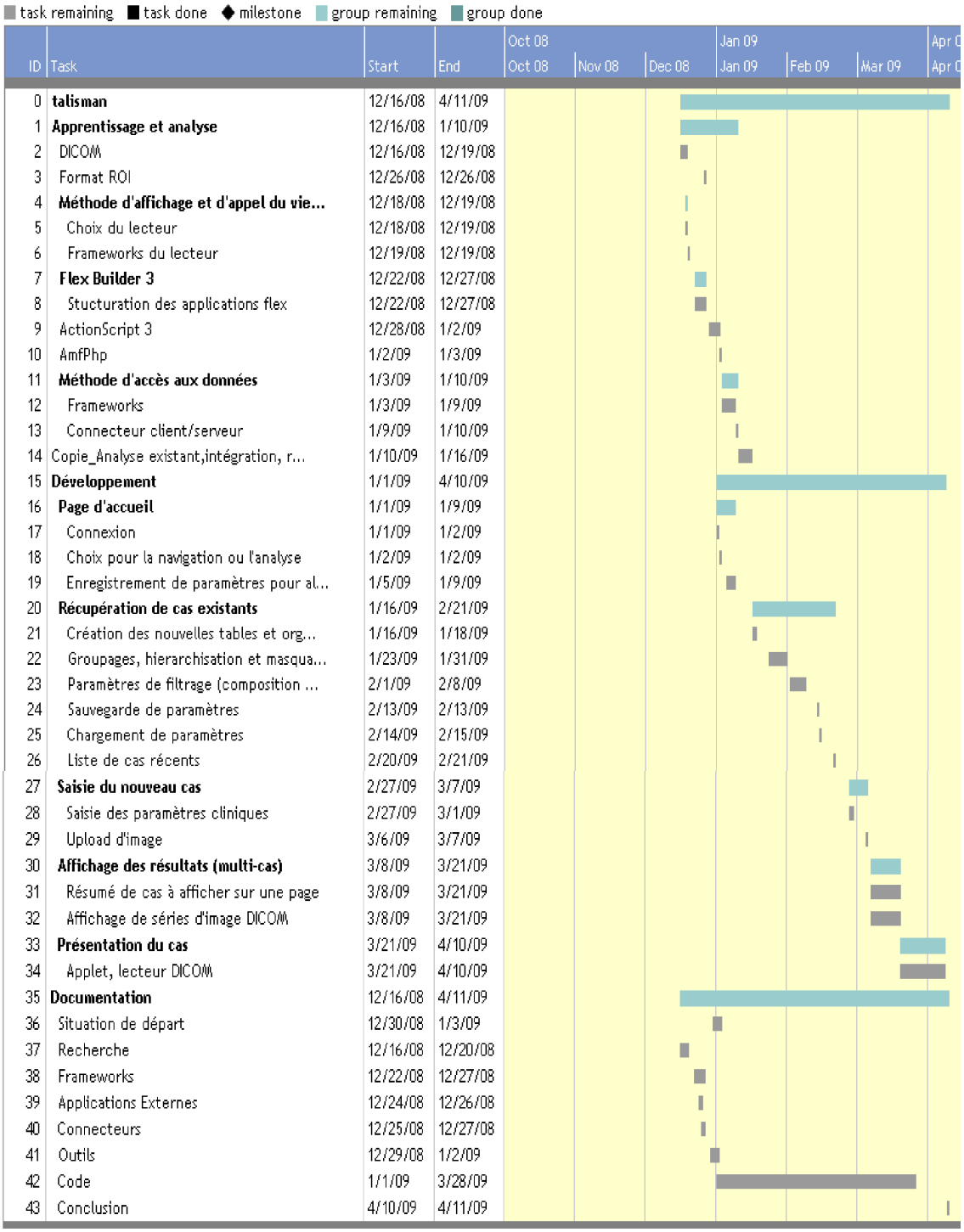
Tâches	Importance	Priorité
Apprentissage et analyse		
• Format ROI	Obligatoire	1
• DICOM	Obligatoire	1
• Flex	Obligatoire	1
○ Structuration des applications Flex	Obligatoire	1
• Méthode d'accès aux données	Obligatoire	1
○ Connecteur client/server	Obligatoire	1
○ Framework (livecycle) pour les données	Obligatoire	1
○ PHP	Obligatoire	1
○ MySQL	Obligatoire	1
Développement		
• Page d'accueil		
○ Connexion	Obligatoire	1
○ Choix pour la navigation ou l'analyse	Obligatoire	1
○ Enregistrement de paramètres pour la prochaines connexion	A analyser	4
• Récupération de cas existants		
○ Création des nouvelles tables et organisation des données	Obligatoire	1
○ Groupager, hierarchiser, masquer les paramètres	Obligatoire	1
○ Paramètres de filtrage (composition de la requête sql)	Obligatoire	1
○ Sauvegarde de paramètres	Nice to have	3
○ Chargement de paramètres	Nice to have	3
○ Liste de cas récents	Nice to have	4

Développement		
• Saisie du nouveau cas		
○ Saisie des paramètres cliniques	Obligatoire	1
○ Upload d'image	Obligatoire	1
• Affichage des résultats (multi-cas)		
○ Résumé de cas à afficher sur une page	Obligatoire	1
○ Affichage de séries d'images DICOM	Obligatoire	1
• Présentation du cas		
○ Applet, lecteur DICOM	Obligatoire	1

Documentation		
• Situation de départ	Obligatoire	3
• Recherche	Obligatoire	3
• Outils	Obligatoire	3
• Framework	Obligatoire	3
• Connecteurs	Obligatoire	3
• Applications externes	Obligatoire	3
• Code	Obligatoire	3
• Conclusion	Obligatoire	3

Planification des tâches

Ensuite une planification dans le temps s'est faite avec un diagramme de Gantt.



ID	Task	Start	End
0	**talisman**	12/16/08	4/11/09
1	**Apprentissage et analyse**	12/16/08	1/10/09
2	DICOM	12/16/08	12/19/08
3	Format ROI	12/26/08	12/26/08
4	**Méthode d'affichage et d'appel du vie...**	12/18/08	12/19/08
5	Choix du lecteur	12/18/08	12/19/08
6	Frameworks du lecteur	12/19/08	12/19/08
7	**Flex Builder 3**	12/22/08	12/27/08
8	Stucturation des applications flex	12/22/08	12/27/08
9	ActionScript 3	12/28/08	1/2/09
10	AmfPhp	1/2/09	1/3/09
11	**Méthode d'accès aux données**	1/3/09	1/10/09
12	Frameworks	1/3/09	1/9/09
13	Connecteur client/serveur	1/9/09	1/10/09
14	Copie_Analyse existant,intégration, r...	1/10/09	1/16/09
15	**Développement**	1/1/09	4/10/09
16	**Page d'accueil**	1/1/09	1/9/09
17	Connexion	1/1/09	1/2/09
18	Choix pour la navigation ou l'analyse	1/2/09	1/2/09
19	Enregistrement de paramètres pour al...	1/5/09	1/9/09
20	**Récupération de cas existants**	1/16/09	2/21/09
21	Création des nouvelles tables et org...	1/16/09	1/18/09
22	Groupages, hierarchisation et masqua...	1/23/09	1/31/09
23	Paramètres de filtrage (composition ...	2/1/09	2/8/09
24	Sauvegarde de paramètres	2/13/09	2/13/09
25	Chargement de paramètres	2/14/09	2/15/09
26	Liste de cas récents	2/20/09	2/21/09
27	**Saisie du nouveau cas**	2/27/09	3/7/09
28	Saisie des paramètres cliniques	2/27/09	3/1/09
29	Upload d'image	3/6/09	3/7/09
30	**Affichage des résultats (multi-cas)**	3/8/09	3/21/09
31	Résumé de cas à afficher sur une page	3/8/09	3/21/09
32	Affichage de séries d'image DICOM	3/8/09	3/21/09
33	**Présentation du cas**	3/21/09	4/10/09
34	Applet, lecteur DICOM	3/21/09	4/10/09
35	**Documentation**	12/16/08	4/11/09
36	Situation de départ	12/30/08	1/3/09
37	Recherche	12/16/08	12/20/08
38	Frameworks	12/22/08	12/27/08
39	Applications Externes	12/24/08	12/26/08
40	Connecteurs	12/25/08	12/27/08
41	Outils	12/29/08	1/2/09
42	Code	1/1/09	3/28/09
43	Conclusion	4/10/09	4/11/09

Modification des tâches et nouvelles planifications
Plusieurs événements ont demandé des modifications de la planification.

La planification des tâches à chaque changement aurait demandé trop de temps et d'énergie pour être mise à jour et maintenue.

Cette constatation m'a poussé à utiliser un autre moyen pour la gestion mon projet. La gestion s'est faite par des listes de priorité avec mise à jour par itération.

Chaque semaine une liste de priorité était mise à jour et établie en fonction des projections faites par rapport aux tâches les plus importantes.

Modifications

Visionneuse 3D
La visionneuse étant la pièce la plus importante de l'application, son choix devait se faire au début du travail. Les premières recherches et essais avec des libraires graphiques ont mit à l'évidence que la création d'une visionneuse 3D avec les librairies à disposition et mon expérience dans le domaine n'allait pas apporter quelque chose de mieux que les outils à disposition fait par des spécialistes. Les contraintes du langage utilisé par la visionneuse ont été faites assez rapidement, la recherche de la visionneuse n'a pas pris beaucoup de temps, mais la livraison de source s'est fait assez tard puisque nous avons reçu les sources le 30 mars.

Format .seg
La livraison des sources a permis de découvrir un format très intéressant pour les régions d'intérêts et surtout un format très bien optimisé.

Cela a changé considérablement le projet puisque ce format sera pris comme référence pour le stockage des régions d'intérêts.

Visionneuse 2D et saisie des ROI
La nouvelle visionneuse inclut les outils de saisie de régions d'intérêts. La visionneuse 2D n'est donc plus comprise dans l'application.

Routine de calcul de similarité par images
En ce qui concerne la routine de calcul de similarité par images, par volumes de type tissus, son appel devait se faire dans un point de menu de l'application avec une gestion d'événements qui allait permettre de voir sa progression.

La routine peut être intégrée dans la nouvelle visionneuse 3D.

Communication visionneuse, Flex et base de données
Plusieurs modifications et changement de direction sont apparus dans les communications entre les différentes parties de l'application.

La communication entre Flex et Java à demandé beaucoup de recherches qui se buttées plusieurs fois à la sécurité des applications embarquées dans un navigateur web.

L’upload et le download de fichiers a également demandé plusieurs changements de direction par rapport à la sécurité et à la modification du type de données à transmettre. Dans la version finale, les données sont compressées avant la transmission en ce qui concerne les région d’intérêt.

Méthodes

Existant

Infrastucture

Serveur Apache/2.2.4 (Ubuntu)

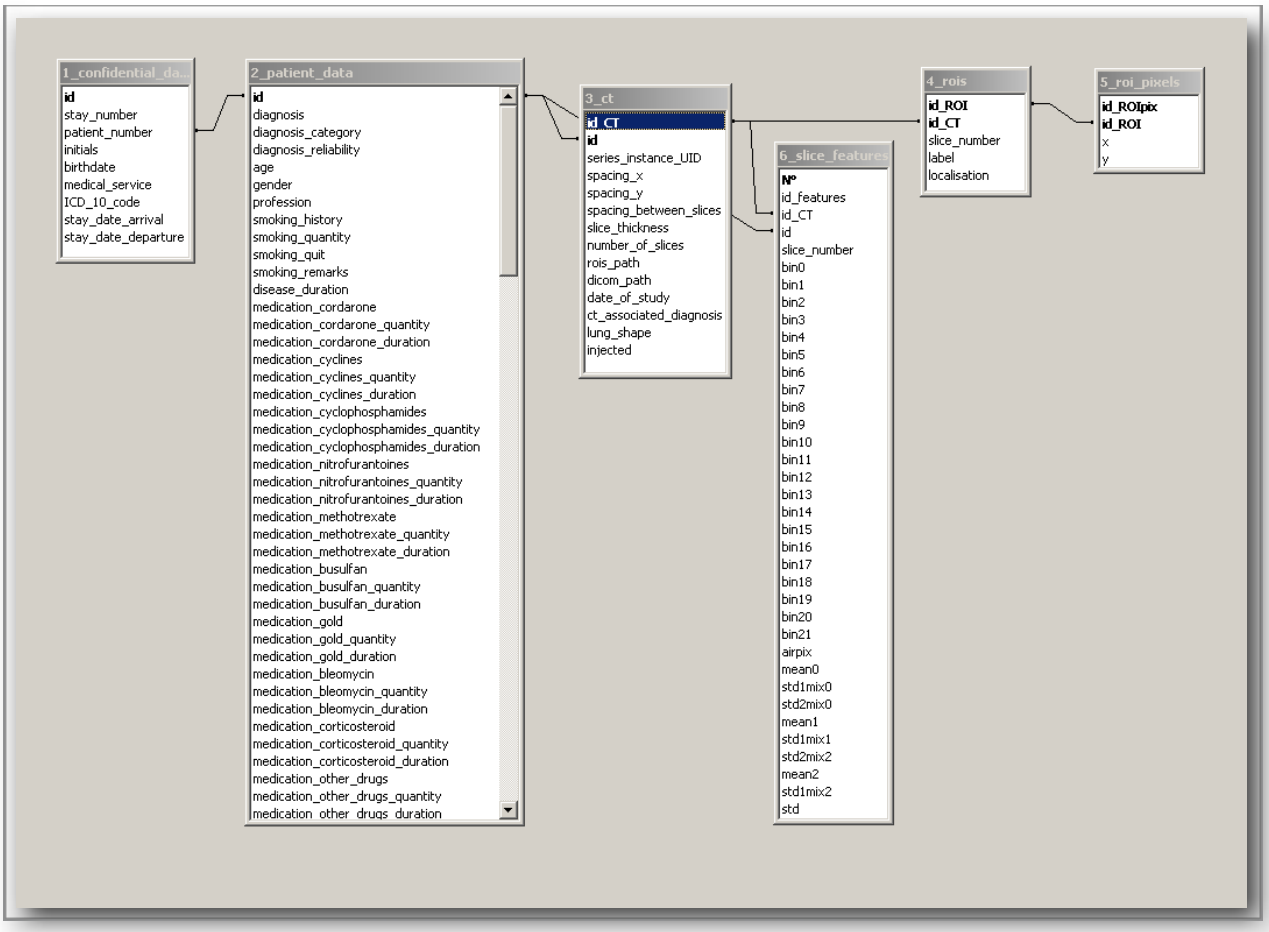
PHP :PHP 5.2.3-1 ubuntu6.4

MySql : mysql Version 14.12 Distrib 5.0.45, for pc-linux-gnu (i486) using readline 5.2

Données

Base de données

Schéma de la base de données MySql



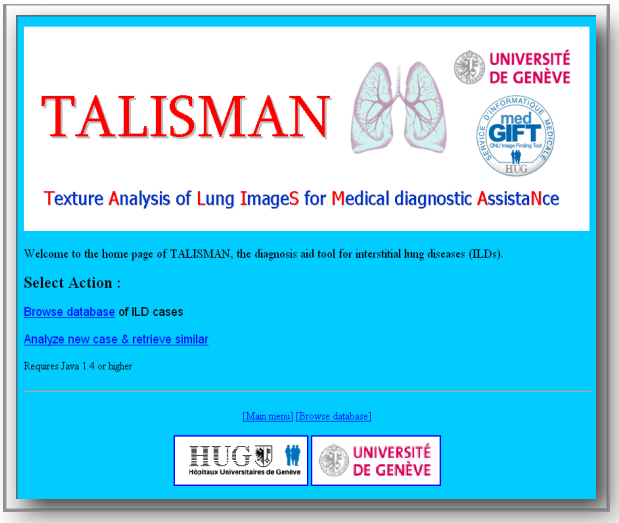
Les tables sont détaillées dans le cahier des charges.

Fichiers

Les régions d'intérêts sont enregistrées dans un fichier texte, la structure de ce fichier comporte tout d'abord des informations concernant la série avec Study, Series, SpacingX, SpacingY et SpacingZ.
Ensuite pour chaque région d'intérêt un label, une localisation, le numéro de slice et le nombre de point de contour de la région d'intérêt dans la slice, suivi par tous les points des coordonnées x et x du contour.

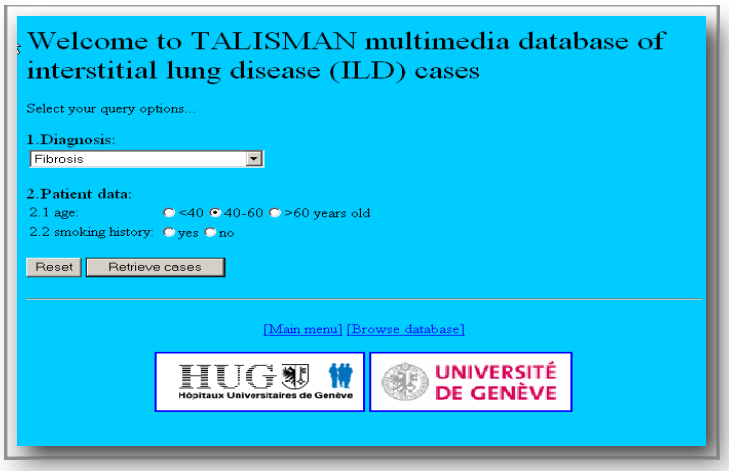
Application

L'application prototype permettant de consulter les informations des données cliniques a été créée avec des pages PHP et permet de faire une recherche sur les critères âge et diagnostic des patients.

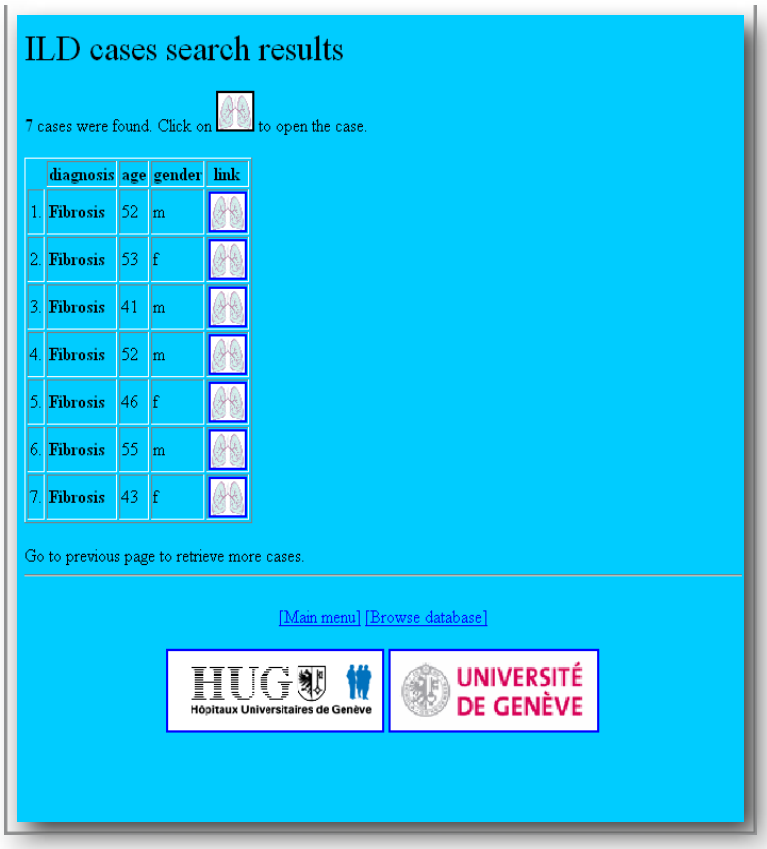


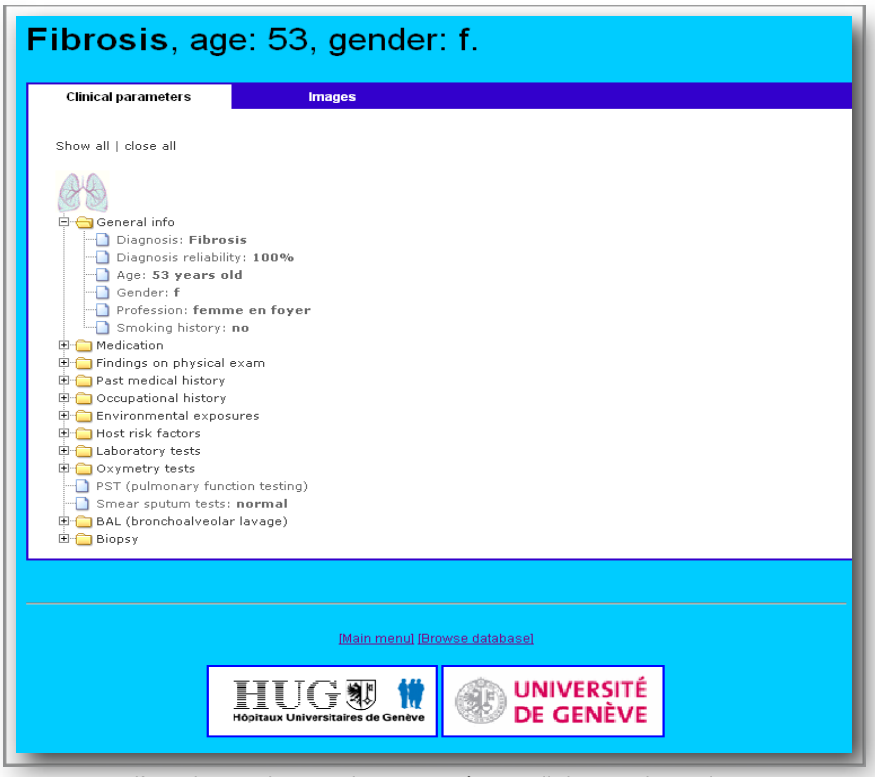
L'analyse d'un nouveau cas et la récupération des cas similaires ne sont pas implémentés.

L'option « Browse database » permet de filtrer les données cliniques par diagnostique, âge et l'historique de fumeur du patient.

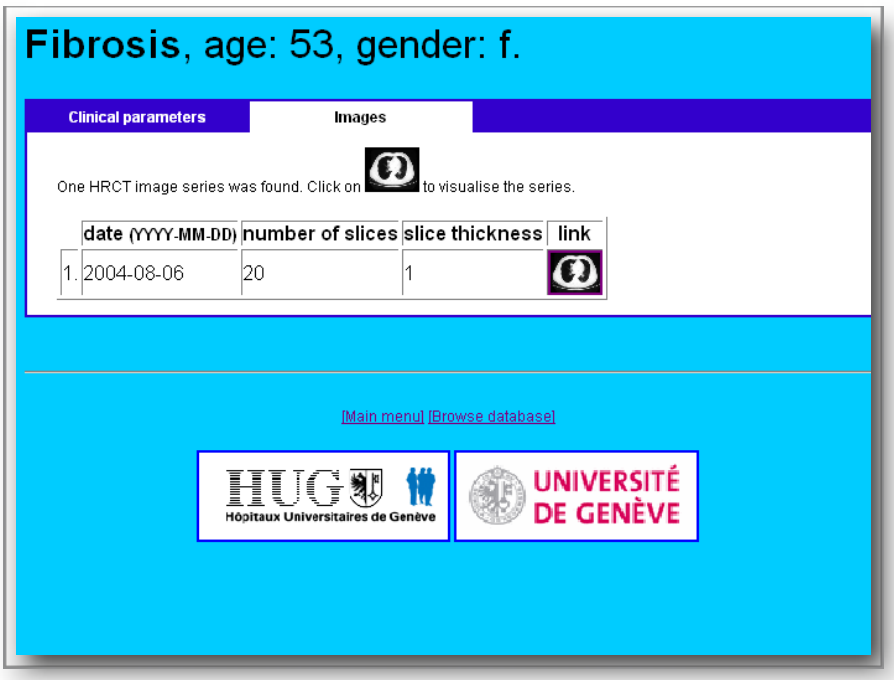


Une liste de patient est retournée.



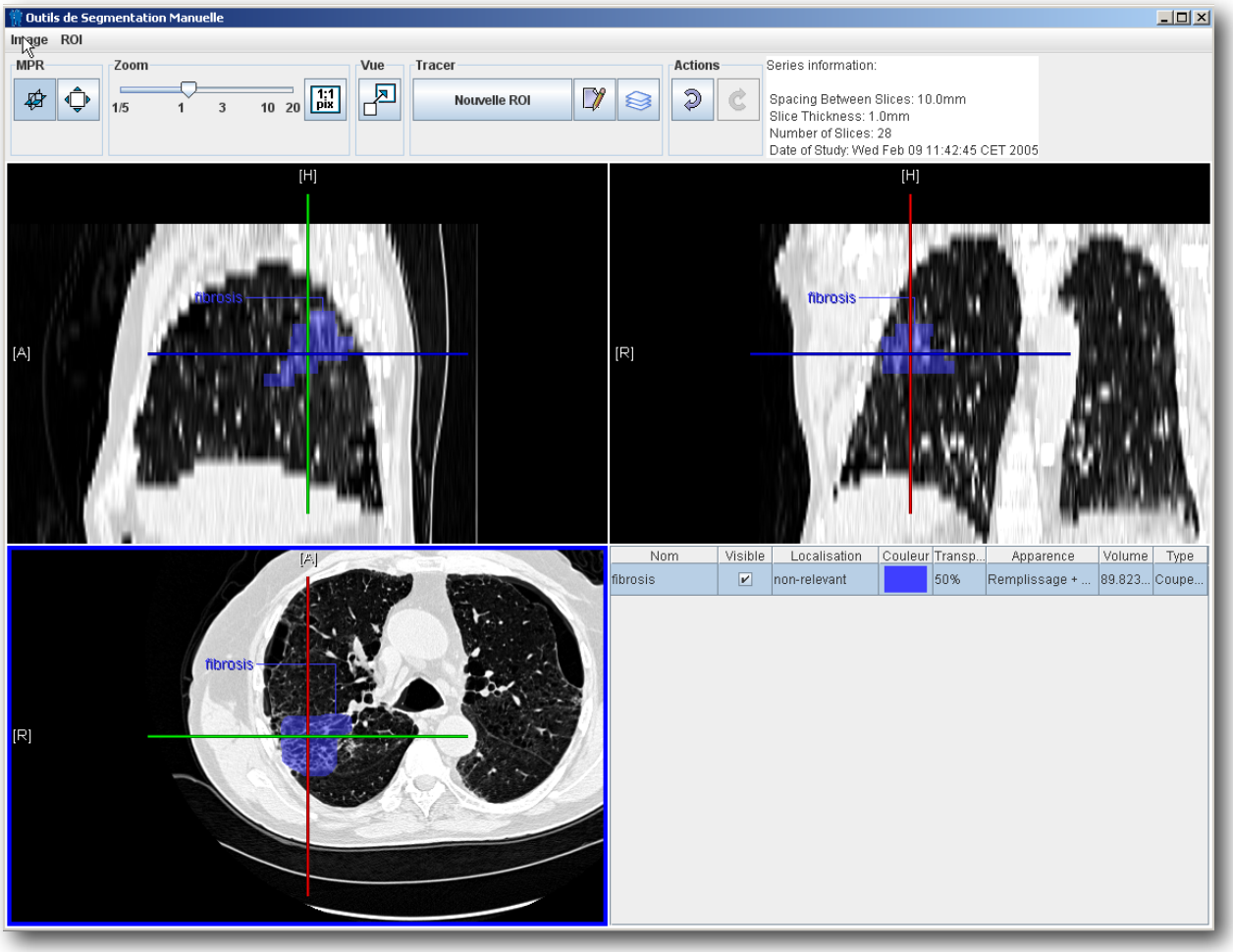


La sélection d'un cas permet l'affichage de tous les paramètres cliniques du patient.
L'onglet Image permet de visualiser la série d'images.



Le développement de cette interface s'est arrêté là. Le lien entre cette page et l'outil de visualisation des images n'est pas directement fait.

L'outil de visualisation de la série d'image DICOM comporte 3 vues générées par la série d'images. Ces vues vont permettre de visualiser, localiser et tracer des ROI.



Analyses et recherches

Généralités

Comment le faire le choix d'une solution RIA ?

AJAX et Applet-like

Dans les solutions proposées on distingue deux groupes bien distincts.
D'un coté les technologies AJAX qui basée sur le langage JavaScript et la communication HTTP asynchrone.
Et de l'autre les "applet-like" qui utilisent le navigateur comme hôte. Très ressemblantes à un client lourd qui utilisent peu ou pas du toutes les capacités du navigateur.

Une solution totale AJAX s'orientera donc vers une typologie client/serveur. Toute la présentation et la logique est hébergée par le navigateur, le serveur n'exécute que les points de service dont le client a besoin.
En hybridant la techonologie AJAX, une partie important de la logique est confiée au serveur. (AJAX avec framework web classique (ASP.NET, Struts, JSF, PHP...)

Les technologies "applet-like"

Venant des applet Java, aujourd'hui flash, Silverlight et JavaFX représentent cette catégorie.
Elles offrent des appels à des serveurs de plusieurs types, elles permettent le "binding" et utilisent un système déclaratif pour la construction des interfaces mxml, xaml json-like...

Critères de choix :
Les valeurs sûres pour le futur
L'interface utilisateur
La facilité de développement, en équipe, maintenance
Les performances robustesse et sécurité
Le portage

Les solutions AJAX pure ont été tout de suite écartées, parce qu'elles ne permettent pas un vrai développement en équipe, leur outillage est rudimentaire et ces solutions sont trop dépendantes des navigateurs.
Pour ce qui est des solutions AJAX hybride et applet like, dans le cadre de Talisman en projetant l'intégration d'une applet Java, l'idéal aurait été bien sûr une solution JSF (Java Server Faces).
Ayant déjà travaillé avec JSF et Java en client lourd, mon intérêt n'était pas très grand pour ce type de solution.
Les solutions ASP.NET avec extension AJAX, javaFX et Flex étaient les plus intéressantes.
Sans essayer de faire un choix technique, bien qu'il puisse est être argumenté par les contraintes matériel, mon choix s'est porté sur Flex d'Adobe. J'ai eu l'occasion de créer quelques petites applications avec cet outil . L'intérêt dans ce travail était de faire une découverte en profondeur cette solution.

Flex

C'est la solution de développement de Macromedia créée en 2004, reprise par Adobe qui est à l'origine de Flex.
Le modèle de programmation de Flex permet d'utiliser un système déclaratif pour les interfaces (MXML) et se base sur l'ActionScrip.

Flex produit des fichiers .swf lisible par les lecteurs flash (Adobe Flash player).

Avant la création de Flex, Macromedia proposait uniquement la plate-forme Flash qui permet de créer des animations flash, mais cette plate-forme est pratiquement inutilisable pour le développement d'application et d'interface utilisateur.
Les premières versions Flex de Macromedia se vendaient sous licence CPU la compilation des fichiers swf ne se faisaient uniquement au niveau serveur.
La version Flex 2.0 d'Adobe en octobre 2005 livrait tout d'abord un SDK gratuitement avec un compilateur en ligne qui permet de construire et déployer les applications et une version de Flex Builder sur la plateforme Eclipse open-souce.

En février 2008 Adobe livre le SDK Flex 3 sous licence Mozilla Public Licence et Adobe AIR qui permet de créer des clients lourds.

MXML
Créé et développé par Macromedia puis repris par Adobe Systems pour Flex, il étend XML et permet de décrir la présentation des interfaces utilisées dans une application Flex.

ActionScript 3.
L'ActionScript 3.0 est un langage orienté objet permettant de créer des applications lisibles dans un client flash (lecteur ou Adobe AIR). Sa syntaxe est très proche du Java ou du C#.

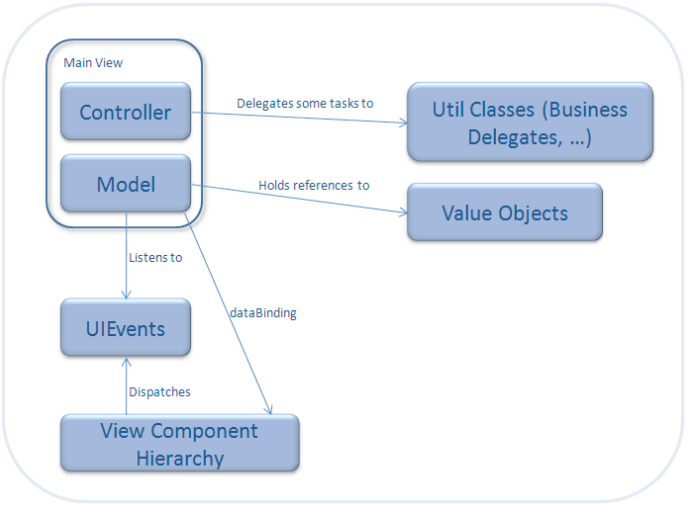
L'origine du langage est basée sur la 4ème version des spécifications du langage ECMAScript .

Les fonctionnalités à retenir d'ActionScript sont :
- le support complet des constructions orientées objet les plus courantes, classes, objets et interfaces
- le modèle d'exécution à thread unique
- la vérification du typage au moment de l'exécution, avec la vérification facultative des types à la compilation
- la création dynamique à l'exécution de fonctions constructeur et variables.
- la gestion et le traitement des exceptions
- le paquetage et les espaces de nommages
- les expressions régulières

Pourquoi utiliser un framework ?

Avec le développement Flex en utilisant l'actionscript, le mxml et en utilisant des tag script dans les mxml on peut très rapidement créer même avec ce type d'architecture un code difficilement maintenable et transmettable.

Le modèle Flex sans framework s'apparente au modèle suivant qui pour une très petite application est tout à fait maintenable et productif.



Dans la vie d'un projet on rencontre très rapidement des contraintes de temps et techniques qui vont nous obliger à créer des solutions temporaires tout en sachant que le temps manquera pour les retouches et la structuration du code.
La vision d'un projet change aussi durant le projet et on apprend à organiser et structurer notre code toujours mieux.
Mais le temps et l'énergie manque parfois pour restructurer et remanier ce qui à déjà été fait et surtout ce qui fonctionne.
De plus dans un développement en équipe le principal problème rencontré avec les nouveaux venus est la compréhension de la structure du projet et son fonctionnement.
C'est ici que le choix d'un framework prend son sens.

Un framework est une infrastructure logicielle qui va faciliter la conception de notre application par l'utilisation de bibliothèques qui vont "cadrer" le développement.
Il va permettre d'organiser le code, faciliter le développement, le debugging et la maintenance.

Le plus important est qu'un framework va permettre, s'il est respecté, de transmettre le code.
L'utilisation d'un framework connu et documenté permettra à un développeur de facilement s'intégrer dans une nouvelle équipe et cela même si ne connaît qu'un autre framework utilisé dans la nouvelle équipe.

Accès aux données

L'accès aux données avec Flex.

Il y a plusieurs méthodes disponibles actuellement pour accéder à des données depuis Flex actuellement :

Les WebServices

Ils permettent de s'interfacer avec un webservice au format WSDL via SOAP.

Il est tout à fait possible de créer un web service en PHP tournant sur linux. L'appel de web services est très facile et demande très peu d'investissement.

Dans le client Flex il suffit de définir la liste des WebServices en MXML avec WebService MXML.Ensuite wsGetXXX traitera les données récupérées du WebService et les stocke dans un objet Bindable

Coté client l'utilisation est simple mais coté serveur l'utilisation est moins simple et l'utilisation et l'intégration de framework spécifique est nécessaire.

Il est possible de typer les type primitifs d'Actionscript 3 et quelques types complexes mais pas de type personnalisés.

Les gros désavantages de cette méthode sont sa lourdeur du point de vue performance due au format XML et le peu de protection qu'elle permet.

HTTPService

L'HTTPService permet d'effectuer des requêtes HTTP sur une url et d'en récupérer la réponse.

On peut récupérer qui ne sont pas typées (typage possible avec JSON), son utilisation est identique à un WebService sauf que l'on a qu'une seule fonction par URL.

C'est encore plus rapide à mettre en place que le WebServices, mais encore moins fiable et pas sécurisée.

Il faut limiter son utilisation à de petites tâches.

RemoteObject

Il permet de faire transiter des objets du serveur au client. En général un serveur J2EE est utilisé.

Un objet est envoyé au serveur qui renvoie un objet client. L'avantage de cette méthode est qu'aucune transformation des données est effectuée avant les envois. Le protocole se charge de la sérialisation et dé sérialisation des objets. L'application sera donc facilement maintenable en utilisant un framework adapté qui permet de séparer les couches.

Des autres solutions proposées telle que Flex Data Services demandent obligatoirement des serveurs d'application et ont un prix élevé.

Dans le cadre de ce projet l'utilisation d'un serveur d'application est écartée vu que l'accès aux données est réduit, ne demande que très peu de modification et de maintenance.

Dans cette application le but est de consulter des données d'une manière simple de la manière d'un WebService mais sans les problèmes de performances lié à XML. Le tout en open source et si possible bien adapté aux ressources disponibles.

Dans les solutions alternatives pour la partie serveur AmfPhp remplit toutes ces attentes d'une part parce qu'il utilise le format amf propriétaire du lecteur flash et donc pas le XML, qu'il ne demande pas de transformation d'object avant l'envoi ou la réception et d'autre par parce qu'il est très facilement implémentable.

Format AMF

Le Format AMF "ActionScript Message Format" est un format binaire optimisé pour les échanges client serveur. Il a été développé à l'origine par Macromedia et il a été un élément essentiel de la technologie Flash Communication Server aujourd'hui Flash Media Server.

Les échanges optimisés permettent de réduire la bande passante de 10 fois par rapport aux services qui échangent des données complexes en XML ou SOAP.

Déployée sous la forme de composants gratuits, Flash Remoting ajoute à Flash la capacité d'appeler des méthodes d'un service distant de manière simplifiée, le tout au format AMF.

Le rapport poids performances de l'interprétation du flux AMF par le Flash player est plutôt intéressant d'une part par la compression du format et d'autre par parce qu'il est natif au Flash player.

Grâce au format AMF, le lecteur Flash se charge de sérialiser et désérialiser les données ActionScript nativement. Nous pouvons échanger des données complexes typées, sans se soucier de la manière dont cela est réalisé. A la transmission de données, le lecteur encode un paquet AMF binaire compressé contenant les données sérialisées, puis le transmet au script serveur par la méthode POST par le biais du protocole HTTP.

Lorsque le lecteur Flash transmet un paquet AMF à un script serveur, celui-ci n'est pas par défaut en mesure de le décoder. Des projets appelés passerelles remoting ont ainsi vu le jour, permettant à des langages serveurs tels PHP, Java ou C# de comprendre le format AMF. Chaque passerelle est ainsi liée à un langage serveur et se charge de convertir le flux AMF transmis en données compatibles. Afin de développer ces passerelles, le format AMF a du être piraté par la communauté afin de comprendre comment le décoder. Initialement non documenté, Adobe a finalement rendu public les spécifications du format AMF en décembre 2007. Toute personne souhaitant développer une passerelle pour un langage spécifique peut donc se baser sur ces spécifications fournies par Adobe.

Les différents projets amf actuels sont :

Nom	Langage	Open Source	Lien
AMFPHP	PHP	oui	http://www.amfphp.org
WebOrb	Java, .NET, .NET, Ruby, PHP Non	non	http://www.themidnightcoders.com/
RubyAMF	Ruby	oui	http://www.rubyamf.org/
AMF.NET	.NET	oui	http://amfnet.openmymind.net/
OpenAMF	Java	oui	http://www.rubyamf.org/
Fluorine	.NET	oui	http://fluorine.thesilentgroup.com/

Dans notre cas nous allons nous intéresser à AmfPhp

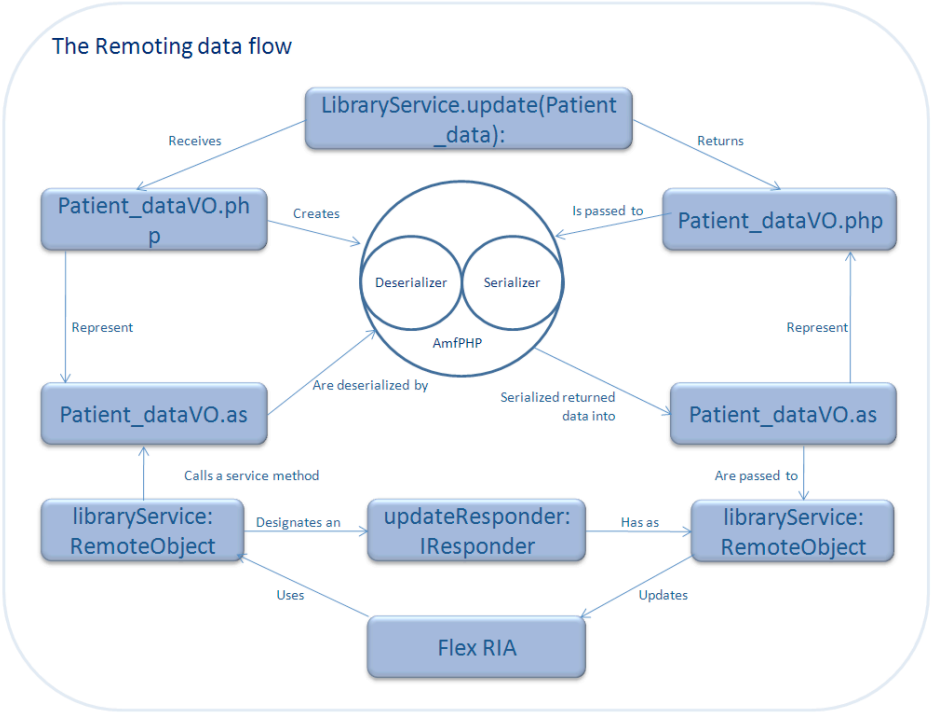
AMFPHP

AMFPHP est une librairie PHP open source qui permet de connecter Flex ou Flash à un backend en PHP, en utilisant le format AMF.

AMFPHP va automatiquement sérialiser les données échangées entre le client et le serveur. En envoyant par exemple un type booléen, il va détecter ce type et le convertir en type compatible côté serveur.

En envoyant un type Array vers votre service, AMFPHP convertira automatiquement ce tableau en tableau compatible pour le langage utilisé côté serveur.

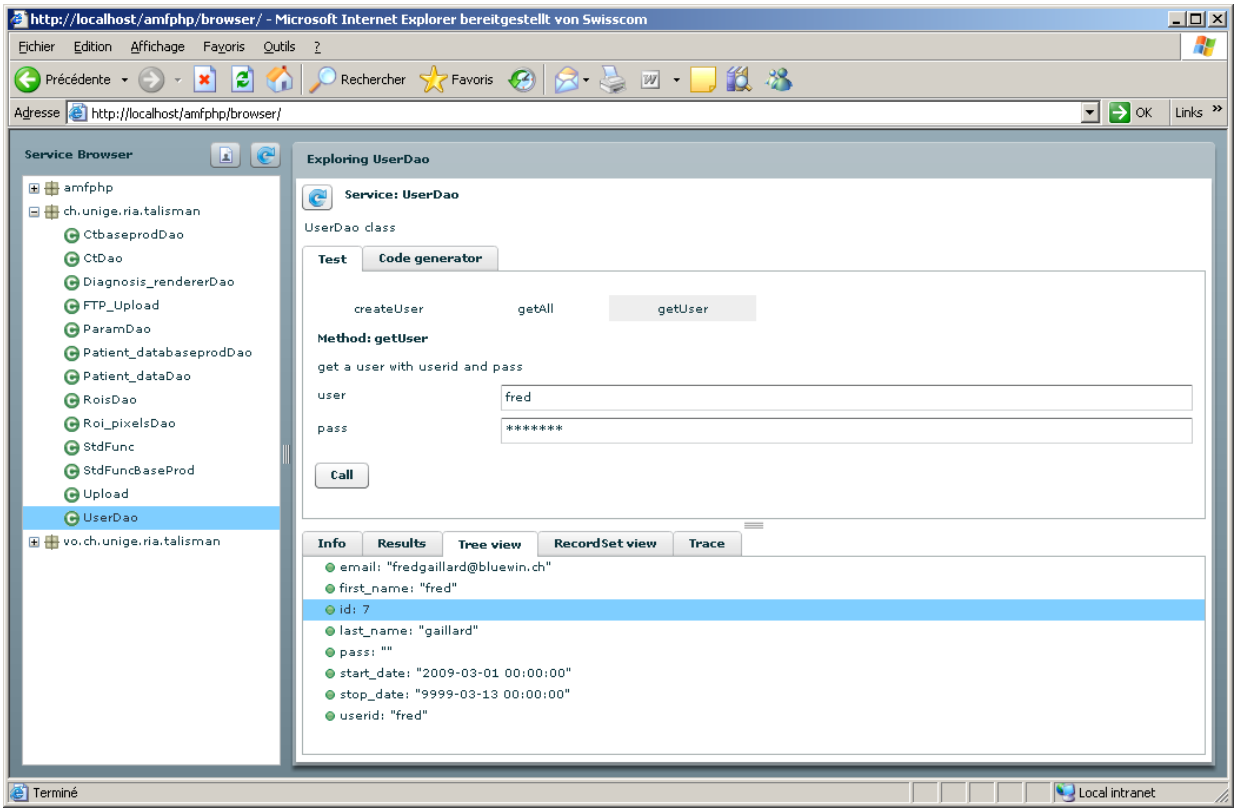
Type ActionScript	Type PHP	Notes	Sérialisation automatique
null	Null		Oui
Number	Double		Oui
Boolean	Boolean		Oui
String	string		Oui
Array	array		Oui
Object	array		Oui
XML	string		Oui
Date	float	Unix timestamp	Oui
ArrayCollection	Ressource MySQL	PHP à Flex	Oui



Du côté serveur il suffit d'une compatibilité PHP4 ou PHP5.
L'installation de la passerelle se fait en copiant simplement le projet la version 1.9 beta 2.

Service Browser

AmfPhp contient un outil très intéressant qui va permettre de tester directement ses services sans aucun code client.



Le fonctionnement d'AMFPHP se fait de la manière suivante, une classe PHP, le service est définie du coté serveur.

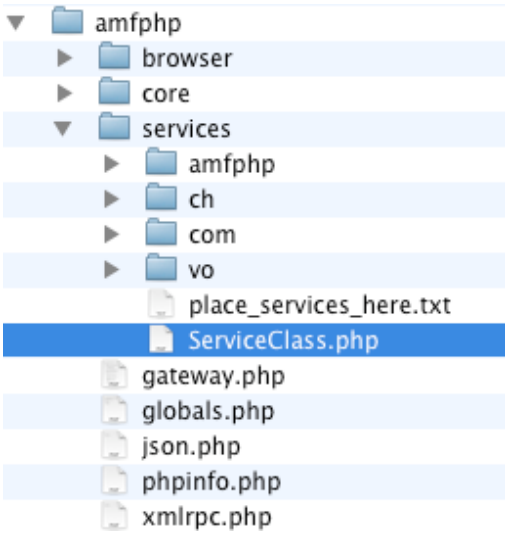
Elle contient différentes méthodes que notre application client Flex pourra appeler très simplement par la classe RemoteObject comme si elle appelle des méthodes ActionScript

Du coté serveur donc un service

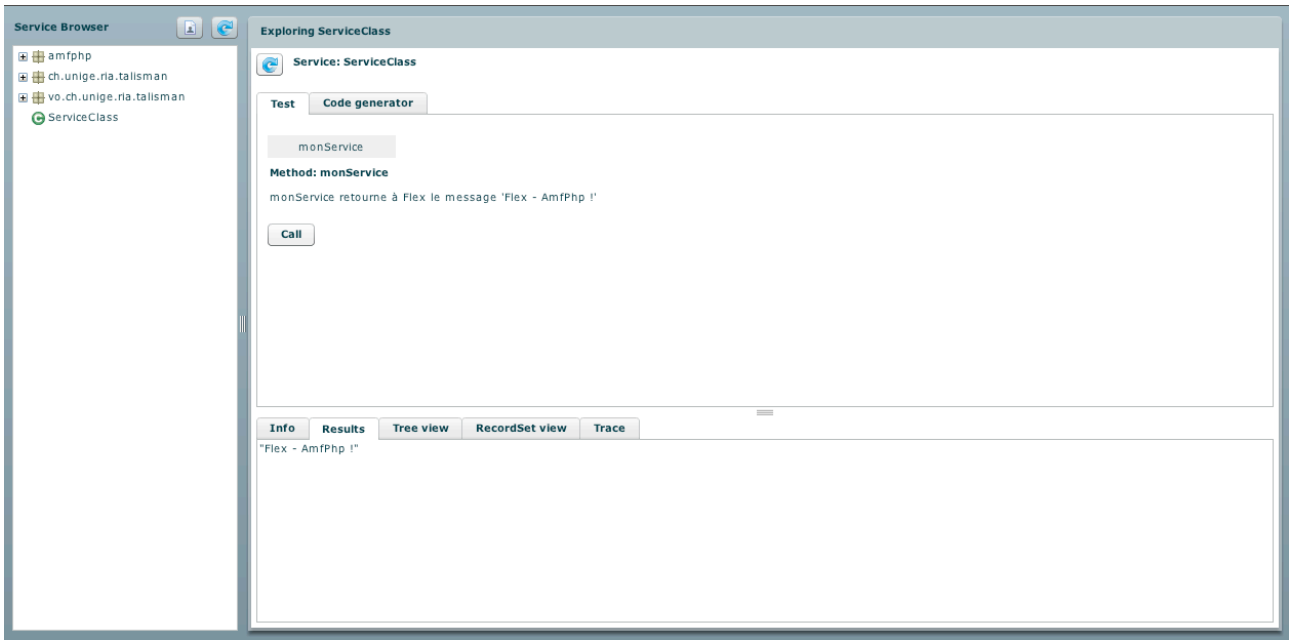
```
<?PHP

class ServiceClass
{
    function ServiceClass()
    {
    //constructeur
    }

    /**
    * monService
    * retourne à Flex le message 'Flex - AmfPhp !'
```

```
    */
    function monService()
    {
        return "Flex - amfPhp !";
    }
}
?>
```

Cette classe PHP sera ajoutée simplement dans le répertoire services d'amfphp



Le test va se faire très facilement avec le service browser en le chargeant. La classe ServiceClass est visible et on peut appeler la méthode.



Du coté Flex on crée une application

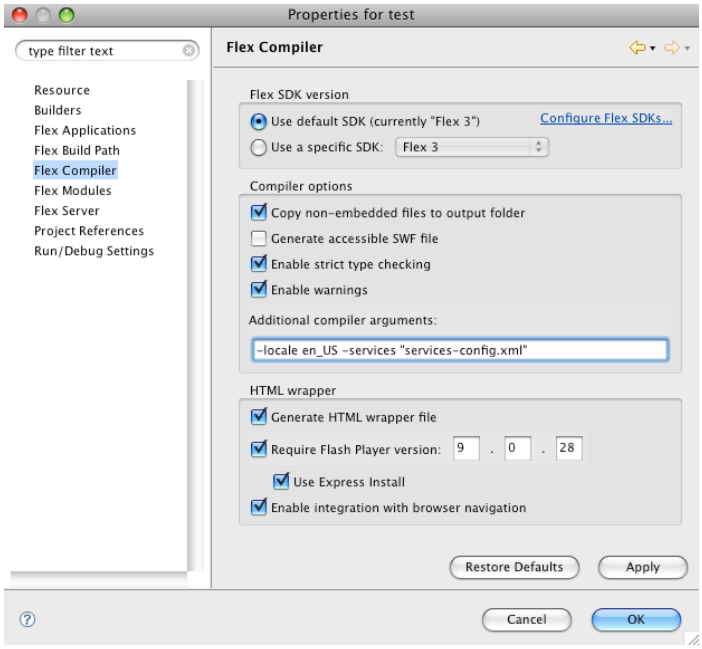
```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
               layout="absolute" >

<mx:RemoteObject id="services" destination="amfphp" source="ServiceClass">
               <mx:method name="monService"
              result="monService_resultHandler(event)"
              fault="faultHandler(event)"/>
</mx:RemoteObject>

<mx:Script>
<![CDATA[

    import mx.rpc.events.FaultEvent;
    import mx.rpc.events.ResultEvent;
    import mx.controls.Alert;

    /**
    * handleFault
    * Si erreur, on l'affiche
    */
    private function faultHandler(fault:FaultEvent):void
    {
        Alert.show(fault.fault.faultString, fault.fault.faultCode.toString());
    }

    /**
    * monService
    * Appel de la methode monService
    */
    public function monService() :void
    {
        services.getOperation('monService').send();
    }

    /**
    * monService_resultHandler
    * RŽsultat pour monService
    */
    private function monService_resultHandler(evt:ResultEvent) :void
    {
        infos.text = evt.result.toString();
    }
]]>
</mx:Script>

<mx:TextArea x="10" y="10" id="infos"/>
<mx:Button x="10" y="62" label="Mon Service" click="monService()"/>
</mx:Application>
```

Cela ne suffit pas, il faut encore écrire un fichier services-config.xml qui se trouvera à la racine de l'application et qui permettra de spécifier la «Channel» et la destination du service

```
<?xml version="1.0" encoding="UTF-8"?>
<services-config>
               <services>
```

```
                        <service id="amfphp-flashremoting-service"
class="Flex.messaging.services.RemotingService"
messageTypes="Flex.messaging.messages.RemotingMessage">
                                <destination id="amfphp">
                                        <channels>
                                                <channel ref="local-amfphp"/>
                                        </channels>
                                        <properties>
                                                <source>*</source>
                                        </properties>
                                </destination>
                        </service>
                </services>
                <channels>
                        <channel-definition id="local-amfphp" class="mx.messaging.channels.AMFChannel" >
                                <endpoint uri="http://localhost:80/amf/gateway.PHP"
class="Flex.messaging.endpoints.AMFEndpoint" />
                        </channel-definition>
                </channels>
</services-config>
```

Pour lier ce fichier à l'application il suffit de le donner en paramètre de compilation avec l'écran de propriété sur le projet



Ensuite l'appel du service renvoie la string du code PHP.

Accès aux ressources locales

Dans cette application il sera nécessaire de faire un accès aux ressources locales dans le cas du choix d'une série d'image d'un patient et pour trouver les similarités de celle-ci par rapport aux séries d'images stockées dans la base.
Les premières analyses du work flow de l'application demandaient une sélection du répertoire d'images par Flex, puis l'envoi de l'algorithme de similarité sur ces images sélectionnées.
Après quelques recherches on remarque très vite qu'il n'est pas possible de faire une sélection de répertoire sur le système de fichier local avec Flex. Un upload de fichier peut se faire sans problème comme avec le bon vieux <input name="userfile" type="file">

Mais dans ce cas, elle est inutile puisque l'on veut utiliser la puissance de calcul du client et donc sélectionner des images dans les systèmes de fichiers locaux sans dialoguer avec un server.
Une solution RIA ne dispose pas des mêmes capacités qu'un client lourd. La nature du déploiement d'une RIA impose une architecture « applet like » à base de machine virtuelle avec un framework plutôt contraint en taille. Un client lourd Java utilise un framework qui contient plus de 7000 classes.
C'est une des différences qui va permettre à un client lourd d'accéder aux ressources locale de la machine.
Adobe à palier au problème en introduisant sont produit Air qui permet d'exécuter une application Flex localement. Air ajoute toute une série de classes aux classes utilisées par Flex, ces classe vont permettre par exemple d'accéder aux fichiers du système local, d'utiliser une base locale, d'utiliser le drag and drop avec l'hote.
On peut faire une comparaison entre Flex, AIR et applet Java, client lourd Java Swing.
Mis à part que l'applet, elle, permet l'accès aux fichiers locaux.
La SandBox de Flash interdit ne permet que l'upload de fichier mais avec des restrictions.
Une des possibilité serait de faire un upload sur le serveur des fichiers à utiliser puis des les downloader avec Flex mais l'aller retour des fichiers n'est pas logique, coûte trop de temps et vis à vis du client l'enregistrement des ses images sur un serveur avant leur utilisation peut poser des problèmes de confidentialité.
De tout manière l'utilisation d'une fonction d'upload avec Flex pose le problème de la sélection de répertoire local qui n'est pas possible, aucun moyen permet de récupérer le chemin du répertoire local.
La seule alternative que nous avons est d'utiliser une applet pour permettre la sélection d'un répertoire local ou

se trouveront les images à utiliser avec la visionneuse.
L'applet inclura simplement la sélection d'un répertoire avec :

```
JFileChooser chooser = new JFileChooser();
chooser.setDialogTitle("Select target directory");
chooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
int returnVal = chooser.showOpenDialog(parentFrame);
if(returnVal == JFileChooser.APPROVE_OPTION) {
        myDirectory = chooser.getSelectedFile();
}
```

L'accès aux ressources locales demande encore une opération qui consiste à donner les autorisations à l'applet.
Une applet a des restrictions de sécurité, par exemple, elle ne peut pas lire ou écrire un ficher sur le disque local de la machine qui la lance, elle ne peut pas se connecter via socket à un autre host que le local host, etc…
En signant l'applet on va l'autoriser à accéder aux ressources locales et elle pourra se comporter comme un client lourd swing.
La signature d'une applet demande 2 étapes, tout d'abord la création d'un certificat avec l'outil keytool.exe qui se trouve dans les jdk de Java et ensuite, la signature des jars en utilisant le certificat.

Génération du certificat

```
Keytool –genkey –alias cleTalisman
```

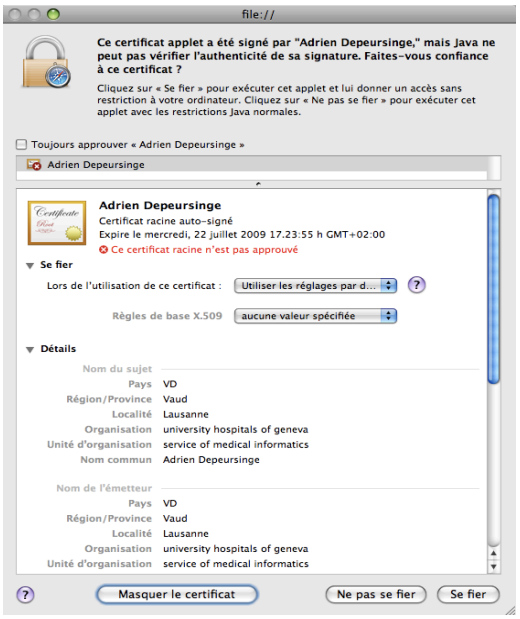
-genkey active la génération des clés publique et privée
-alias permet de donner un nom à la clé
A l'envoi de cette commande, différents paramètres sont demandés, un mode passe pour le keystore, le nom et prénom du signataire, le nom de l'organisation, la ville le pays et code du pays.

Signature des jars

```
Jarsigner –verbose AppletTalisman.jar cleTalisman
```

L'option –verbose va permettre de montrer les classes qui sont signées donner un avertissement sur la durée de la validité de cette signature

Ensuite au premier appel de l'applet le navigateur va afficher une boite de dialogue
Cela peut changer d'un navigateur à l'autre mais en principe les informations sur l'éditeur et des options d'exécution à l'utilisation de ce certificat sont possible.
Dans le cas de certain navigateur, un mot de passe administrateur va même être demandé pour l'utilisation du certificat.



Communication Java et AmfPhp

La communication entre l'Applet Jave et les bases de données MySql doit se faire dans l'Applet Java pour enregistrer les résultats des analyses sur les types de tissus.

Avant de chercher une solutions, nous pouvons déjà avec Java nous assurer qu'il n'y aura pas de problème pour trouver une solution avec JDBC par exemple pour se connecter à la base MySql.

Mais le plus logique serait de communiquer avec les services existant de la même manière que Flex à AmfPhp.

Adobe permet avec ses librairies BlazeDS de le faire très simplement

```
AMFConnection amfConnection = new AMFConnection();

String url = "http://medgift.unige.ch/talismanAmfServices/gateway.PHP";

...
    amfConnection.connect(url);
...
    Object result = amfConnection.call("ch.unige.ria.talisman.RoisDao.create", param1, param2, param3, ...);
...
amfConnection.close();
```

Très peu de documentation existe sur cette librairie et la page d'Adobe traitant du sujet est assez sommaire.

Communication Java et Flex

Flex et Java

La communication entre client Flex et applet Java

L'Application Flex devra communiquer avec du code Java contenu dans une applet. La communication devra se faire dans les deux sens à priori.

Une première analyse pour cibler et essayer de d'optimiser les recherches dans ce domaine permettra de d'imaginer des recherches pour la communication avec le "au pire" qui sera par exemple la communication avec le serveur de donnée pour interagir entre les clients, ou avec une base de données ou encore avec des fichiers locaux ou sur serveur dans un style de « message queue », jusqu'à l'idéal" qui serait un bridge complètement transparent entre Java et Flex.

Tout cela en assurant qu'une des ces solutions fonctionne absolument

Communication Flex et Java

La communication entre Flex et Java est utilisée couramment et la tendance actuelle est d'utiliser la richesse d'un client Flex pour un serveur Java.

La communication se fait de client à serveur.

Dans notre cas les programmes Java à exploiter doivent fonctionner sur le client pour exploiter les performances des pc.

En faisant des recherches on remarque très vite que l'intégration d'un client Java(applet) dans un client Flex n'est pas courant.

Une première solution est de partager la page qui encapsule le client Flex avec une applet Java. La communication et le passage de paramètres se feront par javascript ou le FAB Flex ajax bridge.

External Interface

Une solution pour communiquer de Flex à d'autres application est d'utiliser l'ExternalInterface qui permet d'exécuter des méthodes JavaScrip depuis Flex ou des méthodes Flex depuis JavaSript tout en passant des arguments d'un coté à l'autre.

Une fonction JavaScript dans la page hébergeant Flex

```
<script>
function transfertObject(obj) {
}
</script>
```

et dans le code Flex

```
...
if (ExternalInterface.available){
        ExternalInterface.call("transfertObject", objtToTransfert);
}
...
```

Pour le passage inverse

```
function getFlexApp(flexAppName){
        if (navigator.flexAppName.indexOf("Microsoft") != -1){
                return window.[flexAppName];
        }else{
                retrun document[flexAppName];
        }
}

function callFlex() {
        var strParam = "Hello!";
        getMyApp("flexAppName").flexFunction(strParam);

<div id="fromJS">
<button onclick="callFlex();">click callFlex </button>
</div>
```

Et au niveau de Flex on va utiliser la méthode addCallback() qui va permettre d'enregistrer les méthodes exposées par Flex et accessible depuis le Javascript

```
...
//A l'initialisation
ExternalInterface.addCallback("flexFunction", communicationFunction);
...
public function communicationFunction(strParam:String):String {
        //
}
```

L'ExternalInterface est tout à fait utilisable pour l'appel d'un nombre limité de fonction, mais s'il faut communiquer et mettre à disposition beaucoup de méthodes, cela devient complexe à coder et à maintenir.
Une option existe depuis peu dans Flex builder un bridge Ajax, le Flex Ajax Bridge.

FAB

Le FAB permet de manipuler des objets Flex AS depuis JavaScript sans coder aucune ligne.
Après la génération du script permettant de faire le pont entre JavaScript et Flex, il suffira d'instancier le bridge dans le code Flex.

```
<fab :FABridge xmlns:fab="bridge.*" />

<mx:TextInput id="textInputId"/>
```

Ensuite l'utilisation dans le JavaScript se fera très simplement par la classe de script générée par Flex

```
<script>
function callFlex(){
		var flexApplication = FABridge.flash.root();
		var flexTextInputId = flexApplication.getFlexTextInputId();

}
</script>
```

Ensuite c'est la communication entre une applet Java et JavaScript qui va nous intéresser.
Tout d'abord les communications entre les différentes applications ne vont pas nécessiter des transferts de type complex, l'utilisation des types string,int ou string uniquement sera tout à fait suffisante.

De Javascript à Java on peut facilement appeler une méthode avec :

```
<applet code="Applet.class" name="applet"></applet>
...
<input type=button value="Call JavaScript"
	onclick="if (applet) {applet.callJavaScript();}">
...
```

Dans l'applet une méthode publique doit être créée :

```
public void callJavaScript() {
		...

}
```

Et dans le sens inverse :

```
...
<applet mayscript code="callJavaScript.class" name=applet></applet>
...
```

Et du côté Java la classe devra utiliser les méthodes de la classe JSObject :

```
JSObject win = JSObject.getWindow(this);
win.eval(javaSriptFuncion);
```

Cette solution assez simple permettra la communication mais demande de faire la compatibilité avec les navigateurs, il faudra inclure les classes JSObject spécifique à Internet Explorer

Merapi

Merapi une toute nouvelle interface en version beta qui a pour but de faire un pont entre Java et Flex fait ce travail.

Son principe est assez simple : il s'agit de mettre en place une sorte de passerelle entre les applications Flex/AIR et Java, afin d'utiliser la puissance et les possibilités bien plus nombreuses qu'offre Java. Ainsi, on peut envisager de lancer des applis externes

En jetant un œil sur les librairies utilisées par Merapi et son mode de fonctionnement, on comprend très vite qu'elle se base sur une communication avec des sockets.

Un rapide test en local montre la simplicité de cette interface qui parait idéale.

Coté Flex, on instancie le bridge

```
<merapi:BridgeInstance id="bridge" result="handleResult(event)"/>
```

Et on envoie un message

```
var message:Message=new Message();
message.data=textId.text;
message.type=typeId.text;
Bridge.instance.sendMessage(message);
…
<mx:TextInput id="textId" x="35" y="89"/>
```

Le type de message va permettre de catégoriser les messages et spécifier le type les objets envoyés bien pratique dans les cas d'envoi de collection d'objet (Array).

La réception se fera avec la méthode définie dans le tag result de l'instanciation du Bridge

```
private function handleResult(event:ResultEvent):void {
            var message:IMessage=event.result as IMessage;
            receiveId.text=message.data.toString();
}

//Pour l'envoi d'un message en Java, il faut tout d'abord instancier le bridge avec
Bridge bridge = Bridge.getInstance() ;
Message message = new Message() ;
Message.setData("Message from Java");
Bridge.sendMessage(message);
//Et pour la réception dans Java
Bridge.getInstance().registerMessageHandler(' Reply ', messageHandlerInstance) ;

Public void handleMessage(IMessage message){
            System.out.println(message.getData()) ;
}
```

Les premiers essais en local très motivant se sont conclus malheureusement au passage de Flex en localhost par des erreurs de connexions et l'impossibilité de traverser la sécurité du Flash player.

La communication se faisant par socket, on peut imaginer que l'interface permet la communication entre un client Flex et une applet, mais le Flash player ne le permet pas et en utilisant les socket en réseau on remarque que Flex et le lecteur flash ne permettent pas de créer un serveur de socket mais seulement d'être le client d'un

serveur de socket.

La dernière solution à envisager est la communication avec des sockets, plus précisément avec des xmlsocket, l'applet sera le serveur et la partie Flex le client.
Un premier test montre que la communication se fait assez simplement mais tout de suite en déplaçant le client Flex sur un serveur, la sécurité pose problème.
La documentation d'Adobe explique que la sécurité doit être gérée avec un fichier de configuration placé sur le serveur hébergeant la partie Flex.
Plusieurs tests et essais ont été faits sur divers flash player mais sans résultats.
Après plusieurs recherches d'autres solutions ont étés découvertes, elles proposent toutes de faire en sorte que si un message de type <policy-file-request/> est reçu par le serveur, il faut lui retourner ce fichier cross-domain.xml.
Un solution possible est de créer un deuxième serveur écoutant le port 843 et renvoyant le fichier cross-domain comme une string et terminé par un '\0' à chaque fois qu'il reçoit un <policy-file-request/>.

```
server.sendAll(
    "<?xml version=\"1.0\" encoding=\"UTF-8\"?>" +
    "<cross-domain-policy>"+
    "<allow-access-from domain=\"*\" to-ports=\"*\" secure=\"false\" />"+
    "<site-control permitted-cross-domain-policies=\"master-only\" />"+
    "</cross-domain-policy>",
    "" + charCur[0]
);
```

Cette solution pose problème malheureusement à la réception des messages qui, parfois sont tronqués pour une raison inconnue pour l'instant. Le temps manquant pour résoudre ce problème la solution utilisée finalement s'inspire de serveur de jeux ou de chat.

Du coté Flex

```
var connexion:XMLSocket = new XMLSocket();
connexion.connect("http://localhost", 8080);
```

L'écoute des événements se fait en ajoutant des écouteurs sur la connexion

```
connexion.addEventListener ( Event.CONNECT, connexionReussie );
connexion.addEventListener ( Event.CLOSE, fermetureConnexion );
connexion.addEventListener ( DataEvent.DATA, receptionDonnees );
connexion.addEventListener ( IOErrorEvent.IO_ERROR, erreurIO );
connexion.addEventListener ( SecurityErrorEvent.SECURITY_ERROR, erreurSecurite );
```

L'envoi de message se fera en appelant la méthode send sur la connexion

```
function envois_donne( message : String ):void{
   connexion.send (message) ;
}
}

function connexionReussie ( pEvt:Event ):void
{
   //affichage "connexion réussie";
}
```

Du coté Applet Java qui fait office de server de Socket nous retrouvons

```
public class Server {
...
   try
   {
      Integer port = new Integer(17500);
      ServerSocket ss = new ServerSocket(port.intValue());

      printWelcome(port);

      while (true)
      {
      new ClientThread(ss.accept(),server);
      }
   }
   catch (Exception e) {
      System.out.println("Une erreur c'est produite lors de la creation du server..");
   }
   }



public class ClientThread implements Runnable{
   ...

         if(message.equals("<policy-file-request/>")){
         System.out.println("tentative de header");
         server.sendAll("<?xml version=\"1.0\"?><cross-domain-policy><allow-access-from domain='*' to-ports='17500'
/></cross-domain-policy>",""+charCur[0]);
         }
         System.out.println("msg : "+message);
```

```
if(charCur[0]=='\u0000')
server.sendAll(message,""+charCur[0]);
else server.sendAll(message,"");
message = "";
```

Générateur

Générateur Cairngorm
Plusieurs outils sont disponibles pour générer des structures et du code pour Cairngorm.
Cairngen va permettre sous la forme d'une tâche ANT et d'un fichier de properties de créer l'arborescence pour cairngorm. Il s'intègre à Flex Builder ou Eclipse.
CairngormCreator est un outil de génération en ligne qui en plus de la création de l'arborescence et des classes cairngorm crée une interface graphique.

FCG : a Flex Code Generator
FCG est une application AIR qui lit les sources Java et PHP des objets coté serveur et crée les fichier coté Flex en plus des classe cairngorm.

crazedCoders PHP 5.5.2/Cairngorm 2_2_1/Flex 2.01 Code Generator fait une génération en ligne en se connectant à notre base et génère la structure cairngorm ainsi que les scripts serveur.

IDE Factory
Ce plugin que l'on peut intégrer dans Flex permet de générer la structure par exemple d'un nouvelle commande, d'un event. En fait le tool crée des classes en étendant ou implémentant les bonnes classes de cairngorm

Le but de ces outils est de supprimer les parties redondantes, aucun n'est parfait, pour ma part, j'ai essayé d'utiliser FCG qui n'acceptait pas tous les types de données utilisés dans l'application, ce qui me demandait des retouches à chaque nouvelle génération.
Ensuite j'ai essayé crazedCoders PHP 5.5.2/Cairngorm 2_2_1/Flex 2.01 Code Generator fait en PHP qui m'a posé de problèmes avec les noms des tables contenant des caractères «_ ». J'ai modifié les sources et j'ai remarqué que les setters dans les classes VO n'était pas fait pour certains type de données mal gérés…
De plus aucun générateur prétendant gérer les contraintes de clés étrangères ne fonctionne correctement, et même dans certains cas, par exemple avec le crazedCoders PHP 5.5.2/Cairngorm 2_2_1/Flex 2.01 Code Generator, une table contenant une clé étrangère va générer une classe qui contiendra dans tous les types de relation une collection d'instance de classe concernant la clé étrangère.
Tout cela pour montrer que la génération est peut-être utile au début d'un projet pour faire des choix de structure et voir ce que nous proposent les générateurs, sinon, l'outil parfait n'existe pas et le temps passé ou perdu à le configurer, debugger et l'adapter à nos contraintes logiciel et matériel est trop important dans la majorité des cas.
Les cotés positifs, sont l'apprentissage du framework par la génération d'application d'essai.

Dans Talisman j'ai essayé de faire en sorte que l'on puisse faire des générations pour les opérations standards CRUD (create,read, update and delete).
Cela demande une séparation des models locators pour permettre des nouvelles générations, mais aussi plusieurs opérations comme la suppression des contraintes sur les table pour une génération sans erreur, le contrôle pour savoir si tous les types de données sont bien supportés, la modification des packages, le formatage du code...
Pour de petites modifications, une modification manuelle sera plus sûres et moins coûteuse.

Securité avec AmfPhP

La sécurité et AmfPhp
Que peut-on prendre comme mesure pour rendre AmfPhp plus sûr.
Tout d'abord il faut supprimer ou interdire l'accès au Service Browser pour éviter que l'on puisse connaître tous les services à dispositions et surtout pour éviter l'utilisation des services.
Le service DiscoveryService.PHP qui permet au Service Browser de trouver tous les services à disposition, devrait être supprimé ou être sécurisé.
Ensuite à la mise en Production la propriété PRODUCTION_SERVER du fichier gateway.PHP qui se trouve dans le dossier racine d'AMFPHP doit être mise à true. Cela va permettre de désactiver à distance le traçage et le débogage des entêtes.
define ("PRODUCTION_SERVER", true);
La version 1.9 d'AmfPhp offer une nouvelle fonctionnalité qui permettra d'authentifier le client appelant pour lui donner l'autorisation d'utiliser le service.
Il suffit de créer une fonction « beforeFilter » dans les classes service fonction publique beforeFilter ($ function_called.)
A chaque appel de la fonction de service, cette fonction sera appelée avant chaque appel de fonction d'un service, si la méthode beforeFilter retourne vrai, le service pourra être appelé, sinon, une erreur sera retournée.
C'est à l'intérieur de beforeFilter qu'une authentification sera faite.
Dans le fichier Authenticate.PHP se trouvant dans \core\shared\util la méthode de connexion.

```
function login($name, $roles) {
if(!session_id())
{
session_start();
}
$_SESSION['amfphp_username'] = $name;
$_SESSION['amfphp_roles'] = $roles;
}
```

Dans la fonction beforeFilter il suffit ensuite d'appeler une variable créée auparavant dans les classes de service qui contient les rôles qui ont les droits d'appeler les services de cette classe.

```
public function beforeFilter($function_called)
{
$memberName = $function_called."Roles";
return (@$this->$memberName) ? Authenticate::isUserInRole($this->$memberName) : true;
}
```

La méthode beforeFilter vérifiera si tout d'abord la fonction xxx. « Roles » existe et si oui, l'utilisateur appelant devra faire partie des ces rôles.
Si la fonction xxx. « Roles » n'existe pas, l'authentification ne sera pas faite.

Depuis AmfPhp tous les services exposés sont des fichiers PHP, la configuration du serveur devra suivre bien sûr les directives de sécurité de PHP pour assurer une sécurité optimale.
Ensuite un tunnel ssl pourrait être utilisé pour parfaire la sécurité.

Choix des outils et solutions

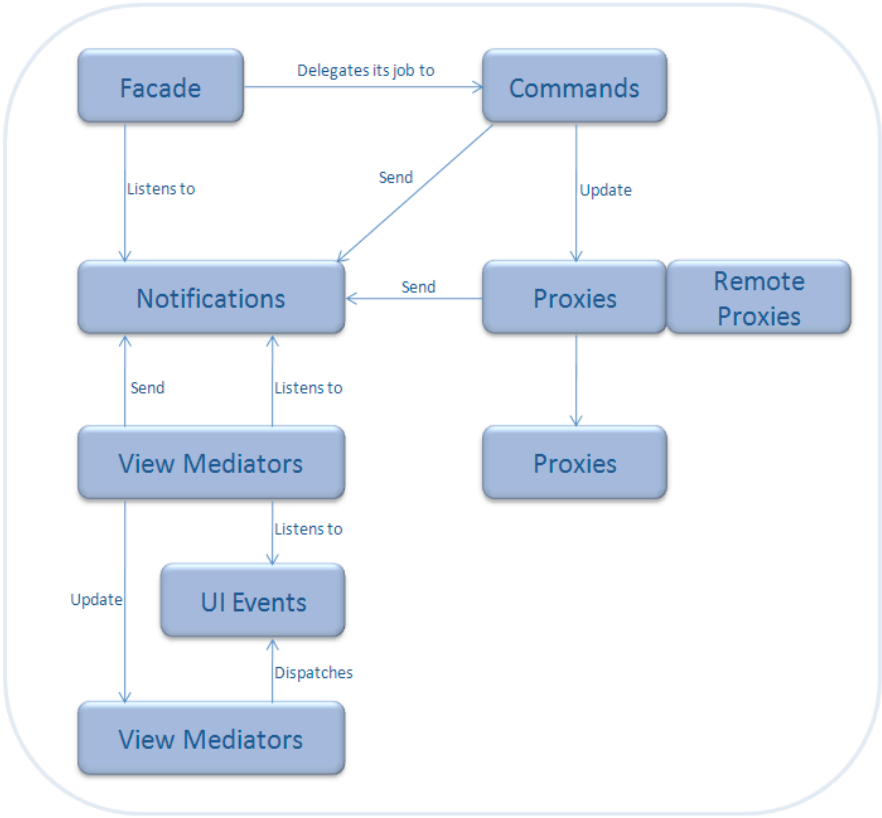
Choix d'un framework.

Les framework sur le marché pour Flex AS3 sont Cairngorm, PureMVC, LowRa, ARP, MVCS, Flest, Model-Glue: Flex, ServerBox Foundry, Guasax, and Slide.

Les plus recommandés, mûrs et documenté sont actuellement Cairngorm, Pure MVC.

PureMVC

Ce framework n'a pas été conçu pour Flex. La philosophie MVC est donc respectée à l'extrême. Un projet PureMVC découpe votre code en plusieurs packages et en de nombreuses classes. PureMVC profite aussi d'une large communauté de plus en plus active.
Il est aussi très adapté au développement par équipes.
La faiblesse de PureMVC est qu'il ne tire pas partie des forces du framework Flex (il ne profite pas par exemple des forces du langage MXML).
D'autre part PureMVC est jugé plus difficile à appréhender que d'autres frameworks, la courbe d'apprentissage est plus étendue.



Cairngorm (Cairngorm microarchitecture)
C'est le framework MVC historique et Flex et le plus connu.
Il reprend les codes du monde Java et se concentre sur trois domaines: la gestion des actions de l'utilisateur, les interactions avec le serveur et la logique métier tout en gérant le contexte de l'utilisateur.
Les classes principales sont le ModelLocator, le ServiceLocator, la logique métier et le FrontController. La force principale de Cairngorm est sa réputation dans la communauté, le fait qu'il soit un projet Open Source Adobe supporté et avec une communauté active.
Il est léger, il est composé d'une vingtaines de classes et soutenu par Adobe de manière officielle.

Il est très adapté au développement en équipe.

Sa faiblesse se situe au niveau du nombre élevé de classes à coder (chaque événement est relié à une commande, donc une classe à écrire par événement). Une autre limitation est le fait que chaque événement doit avoir sa propre classe de commande, donc nous sommes limités à un répondeur par événement.

L'idéal serait de démarrer des petits projets avec Cairngorm et ensuite avec un peu d'expérience de passer à PureMVC si le projet grossit.

Dans le cadre de ce travail c'est Cairngorm qui est donc choisit

Cairngorm propose une implémentation de différents patterns :

Le pattern front controller, singleton, command, modèle vue contrôleur (MVC) et le pattern listener

Ces patterns vont permettre de résoudre les problèmes de récupération d'événements utilisateur, d'encapsulation de la logique métier et des interactions avec les services et de la gestion et représentation des données côté client.

L'architecture proposée par Cairngorm est composée par différent package :

VO
Object Value ou DTO (Object Transfert Value)
Ils permettent de définir la structure et représenter une entité logique de notre application.
Ils vont permettre d'effectuer l'interface entre les données du serveur et les vues Flex sur
lesquelles seront affichées les informations.
Le pattern Value Object définit de manière sémantique la structure des données.

Model
Le ModelLocator
Cette interface basée sur le pattern singleton, centralise les accès aux structures de données et permet d'assurer l'unicité de l'instance d'un objet et son accès par les vues.
Le ModelLocator de Cairngorm encapsuler les données du client dans une structure et d'assurer aussi la séparation entre les données et les vues.
Les données sont « Bindable » cela offre l'avantage de la répercussion de la mise à jour automatique les vues sans. *** à revenir...

Vue (view)
Représente la couche de présentation des données pour l'utilisateur tout en contrôlant les saisie et en déclenchant les événements de l'utilisateur à la couche Event

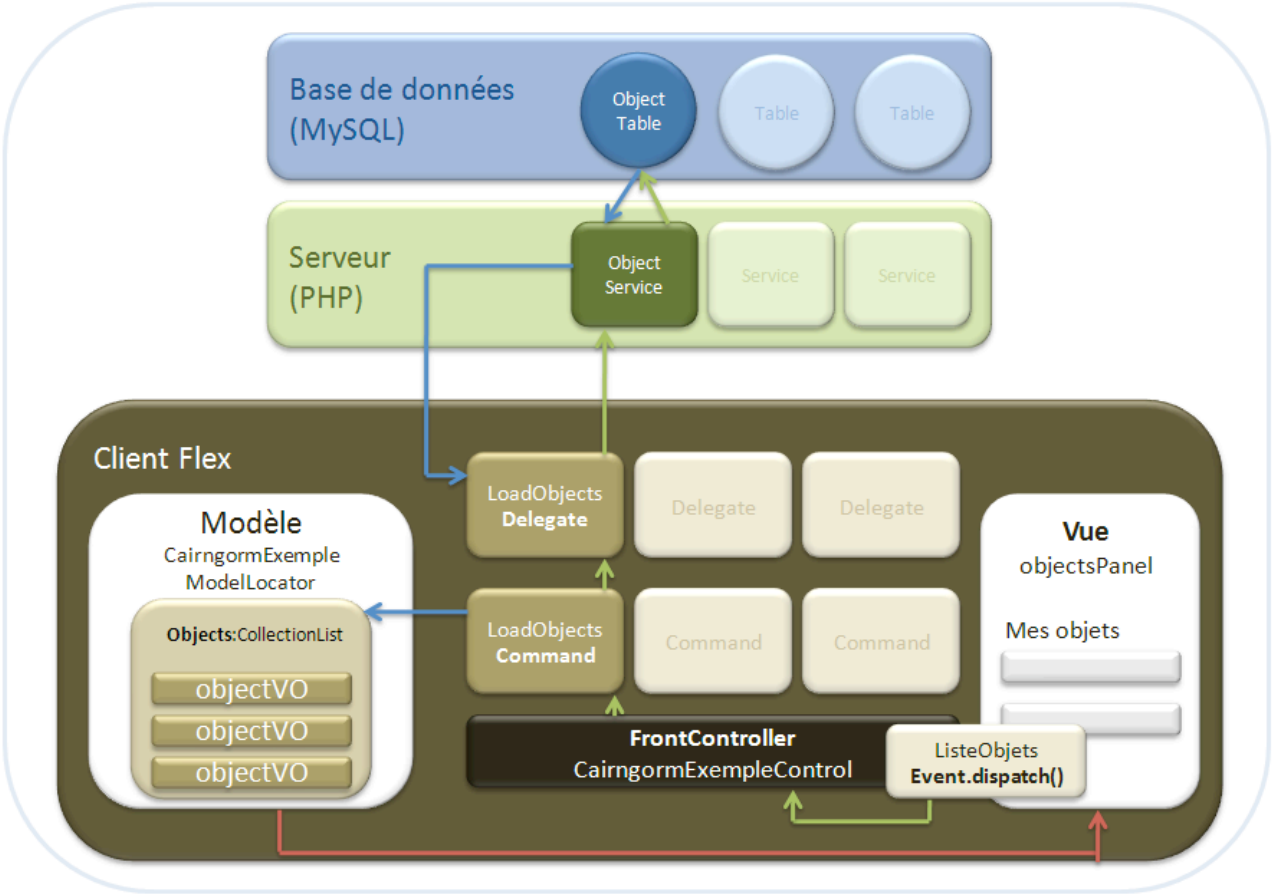
Event
Ici se gère les événements et le transfert des données entre la présentation (View) et les commandes

Control
Basé sur le pattern FrontController. Cette couche est une composition entre les patterns MVC et Commande.
Il associe un événement à un type de traitement en encapsulant les traitements dans des commandes.
On peut donc identifier tous les événements de notre application par les classes contenue dans ce package.

Command
Une commande doit implémenter la fonction execute(event : CairngormEvent) : void appelée par le FrontController.
Cette commande va déléguer l'appel serveur par la classe Delegate qui lui retournera les données.

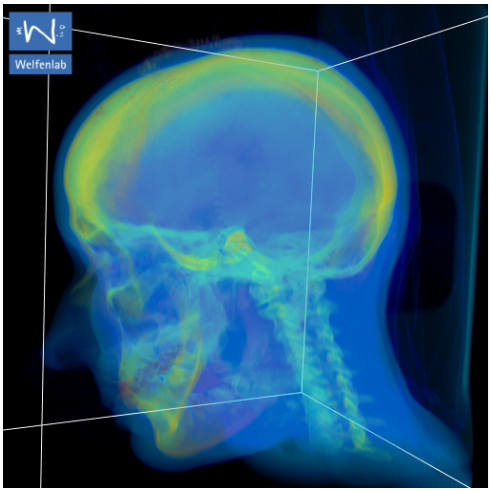
Business
Le business concentre les service et appels à des procédures distantes.
Il contient les Delegates (délégués) qui centralisent les définitions de méthodes distantes d'accès aux données. (liste de classes déléguées à l'encapsulation des appels des services distants)..
Les Services locators qui concentrent les services distants.

Il existe différents types de services distants. On peut citer: WebServices, REST, RemoteObject ….
La définition des ServiceLocator se fait à travers un fichier mxml où sont spécifiés les différents objets distants ainsi que leur source.



Cairngorm propose aussi un DiagramExplorer qui permet de visualiser les actions sur l'interface et tout le chemin des messages à travers les différentes couches du modèle tout en montrant les parties du code intéressantes.

Choix d'un visualisateur 3D



Choix de la visionneuse DICOM

Le choix du viewer 3 dépendait de certaines contraintes. La portabilité de l'application et l'intégration de la visionneuse dans une page web ou dans Flex builder a restreint les possibilités des langages utilisés pour le lecteur. Le seul langage qui permet cette souplesse est Java.

Java va permettre faire fonctionner une visionneuse 3D sur toutes les plateformes sans installation particulière en principe, il est connu par la majorité des programmeurs et il a été utilisé pour la création du viewer 2D .
Un désavantage de ce langage est sa lenteur par rapport à des langages comme le C.
Un des points importants pour le choix du lecteur était aussi la facilité de son intégration dans l'application, la recherche de lecteur en open source était aussi primordiale.
Le choix des lecteurs 3D est assez pauvre.
Le site très intéressant http://www.idoimaging.com/ regroupe pratiquement tout sur l'imagerie médicale en code ouvert.
Une recherche sur ce site met très vite en avant que les lecteurs DICOM 3D en Java sont assez rares.
La première idée était de trouver juste une visionneuse intégrable dans la visionneuse 2D déjà existante comme 4ième vues.
L'utilisation d'une librairie comme ImageJ aurait été intéressante avec ses macros, son langage script et la modularité de ses plug-ins, mais le rendu des démonstrations n'était pas assez intéressant et le peu de connaissances dans ce domaine m'aurait demandé un trop gros effort pour un résultat certainement en dessous des viewer spécialisés.
Après quelques recherches nous avons découvert YaDiV un viewer très intéressant en Java et utilisant en plus les nouvelles librairies Java 3D.
Les informations disponibles pour ce viewer était un descriptif et quelques screenshots, mais le développeur de ce viewer promettait une version code ouvert assez rapidement.
Après discutions avec Karl-Ingo Friese nous avons pu avoir une version à code fermé à la mi-février de YaDiV qui comblait nos attentes par rapport aux rendus et aux fonctionnalités.

Karl-Ingo Friese pensait livrer une première version open source en début d'année mais après quelques échanges de e-mail, nous nous sommes rendu compte qu'une version ne serait pas livrée à temps pour l'intégrer dans notre application.
C'est pourquoi nous avons proposé à Karl-Ingo Friese de nous livrer les sources sous NDA et c'est ce qui a été fait en prenant un peu de temps mais les sources nous ont été livrés finalement le 30 mars.

Avant l'intégration de YaDiV nous projetions d'intégrer dans l'affichage des données d'un patient, la possibilité d'afficher les données cliniques, le viewer 2D permettant de créer des ROI et la vue 3D dans 3 onglets différents.
Une des zones d'ombre était le format de fichier .seg utilisé par YaDiV que nous comparions à des ROI.
Après analyse de la structure des fichiers utilisés par YaDiV pour représenter les « régions d'intérêt ».

La représentation des régions d'intérêt était projetée avec les quatre tables rois référençant les type de régions d'intérêt, roi_pixels contenant les points des contours des régions d'intérêts et ct pour référencer les images.

Le format .seg utilisé par YaDiV est une représentation de BitCut qui va spécifier pour chaque voxel si oui ou non ce voxel fait partie du segment.

Il a été décidé d'utiliser ce format pour le stockage des régions d'intérêts.
De plus la modularité de YaDiV qui permet d'intégrer très facilement ses propres routines et de désactiver les options qui ne nous intéressent pas, nous a donné la possibilité d'intégrer les étapes d'initialisation des images à comparer et l'appel de son algorithme de similarité dans YaDiV.

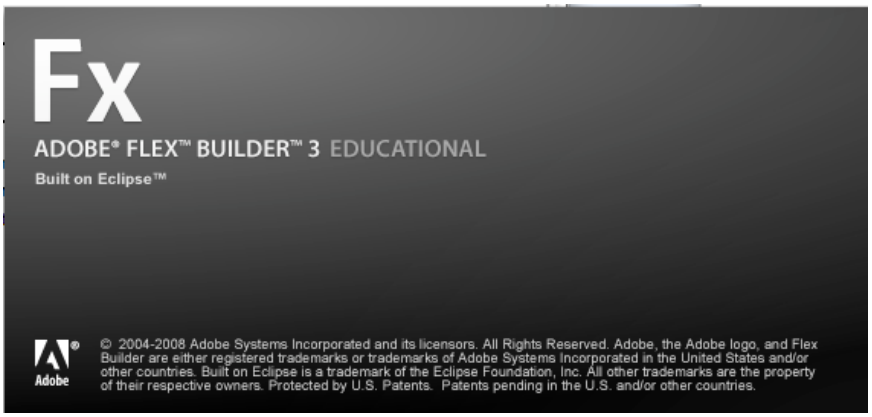
Résultats

Outils

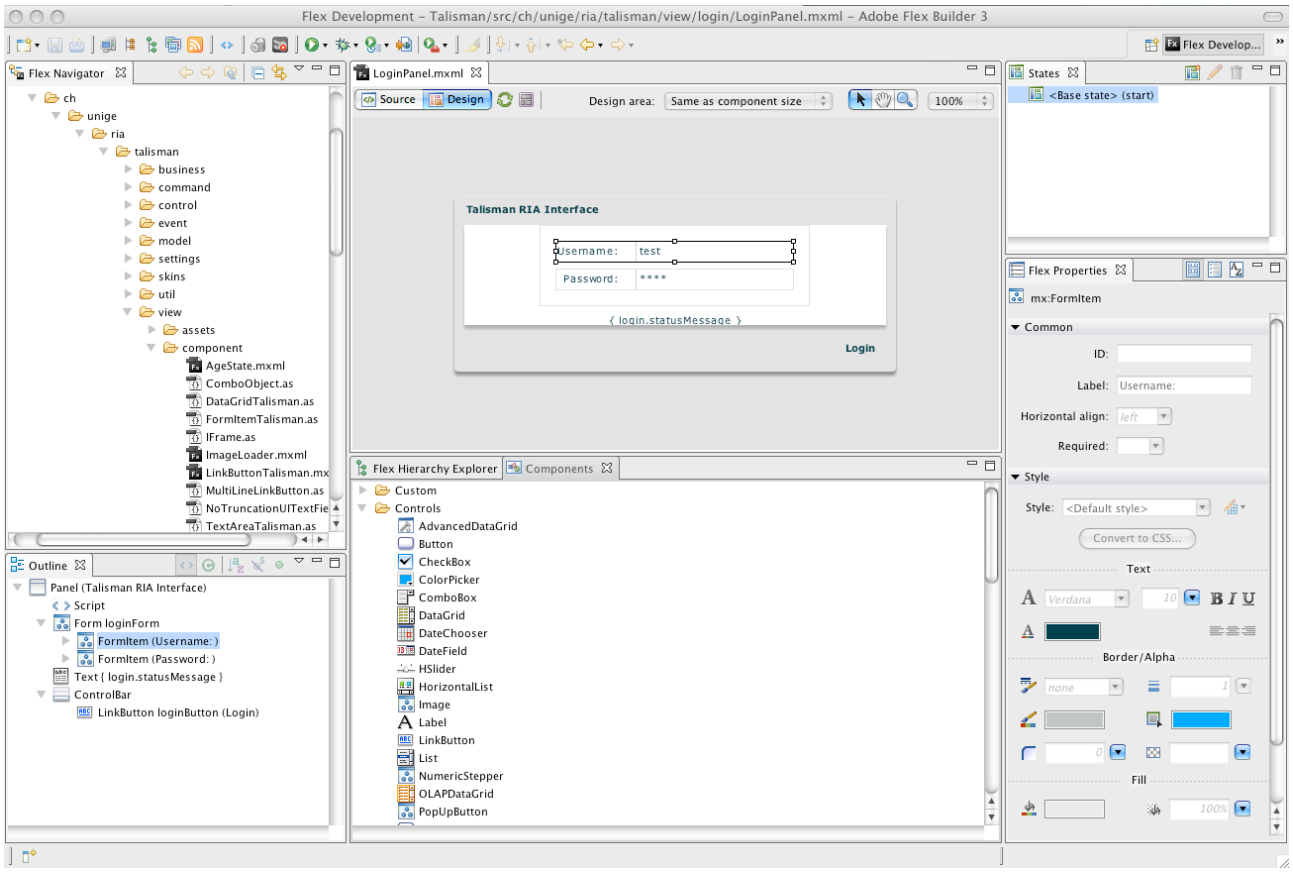
Le développement a été effectué sur deux plates-formes avec plusieurs outils qui différent selon le système d'exploitation utilisé. L'outil principal de développement étant basé sur Eclipse, il n'y a eu aucun problème de transfert de projet d'un système d'exploitation à l'autre.

Mac OS X

Flex Builder 3 (licence étudiant)

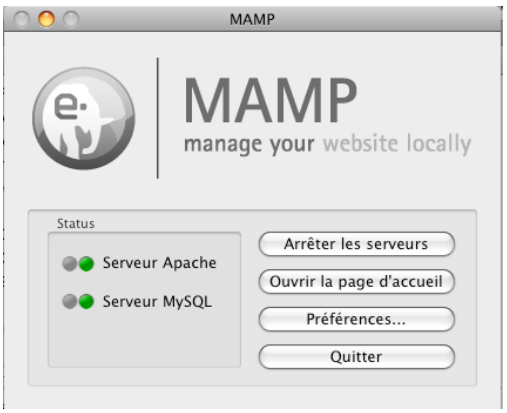


C'est l'outil de développement pour Flex utilisé pour le développement avec ActionScript et Mxml, il est très similaire à la version Eclipse pour Java.



Dans cette copie d'écran une vue en mode design qui n'est pas très courante avec Eclipse pour Java sinon les habitués de Java s'y retrouveront très rapidement, l'ActionScript étant assez similaire.

MAMP



Sous mac Os X il va permettre comme EasyPHP d'exploiter un serveur Apache avec du PHP.

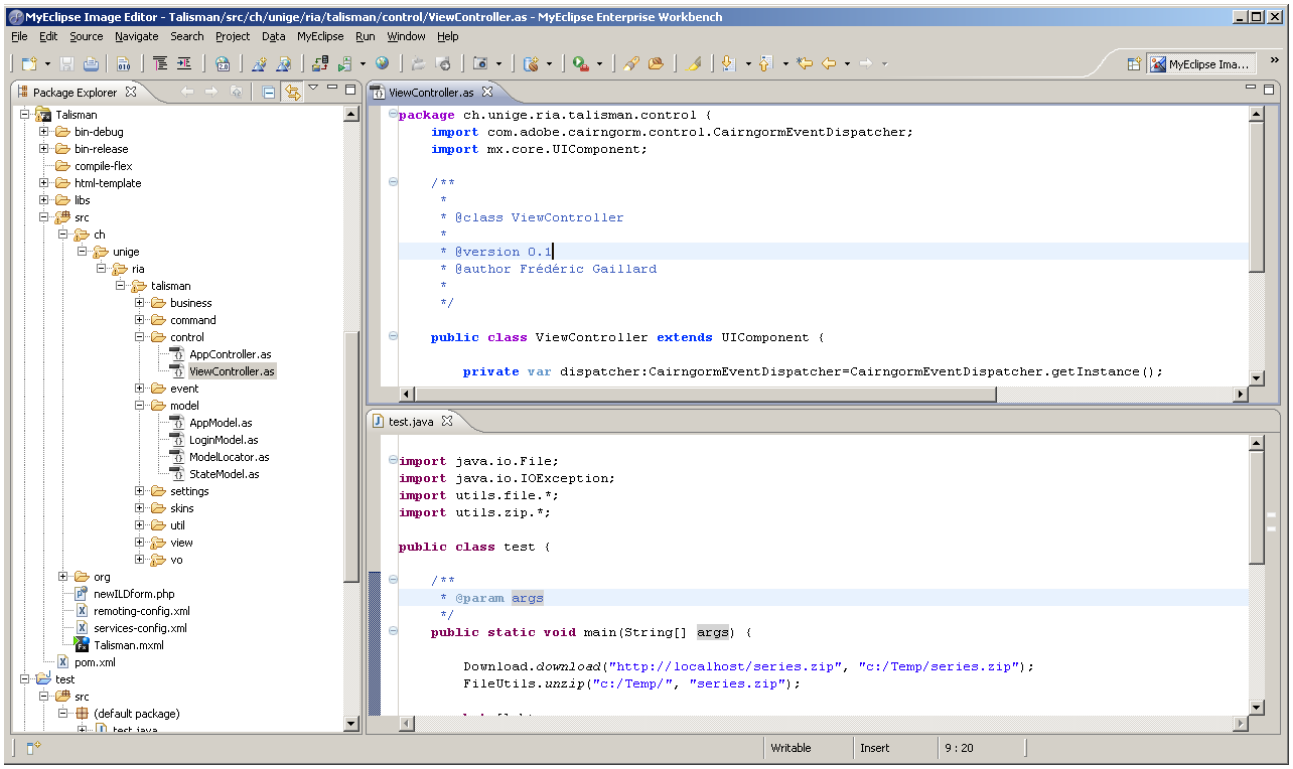
SumItron

C'est un petit éditeur de texte assez performant qui a été utilisé pour coder le PHP.

Microsoft Windows XP

Eclipse v 3.3 avec le Plug ins Flex Builder 3 (licence étudiant)

L'utilisation d'Eclipse pour Java avec le plug ins Flex Builder 3 est peut-être la meilleure solution pour le développement de cette application qui demande l'utilisation de flex et aussi des routines en Java et l'interaction des deux ensemble.



Dans cette copie d'écran ci-dessus deux types de projets cohabitent dans l'explorer de projet.

Un plugin PHP pourrait être installé et l'outil serait parfait pour ce type de développement.

EasyPHP

Similaire à MAMP sous mac Os X mais un peu plus souple et performant.

UltraEdit

Un très bon éditeur de texte sous pc, utilisé pour la programmation PHP.

Microsoft Excel

Excel a été très utile pour la manipulation des données de la structure de la table patient data qui contient près de 150 champs

Pages

La documentation est créée sous Pages ’09, ce choix est fait pour, encore une fois, comme pour Flex, me permettre de découvrir un nouveau produit.

Librairies et composants

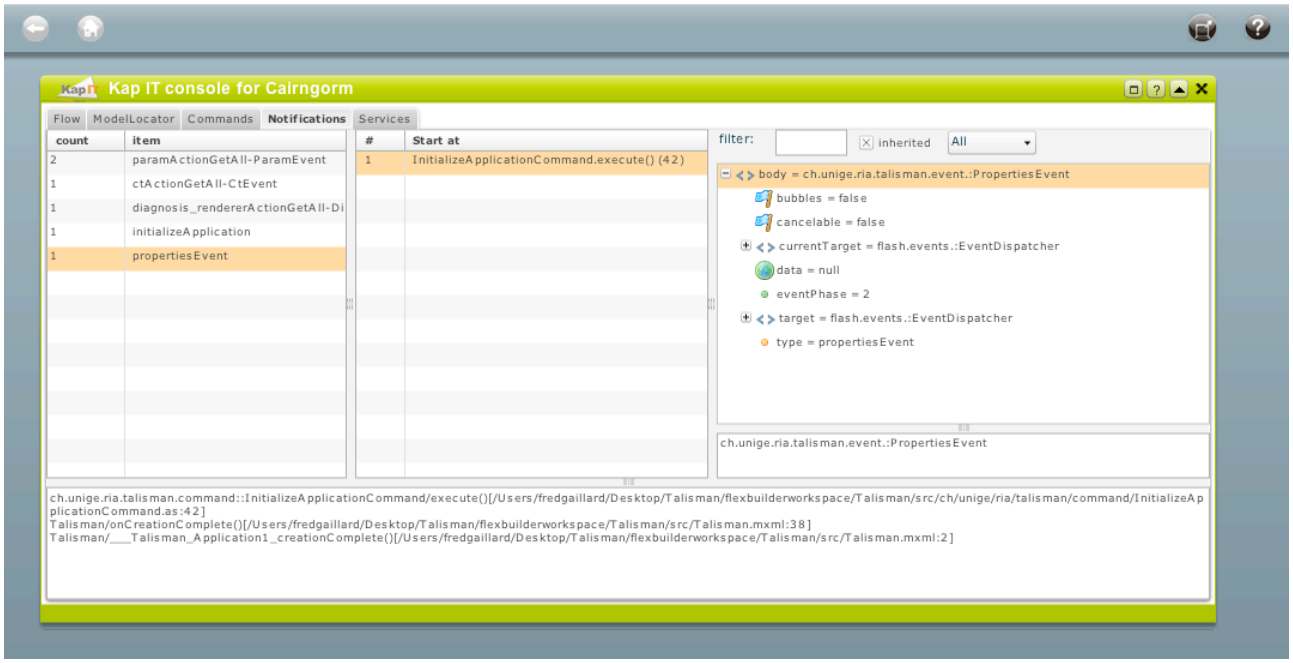
Les différentes librairies et composants utilisés dans l'application sont ajoutés sous forme de librairies ou simplement de classes ActionScript

Cairngorm

Sous forme de fichier swf, c'est la librairie Cairngorm écrite en ActionScript 3.

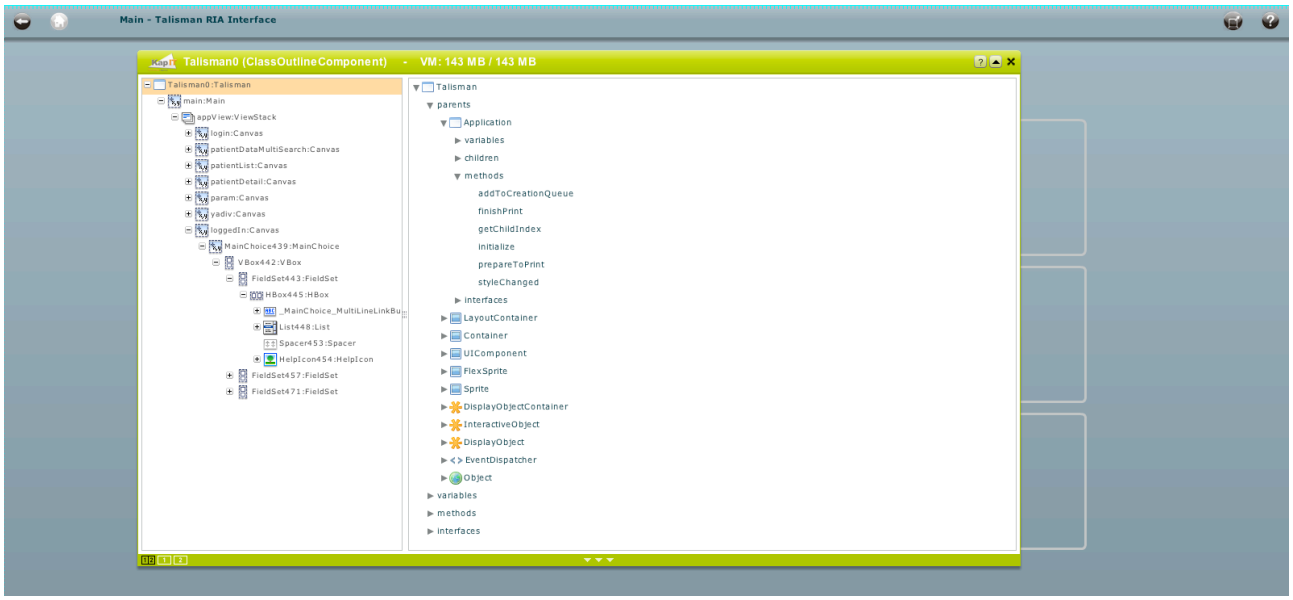
Cairngorm Console

C'est une libraire en swf produite par Kap Lab, qui va donner une aide au développeur Cairngorm, elle va permettre de monitorer et inspecter tous les objects et évènements Cairngorm.



KapInspect

Toujours sous forme de swf, cette libraire de Kab Lab va permettre d'inspecter tous les objets graphiques des applications Flex, de détailler leur binding leur renderer, les événements déclenchés sur les composants graphiques et même de modifier leurs propriétés, sans répercussion sur le code, pour se rendre compte de modification graphique en évitant la recompilation.



FlexLib et Jwolib, Degrafa

Ce sont des libraires graphiques open source qui étendent et améliorent les composants Flex de base.

Elle vont permettre dans l'application l'utilisation de quelques composants, par exemple les cadres avec titres, qui ne sont pas existants dans les composants de base Flex et qui demanderaient un trops grande énergie et connaissance du framework de base de Flex.

Degrafa_Flex

IView et TypedArrayCollection de CrasedCoders

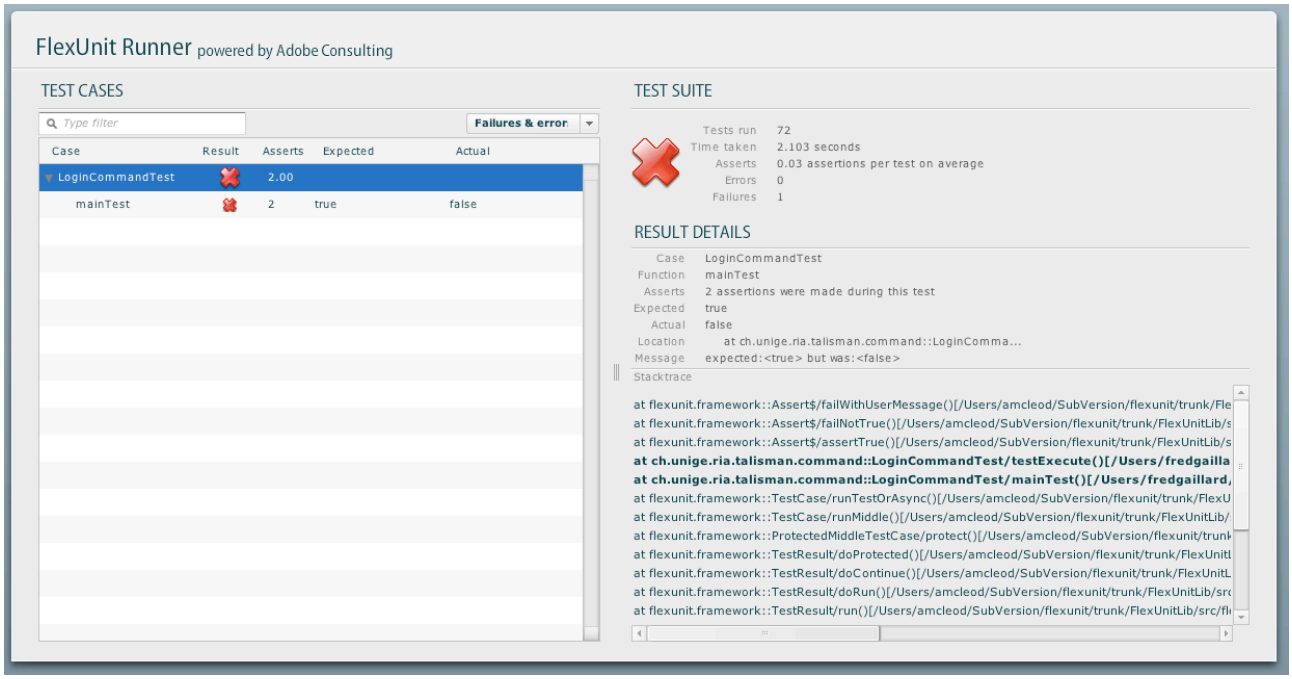
ExternalMouseWheelSupport d'Ali Rantakari

Cette classe permet de rendre accessible à Flex l'utilisation de la roulette de défilement vertical sur une souris.

IFrame.as, Alistair Rutherford,

C'est cette classe composant qui va permettre de contenir l'applet Java

FlexUnit



C’est un framework de test unitaire pour Flex et ActionScript en application et librairie, il est similaire à JUnit pour Java. Il comprend une interface graphique très utile.

Développement

Migration des données

Adaptation du modèle de base de données.

Différentes opérations ont été effectuées pour adapter les données de la base MySql à la nouvelle base de données.

Ces opérations ne devaient pas être trop importantes puisque les tables sont utilisées par l’outil de saisie des patients et les routines de recherche de similarités.

Base de données initiale

La base initiale se constitue de cinq table :

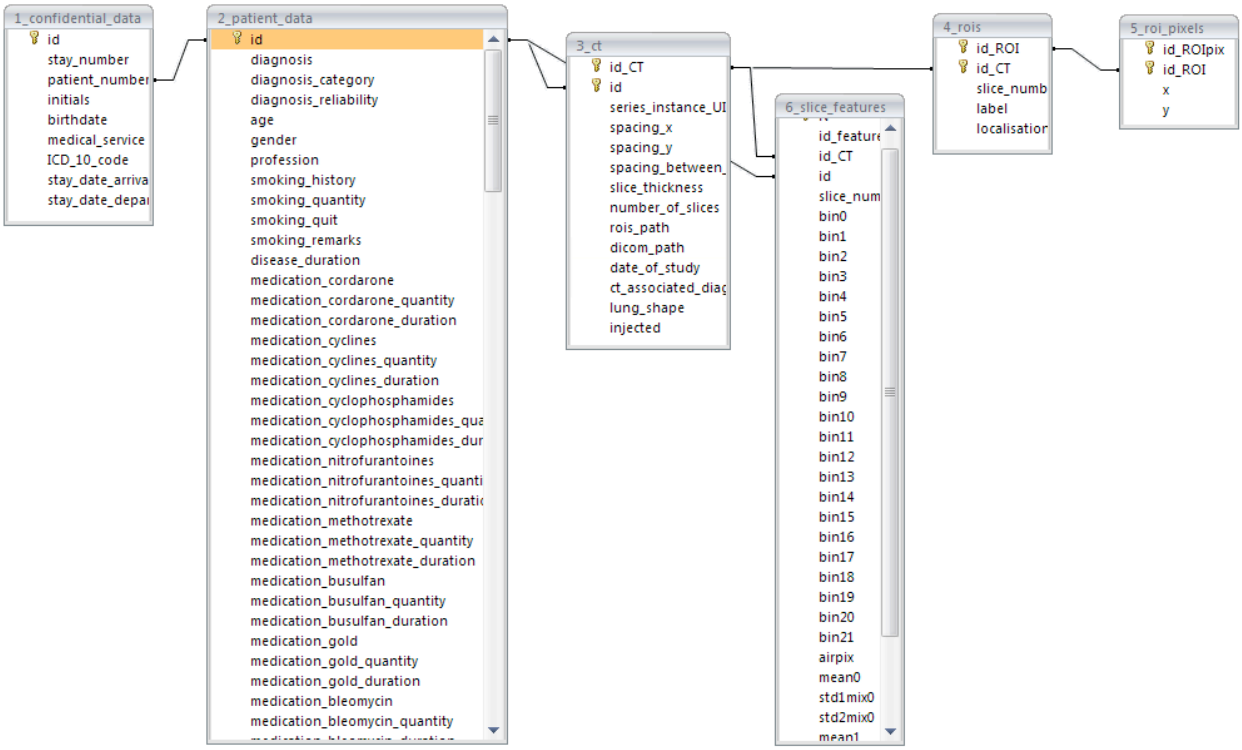


Table confidential_data

Cette table ne sera pas utilisée ni disponible.

Nom des tables

Tout d'abord le nom des tables est modifié, la numérotation avant le nom des tables est supprimée, elle n'a pas de sens dans la nouvelle application puisque la table 1_confidential_data ne sera pas utilisée.

Paramètres

En analysant la table Patient_data et la table Ct on remarque que des types enums sont utilisé, il sont difficilement utilisables et maintenables, ils sont remplacés par des clés étrangères liées à une nouvelle table de paramètres.

Les champs contenant des enums « null, yes, no » sont remplacés par un type « tinyint » qui permet les valeurs nulles.

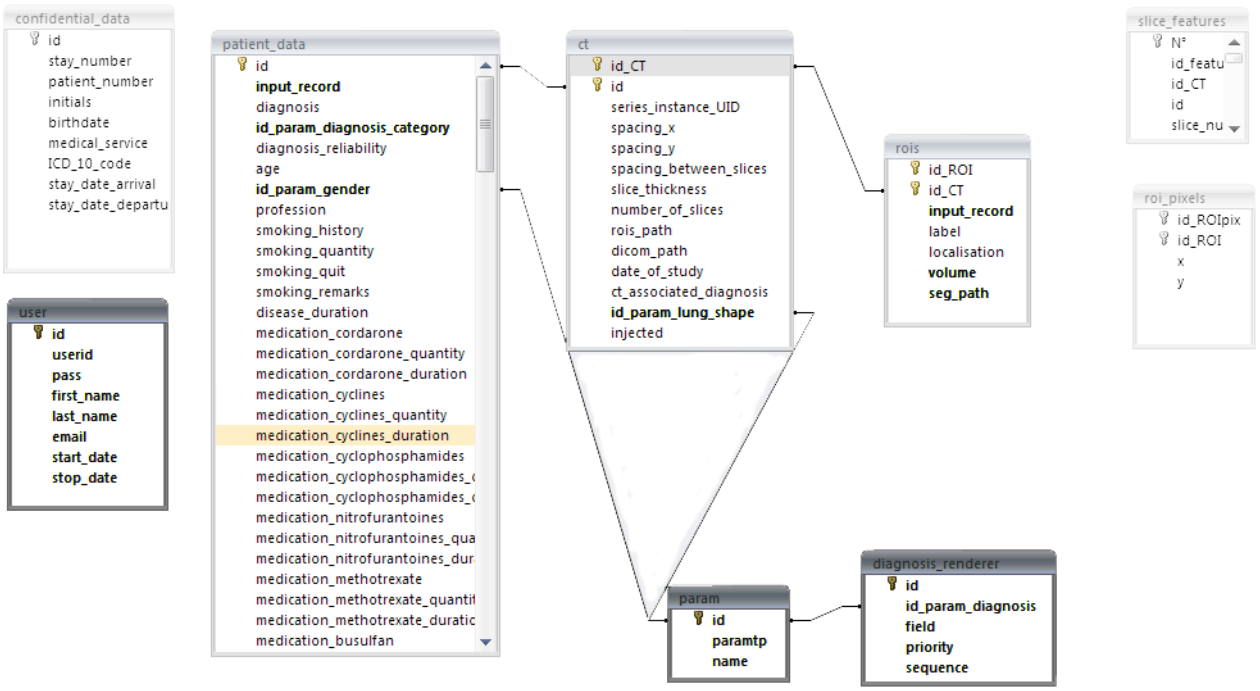
La nouvelle table paramètres contient un id, un paramtp qui contient le nom du champs qui contenait les enums et toutes les valeurs des enums.

Diagnosis_renderer

Une nouvelle table Diagnosis_renderer est ajoutée pour permettre d'afficher et de regrouper, selon les diagnostiques, les données cliniques selon trois priorités et avec un ordre prédéfini.

Table user

Une nouvelle table user est créée, elle permettra plus tard, l'enregistrement d'utilisateur avec des mots de passe et d'enregistrer les différentes analyses des utilisateurs ainsi que des configurations de leur environnement.



Modification des structures des tables

Patient_data

La table patient_data contiendra un nouveau champs input_record qui va permettre de filtrer les patients de base dans la perspective de nouveaux ajouts de données ou de sauvegarde temporaire d'enregistrement saisis pour la recherche de similarités.

Rois

La table rois va contenir un nouveau champs input_record qui aura le même rôle que dans la table patient_data.

Un nouveau champs volume est ajouté, il contiendra la proportion de volume de tissu trouvé.

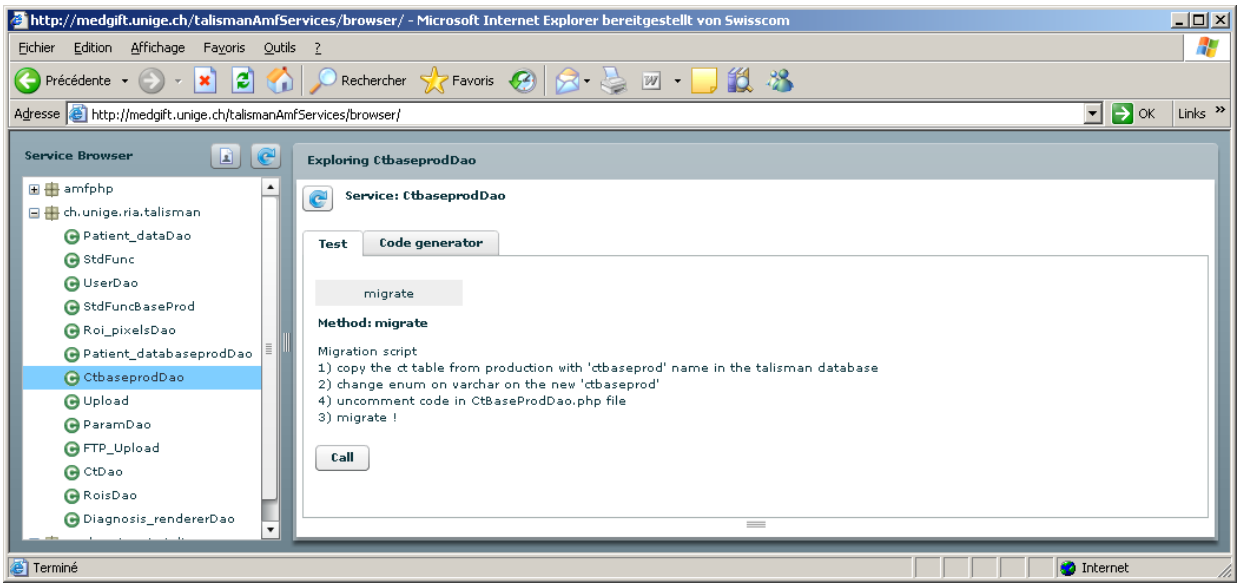
Le seg_path contient le chemin des fichiers .seg contenant les « regions d'intérêt ».

Création de la table param et des foreign keys

La migration des données dans les nouvelles table va se faire par des scripts PHP pour associer les bonnes clés au champs qui contenaient des params.

Ces scripts sont intégrés comme services dans amfphp pour permettent de les rejouer très facilement.

Par exemple pour la modification de la table Ct :



On peut créer un appel avec des commentaires qui suivent les standards javadoc.

Pour ces fonctions de migration le PHP est très utile il permet facilement de récupérer les noms des champs commençant par « id_param_ » et de faire une jointure sur la table param pour rapatrier la clé correspondante.

```
/**
* Migration script
* 1) copy the ct table from production with 'ctbaseprod' name in the talisman database
* 2) change enum on varchar on the new 'ctbaseprod'
* 4) uncomment code in CtBaseProdDao.PHP file
* 3) migrate !
*/
public function migrate(){

		$allParams = CtbaseprodDao::getAllParam();
		$param =CtbaseprodDao::getAll();

		foreach( $param as $ct ) {

			StdFunc::logString($ct->injected);
			CtbaseProdDao::test_CreateOneForMigration(
				$ct->id_CT
				,$ct->id
				,$ct->series_instance_UID
				,$ct->spacing_x
				,$ct->spacing_y
				,$ct->spacing_between_slices
				,$ct->slice_thickness
				,$ct->number_of_slices
				,$ct->rois_path
				,$ct->dicom_path
				,$ct->date_of_study
				,$ct->ct_associated_diagnosis
				,CtbaseProdDao::getParamIdByTpAndName("lung_shape",
				$ct ->lung_shape, $allParams)
				,CtbaseProdDao::getYesNo($ct -> injected));
```

```
                }
                return 'migration completed';
        }

        /**
        * getParamIdByTpAndName
        *
        * @param : paramtp  - the param type
        * @param : name     - the param name
        * @param : allParam - the list of param
        *
        * @return: the id of param or -1
        */
        public function getParamIdByTpAndName($p, $n, $allParams) {

                if ($n == ""){
                        return -1;
                }
                foreach( $allParams as $param ) {
                        if (strstr($param -> paramtp, $p) && strstr($param -> name, $n)){
                                return $param->id;
                        }
                }
                return -1;
        }
```

Développement

Généralités

ActionScript 3.0

Pour décrire très rapidement l'ActionScript 3.0, on peut faire une petite comparaison entre Java 5.0 et ActionScript 3.0.

Le tableau en annexe détaille toutes les différences de concept et de construction de language. Pour un programmeur Java ou C#, il suffira de très peu de temps pour être dans le «bain».

Les différences rencontrées le plus souvent et un peu déroutante sont l'initialisation des variables, la classe est passée après la variable précédée de var, les paramètres des déclarations de fonction suivent la même règle.

```
public function clickPressed(event:Event) : void {

        var firstName:String=”John”;
}
```

Les variables non déclarées :

```
var maVariable:* ; // le type * est appliqué
```

Tous les types primitifs en ActionScript 3.0 sont des objets.

Les lignes suivantes sont équivalentes :

```
var age:int = 25;
var age:int = new int(25);
```

Les types complexes Array, Date, Error, Function, RegExp, XML et XMLList sont intégrés au langage.

En ayant la connaissance de ces quelques différences on peut déjà très bien débuter en ActionScript 3.0.

Pour ce qui est de l'utilisation de l'éditeur Flex Builder 3 ou bien le plugin Flex Builder 3 pour ActionScript pas de surprise en comparaison à Java sous Eclipse.

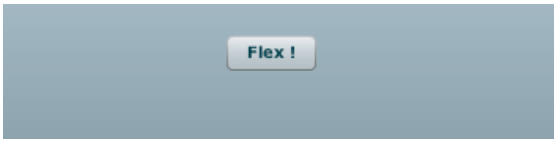
MXML

MXML est un langage XML qui sert à agencer des composants d'interface utilisateur pour les applications Flex.

Il est possible d'utiliser le MXML pour définir de manière non déclarative des aspects non graphiques d'une application comme l'accès à des sources de données.

Très brièvement un petit exemple qui montre la simplicité du langage :

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   horizontalAlign="center" verticalAlign="center">

   <mx:Button id="myButton" label="Flex !" />

</mx:Application>
```



Une gestion de l'évènement click sur le bouton se fait en ajoutant tout d'abord à la balise la propriété «click» et le nom de la fonction qu'elle va appeler.

```
<mx:Button id="myButton" label="Flex!" click="clickPressed()" />
```

Ensuite dans un tag <mx:Script> on ajoute le code de la méthode clickPressed en ActionScript.

```
<mx:Script>
      <![CDATA[
         private function clickPressed ():void
         {
            trace(“Button Flex ! pressed”);
         }
      ]]>
   </mx:Script>
```

Initialisation

ActionScript ou MXML

Avec Flex, le language MXML peut être remplacé par de l’ActionScript et vice versa.

Par exemple <mx:Button id=»myButton»/> peut être remplacé par en ActionScript par var myButton:Button = new Button().

L’avantage avec MXML se trouve bien sûr dans la composition, la lecture du code.

Pour les simples composants et tous les écrans principaux de l’application c’est MXML qui est utilisé, sinon dans le cas de renderer et de composants plus complexes qui demandent des intégrations avec des évènements ou bien l’instanciation de valeurs, c’est l’ActionScript qui est choisit.

L’appel d’une classe graphique représentant un composant ou bien d’un composant MXML se fera exactement de la même manière.

Le simple ajout d’une variable public dans une classe composant ActionScript par exemple une classe MonComposant.as pourra être utilisée simplement dans MXML par :

```
<view:MonComposant />
```

Si mon composant contient des variables plublic, on pourra y accéder avec :

```
<view:MonComposant  maVariable=’‘Flex”/>
```

L’interface Flex Builder 3 qui permet l’utilisation de MXML est conviviale et simple d’utilisation.

On peut passer du mode Design au mode Source en sélectionnant par exemple un composant pour le retrouver sélectionné d’un coté comme de l’autre.

La touche F4 permet en mode Design de faire un Show Surround Containers qui permettra de visualiser directement la composition des containers de notre écran.

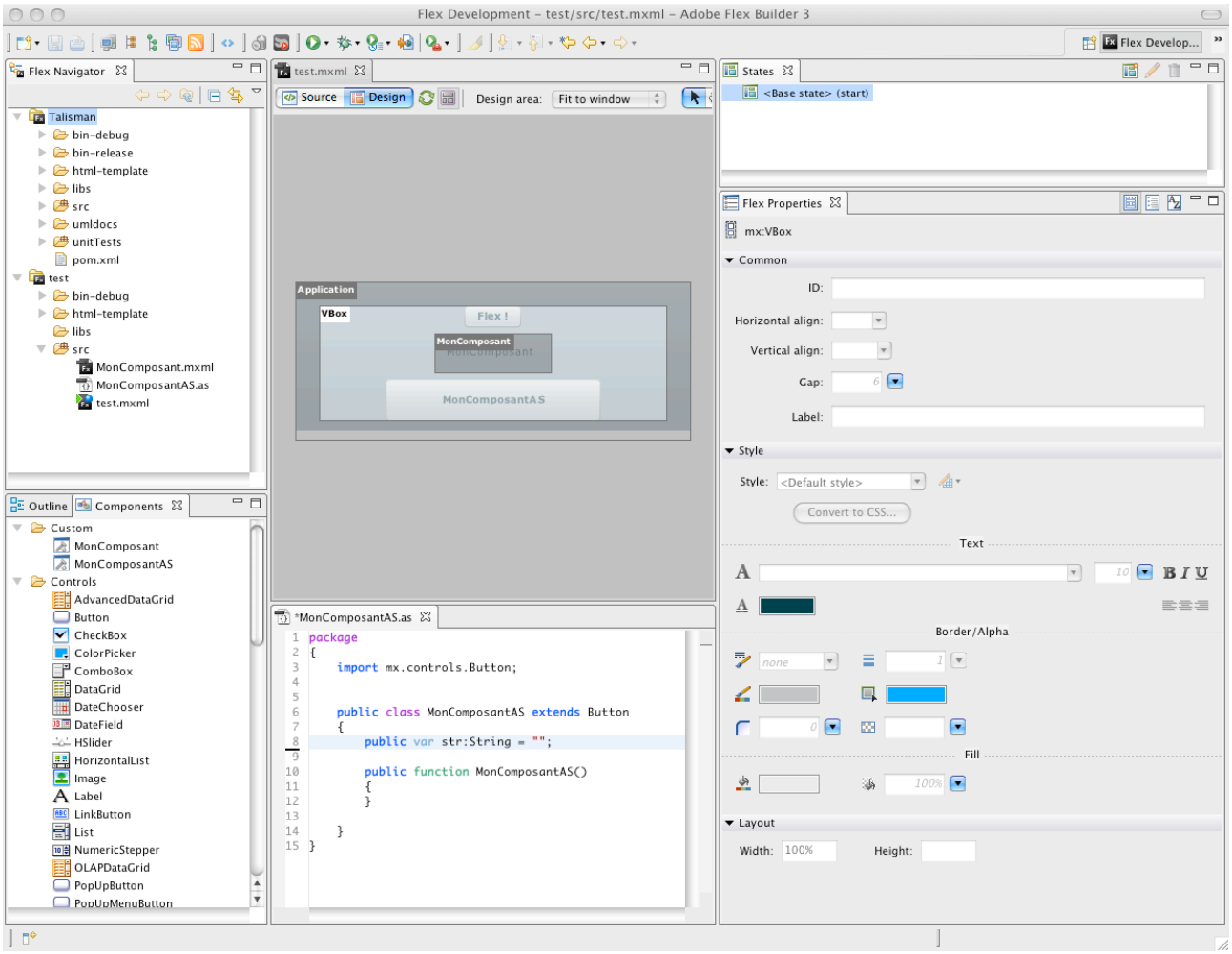
De plus les composants «Custom» sont visibles et directement intégrés comme d’autres composants.

Cette exemple démontre ce qui a été dit plus haut, la vue avec le Show Surroud montre un écran composé de plusieurs composants dont deux composants «Custom» qui ont été déposés sur la page avec un drag and drop depuis le sélecteur de composants en bas à gauche.

Les deux composants «Custom» sont en fait les deux classes que l'on voit dans le Navigateur de classe en dessus. Le composant «MonComposantAS» est ouvert dans l'éditeur ActionScript également. Le panneau de droite «Flex Properties» va permettre en quelque sorte d'éditer les propriétés des tags MXML.

La propriété str de la classe MonComposantAS.as sera accessible simplement avec :

```
<ns1:MonComposantAS str="MonComposantAS" />
```

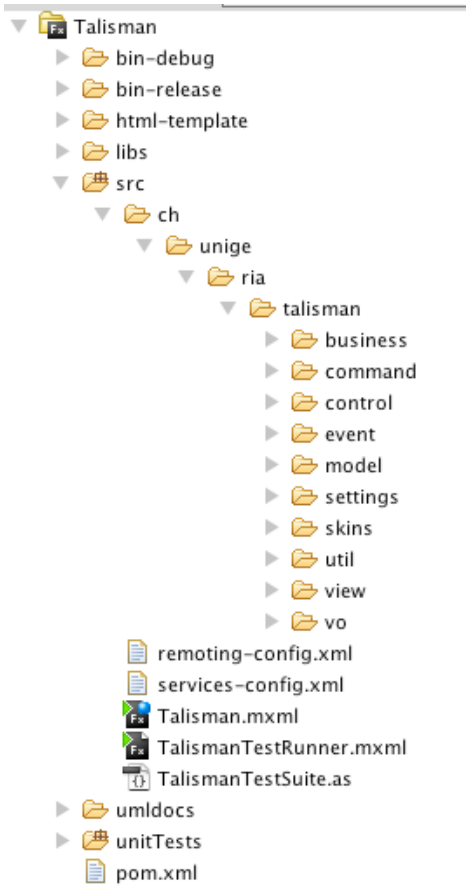


Cairngorm et Structuration de l'application

Il n'y a pas beaucoup de documentation sur la struturation d'une application Flex à disposition, mis à part les regroupements évidents par package ou bien les regroupements des ressources. Les exemples trouvés sont très souvent de petites applications ou sites, et l'ajout de quelques cas d'utilisation dans ces structures font ressortir très vite beaucoup de problèmes.

Heureusement Cairngorm résout très vite ce genre de problèmes en ne laissant que très peu de choix quant à la structure des répertoires et packages de notre application.

Très simplement des regroupements par les principaux «acteurs» de son modèle :

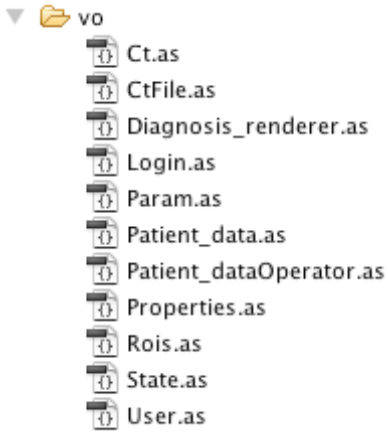


Un choix pourrait se faire en créant une structure par cas d'utilisation mais dans le cas de cette application une structure contient les différents cas d'utilisation.

Dans cette application les paquets settings,util et vo pourraient être séparés des paquets cairngorm pour la lisibilité mais dans ce cas il faudrait seulement le faire dans le cas d'un gros projet qui sépare les cas d'utilisation à la base de ses packages.

Le package vo

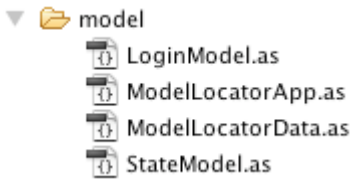
Le package vo ou value object contient les objets qui représentent les entités logiques de l'application.



Les vo seront l'interface entre les données distantes et les objets Flex.

Les classe Ct, Diagnosis_renderer, Param, Patient_data, Rois et User, représentent des objets de base de données. En revanche les classes Login, Properties et State sont des classes métiers qui ne font pas d'interface avec les tables de la base de données.

Le package model



Basés sur le pattern singleton, les ModelLocator vont encapsuler les données du client dans une structure et séparer les donnes des vues.

C'est un singleton qui contiendra tous les vo. C'est ce point unique qui permettra l'accès aux données depuis les vues. La classe est précédée du tag [Bindable] ce qui lui offre l'avantage de lier les données du model aux composants des vues.

Toutes les mises à jour des données seront automatiquement répercutées sur les vues.

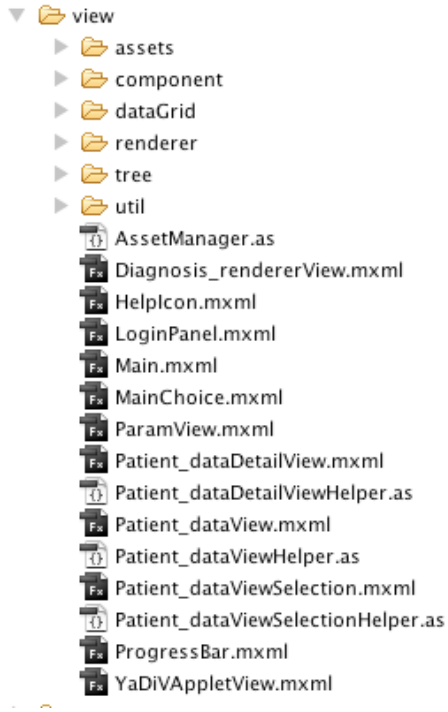
Les classes LoginModel et StatesModel qui représentent des objets qui ne sont pas liés à la base seront référencés dans le ModelLocator. Ces classes contiennent en plus de leur vo, des informations propres à la l'appel des services distants ou à la session en cours.

Le ModelLocatorApp est la classe principale qui regroupe l'accès au données de l'application et le ModelLocationData est la classe qui regroupe l'accès sur les données venant des bases.

Cette séparation est aussi faite par rapport au générateur.

Le package view

Ce package contiendra les vues, toutes les interfaces graphiques de l'application :



Le package controller

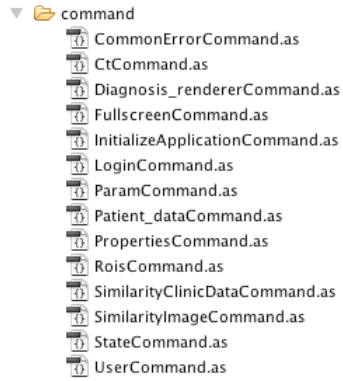
Ses classes implémentent le pattern FrontController qui est composé des pattern Commande et MVC.

Lorsqu'une action génère un évènement, ce dernier est capté par le frontcontroller, le frontcontroller associe ensuite chaque évènement à une commande.

Ce package se compose uniquement de la classe AppController.as qui contient un constructeur qui fait des ajout de commandes, les commandes à écouter.

Le package command

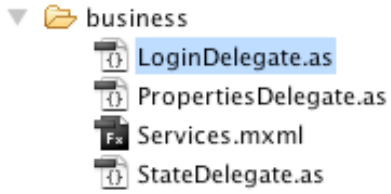
Ce package contient toutes les commandes suivantes :



Les commandes sur les objets de la base sont regroupées dans les classes correspondantes aux objets, sinon les principales commandes de l'application.

Le package business

Ce package contient les delegates qui sont chargés d'encapsuler les appels aux méthodes des services distants :



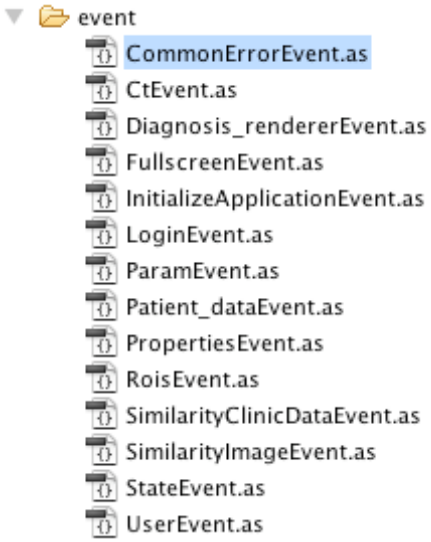
Dans cette application trois delegates sont utilisés, le login et properties et le state.

Le login et le properties déclenchent tous les deux des accès distants.

Les différentes opérations sur les vo de base n'ont pas de delegate ici, ceci n'étant d'une part pas très utile et d'autre part j'ai décidé de ne pas les implémenter pour permettre une génération de code plus facile.

Le package event

Il contient tous les énènements associés aux commandes :



En général ces classes étendent la classe CairngormEvent et contiennent un constructeur faisant un appel au constructeur de la classe qu'il étend, en lui passant une variable static permettant de définir l'événement.

Générateur

J'ai essayé de combiner l'utilisation de générateur avec cairngorm pour la création commandes de base pour les opérations de base sur les tables de l'application.

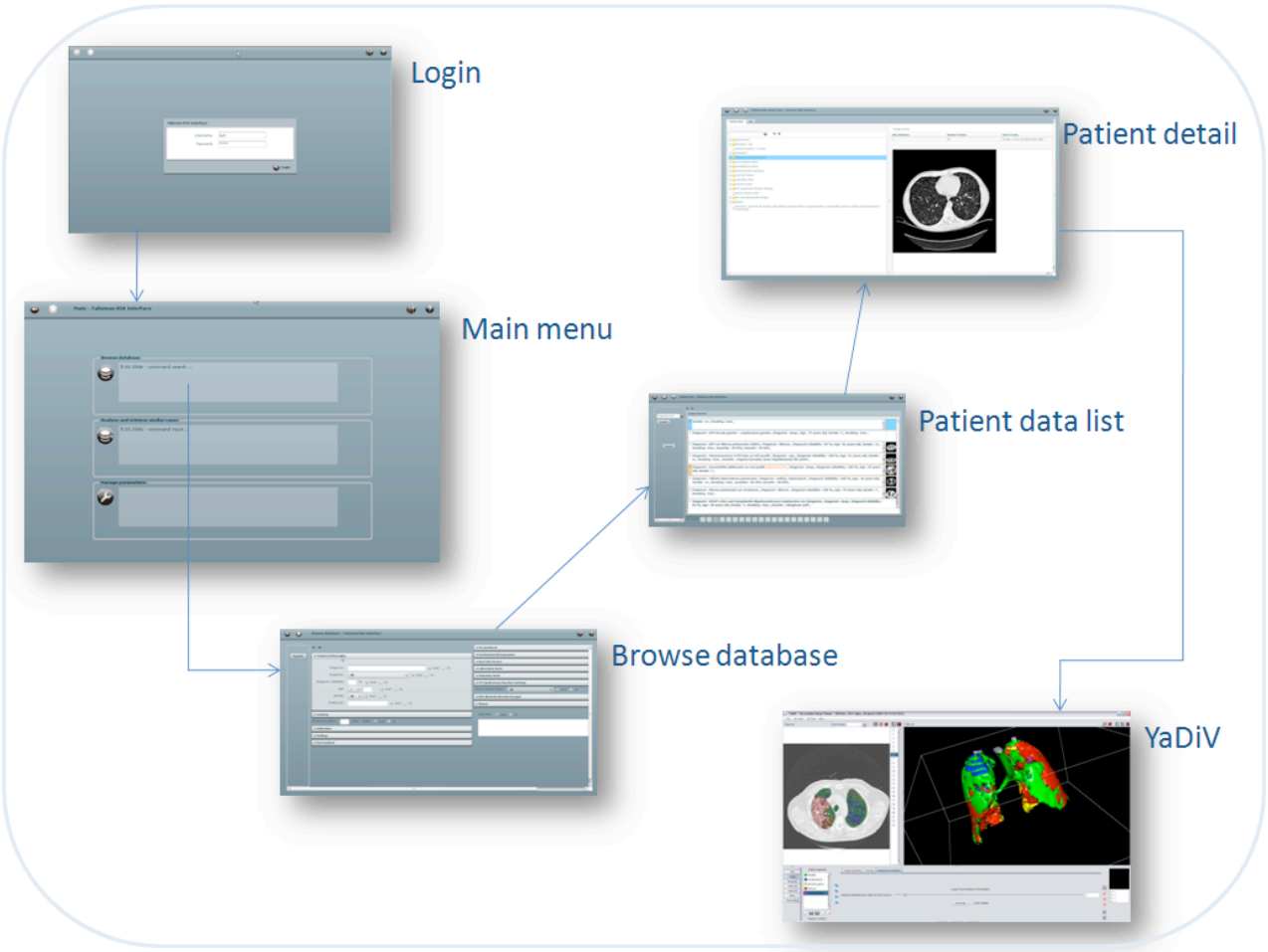
Cela demande une séparation des modelLocator de l'application.

Un model créé pour les commandes générées pour les opérations de bases, les autres models concerneront toutes les commandes et opérations de l'application.

Structure globale de l'application

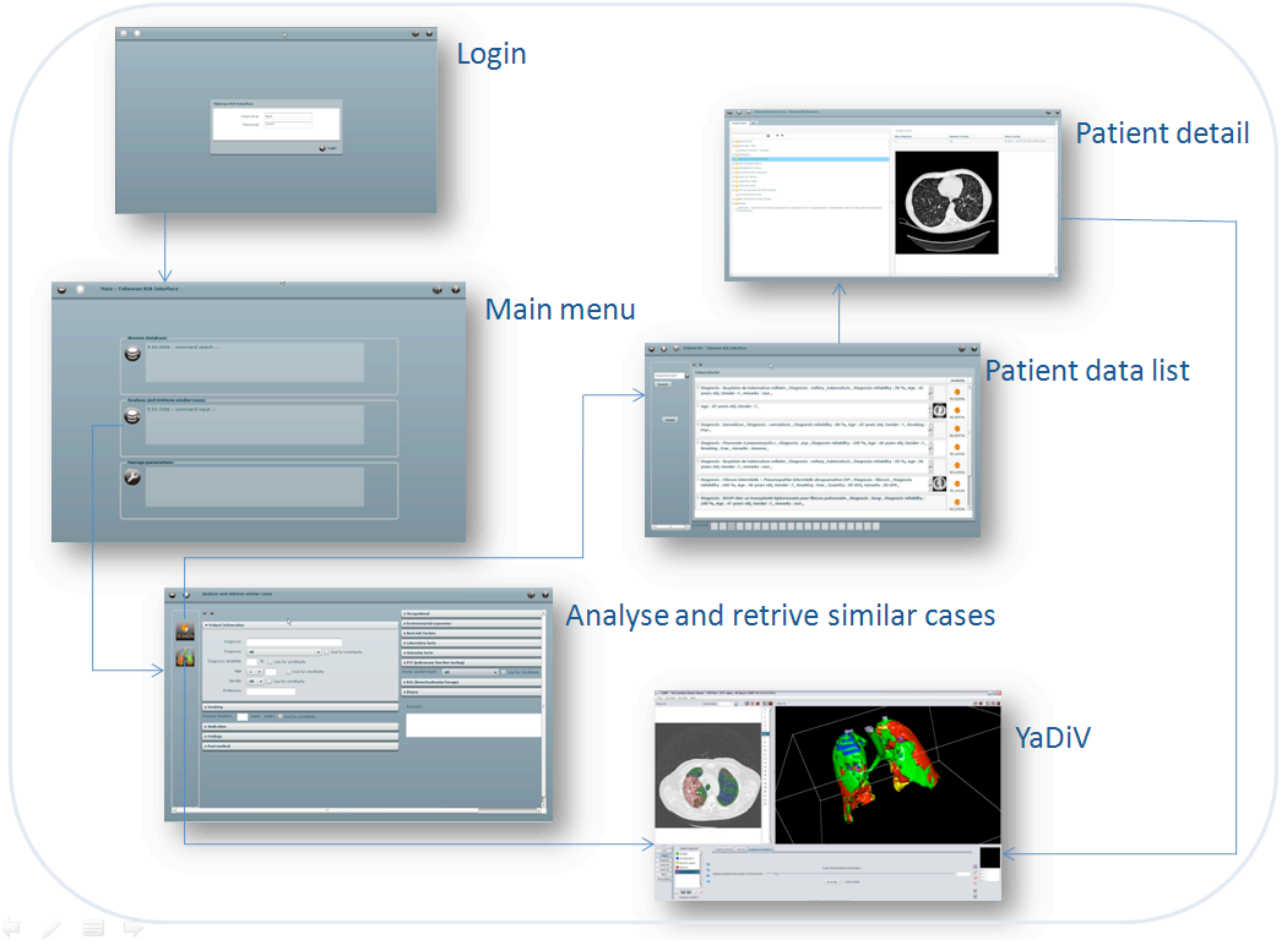
Le work flow de l'application se divise en trois parties

Browse database



Après la connexion de l'utilisateur, l'option du menu «Browse database» va permettre l'ouverture de l'écran de composition de requête de recherche sur tous les cas de la base, une fois la recherche lancée, une liste de patient est affichée. De cette liste un détail du patient peut s'ouvrir ainsi que la visionneuse permettant d'afficher la série d'images concernant le patient.

Analyse and retrieve similar cases



La deuxième option «Analyse et retrieve similar cases» permet de faire des similarités par rapport aux données cliniques ou aux images d'un patient saisi.

Le premier bouton représentant un «pie char» va permettre de faire des similarités avec les données cliniques du patient en cochant les cases «Use for similarity» devant les paramètres intéressants.

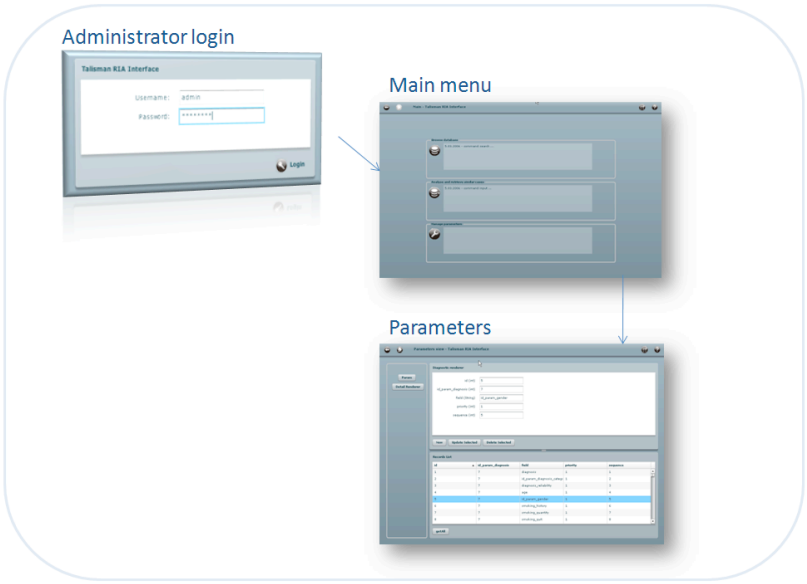
Une liste de patients s'affiche contenant le pourcentage de similarité par rapport au patient saisi.

Le deuxième bouton représentant des poumons segmentés va ouvrir l'applet YaDiV qui permettra de segmenter les poumons sélectionnés par l'utilisateur puis de lancer une routine qui donnera le pourcentage de volume par type de tissu des images de l'utilisateur.

Une fois les volumes par type de tissu calculés, la liste des similarités est rafraîchie et les similarités de volume par type de tissu sont affichées.

Manage parameters

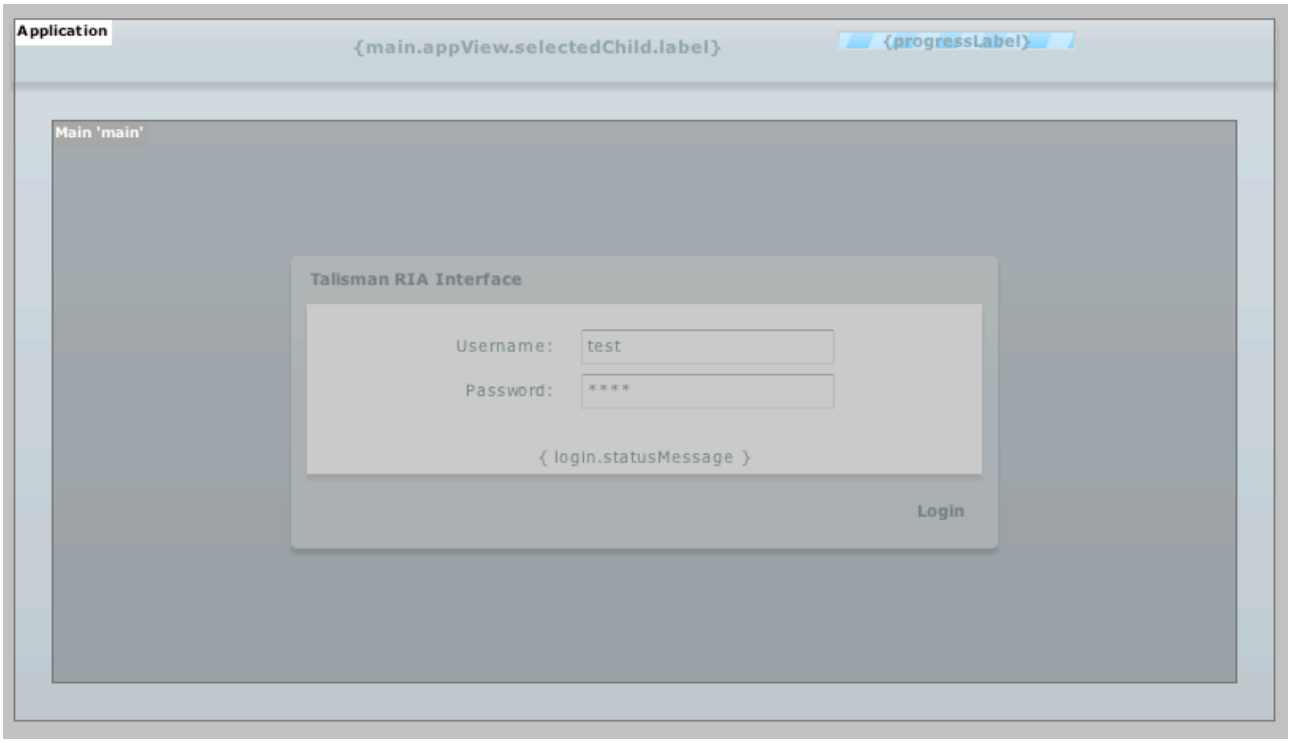
La dernière options permet de gérer les paramètres de l'application, pour le moment les tables «param» et «diagnostic_renderer» sont maintenable dans cette option.



Structure des views

La structure adoptée pour les objects graphiques, les views est la suivante :

Tout d'abord la classe Talisman.mxml est composée d'une barre d'application qui va accueillir différents boutons dont les boutons de navigation et d'aide, un texte indiquant l'écran sélectionné et une barre de progression, le container principal contiendra la classe MainView.mxml.



La classe MainView.mxml est composée d'un objet viewStack qui permet d'empiler les vues.

Toutes les vues de l'application qui se retrouvent dans les packages view sont empilées dans ce viewStack.

La classe LoginView affiche l'écran de connexion.

La classe MainChoiceView affiche l'écran d'accueil.

La classe Patient_dataSelectionView.

La classe Patient_dataSelectionView permet d'afficher la création de la requête de recherche et de similarité.

La classe Patient_dataView affiche les listes des patients recherchés et la similarité des patients.

La classe Patient_dataDetailView le détail du patient.

Et YaDiVAppletView l'applet 3D.

Initialisation de l'application

Au démarrage de l'application, plusieurs opérations sont effectuées. Elles vont permettre d'initialiser l'application et de charger toutes les données indispensables pour le fonctionnement de l'application.

Dans la classe Talisman.mxml la première méthode exécutée à l'ouverture de l'application est la méthode onCreationComplete(). Son appel n'est pas tout à fait évident à trouver, en fait il se trouve dans le premier tag <mx:Application> de la classe. On peut voir aussi qu'à la fin de la création de la classe, la méthode init() sera aussi lancée.

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
...
 applicationComplete="init(event);"
 creationComplete="onCreationComplete(event);"
...
 xmlns:component="ch.unige.ria.talisman.view.component.*">
```

La méthode onCreationComplete va déclencher la commande InitializeApplication.

```
private function onCreationComplete(event:Event):void {
		...
		var eventInit:InitializeApplicationEvent=new InitializeApplicationEvent();
		eventInit.dispatch();
}
```

Sans détailler l'enchaînement de l'appel d'une commande (qui est fait plus bas), la commande InitializeApplication va déclencher plusieurs actions et commandes.

```
Utils.initStyles();
var evPro:PropertiesEvent=new PropertiesEvent();
evPro.dispatch();

var evDiag:Diagnosis_rendererEvent=new
Diagnosis_rendererEvent(Diagnosis_rendererEvent.DIAGNOSIS_RENDERER_ACTION_GET_ALL, null);
evDiag.dispatch();

var evP:ParamEvent=new ParamEvent(ParamEvent.PARAM_ACTION_GET_ALL, null);
evP.dispatch();
```

Tout d'abord la méthode initStyles qui charge une librairie .swf générée à partir d'un fichier .css qui se trouve dans le package skins.

Ensuite les renderers pour les diagnostiques et les paramètres sont chargés, ces données seront indispensables pour l'affichage des information des patients.

Creation Policy

Flex permet plusieurs modes de création ou de chargement de ses objets graphiques.

En général le mode creationPolicy=’‘auto” est utilisé et si rien n’est spécifié dans le tag <mx:Application/> ou dans les tags composants, c’est ce mode de création qui est utilisé. Dans ce mode les composants sont instanciés seulement à l’appel de l’état.

Dans le mode creationPolicy=’‘all”, tous les composants sont instanciés à l’initialisation.

Plusieurs choix sont possibles, soit les composants sont créés quand on a besoin de les utiliser, soit tout est créé au démarrage ce qui impliquera un plus long temps au démarrage.

Dans cette application, l’écran de recherche qui permet de composer la requête pour la recherche et la similarité demande beaucoup de temps pour son chargement.

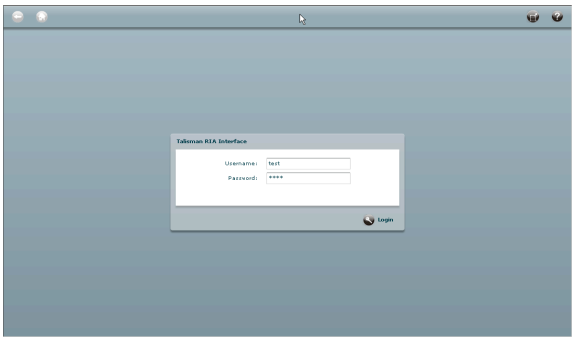
Une instanciation à l’appel de cet écran est beaucoup trop longue, c’est donc le mode de chargement qui instancie à l’ouverture de l’application les composants qui est choisit.

Dans la classe écran Patient_dataSelectionView un mode creationPolicy=’‘queued” est utilisé pour tous les composants avec un index de creation (creationIndex=”2” qui permet de donner un ordre de création à l’écran.

L’écran Patient_dataSelectionView sera créé avec ce mode “queued” dès l’initialisation de l’application et pendant même la connexion, il se peut même que, dans des cas de transfert très lents des données, l’affichage des composants de cet écran ne soit pas complètement terminé à l’ouverture cette vue.

Page d'accueil

Connexion



Une table user a été créée dans la base et les services de création de user et des tests de mot de passe crypté sont faits mais ne sont pas utilisés pour l’instant. Leur utilisation faisait partie d’un «nice to have» du cahier de charges.

Dans un premier temps donc, un minimum est fait avec service Http, ce qui permet entre parenthèse de tester ce type de service.

Du coté serveur un simple fichier login.PHP va contenir :

```
<?PHP
if (($_POST[“username”] == “test” && $_POST[“password”] == “test”)
		$_POST[“username”] == “admin” && $_POST[“password”] == “admin”)){
			print “true”;
		} else {
			print “false”;
}
```

Ensuite un service Http est définit dans le fichier Services.mxml qui se trouve dans le package business :

```
<mx:HTTPService id="loginServices" url="http://medgift.unige.ch/talismanAmfServices/login.PHP"
		showBusyCursor="true"
		method="POST"
		resultFormat="text"/>
```

Du coté client le déclenchement de l'événement du login va démarrer sur le bouton login de l’écran de connexion avec le tag :

```
<mx:LinkButton icon="{AssetManager.loginIcon}" toolTip="Login" label="Login" id="loginButton"
					enabled="{ !login.isPending }"
					click="loginUser()"/>
```

La méthode cloginUser() est définie dans un tag <mx:Script> :

```
public function loginUser():void {
		var loginVO:Login=new Login();
		loginVO.username=username.text;
		loginVO.password=password.text;
		var event:LoginEvent=new LoginEvent(loginVO);
		event.dispatch();
}
```

La classe Controller suivant le pattern FrontController a ajouté la command login dans son constructeur. Ce qui lui permettra de détecter l'événement généré par le login et contacter la commande :

```
addCommand( LoginEvent.EVENT_LOGIN, LoginCommand );
```

La classe LoginCommand est appelée par la fonction execute

```
public function execute( event : CairngormEvent ) : void {
			model.login.isPending = true;

			var delegate : LoginDelegate = new LoginDelegate( this );
			var loginEvent : LoginEvent = LoginEvent( event );
			delegate.login( loginEvent.loginVO );
}
```

La classe Login commande délègue l'appel à la classe LoginDelegate qui va appeler le service Http.

Le constructeur de la classe demande au ServiceLocator le service «loginServices» :

```
public function LoginDelegate(responder:IResponder) {
                        this.service=ServiceLocator.getInstance().getHTTPService("loginServices");
                        this.responder=responder;
}
```

L'appel sur le service :

```
var token:AsyncToken=service.send({username:loginVO.username, password:loginVO.password});
token.addResponder(this);
this.loginVO=loginVO;
```

Le LoginDelegate implemente la classe IResponder ce qui lui impose une méthode result :

```
public function result(data:Object):void {
        var event:ResultEvent=ResultEvent(data);
        if (event.result == "true") {
                responder.result(this.loginVO);
        } else {
                responder.fault(null);
        }
}
```

La classe vo Login.as permettra ensuite de faire en sorte que l'option Parameters de la classe MainChoice.mxml soit active ou non selon l'utilisateur connecté, dans le cas du fichier login.PHP montré plus haut, c'est pour l'utilisateur «admin» que l'option est activée.

Bouton de navigation et menu principal

Navigation

La barre de navigation de l'application contient un bouton «flèche gauche» qui permet de passer à l'état précédent et un bouton «home» qui permet d'accéder au menu principal.

Les actions générées par ces boutons sont des commandes de la classe StateCommande.

Par exemple, le code suivant permet d'accéder au menu principal :

```
public function stateToMain():void {
        var state:State=new State();
        state.state=ModelLocatorApp.STATE_MAIN;
```

```
		new StateEvent(state).dispatch();
}
```

Le troisième bouton qui n'est actif qu'après une saisie dans les écrans «Browse database» et «Analys et retrive similar cases» va permettre d'afficher la saisie qui a été faite pour permettre de comparer les paramètres saisis à la liste des résultats par exemple.

Le bouton qui permet d'utiliser le mode plein écran exécute la commande FullscreenCommand. Cette commande affiche une erreur «le mode plein écran n'est pas autorisé» avec certain Flash Player, et dans d'autre cas, seule la souris est utilisable et pas le clavier pour des raisons de sécurité.

Le dernier bouton concerne l'aide, il ne génère pas de commande mais profite du «Binding».

Le composant HelpIcon.mxml :

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Image click="showAlert()" xmlns:mx="http://www.adobe.com/2006/mxml" source="{
AssetManager.helpSmallIcon}" visible="{ENABLED}">
	<mx:Script>
		<![CDATA[
			import ch.unige.ria.talisman.view.AssetManager;
			import mx.controls.Alert;

			public var helpText:String="";


			[Bindable]
			public static var ENABLED:Boolean=false;

			private function showAlert():void {
				Alert.show(helpText, "help");
			}
		]]>
	</mx:Script>
</mx:Image>
```

Il contient une variable «Bindable» ENABLED. Un click sur le bouton d'aide sur la barre d'application lance la fonction suivante :

```
private function toggleHelp():void {
	HelpIcon.ENABLED=!HelpIcon.ENABLED;
}
```

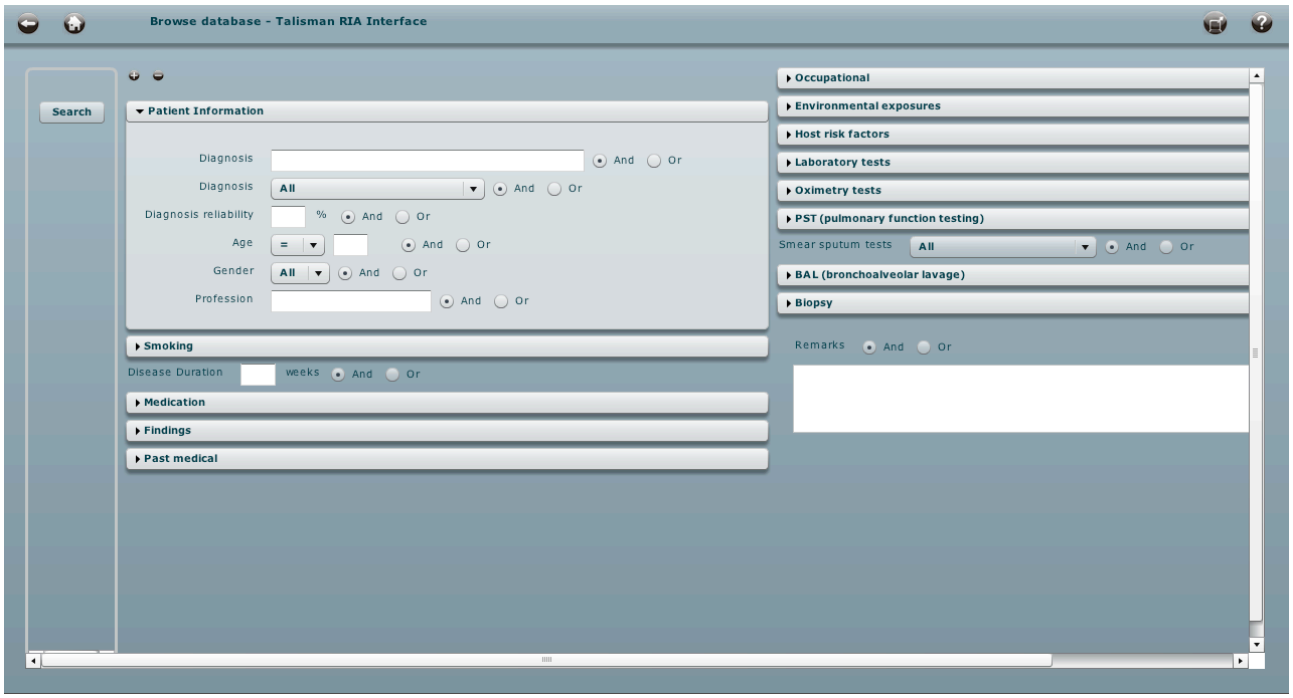
Le «Binding» va répercuter la modification sur tous les écrans contenant le composant HelpIcon.mxml automatiquement et les icônes d'aide vont s'afficher.

Menu principal

Le menu principal contient 3 options qui vont déclencher un événement State qui permettra d'afficher l'écran de récupération des cas existants «Browse database» ou bien l'écran qui permet de faire des similitudes avec les données de la base.

La troisième option permet d'ouvrir l'écran de modification des paramètres.

Récupération de cas existants - Browse database



La récupération des cas existants va permettre de composer une requête sur la base de données MySql en incluant tous les paramètres de la table Patient_data. Chaque champs est suivi d'un radio button «And Or» qui permettra d'inclure des conditions dans la composition de la requête.

L'événement généré sera une commande Patient_data avec un objet vo d'application PatientOperator.

```
var ev:Patient_dataEvent=new
Patient_dataEvent(Patient_dataEvent.PATIENT_DATA_ACTION_SEARCH_CRITERIA_PATIENT_DATA_OP
ERATOR, this);
ev.paramItemOperator=patientOperator;
ev.dispatch();
```

Cet Objet va permettre de gérer les radio button «and or» sélectionnés pour la requête.

La commande Patient_dataCommand fait partie des services remote object qui diffèrent un peu des services http.

Les classes de commande contiennent plusieurs commandes filtrées par les types d'événements.

Les delegates ne sont pas implémentés et un Model est spécialement créé pour faciliter la génération.

Les services remote objet demandent la création d'objet vo étroitement lié au table de la base MySql.

Coté serveur la classe Patient_data contient une méthode qui permet de créer la requête en fonction des paramètres passés.

```
/**
*
* The méthode getAllByCriteriaPatient_dataOperator...
*
*
**/

public function getAllByCriteriaPatient_dataOperator( $patient_dataOperator ) {
                $query =  "SELECT id, diagnosis,

                ...

                remarks  FROM patient_data";


                $queryTableCol="SHOW COLUMNS FROM patient_data";

                $resultCol = mysql_query($queryTableCol) or die(mysql_error());

                StdFunc::logString($sql);

                while ($column = mysql_fetch_row($resultCol)) {
                        $operatorFunc = $column[0];
                        $operatorFunc .= "Operator";

                        $condition = $patient_dataOperator -> $column[0];

                        if(!is_null($patient_dataOperator -> $column[0]) &&
                        ($patient_dataOperator -> $column[0]!= -9999)&&
                        !((strstr($column[0], "id_param") && intval($condition) == -1))) {


                                if (strlen(trim($where))>0){
                                        //OR == 1

                                        if ($patient_dataOperator -> $operatorFunc == 1){

                                                $where .= " OR ";
                                        } else {
                                                $where .= " AND ";

                                        }
                                }


                                if (strstr($column[1],"text")){
                                        $where .= "$column[0] LIKE '%" .$condition ."%'" ;
                                }else {
                                        if (strstr($column[0], "age")){
                                                $where .= "$column[0] " .$condition;
                                        } else{

                                                $where .= "$column[0] = " .$condition ;
                                        }
                                }
                        }
```

```
                }

                if (strlen($where)>0){
                        $query .= " WHERE " .$where;
                }


                $query .=";";
                $firstFlag = true;
                foreach( Patient_dataDao::$loaded as $key=>$value) {
                        if( ! $firstFlag ) {
                                $query = " AND ";
                        }
                        $query .= " [id] != $value->id ";
                        $firstFlag = false;
                }


                StdFunc::logString($query);

                $rid = mysql_query( $query, $this->dbConn );
                $ret = ( $this->returnQueryRows( $rid ) );

                        return( Patient_dataDao::$loaded );
        }
```

Le gros problème rencontré avec cette table est le nombre de ses paramètres.

La combinaison PHP et MySql va permettre dans cette méthode à l'aide de la requête :

```
"SHOW COLUMNS FROM patient_data";
```

qui retourne les champs suivants :

Field	Type	Null	Key	Default	Extra
id	int(20)	NO	PRI	NULL	auto_increment
input_record	tinyint(1)	NO		0	
diagnosis	text	NO			
id_param_diagnosis_category	int(3)	NO	MUL		
diagnosis_reliability	int(3)	NO		0	

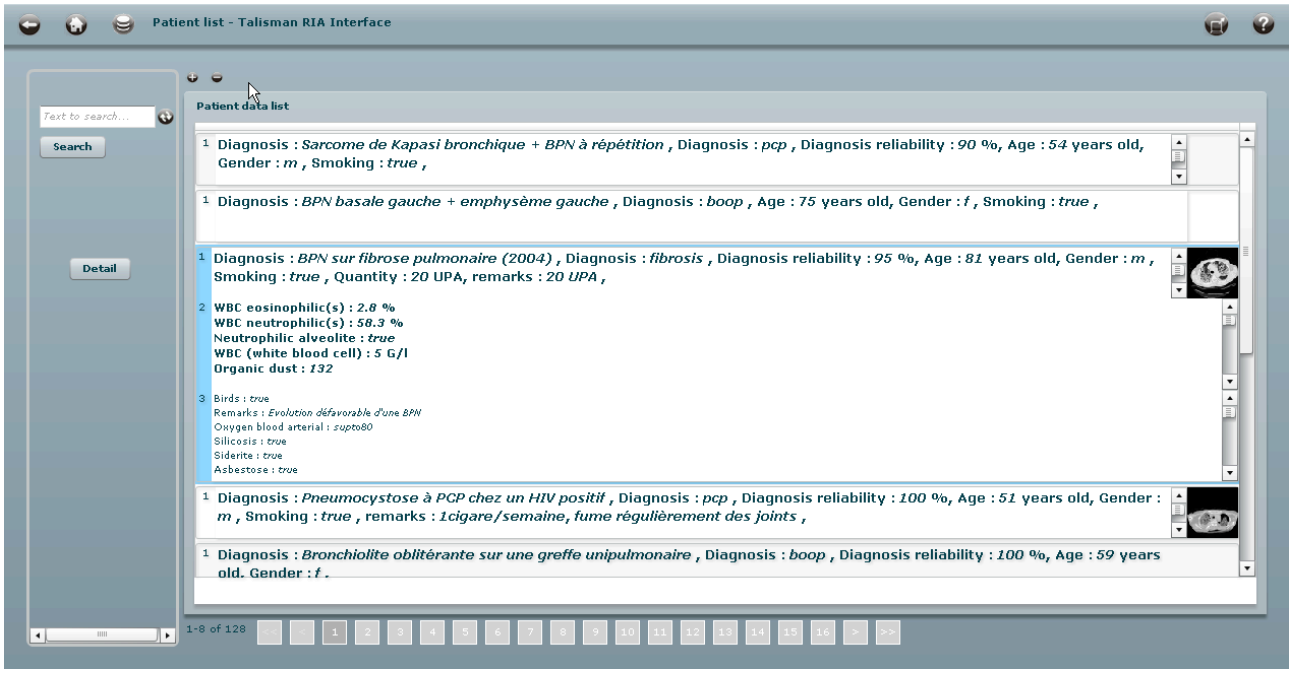
age	int(3)	NO		0
id_param_gender	int(3)	YES	MUL	NULL
profession	text	NO		
smoking_history	tinyint(1)	YES		NULL
smoking_quantity	int(5)	YES		NULL
smoking_quit	int(5)	YES		NULL
...	...	...		...

Ces champs vont permettre de connaître le type de données et de faire des appels avec les noms sur la classe patient_dataOperator passée en paramètre.

Par exemple un «LIKE» est appliqué aux conditions des champs text. Dans le cas des champs qui commencent par id_param un test peut-être fait pour gérer les cas de la sélection «all» dans les combobox des paramètres associés.

Affichage des résultats (multi-cas)

L'affichage des résultats se fait dans une liste trois états, ces trois états regroupent les informations des patients selon des priorités dépendantes des diagnostics.



Les priorités et ordre d'apparition de chaque champs sont définis dans la table Diagnosis_renderer.

Selon le diagnostic l'ordre des champs va être chargé et les champs seront affichés dans les trois états de la liste de patient.

Pour chaque champs à afficher, un label doit lui être attribué et parfois un texte suivant la valeur du champs, comme un «%» ou «years».

Pour associer ces valeurs aux champs et leur contenu, un structure XML est créée dans la classe Patient_data.

```
<element eid="root" dspName="root" dspEnd="" type="dir">
<element eid="General info" dspName="General info" dspEnd="" type="dir">
<element eid="diagnosis" dspName="Diagnosis" dspEnd="" type="prop"/>
<element eid="diagnosis_reliability" dspName="Diagnosis reliability" dspEnd="%" type="prop"/>
<element eid="id_param_diagnosis_category" dspName="Diagnosis" dspEnd="" type="prop"/>
<element eid="age" dspName="Age" dspEnd="years old" type="prop"/>
<element eid="id_param_gender" dspName="Gender" dspEnd="" type="prop"/>
<element eid="profession" dspName="Profession" dspEnd="" type="prop"/>
</element>
<element eid="smoking_history" dspName="Smoking" dspEnd="" enable="" type="dirWithProp">
<element eid="smoking_quantity" dspName="Quantity" dspEnd="UPA" type="prop"/>
<element eid="smoking_quit" dspName="quitted since" dspEnd="years" type="prop"/>
<element eid="smoking_remarks" dspName="remarks" dspEnd="" type="prop"/>
</element>
<element eid="disease_duration" dspName="Disease duration" dspEnd="years" type="prop"/>
<element eid="Medication" dspName="Medication" type="dir">
<element eid="medication_cordarone" dspName="Cordarone" dspEnd=""  type="dirWithProp">
<element eid="medication_cordarone_quantity" dspName="quantity" dspEnd="" type="prop"/>
<element eid="medication_cordarone_duration" dspName="duration" dspEnd="days" type="prop"/>
</element>
<element eid="medication_cyclines" dspName="Cyclines" dspEnd="" enable="" type="dirWithProp">
<element eid="medication_cyclines_quantity" dspName="quantity" dspEnd="" type="prop"/>
<element eid="medication_cyclines_duration" dspName="duration" dspEnd="days" type="prop"/>
</element>
<element eid="medication_cyclophosphamides" dspName="Cyclophosphamides" dspEnd="" enable="" ty-
pe="dirWithProp">
<element eid="medication_cyclophosphamides_quantity" dspName="quantity" dspEnd="" type="prop"/>
<element eid="medication_cyclophosphamides_duration" dspName="duration" dspEnd="days" ty-
pe="prop"/>
...
```

Cette structure va permettre aussi de hiérarchiser les informations.

Etat et priorité

Un «itemRenderer» est créé pour la liste des patients, il va permettre de gérer les états par priorité.

C'est un composant qui peut être lié à chaque ligne de liste des patients.

La première priorité contenant les champs les plus importants par rapport à un diagnostic dans le tag <mx:states> de la classe Patient_dataViewItemRenderer.mxml est représentée par un container <mx:HBox> contenant un label, du texte et une image.

```
<mx:HBox width="100%" doubleClickEnabled="true" doubleClick="toggleState()" horizontalGap="0">
            <mx:Label text="1" height="100%" textAlign="center" id="label1"/>
            <mx:TextArea htmlText="{data.getRendererPriorityOne()}" width="100%" id="txtAreaBase"
            height="100%" borderStyle="none"/>
            <mx:Image height="50" width="50" click="openImageLoaderWindow(event);"
            rollOverEffect="glowImage" rollOutEffect="unglowImage" id="image1">
            <mx:source>{getImagettePath(data.id)}</mx:source>
            </mx:Image>
</mx:HBox>
```

La deuxième priorité fait un add d'une <mx:HBox> contenant le deuxième texte :

```
<mx:State name="stateOne">
            <mx:AddChild>
            <mx:HBox width="100%" doubleClickEnabled="true" doubleClick="toggleStateOne()" height="90"
            horizontalGap="0">
            <mx:Label text="2" height="100%" width="15" textAlign="left"/>
            <mx:TextArea id="txtAreaExpanded" htmlText="{data.getRendererPriorityTwo()}" width="100%"
            height="100%" borderStyle="none"/>
            </mx:HBox>
            </mx:AddChild>
</mx:State>
```

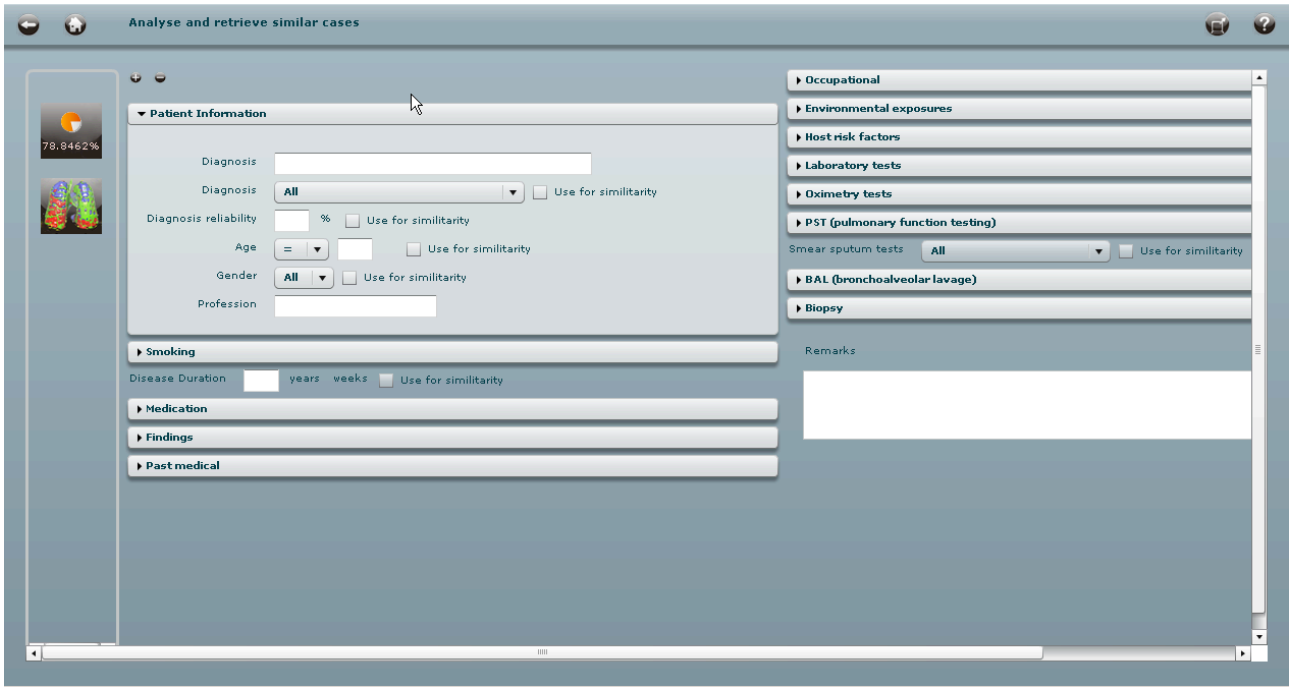
La troisième priorité est ajoutée :

```
<mx:State name="stateTwo" basedOn="stateOne">
<mx:AddChild position="lastChild">
            <mx:HBox width="100%" height="90" horizontalGap="0">
            <mx:Label text="3" height="100%" width="15"/>
            <mx:TextArea id="txtAreaExpandedDetail" htmlText="{data.getRendererPriorityTree()}"
              width="100%" height="100%" borderStyle="none"/>
                                        </mx:HBox>
                                </mx:AddChild>
                        </mx:State
```

Recherche des cas similaires - Analyse and retrieve similar cases

L'écran de recherche des cas similaires par rapport aux paramètres cliniques permet de sélectionner chaque paramètre qui sera utilisé pour une similarité, mis à par les chaînes de caractères, tous les paramètres peuvent être sélectionnés.

C'est la classe Patient_dataOperator qui va gérer les paramètres de la table Patient_data qui seront utilisés pour la similarité.



Calcul des similarités des données cliniques

Le calcule des similarités sur se fait en trois étapes, tout d'abord une étape d'initialisation d'une structure va permettre de donner un type et des valeurs min et max pour la normalisation des champs :

```
// structure for name field, simi, typeOfData, useForSimi, depend type -> min, max,
//  for type of data
//  0        String
//  1        int
//  2        boolean
//  3        float
//  4        id_param
if (patientField.length < 1) {
        patientField=new Array(147);
        patientField[0]=["diagnosis", 0.0, 0, 0];
        patientField[1]=["id_param_diagnosis_category", 0.0, 4, 0];
        patientField[2]=["diagnosis_reliability", 0.0, 1, 0, int.MAX_VALUE, int.MIN_VALUE];
        patientField[3]=["age", 0.0, 1, 0, int.MAX_VALUE, int.MIN_VALUE];
        patientField[4]=["id_param_gender", 0.0, 4, 0];
        patientField[5]=["profession", 0.0, 0, 0];
        patientField[6]=["smoking_history", 0.0, 2, 0];
        patientField[7]=["smoking_quantity", 0.0, 1, 0, int.MAX_VALUE, int.MIN_VALUE];
```

```
patientField[8]=["smoking_quit", 0.0, 1, 0, int.MAX_VALUE, int.MIN_VALUE];
patientField[9]=["smoking_remarks", 0.0, 0, 0];
patientField[10]=["disease_duration", 0.0, 1, 0, int.MAX_VALUE, int.MIN_VALUE];
patientField[11]=["medication_cordarone", 0.0, 2, 0];
patientField[12]=["medication_cordarone_quantity", 0.0, 0, 0];
patientField[14]=["medication_cyclines", 0.0, 2, 0];
patientField[15]=["medication_cyclines_quantity", 0.0,

...
```

La deuxième étape consiste à déterminer si les champs ont été sélectionnés pour la similarité et dans le cas des champs numériques de calculer et conserver les valeurs min et max.

```
public static const FIELD:int=0;
        public static const SIMI:int=1;
        public static const TYPE:int=2;
        public static const USE:int=3;
        public static const MIN:int=4;
        public static const MAX:int=5;
        public static const TPSTRING:int=0;
        public static const TPINT:int=1;
        public static const TPBOOLEAN:int=2;
        public static const TPFLOAT:int=3;
        public static const TPIDPARAM:int=4;

        //init structure
        initPatientField();
        //the first loop for the max and min value
        for each (var patient:Patient_data in ModelLocatorData.getInstance().patient_datas) {
                for each (var prop:Array in patientField) {
                        var tt:Object=this[prop[FIELD] + "Simi"];
                        if (this[prop[FIELD] + "Simi"] == "1") {
                                //set in the structure useSimi
                                prop[USE]=1;
                                //if type int or float max and min is used
                                if (prop[TYPE] == TPINT || prop[TYPE] == TPFLOAT) {
                                        if (patient[prop[FIELD]] > prop[MAX]) {
                                                prop[MAX]=patient[prop[FIELD]];
                                        }
                                if (patient[prop[FIELD]] < prop[MIN]) {
                                        prop[MIN]=patient[prop[FIELD]];
                                        }
                                }
                        }
```

Les valeurs du patient saisi pour la comparaison sont aussi prises en compte pour les valeures min et max de chaque valeur à comparer.

La seconde étape calcule la distance :

```
var valMin:Number=0;
var valMax:Number=Number.MIN_VALUE;
var val:Number=0.0;
//the second loop for the distance and max and min distance for each patient
for each (var patient:Patient_data in ModelLocatorData.getInstance().patient_datas) {
    val=0.0;
    for each (var prop:Array in patientField) {
      if (prop[USE] == 1) {
            if (patient[prop[FIELD]] != null) {
                  //if type is int or float
                  if (prop[TYPE] == TPINT || prop[TYPE] == TPFLOAT) {
                        //normalize the input value
                        var pente:Number=1 / (prop[MAX] - prop[MIN]);
                        var offset:Number=-prop[MIN] * pente;
                        var propNormInput=pente * this[prop[FIELD]] + offset;
                        //normalize the database value
                        pente=1 / (prop[MAX] - prop[MIN]);
                        offset=-prop[MIN] * pente;
                        var propNorm=pente * patient[prop[FIELD]] + offset;
                        val=val + ((propNormInput - propNorm) * (propNormInput - propNorm));
                        //if type is boolean or id_param
                  } else if (prop[TYPE] == TPBOOLEAN || prop[TYPE] == TPIDPARAM) {
                        //if values are equals the result is 0 else 1
                        if (this[prop[FIELD]] != patient[prop[FIELD]]) {
                              val=val + 1;
                        }
                  }
        }
    }
}
 var temp:Number=Math.sqrt(val);
  if (temp > valMax) {
   valMax=temp;

 Javapatient.distance=temp;
}
```

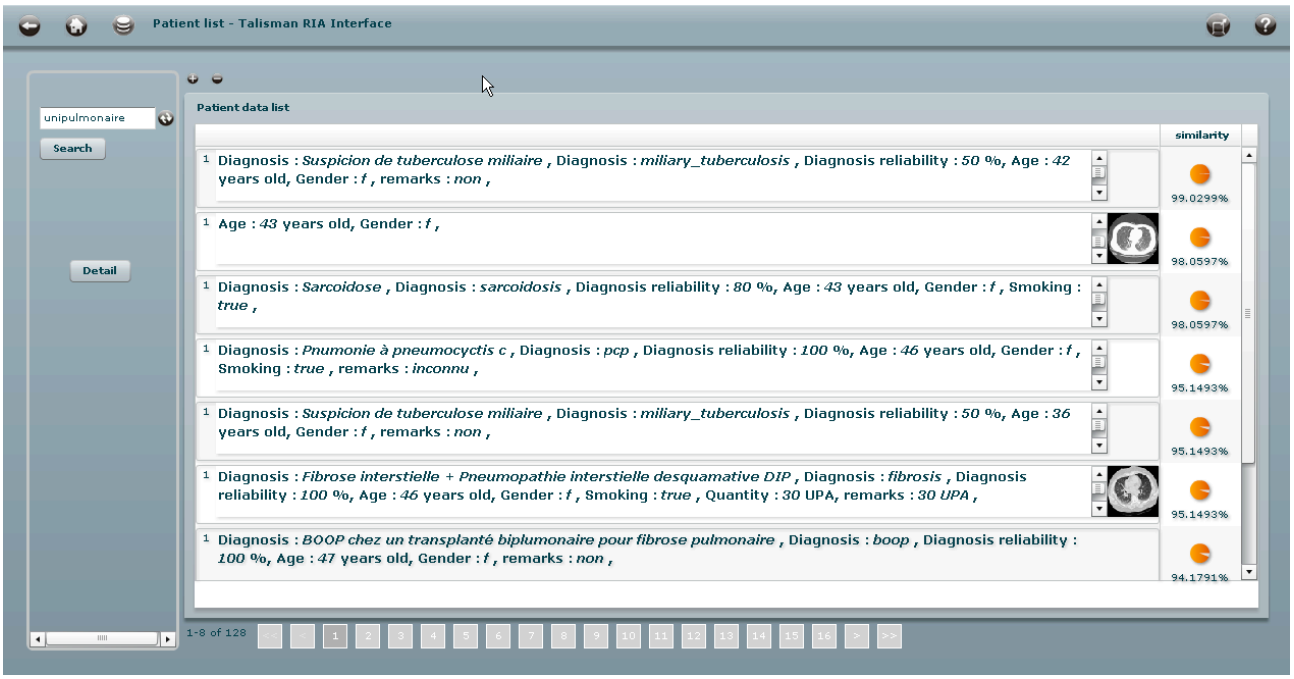
Les valeurs de similarité calculées avec les distances normalisés sont copiées dans un membre des instances des patients.

```
//normalize the value
var pente:Number=1 / (valMax - valMin);
var offset:Number=-valMin * pente;
//the similiarity with the normalized distance
for each (var patient:Patient_data in ModelLocatorData.getInstance().patient_datas) {
            patient.distance=Math.round(10000 * (100 * (1 - (pente * patient.distance + offset)))) / 10000;
}
```

La représentation des valeurs se fait avec un renderer dans la liste des patients.

Le renderer superpose deux «pie chart».

```
<degrafa:GeometryComposition id="barcomp" graphicsTarget="{[this]}">
<!--similarity % pie slice-->
<degrafa:EllipticalArc id="pie1" closureType="pie" arc="{(((dataValue1 / capacity)*100)*360)/100)}"
startAngle="0" height="19" width="19" x="30" y="15">
		<degrafa:fill>
			<degrafa:LinearGradientFill>
				<degrafa:GradientStop color="#FF9800"/>
				<degrafa:GradientStop color="#DB4105"/>
			</degrafa:LinearGradientFill>
		</degrafa:fill>
</degrafa:EllipticalArc>

<!--100% pie -->
<degrafa:EllipticalArc closureType="pie" arc="{(((dataValue2 / capacity)*100)*360)/100)}"
startAngle="{pie1.arc}" height="19" width="19" x="30" y="15">
		<degrafa:fill>
			<degrafa:LinearGradientFill>
				<degrafa:GradientStop color="#EEE"/>
				<degrafa:GradientStop color="#CCC"/>
			</degrafa:LinearGradientFill>
		</degrafa:fill>
</degrafa:EllipticalArc>
</degrafa:GeometryComposition>
<mx:Label text="{data['distance']}%" y="40"  horizontalCenter="0"/>
```



Calcul des similarités par images

Le lancement du calcul des similarités par images se fait en pressant sur l'icône représentant des poumons segmentés.

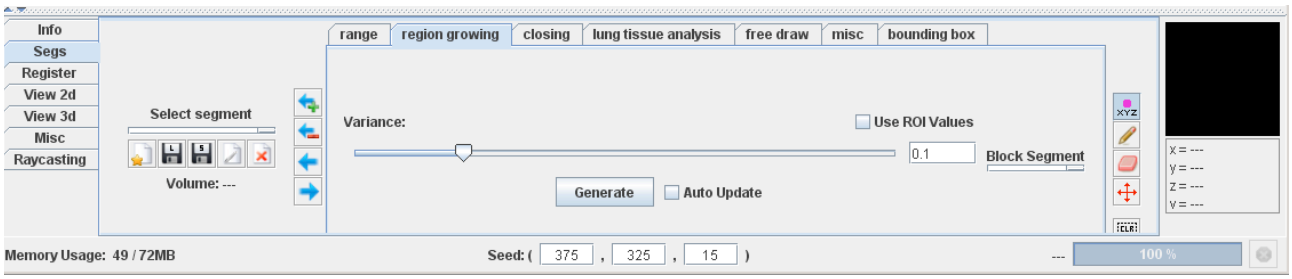
L'ouverture de l'applet YaDiV va se faire et une connexion par XMLSocket est établie entre l'applet et le client Flex qui fait office de serveur de socket.

Flex envoie un message à l'applet YaDiV contenant l'id de la série d'images qui est chargée automatiquement.

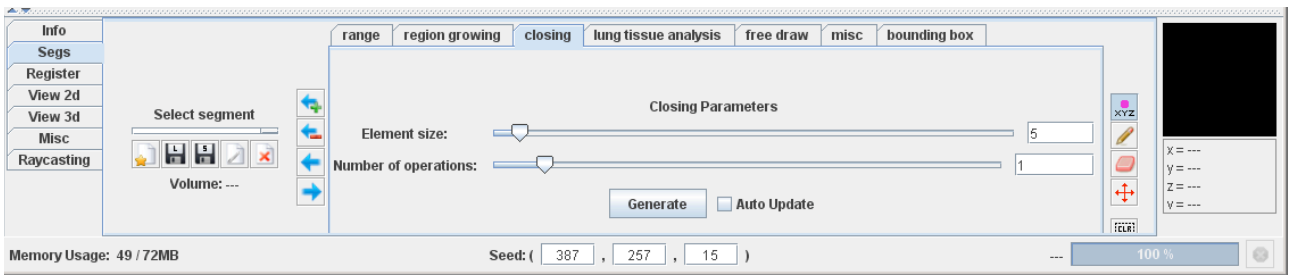
```
private function openYaDiVforSimilarity():void {
		initConnexion();
		send_msg();
}

private function send_msg():void {
		connexion.send("<msg>load new  </msg>");
}
```

Ensuite les différentes opérations et routines doivent être activées par l'utilisateur, il commencera par la sélection de la série d'images qu'il désire comparer ou analyser.

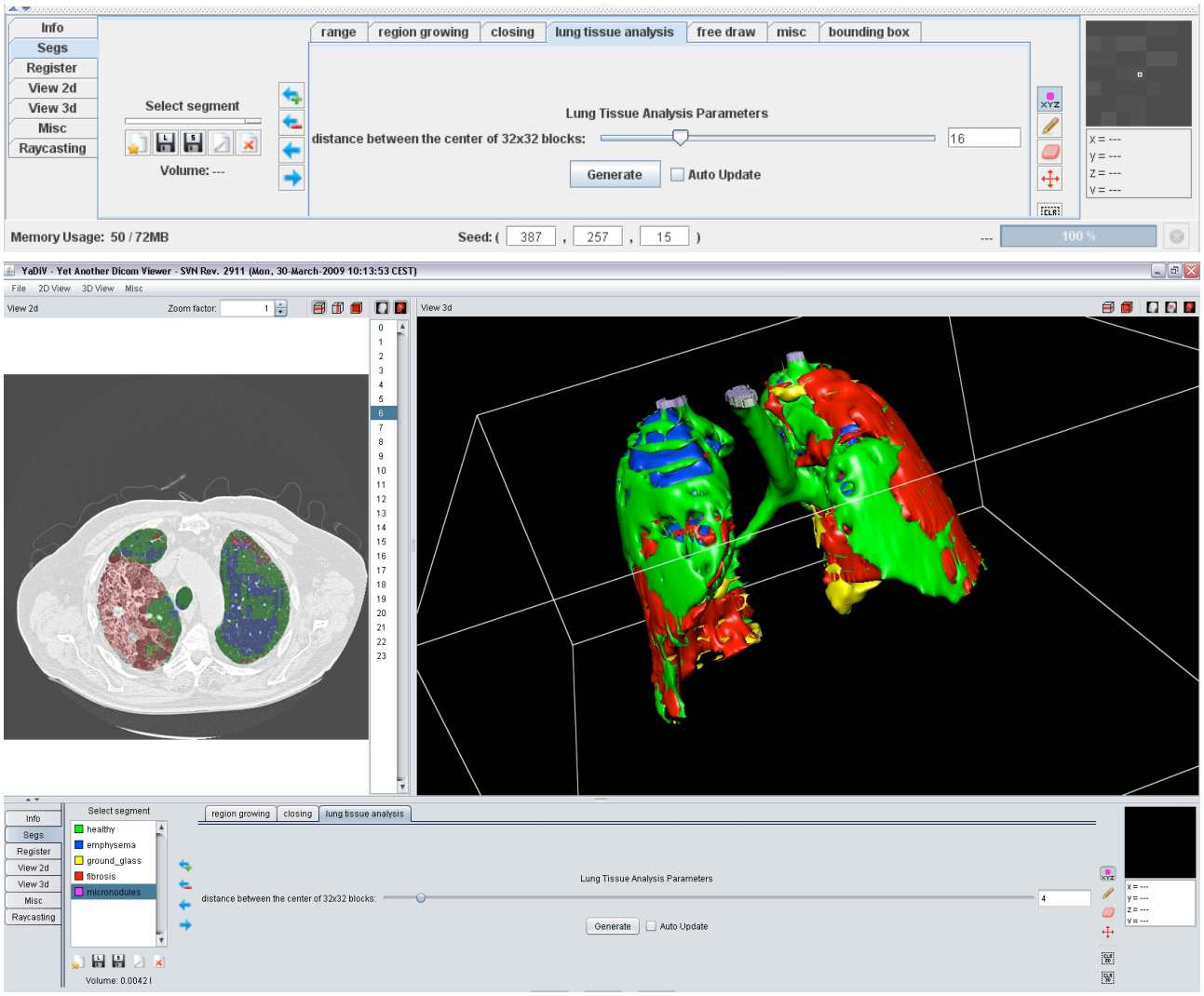
Un fois la série chargée, il lancera une opération de «croissance de région» pour créer un masque pour la segmentation.



Ensuite l'utilisateur utilisera la routine de «closing» qui va combler toutes les régions des poumons.



La dernière opération consiste à exécuter la routine de segmentation des régions pulmonaires.

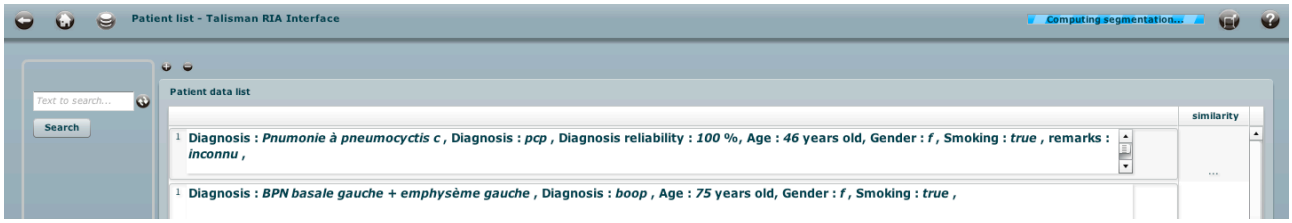


Une fois les poumons segmentés un message est envoyé de l'applet à Flex pour lui transmettre les 5 différents volumes des types tissus calculés.

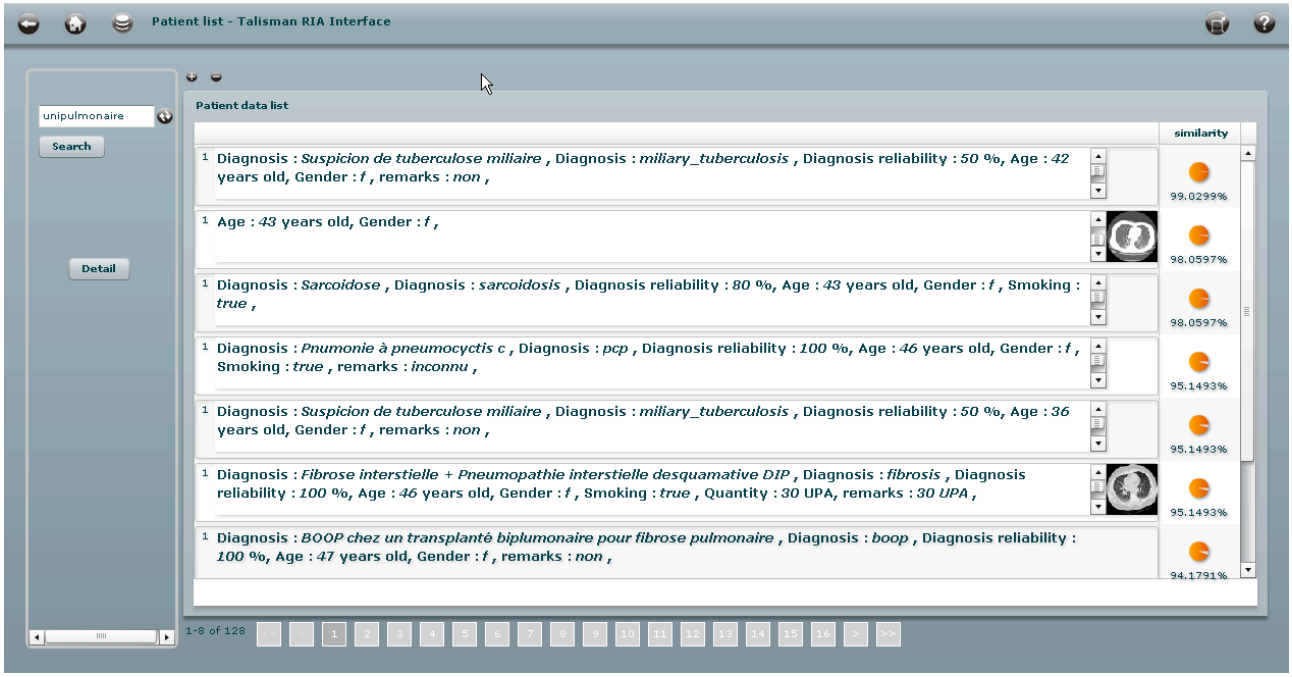
```
server.sendAll("<msg>" + "segmentation terminated" +
            <healthy> + fHealthy + "</healthy>" +
            <emphysema> + fEmphysema + "</emphysema>" +
            <healthy> + fHealthy + "</healthy>" +
            <ground_glass > + fGround_glass + "</ground_glass >" +
            <fibrosis> + fFibrosis + "</fibrosis>" +
            <micronodules> + fMicronodules + "</micronodules >" +
            "</msg>", "");
```

Dans le cas d'annulation de la segmentation dans l'applet YaDiV un message est aussi envoyé à Flex en indiquant simplement «segmentation aborded» dans le tag </msg>

Pendant la segmentation, qui peut prendre beaucoup de temps, l'utilisateur peut retourner dans Flex pour consulter d'autres information, une barre de progression le rend attentif aux calculs en cours.



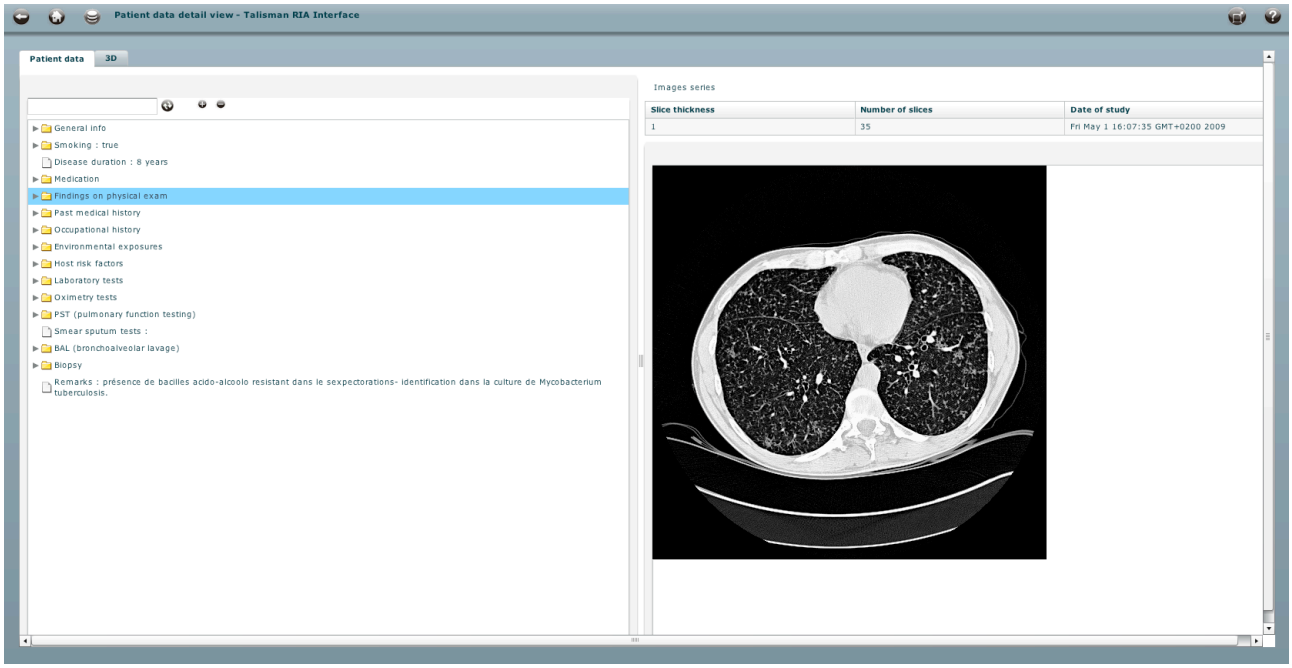
Après la fin de la segmentation et la réception des informations de l'applet YaDiV, Flex calcule et affiche les similarités par rapport aux volumes de tissus.



Ces informations sont utilisées de la même manière que les paramètres cliniques pour afficher les similarités par rapports aux cas enregistrés dans la base de données

Résumé de cas à afficher sur une page

Le détail d'un patient est composé d'un onglet Patient data contenant un arbre représentant tous les paramètres du patient, une table montrant la série d'images du patient et une image représentative de la série :



L'arbre est créé à partir de la structure XML citée plus haut.

Affichage de séries d'images DICOM

L'onglet «3D» va permettre de lancer l'applet YaDiV, l'applet YaDiV se charge avec la série d'images du patient.

Pour le chargement des images une connexion socket est établie avec YaDiV et le message suivant lui est envoyé :

```
private function openYaDiVforSimilarityNew():void {
		initConnexion();
		send_msg();
}

private function send_msg():void {
		connexion.send("<msg>load " + ModelLocator.getInstance().selectedPatient.getCtId() +
"</msg>");
}
```

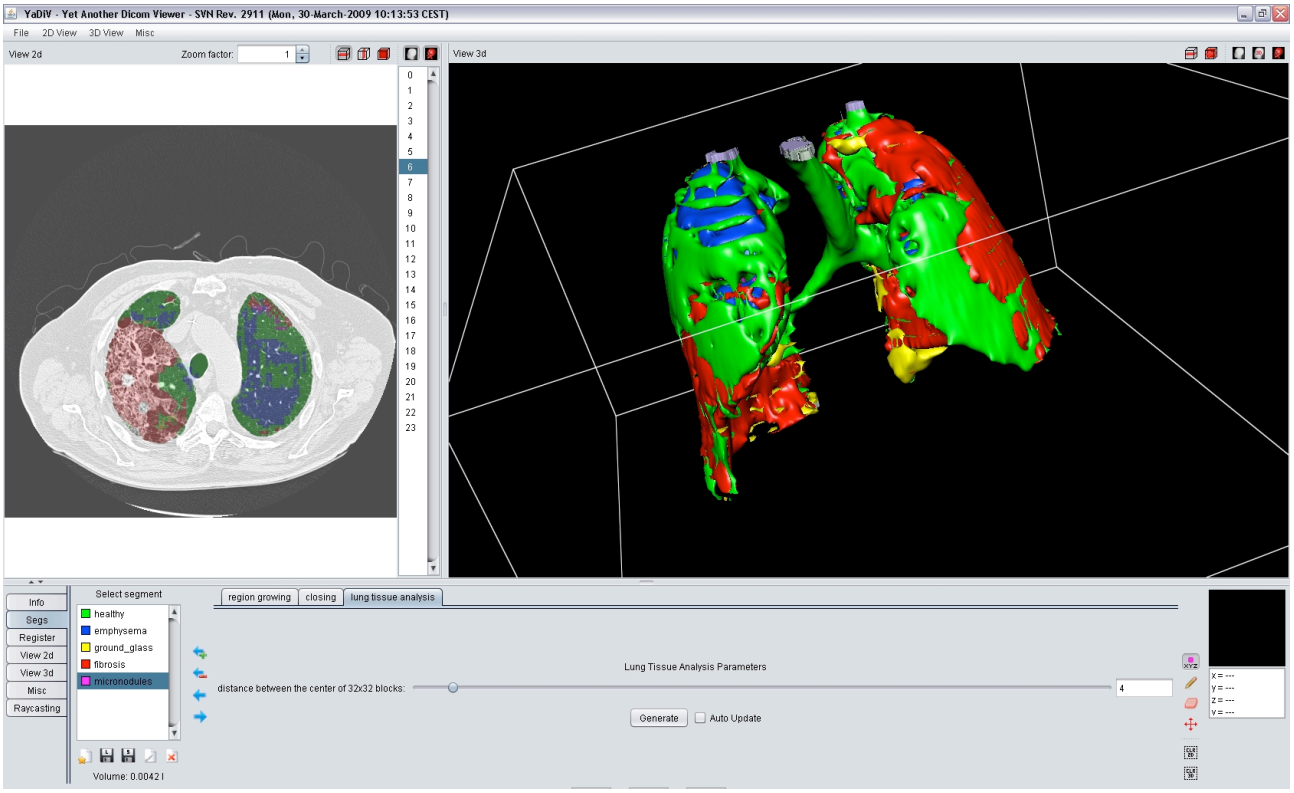
YaDiV se charge de rapatrier la série d'images du patient par rapport en faisant un download de la série d'images en fonction de id de la série passé en dans le message.

```
public static void downloadUnzipAndLoad(int id_CT) {
                        ...

        // Create the AMF connection.
        AMFConnection amfConnection = new AMFConnection();
        // Connect to the remote url
        String url = "http://medgift.unige.ch/talismanAmfServices/gateway.PHP";
```

Un appel de Java est fait sur les services amf pour récupérer le chemin sur le serveur de la série d'images et des segmentations.

```
        ...
        String seriesPath = "";
        ArrayList<String> segFiles = new ArrayList<String>();
        // Make a remoting call and send the result.
        try {
        //amfConnection.call("ch.unige.ria.talisman.RoisDao.getAllBy_Id_CT", id_CT);

        Object[] result = (Object[]) amfConnection.call("ch.unige.ria.talisman.CtDao.getAll", "");
        for (int i = 0; i < result.length; i++)
                if (((Ct)result[i]).getId_ct() == id_CT)
                        seriesPath = ((Ct)result[i]).getDicom_path();

                        result = (Object[]) amfConnection.call("ch.unige.ria.talisman.RoisDao.getAll",
                        "");
                        for (int i = 0; i < result.length; i++)
                                if (((Rois)result[i]).getId_CT() == id_CT)
                                        segFiles.add(((Rois)result[i]).getSeg_path());
        ...

                        // Download and unzip the image series at seriesPath
        ...
                        Download.download(url+seriesPath+"series.zip", localPath+"series.zip");
                        FileUtils.unzip(localPath, "series.zip");
        ...
```

Puis les images et les segmentations sont téléchargées, décompressées sur le poste client et enfin chargées dans YaDiV.



Conclusion

Dans l'ensemble les tâches du cahier des charges ont pu être effectuées, les modifications des structures de base et du format des régions d'intérêts et l'attente de la visionneuse 3D m'ont demandé d'analyser les situations et trouver plusieurs solutions afin d'assurer les fonctionnalités de l'application.

Les premiers exemples et tests des technologies à utiliser pour les différentes fonctionnalités de l'application se sont faites avec des petites tables, comme la table des utilisateurs ou la table de paramètres. Une fois le développement de l'application commencé, j'ai dû faire une remise en question complète de ma manière de concevoir cette application à cause de la table Patient_data qui contient un nombre très important de champs.

Tout d'abord ce nombre de champs a demandé de créer des structures pour hiérarchiser, typer et permettre d'afficher des labels avec les valeurs les champs.

D'autre part, du point de vue performance, j'ai dû abandonner certaines manières de faire et penser à en optimiser d'autre.

La phase de recherche qui a précédé toutes les étapes du développement, m'a permis de faire de nombreuses analyse entre le monde Flex et Java et de découvrir plusieurs aspects du développement web comme la sécurité et les problèmes de performances et de chargement.

La phase de développement qui s'est un peu précipitée à la fin du travail par l'arrivée tardive de YaDiV, m'a permis de me rendre compte que l'outil est stable et qu'il permet une adaptation très rapide.

Avant de commencer mon travail de Bachelor, j'étais très optimiste concernant Flex. L'expérience acquise avec cette application m'a encore renforcé dans ma position.

Mis à part des problèmes de performances dans certains domaines comme l'utilisation de XML ou la compilation.

Son utilisation en entreprise peut se faire de la même manière que Java pour le développement d'un client. Avec une version en plugin, Flex Builder 3 peut cohabiter avec Java et remplacer complètement la partie client Java.

L'application d'un framework pour la structuration, la modularité et la maintenance d'une application ne diffère encore une fois pas beaucoup de java mais apporte vraiment beaucoup pour Flex qui permet de mixer complètement le MXML et l'ActionScript.

Les outils de debugging, d'analyse, d'archivage et de test du code sont très similaires et sont inspirés des outils existants pour Java mis à part les outils.

Pour finir, il faut tenir compte qu'un investissement doit se faire pour la finalisation de cette application. Des tests approfondis sont nécessaires, principalement concernant la partie d'intégration de YaDiV.

Bilan personnel

Le bilan de ce travail de Bachelor est très positif. Ce travail m'a permis de découvrir une multitude de nouveaux outils et d'approfondir mes connaissances dans plusieurs domaines.

Déclaration sur l'honneur

Je déclare, par ce document, que j'ai effectué le travail de diplôme ci-annexé seul, sans autre aide que celle dûment signalée dans les références, et que je n'ai utilisé que les sources expressément mentionnées. Je ne donnerai aucune copie de ce rapport à un tiers sans l'autorisation conjointe du RF et du professeur chargé du suivi du travail de diplôme, y compris au partenaire de recherche appliquée avec lequel j'ai collaboré, à l'exception des personnes qui m'ont fourni les principales informations nécessaires à la rédaction de ce travail.

Saxon, le 3 mai 2009

Frédéric Gaillard

Remerciements

Je tiens à remercier Adrien Depeursinge, Henning Müller et Karl-Ingo Friese pour leur disponibilité et leur encadrement tout au long de ce travail et tout particulièrement mon amie pour sa présence et son soutien.

Bibliographie

Talisman

http://www.dim.hcuge.ch/medgift/01_Talisman_FR.htm

http://www.dim.hcuge.ch/03_Projets_FR.htm

YaDiV

http://www.welfenlab.de/forschung/gebiete/yadiv/

Outils utilisés

Mac OS X

http://www.adobe.com/products/flex/

http://www.mamp.info/

http://sourceforge.net/projects/smultron/

PC

http://www.eclipse.org/

http://www.easyphp.org/index.PHP

http://www.ultraedit.com/

Librairies

http://labs.kapit.fr/

http://labs.kapit.fr/

http://www.crazedcoders.com/

http://www.degrafa.org/

http://code.google.com/p/flexlib/

http://jwolib.riaforge.org/

ActionScript

Colin Moock (2008), Le meilleur d'ActionScript 3. - O'REILLY

http://livedocs.macromedia.com/specs/actionscript/3

http://flash.developpez.com/cours/as3/pratique-actionscript-3/

Acces Base

Connexion automatique entre Flex 3 et MySQL via PHP

http://julien-lestel.developpez.com/tutoriels/flex/connexion-avec-mysql-via-php/

Amf

http://fr.wikipedia.org/wiki/Action_Message_Format

http://labs.adobe.com/technologies/blazeds/

Cairngorm

http://www.cairngormdocs.org/tools/CairngormDiagramExplorer.swf

http://www.tylerbeck.com/CairngormCreator/

http://flex-actu.com/2008/04/cairngorn-par-lexemple/

http://www.davidtucker.net/2007/10/07/getting-started-with-cairngorm-–-part-1/

Coding Conventions

http://opensource.adobe.com/wiki/display/flexsdk/Coding+Conventions

DICOM

http://www.idoimaging.com/index.shtml

http://fr.wikipedia.org/wiki/DICOM

http://dicom.offis.de/dcmtk.php.en

http://www.dicomvcl.com/

DICOM et flash

http://dicom-flash-viewer.qarchive.org/

ImageJ

http://www.developpez.net/forums/d595929/environnements-developpement/eclipse/eclipse-java/eclipse-rcp/debutant-plugin-imagej/

http://www.developpez.net/forums/d449350/java/general-java/faire-plugin-imagej/

http://www.experts-exchange.com/Programming/Languages/Java/Q_23039357.html

http://rsb.info.nih.gov/ij/

http://www.itk.org/HTML/Documentation.htm

http://forums.java.net/jive/message.jspa?messageID=294107

Mango

http://ric.uthscsa.edu/mango/forum/

http://web.ift.uib.no/~erling/medimage.html

http://medical.nema.org/dicom/2007/

Flash to Flex

http://www.webkitchen.be/2008/05/05/the-flash-and-flex-marriage/

Flash and Custom interface

http://doloresjoya.com/blog/

Flex and dynamic load

http://www.adobe.com/devnet/flex/articles/link_load.html

http://www.dreaminginflash.com/2008/04/24/the-mxmlloader-class-dynamic-load-mxml/

http://mannu.livejournal.com/311003.html

http://manishjethani.com/blog/2008/04/02/the-mxmlloader-component/

Flex and Java

http://www.adobe.com/products/livecycle/dataservices/

http://www.actionscript.org/forums/showthread.php3?t=154858

http://scripts.chitika.net/eminimalls/728x90.html

http://flash.mediabox.fr/index.php?showtopic=89161

http://blogs.eyepartner.com/adrian/flex/create-combined-javaflex-project-in-flex-builder/

http://corlan.org/2008/06/05/creating-a-combined-flexjava-project-in-flex-builder-wo-lcdsblazeds/

http://weblogs.macromedia.com/dharfleet/archives/2006/08/calling_java_re.cfm

http://weblogs.macromedia.com/dharfleet/archives/2006/07/java_developmen.cfm

http://flash.mediabox.fr/index.php?showtopic=77421

http://www.actionscript.org/forums/showthread.php3?p=743094

http://codemoiunmouton.wordpress.com/flex/

http://www.artima.com/weblogs/viewpost.jsp?thread=193593

Flex and JavaScript

http://www.actionscript.org/forums/showthread.php3?p=803082

FAB

http://blog.dt.org/

http://www.kellyjo.com/blog/index.cfm/2007/2/20/Accessing-the-Local-File-System-with-Flex

http://ajaxian.com/archives/iris-example-of-combining-java-applets-and-ajax

http://www.browzor.com/blog/?cat=11

http://flash.mediabox.fr/index.php?showtopic=89161

http://javaboutique.internet.com/reviews/macro_flex/

http://www.myot.ca/blog/tag/flex/

Merapi

http://adamflater.net/blogassets/merapi/source/echo/source/Echo.mxml.html

Pagination

http://www.adobe.com/cfusion/communityengine/index.cfm?event=showdetails&productId=2&postId=9263

http://gurufaction.blogspot.com/2007/02/flex-datagrid-paging-example-with.html

http://www.boyzoid.com/blog/index.cfm/2007/5/10/Paginate-an-ArrayCollectionthe-easy-way

Securité

http://securingjava.com

Signed Applet

http://tobiaspatton.wordpress.com/2007/08/29/using-a-signed-java-applet-as-a-flex-helper-part-1/

Flex3

http://labs.adobe.com/wiki/index.php/Flex_3

Maven

http://www.mindtheflex.com/?p=61

Mvc

http://puremvc.org/component/option,com_wrapper/Itemid,160/

PixelMed

http://www.pixelmed.com/

http://www.developpez.net/forums/d417640/autres-langages/algorithmes/traitement-dimages/extraction-dimage-dicom-vers-jpeg-java/

Ressources Flex

http://flex-actu.com/category/flex/flex-ressources/

Components

http://lab.arc90.com/2008/03/collapsiblepanel.php#example

http://dougmccune.com/blog/2007/01/21/draggable-slider-component-for-flex/

http://examples.adobe.com/flex2/exchange/MaskedTextInput/MaskedTextInputExplorer/MaskedTextInputExplorer.html

http://lab.arc90.com/tools/shufflestack/

http://www.rubenswieringa.com/blog/statemanager

http://code.google.com/p/degrafa/

Tutorial

http://www.flex-tutorial.fr/2008/10/28/flex-uicomponent-les-composants-visuels-du-framework-flex-user-interface-components/

Flex Tools

http://merhl.com/flex2_samples/filterExplorer/

http://examples.adobe.com/flex2/inproduct/sdk/explorer/explorer.html

http://examples.adobe.com/flex2/consulting/styleexplorer/Flex2StyleExplorer.html

http://flexonrails.net/stylescreator/public/

http://www.flexibleexperiments.com/Flex/PrimitiveExplorer/Flex2PrimitiveExplorer.html

Annexes

NDA - YaDiV		-	LUH.pdf
Liste des priorité par diagnostic		-	ParamCliniquesImportants.pdf

CD

Contenu du CD

répertoire	Talisman Client	-	sources ActionScript - MXML
répertoire	Talisman Server	-	sources des Services PHP
répertoire	Documentation	-	Documentation de l'application
			TalismanRIA Pdf - Pages
répertoire	Annexes	-	Documentation sur les paramètres cliniques
			ParamCliniquesImportants Pdf
			Rapport sur le projet Talisman
			rapportTalismanAsmaaV4.Pdf
			NDA - YaDiV
			LUH.pdf